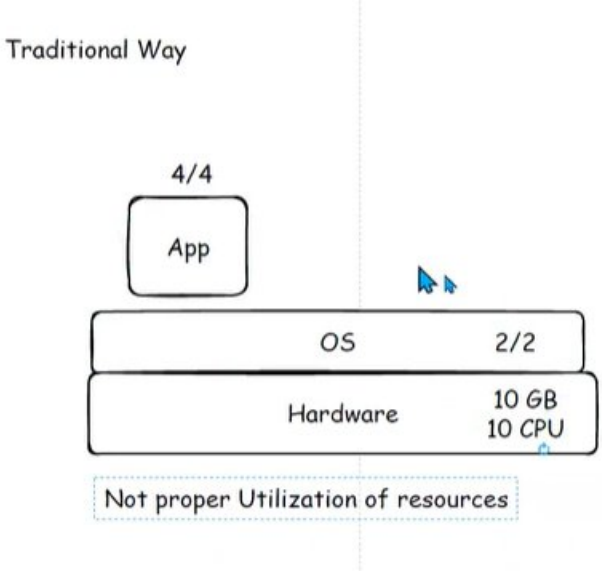
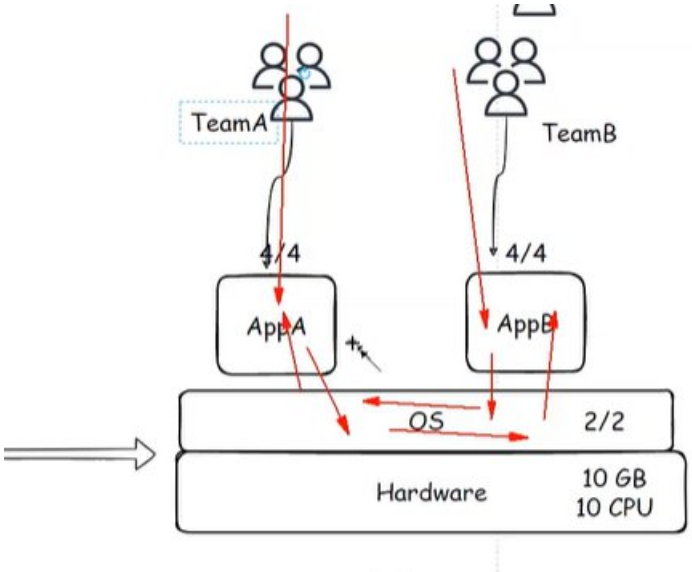
- So there is an app which should be running always for business
- So there are multiple ways to run it
- F**irst is traditional**, in this we will have a hardware where will have OS on this OS we would be running the app, and here we have total 10 GB, but app is using only 4 GB. so in this approach there will **no proper resource utilization**



Screen clipping taken: 05-08-2025 03:57 PM

- So in second scenario, we have 2 teams and as resources are shared both teams can have access to each other's app, **which is not secured**



Screen clipping taken: 05-08-2025 04:00 PM

Pros of traditional approach

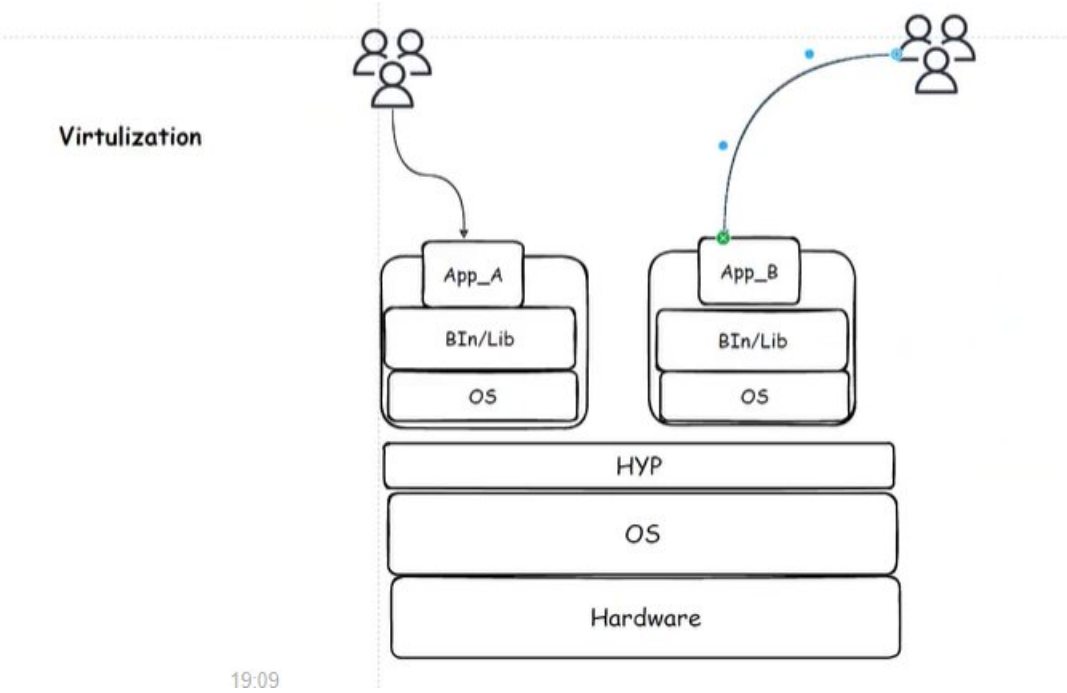
- Simple setup
- No abstraction- direct access to hardware
- Well suited for monolithic apps or legacy workloads

Cons of traditional approach

- Resource Wastage
- Poor isolation
- Scalability is a challenge
- Since isolation is not good we can face environment mismatch

Virtualization

- In virtualization we use hypervisor on a server and then we create multiple virtual machines on top of it



Screen clipping taken: 05-08-2025 04:10 PM

- The app users can't get access to hypervisor

Pros

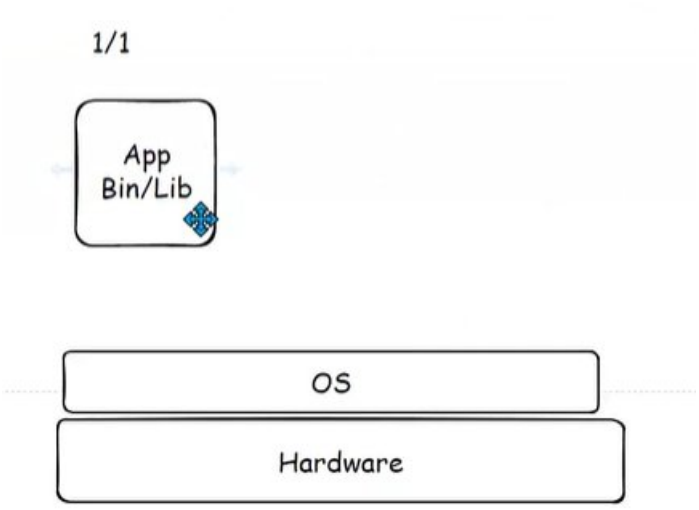
- Better isolation because of hyper-visor
- Improved security
- Infrastructure optimization

Cons

- Cost is higher
- Heavy weight as it comes with fully OS(it comes with all the apps)
- In this we were using the whole OS for an app which was making it slow

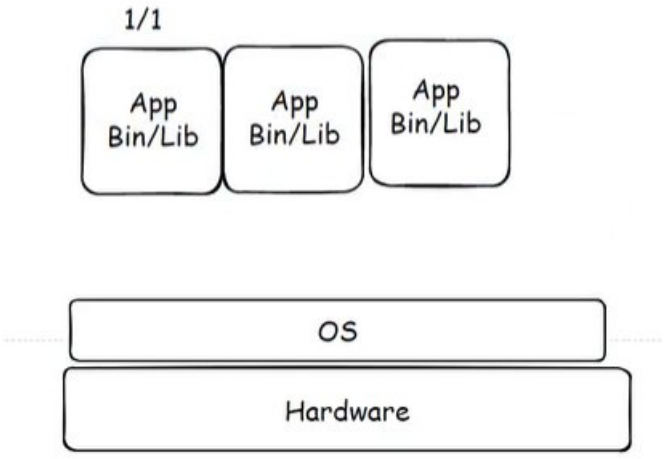
Container

- **A container is a lightweight, standalone,** executable software unit that packages code and all its dependencies so the application runs reliably across different computing environments.
- **In container we need only app and a few binaries which is required to run the code**



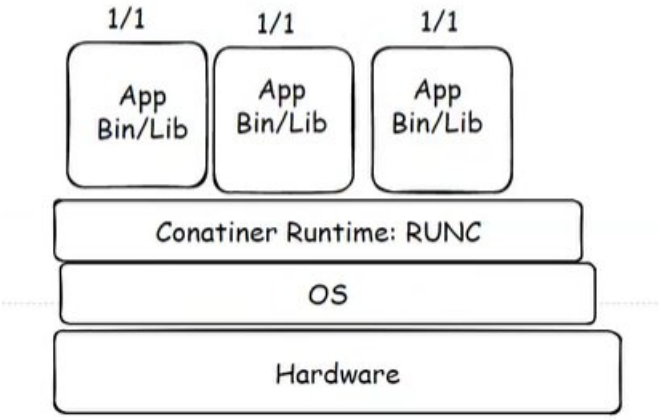
Screen clipping taken: 05-08-2025 04:30 PM

- **We can run more containers on same hardware**



Screen clipping taken: 05-08-2025 04:31 PM

- **For security we have one more layer called container runtime: RUNC(this could be anything docker, podman any container type)**
- It's the core tool responsible for creating and managing containers in many platforms, including Docker.



Screen clipping taken: 05-08-2025 04:35 PM

- Virtualization **uses full OS but container uses partial OS**
- And it use **less memory and storage** as compared to VM on Hyper-V
- We can **migrate app from one hardware to another hardware** as it is not dependent on OS

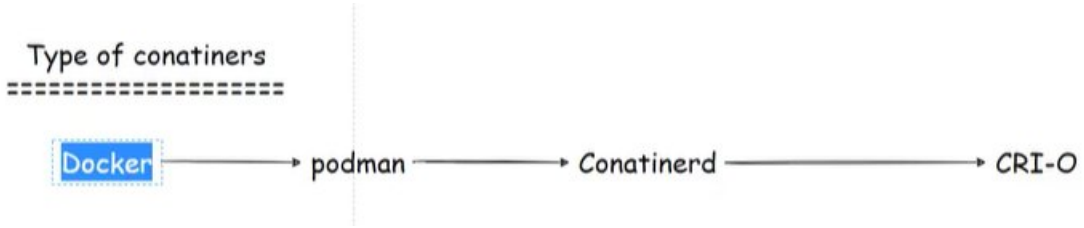
Pros

- Lightweight as containing only few binary and libraries
- Fast startup
- Portability (
- We can migrate app from one hardware to another hardware as it is not dependent on OS)

Cons

- Less isolation than VM as they share the OS

Typer of Containers

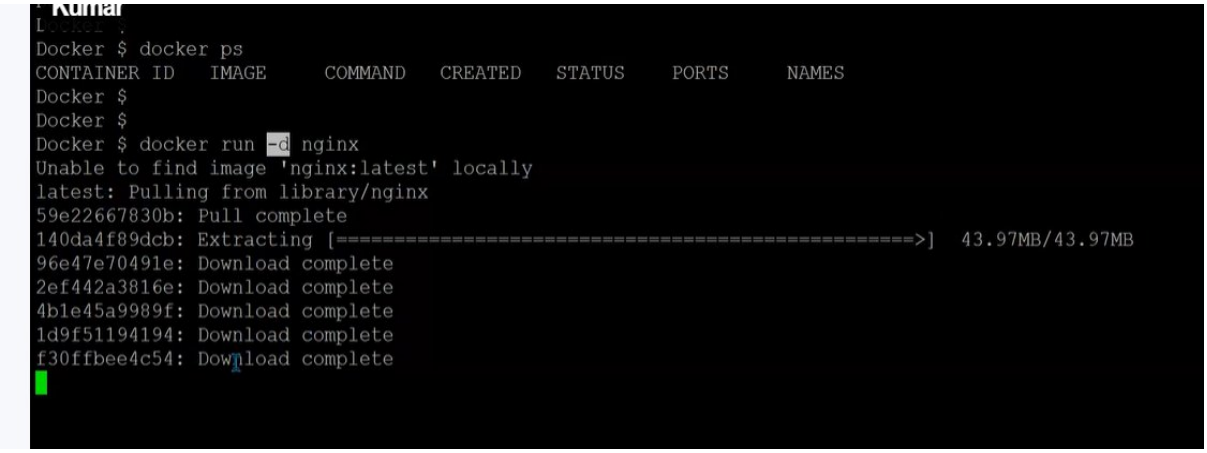


Screen clipping taken: 05-08-2025 04:51 PM

We will discuss docker

- We will install docker, we can install it anywhere, we will install docker on Linux machine
- **Docs.docker.com** is the best place to find anything about docker
- We can use play with docker website to learn about docker

  https://www.docker.com/play-with-docker/
- In lambda we just run the code irrespective of the hardware
- Now to use docker the syntax is **docker + verb**
- **Ps will list the containers**
- **Will install docker with nginx**



Screen clipping taken: 06-08-2025 01:11 PM

- Now will check the container in the list

```
Docker $
Docker $ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
39b3c6777f84   nginx     "/docker-entrypoint.…"   19 seconds ago   Up 18 seconds   80/tcp    eager_black
Docker $
```

Screen clipping taken: 06-08-2025 01:19 PM

- So here name container will pick its own and it will be always funny
- Now we will open a new session and will check the nginx process status , will use **ps -aux | grep nginx**

```
root        3399  0.0  0.0   7076  2072 pts/3    S+   05:44   0:00 grep --color=auto Nginx
root@MyDockerLab:~# ps -aux | grep nginx
root        3250  0.0  0.0  11468  7672 ?        Ss   05:41   0:00 nginx: master process nginx -g daemon off;
message+    3299  0.0  0.0  11932  3268 ?        S    05:41   0:00 nginx: worker process
message+    3300  0.0  0.0  11932  3268 ?        S    05:41   0:00 nginx: worker process
message+    3301  0.0  0.0  11932  3272 ?        S    05:41   0:00 nginx: worker process
message+    3302  0.0  0.0  11932  3272 ?        S    05:41   0:00 nginx: worker process
root        3401  0.0  0.0   7076  2076 pts/3    S+   05:44   0:00 grep --color=auto nginx
root@MyDockerLab:~#
```

Screen clipping taken: 06-08-2025 01:29 PM

- Now will force kill the process by using **kill -9 process Id**

```
root@MyDockerLab:~#
root@MyDockerLab:~# kill -9 3250
root@MyDockerLab:~#
```

Screen clipping taken: 06-08-2025 01:35 PM

- Now will check process again

```
root@MyDockerLab:~#
root@MyDockerLab:~# kill -9 3250
root@MyDockerLab:~# ps -aux | grep nginx
root        3453  0.0  0.0   7076  2036 pts/3    S+   05:46   0:00 grep --color=auto nginx
root@MyDockerLab:~#
```

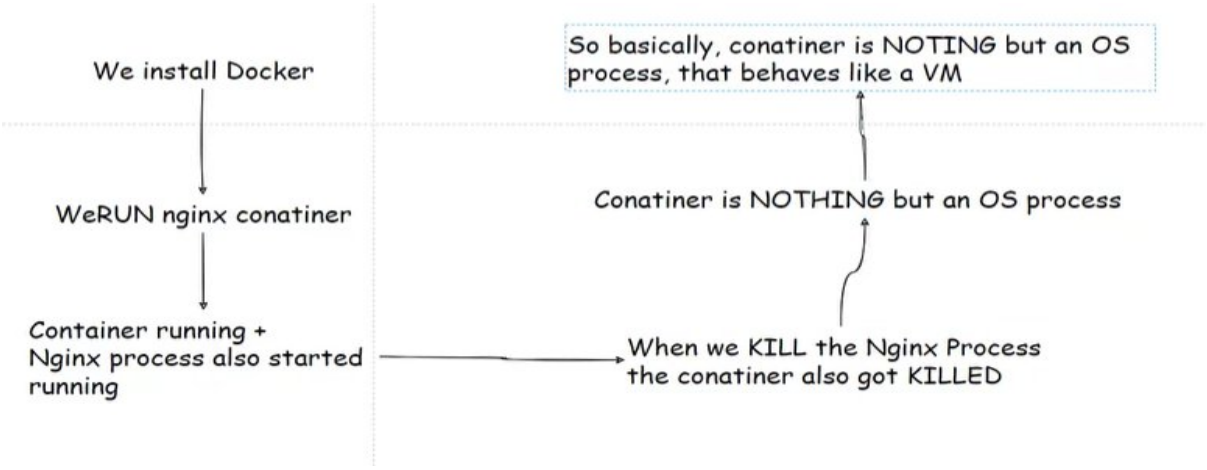
Screen clipping taken: 06-08-2025 01:36 PM

- And if check the docker will also be dead

```
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
39b3c6777f84   nginx     "/docker-entrypoint.…"   4 minutes ago   Up 4 minutes   80/tcp    eager_black
Docker $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
Docker $
```

Screen clipping taken: 06-08-2025 01:37 PM

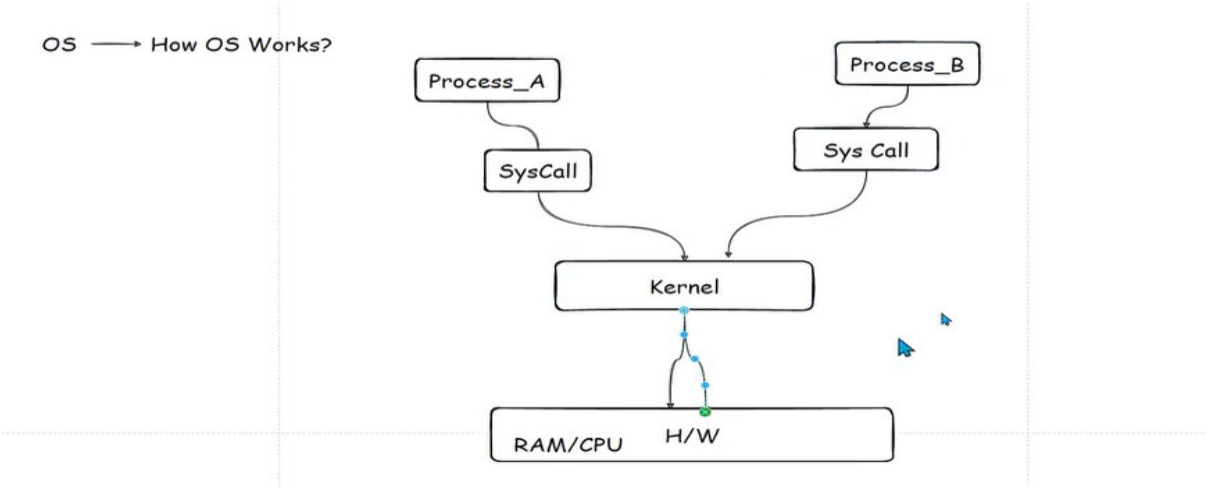
- So as per it the container is nothing but an OS process that behaves like a VM



Screen clipping taken: 06-08-2025 01:41 PM

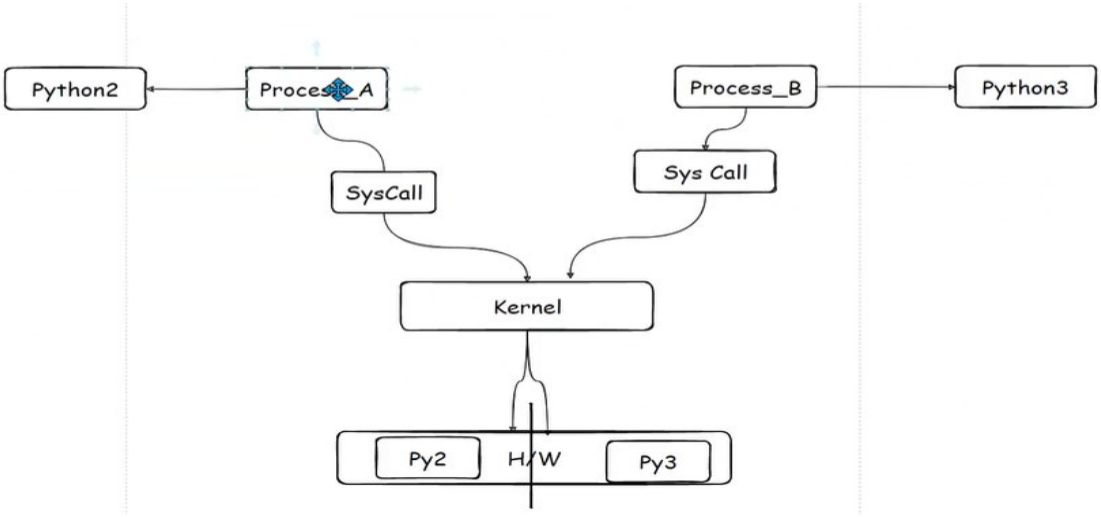
<u>How OS works</u>

- So we have a hardware which contains RAM,CPU and etc, so here , we run any command which runs the process so the process actually need hardware in order to run it
- But this won't get direct access to the hardware this will take the help of kernel who gives process and access to OS, kernel is the heart of OS, but process don't connect directly with kernel there is another processing in between called syscall.



Screen clipping taken: 06-08-2025 01:47 PM

- Suppose process A and process B needs python2 and python3 respectively, and we can't install both on the hardware, so in this we can have partition between hardware so that we can cater the requirement as per the process and this feature is known as Namespaces
- **Namespaces** are a feature of the linux kernel that partition kernel resources such that one set of processes sees one set of resources, while another set of processes sees a different set of resources.



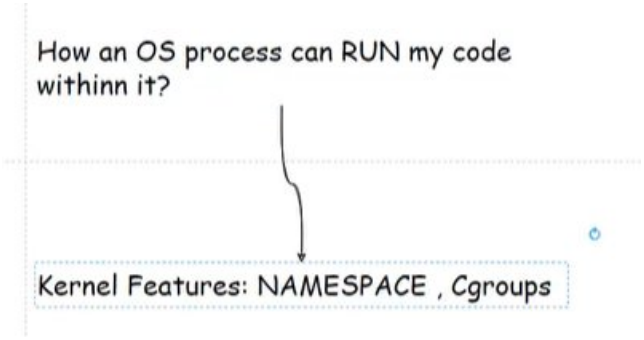
Screen clipping taken: 06-08-2025 01:53 PM

- We use one more feature called **Cgroup** in containers to partition the harddisk for better utilization for resources, here capping size will be decided by
- **cgroup (short for *control group*) is a Linux feature that helps you control how much system resources a process can use.**

Think of it like a manager that says:

“This app can only use 512MB of memory and 50% of the CPU.”

- These features makes container fast



Screen clipping taken: 06-08-2025 01:56 PM

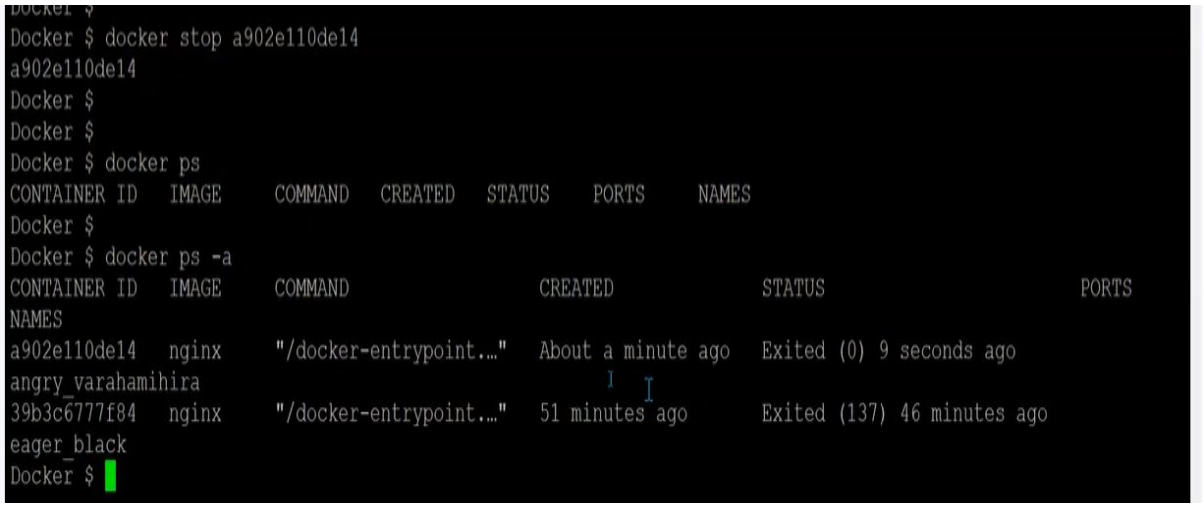
- **So Container Is nothing but an OS process, that behaves like a VM using kernel features namespace and cgroup**

Docker CLI -Core Commands

- Syntax -- Docker + verb + options
- To create a container we will use **docker run Image Name**
- Docker ps will list down the containers
- Docker **ps -a will show running and stopped containers**

Screen clipping taken: 06-08-2025 03:22 PM

- To stop/start we can use **docker stop/start container ID/name**

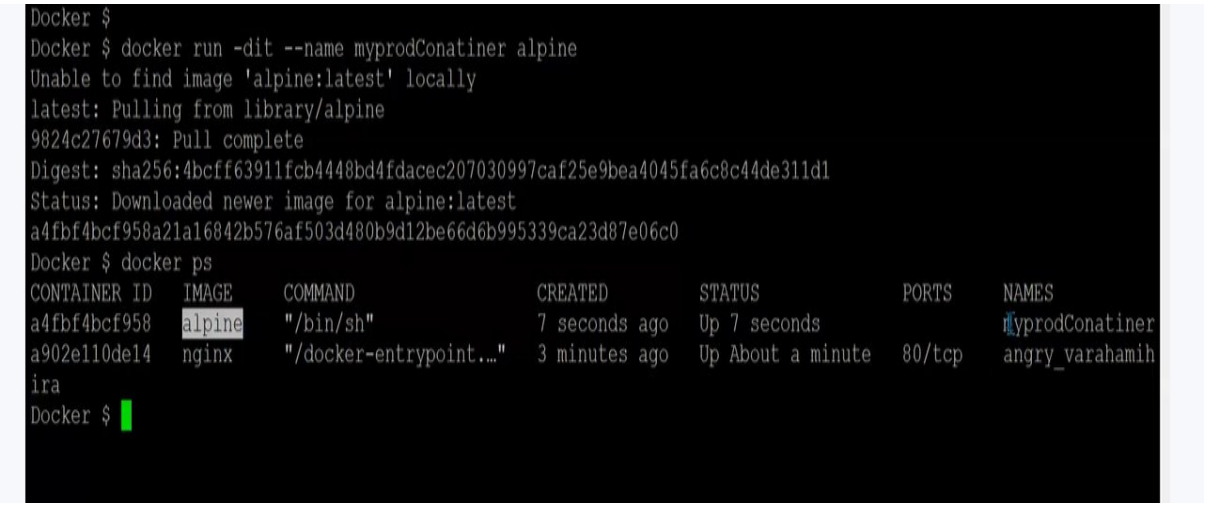


Screen clipping taken: 06-08-2025 03:24 PM

- To check images use **docker images**

Screen clipping taken: 06-08-2025 03:29 PM

- We will create one more container **docker run -dit --name myprodcontainer alpine**



Screen clipping taken: 06-08-2025 03:30 PM

- To delete the container, **first stop**

Screen clipping taken: 06-08-2025 03:31 PM

- Now we will delete using **docker rmi <docker name>**
- N**ow to remove the images, first we will remove the container**

```
Docker $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
alpine       latest    9234e8fb04c4   2 weeks ago   8.31MB
nginx        latest    2cd1d97f893f   2 weeks ago   192MB
Docker $
Docker $
Docker $ docker rmi nginx
Error response from daemon: conflict: unable to remove repository reference "nginx" (must force) - container 39
b3c6777f84 is using its referenced image 2cd1d97f893f
Docker $ docker rm 39b3c6777f84
39b3c6777f84
Docker $ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS         PORTS     NAMES
a4fbf4bcf958   alpine    "/bin/sh"   3 minutes ago   Up 3 minutes             myprodConatiner
Docker $
Docker $
Docker $ docker rmi nginx
```

Screen clipping taken: 06-08-2025 03:34 PM

```
Kumar docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS         PORTS     NAMES
a4fbf4bcf958   alpine    "/bin/sh"   3 minutes ago   Up 3 minutes             myprodConatiner
Docker $
Docker $
Docker $ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS         PORTS     NAMES
a4fbf4bcf958   alpine    "/bin/sh"   3 minutes ago   Up 3 minutes             myprodConatiner
Docker $
Docker $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
alpine       latest    9234e8fb04c4   2 weeks ago   8.31MB
Docker $
```

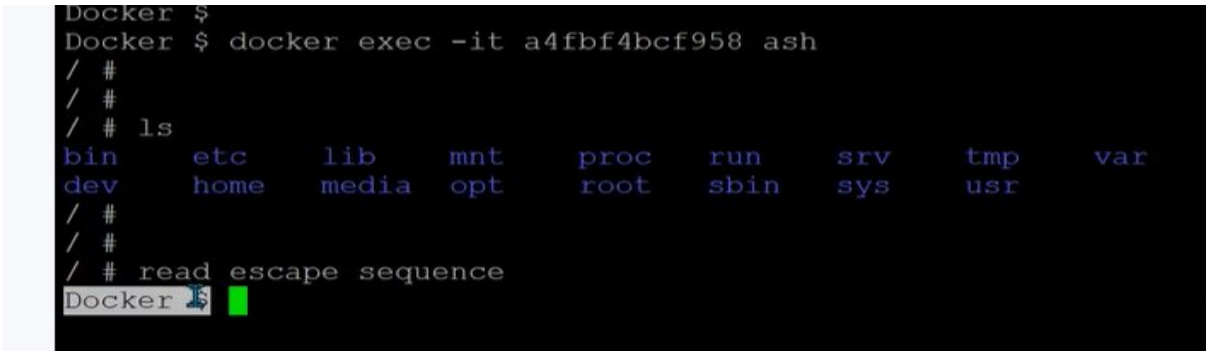
Screen clipping taken: 06-08-2025 03:34 PM

- To go inside the container use **docker exec -it <container id> ash**

```
Docker $
Docker $ docker exec -it a4fbf4bcf958 ash
/ #
/ #
/ #
```

Screen clipping taken: 06-08-2025 03:35 PM

- Here **it means interactive terminal**
- To come out use **ctrl+P+Q**



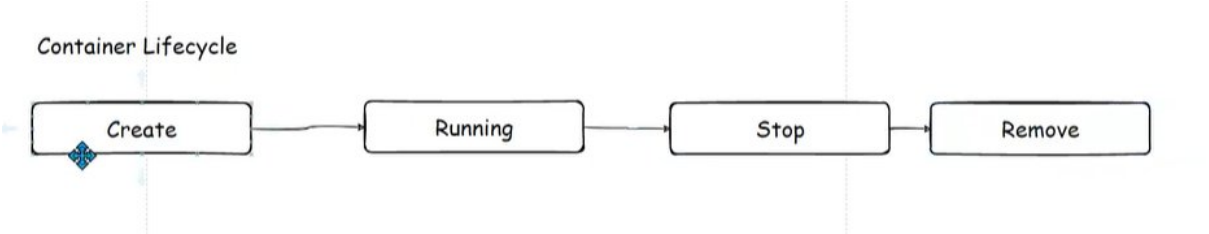
Screen clipping taken: 06-08-2025 03:39 PM

- To check logs use **docker logs <container name>**, no logs are generated yet

Screen clipping taken: 06-08-2025 03:41 PM

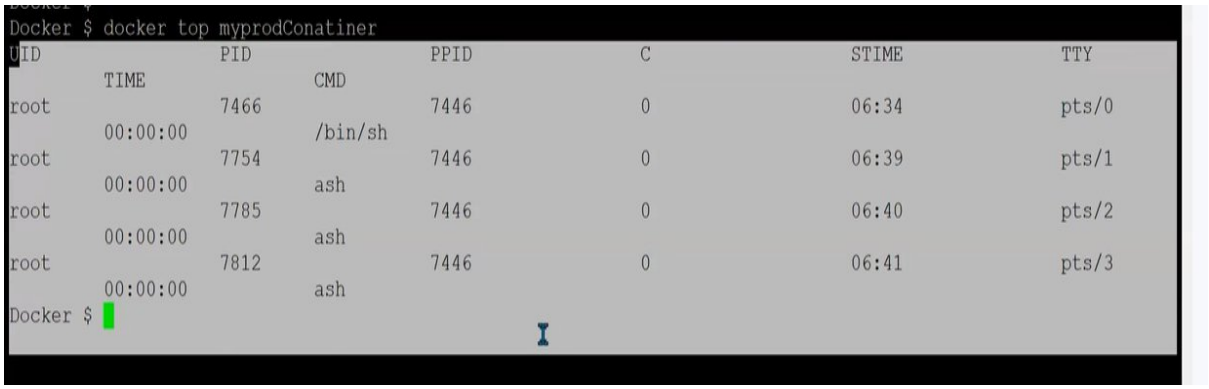
<u>Container Lifecycle</u>

- We will create the container then container will be running and then we will stop the container, and lastly delete the container



Screen clipping taken: 06-08-2025 03:43 PM

- To check which processes are running in the container we will use **docker top <container name>**

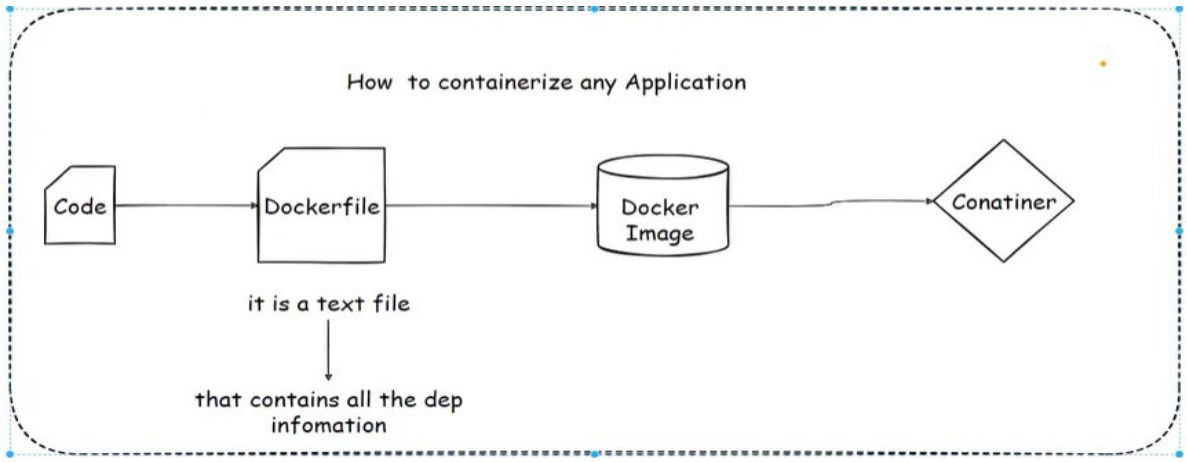


Screen clipping taken: 06-08-2025 03:44 PM

- To deep dive about the container use **docker inspect <container name>**

How to containerize any application

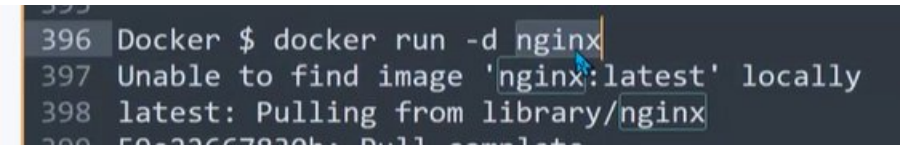
- **From code we create a docker file, right now it is a text file which contains all the dependencies info, so based on code docker file creates image like if code needs java it will create as per it , now docker file will create docker image , using this image we will create container**



Screen clipping taken: 06-08-2025 03:53 PM

Docker Architecture

- The first component is **docker CLI**, where we run all the commands
- Another component is **docker daemon**, so when we install the docker daemon process is started, so every command go to daemon in form of **REST API, and daemon reads it**
- Docker will look for this engine locally, which is why we get this error



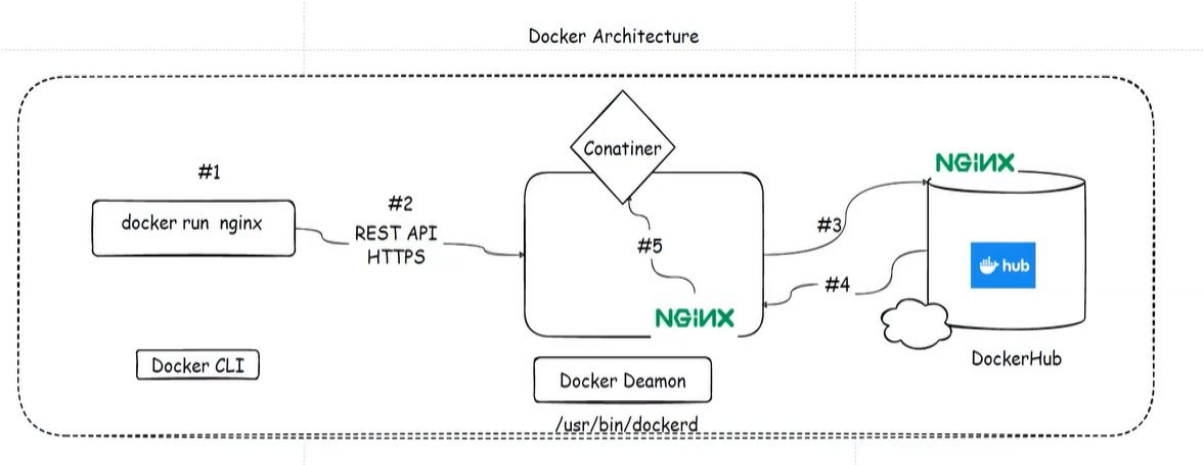
Screen clipping taken: 07-08-2025 10:11 AM

- Then there is another component **docker hub which is maintained by docker company , it is a repository which is available over the internet and contains all the images**
- We should always use official image from hub
- So then docker **deamon go to hub and download the image from there which is why we get this .message**

```
395
396 Docker $ docker run -d nginx
397 Unable to find image 'nginx:latest' locally
398 latest: Pulling from library/nginx
399 59e22667830b: Pull complete
400 140da4f89dcb: Pull complete
401 96e47e70491e: Pull complete
402 2ef442a3816e: Pull complete
403 4b1e45a9989f: Pull complete
404 1d9f51194194: Pull complete
405 f30ffbee4c54: Pull complete
406 Digest: sha256:84ec966e61a8c7846f509da7eb081c55c1d56817448728924a87ab32f12a72fb
407 Status: Downloaded newer image for nginx:latest
```

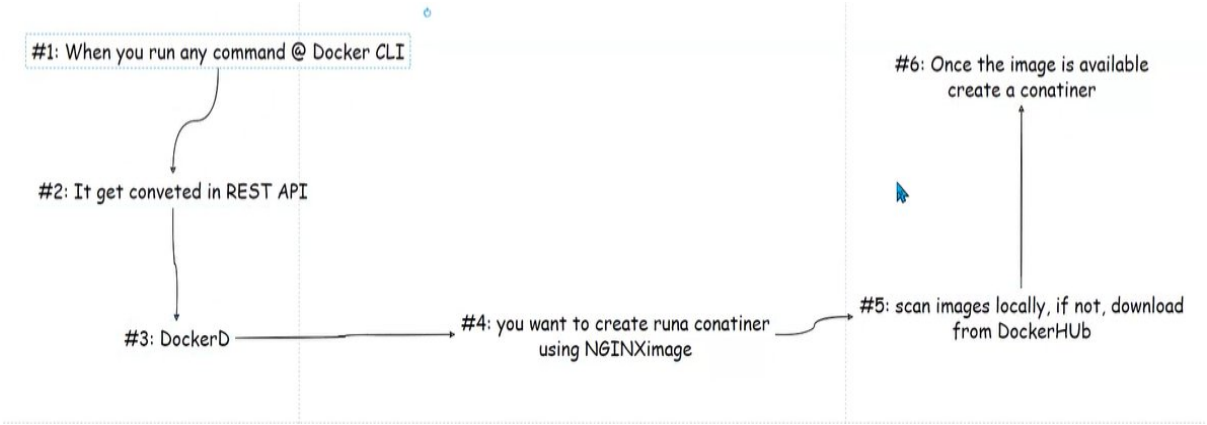
Screen clipping taken: 07-08-2025 10:14 AM

- Once we have the image the container will be created



Screen clipping taken: 07-08-2025 10:29 AM

- Steps when we create a docker



Screen clipping taken: 07-08-2025 10:19 AM

Docker File

- It is nothing but a simple text file that contains list of instructions how to build an image

Docker file has some set of commands

- All the instruction will be in **uppercase**
- The first instruction always start FROM

FROM: ----------> What is your base image

WORKDIR: ----------> working directory inside the container just like C drive for OS, it is not necessary but best practice to mention

COPY: copy to your image ---------->from local machine to ADD: ----------> copy with some extra

RUN:----------> Execute the command at build time of image

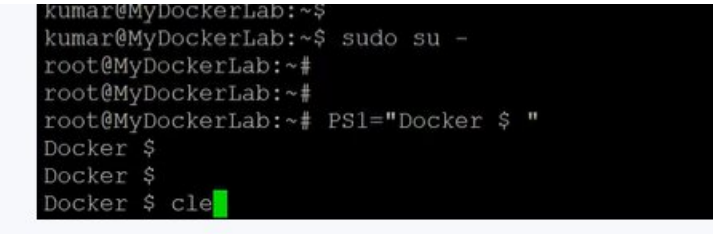
EXPOSE:----------> port opening

CMD:----------> when you launch a container what you want to start

==========

==========

- Docker file is a simple text file that contains set of instructions telling docker how to build and image.
- To name on the CLI use **PS1="Docker $ "**

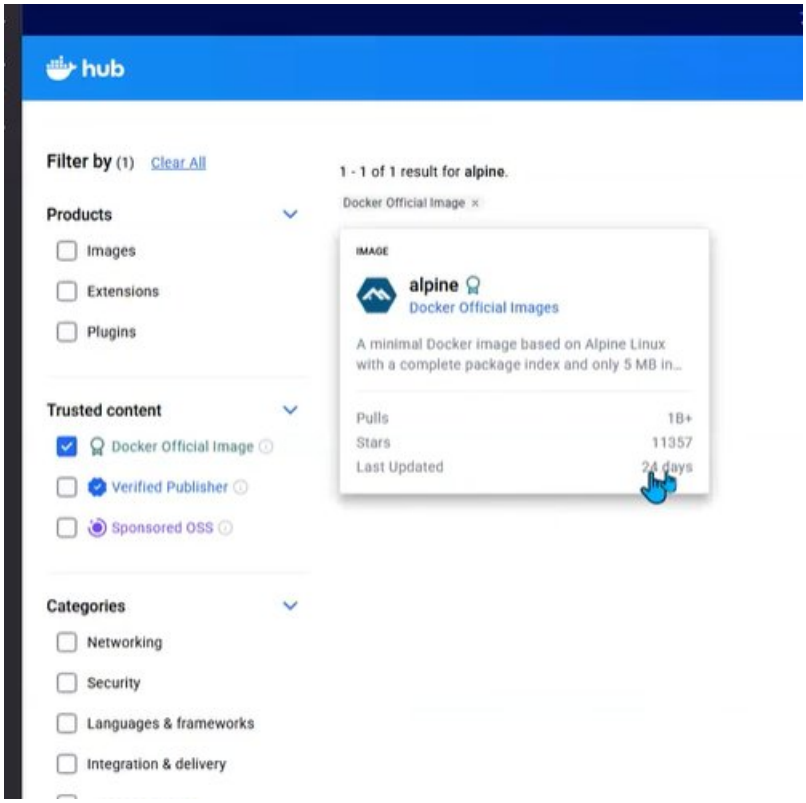


Screen clipping taken: 10-08-2025 05:56 PM

- To check images use **docker images**

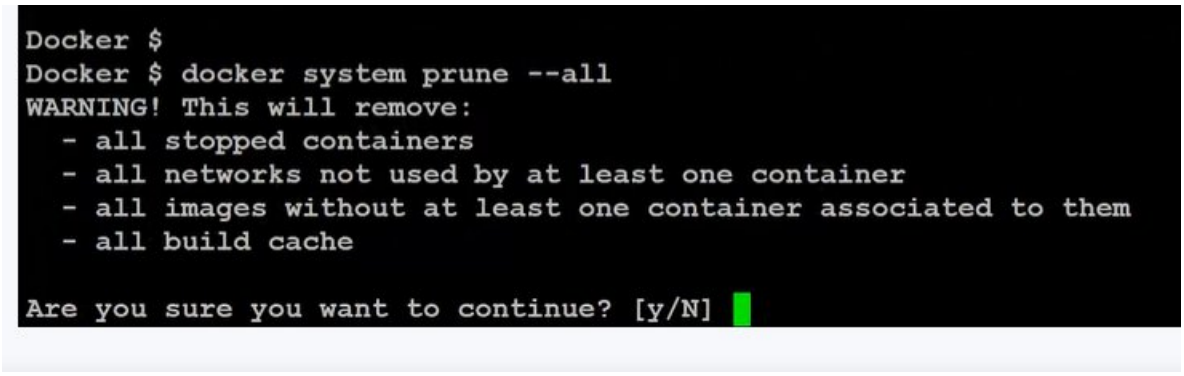
Screen clipping taken: 10-08-2025 05:57 PM

- Here this created time when images was created on docker hub
- To check image on docker hub



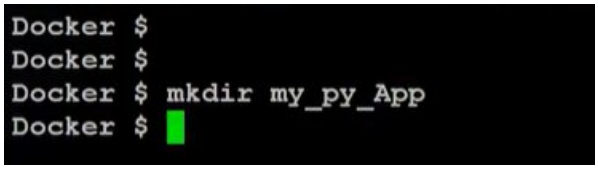
Screen clipping taken: 10-08-2025 05:59 PM

- To create our own image we will write the code
- To remove everything we will use **docker system prune --all**



Screen clipping taken: 10-08-2025 06:04 PM

- First will create a directory, in this we will create a code and it will be in **python** and in real time this will be provided by developer

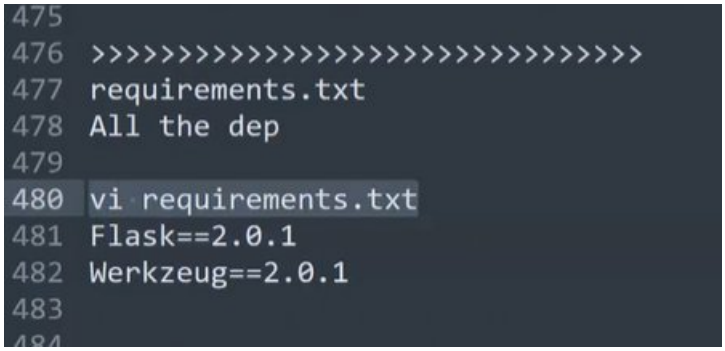


Screen clipping taken: 10-08-2025 06:07 PM

- Will cd to directory
- We will create a file and write the code below

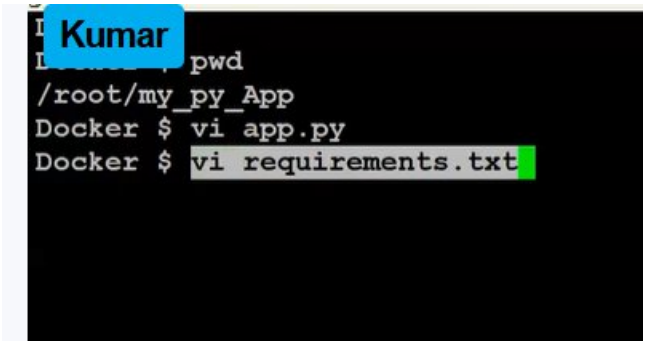
```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, this is a simple web application!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- **Here** this application will write the given msg on this application will listen to 5000 port number
- Now we will create docker file, this will list all the things which is required to run this code
- Now for this code we need python image
- Will go to docker hub and will search for python image, we can ask our developer which exactly image we need, so right now we will take the basic image do slim represents only python basic one
- Workdir is nothing but the directory where all the dependieces are installed just like C drive for OS, it is not necessary but a good practice
- COPY , for this we need source and destination,so here we are copying everything in Workdir
- Now for python we need requirements.txt file which contains all dependencies to run the code



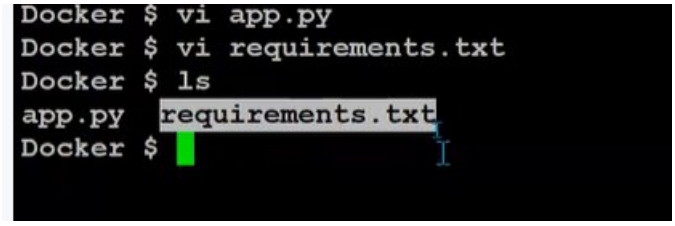
Screen clipping taken: 11-08-2025 05:27 PM

- We will create this file now and will put the above dependencies in it



Screen clipping taken: 11-08-2025 05:29 PM

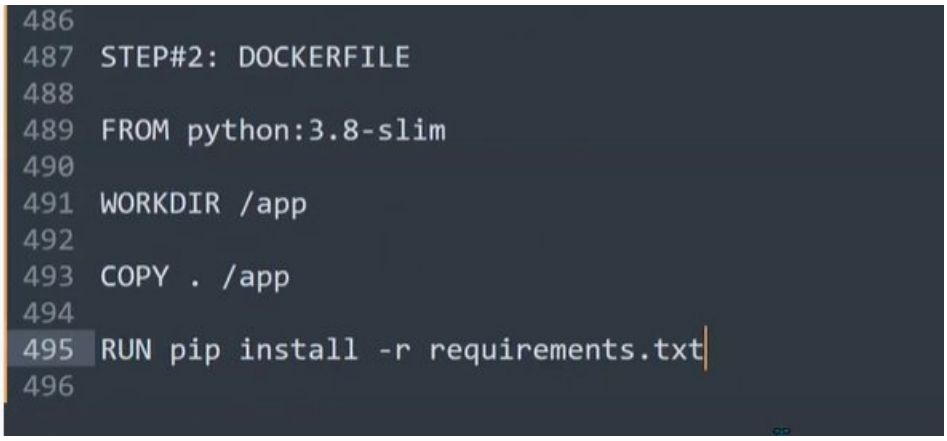
- So we have 2 files now



Screen clipping taken: 11-08-2025 05:30 PM

- The first one contains the code and the other one contains dependencies
- So these dependencies should be installed in container also so for that we will use RUN

RUN pip install -r requirements.txt. Here **pip is the package manager for python>**



Screen clipping taken: 11-08-2025 05:32 PM

- Now we need to expose the code

EXPOSE 5000

- Now we will use CMD to run the command

CMD ["python", "app.py"], here it will run this code with python

- So these are the instruction we need to give to docker file

STEP#2: DOCKERFILE

FROM python:3.8-slim

WORKDIR /app

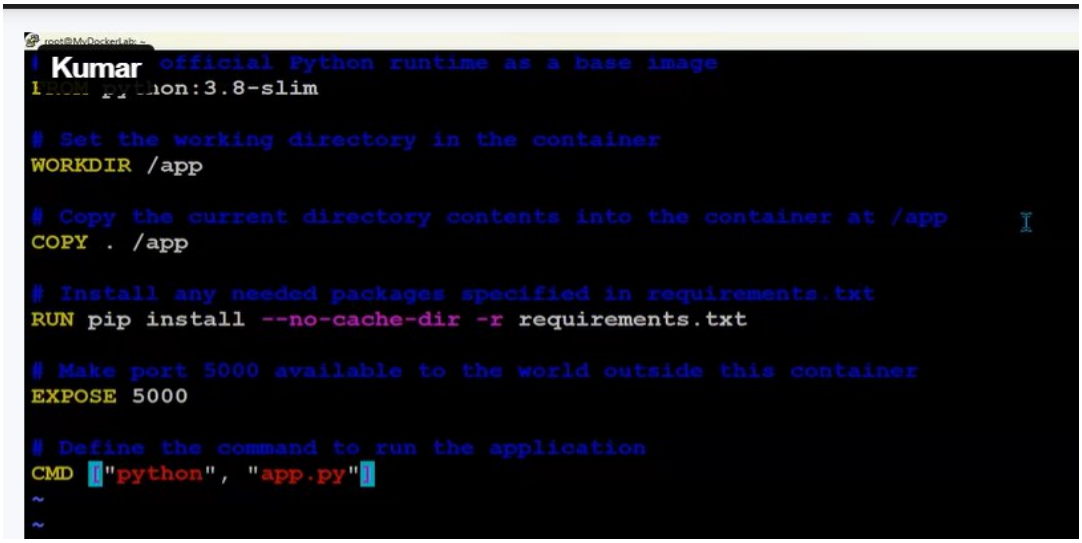
COPY . /app

RUN pip install -r requirements.txt

EXPOSE 5000

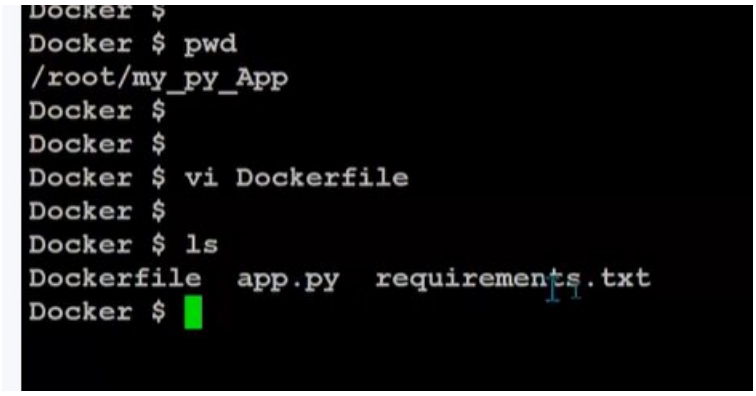
CMD ["python", "app.py"]

- Now we will create docker file

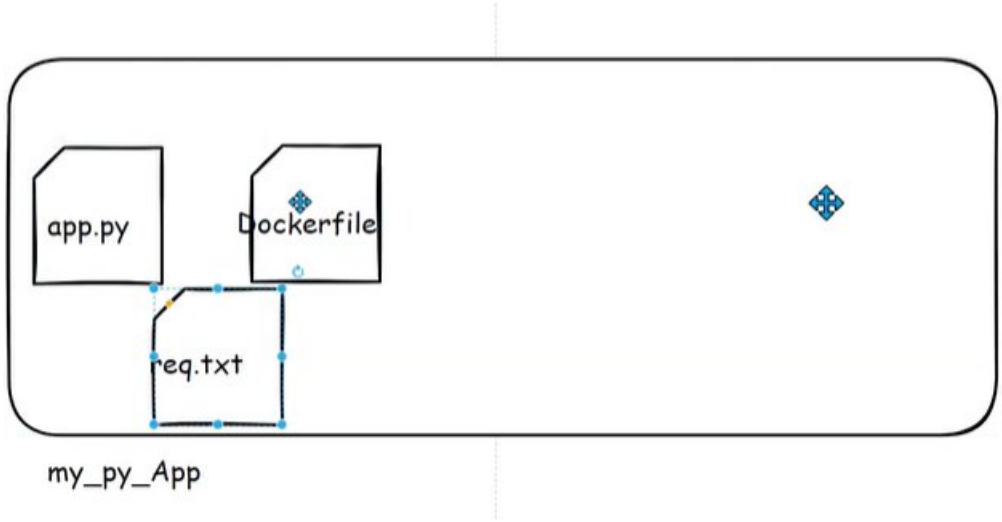


Screen clipping taken: 11-08-2025 05:39 PM

- Now we have these files

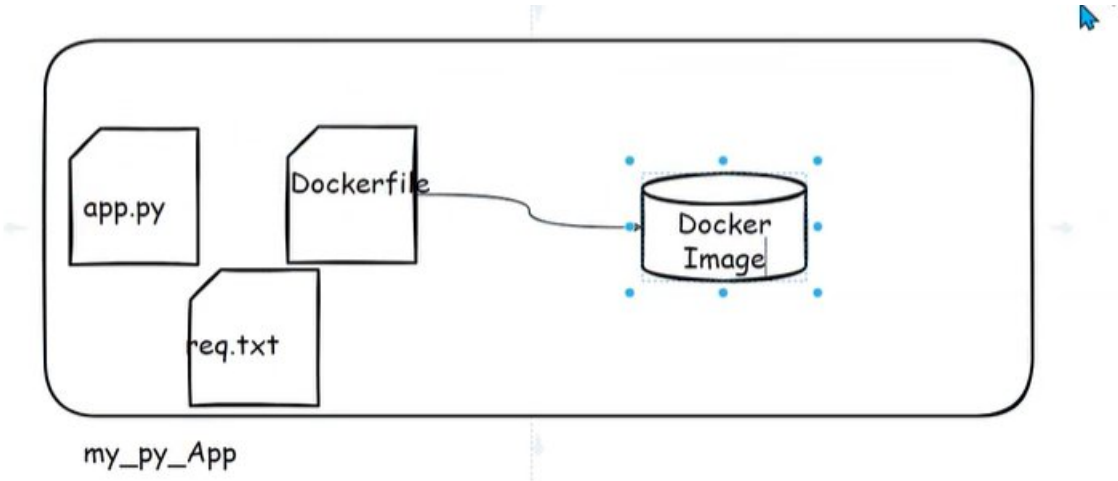


Screen clipping taken: 11-08-2025 05:40 PM



Screen clipping taken: 11-08-2025 05:40 PM

- Now using docker file we will create a docker image which will be stored in the desktop



Screen clipping taken: 11-08-2025 05:41 PM

- We use **Build command to build an image**

```
329   ps          List containers
330   build       Build an image from a Dockerfile
```

Screen clipping taken: 11-08-2025 05:42 PM

- For docker image we have these sections

```
506
507 REPOSITORY    TAG         IMAGE ID    CREATED    SIZE
508
```

Screen clipping taken: 11-08-2025 05:43 PM

- We will use the command below

docker build -t simple-web-app

```
Docker $ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
Docker $ docker build -t simple-web-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  4.096kB
Step 1/6 : FROM python:3.8-slim
3.8-slim: Pulling from library/python
302e3ee49805: Pull complete
030d7bdc20a6: Pull complete
```

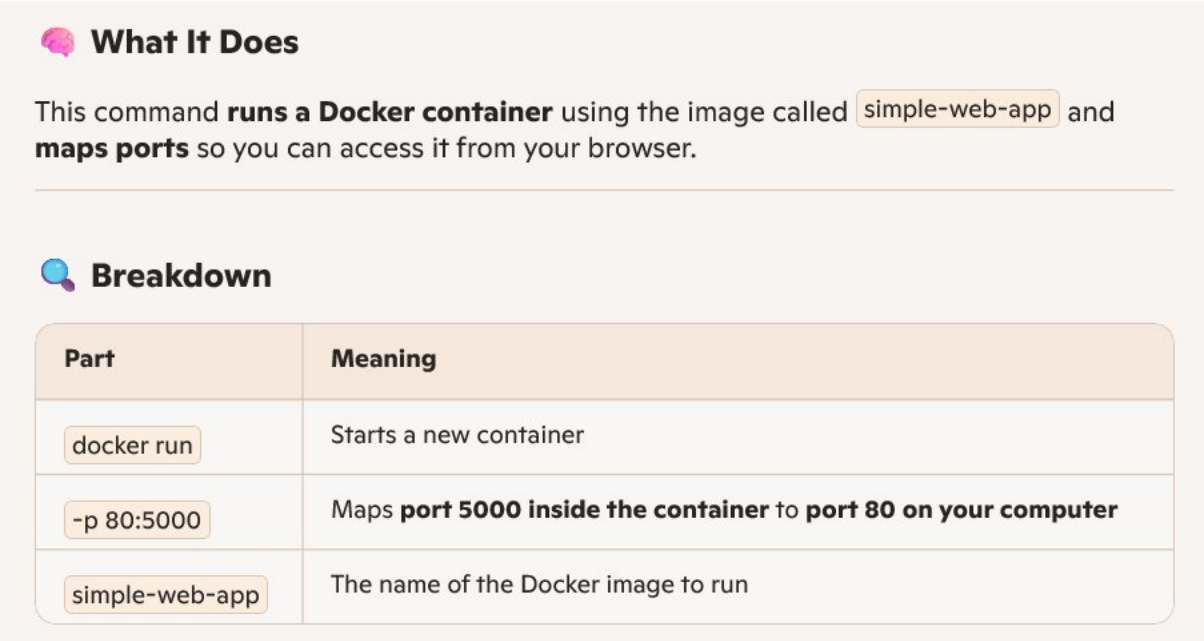
Screen clipping taken: 11-08-2025 05:45 PM

- Now will check the images

```
Successfully tagged simple-web-app:latest
Docker $ docker images
REPOSITORY       TAG         IMAGE ID       CREATED          SIZE
simple-web-app   latest      ace52314009e   31 seconds ago   135MB
python           3.8-slim    b5f62925bd0f   11 months ago    125MB
Docker $
```

Screen clipping taken: 11-08-2025 05:46 PM

- Here we see python also got downloaded as this is a basic python
- Now will create container using **docker run -p 80:5000 simple-web-app**



Screen clipping taken: 28-10-2025 10:21 PM

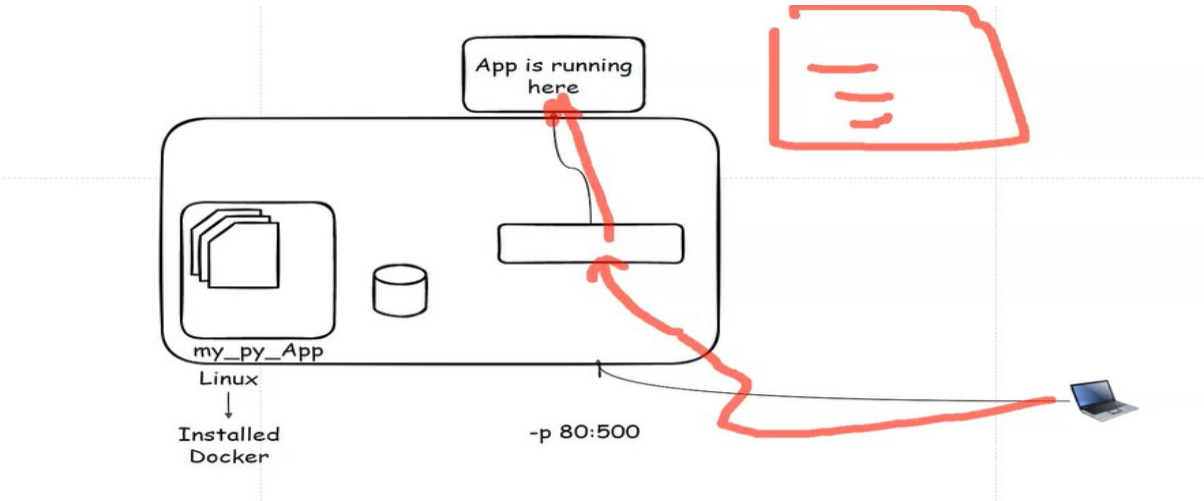
 What Happens

- The app inside the container is listening on **port 5000**
- You can access it from your browser at http://localhost:80
- Docker forwards traffic from your machine's port 80 to the container's port 5000

```
Docker $ docker run -p 80:5000 simple-web-app
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.17.0.2:5000/ (Press CTRL+C to quit)
```

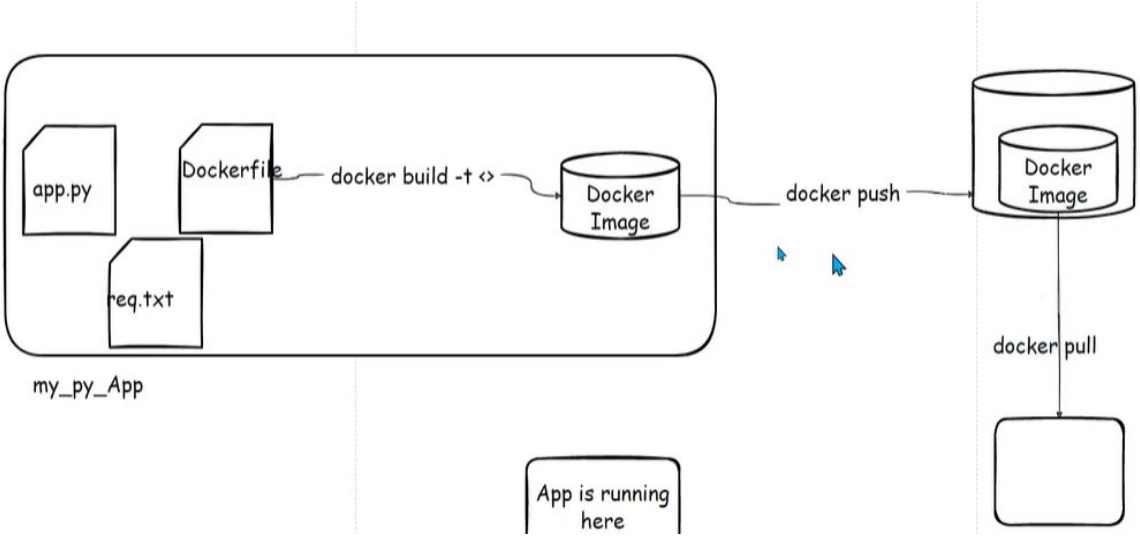
Screen clipping taken: 11-08-2025 05:50 PM

- Now in below diagram we have the container installed on the Linux machine and now we want to connect the laptop to the app which running inside the container and it does not have the public IP, so in this the traffic first will go from laptop to base machine and from there it will go to container, which is why we exposed port number so port number 80 will work for base machine



Screen clipping taken: 11-08-2025 09:45 PM

- So whenever **we build any image we upload to docker hub using docker push,** so that anyone can use this image
- And to download the image we will use **docker pull**



Screen clipping taken: 11-08-2025 09:47 PM

- So we will stop the docker now

```
root@MyDockerLab:~# docker ps
CONTAINER ID   IMAGE           COMMAND          CREATED          STATUS          PORTS
               NAMES
e02c6e1174cc   simple-web-app   "python app.py"   47 seconds ago   Up 46 seconds   0.0.0.0:80->5000/tcp, [::]:8
0->5000/tcp    sad_tesla
root@MyDockerLab:~# docker stop sad_tesla
```

Screen clipping taken: 11-08-2025 09:48 PM

- Now we will delete this container

```
root@MyDockerLab:~# docker stop sad_tesla
sad_tesla
root@MyDockerLab:~#
root@MyDockerLab:~#
root@MyDockerLab:~# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@MyDockerLab:~# docker ps -a
CONTAINER ID   IMAGE           COMMAND           CREATED         STATUS                      PORTS     NAMES
e02c6e1174cc   simple-web-app   "python app.py"   8 minutes ago   Exited (137) 9 seconds ago            sad_te
sla
root@MyDockerLab:~# docker rm e02c6e1174cc
e02c6e1174cc
root@MyDockerLab:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@MyDockerLab:~#
```

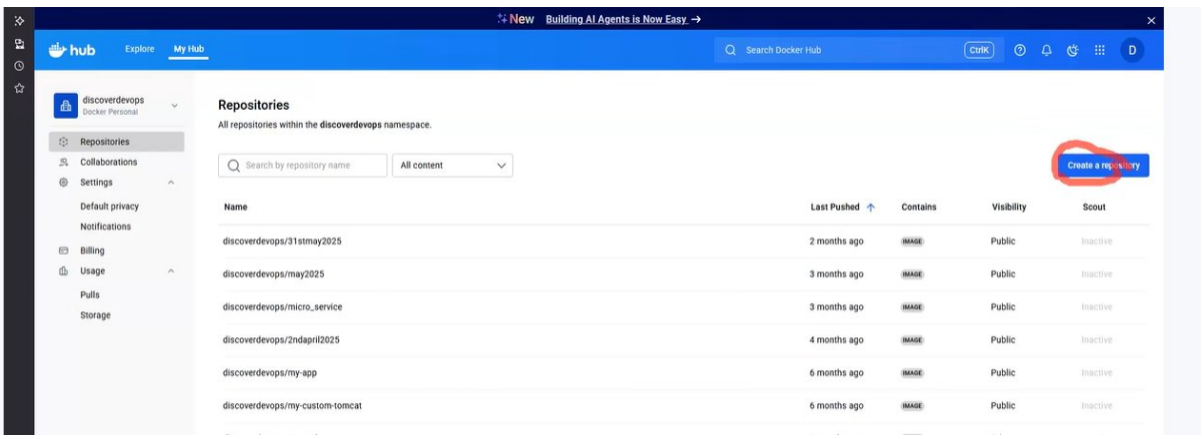
Screen clipping taken: 11-08-2025 09:49 PM

- But images will be there

```
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@MyDockerLab:~# docker images
REPOSITORY       TAG         IMAGE ID       CREATED          SIZE
simple-web-app   latest      ace52314009e   11 minutes ago   135MB
python           3.8-slim    b5f62925bd0f   11 months ago    125MB
root@MyDockerLab:~#
```

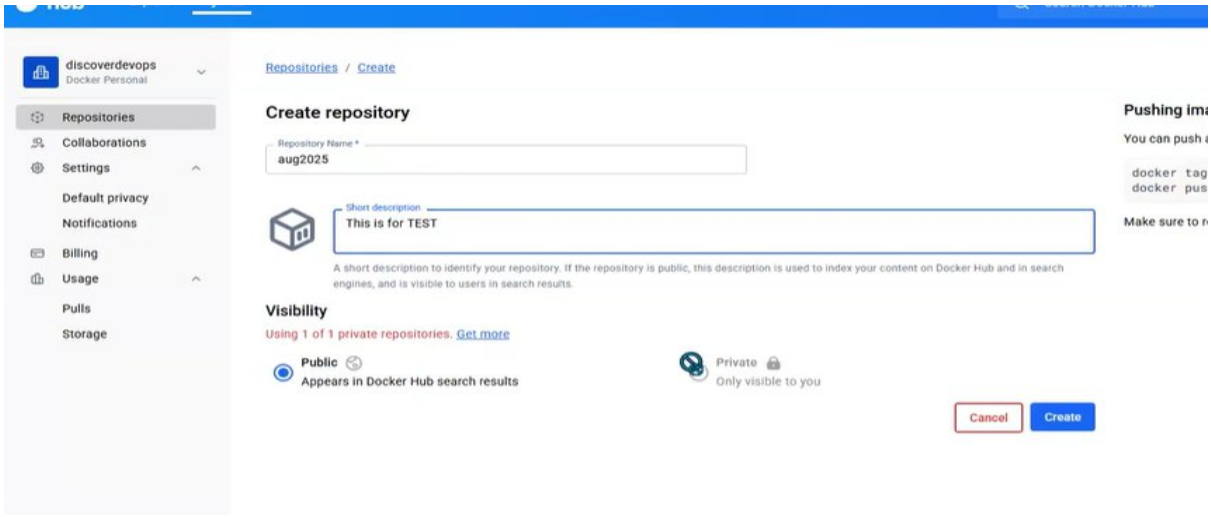
Screen clipping taken: 11-08-2025 09:49 PM

- In docker we can create **1 private and many public repositories**
- We will go to dockerhub and click on create repository



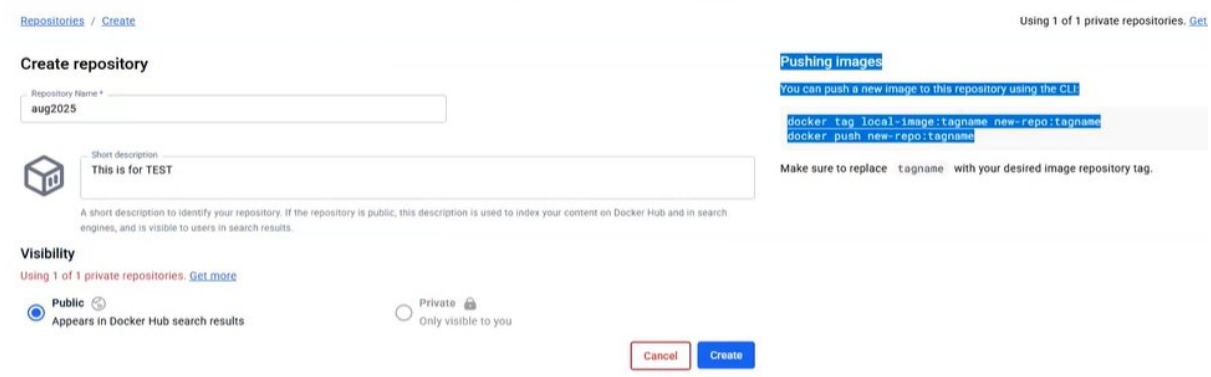
Screen clipping taken: 11-08-2025 09:51 PM

- And give name and select private or public



Screen clipping taken: 11-08-2025 09:52 PM

- On the right side it will give commands to push the images



Screen clipping taken: 11-08-2025 09:52 PM

- You can push a new image to this repository using the CLI:

docker tag local-image:tagname new-repo:tagname

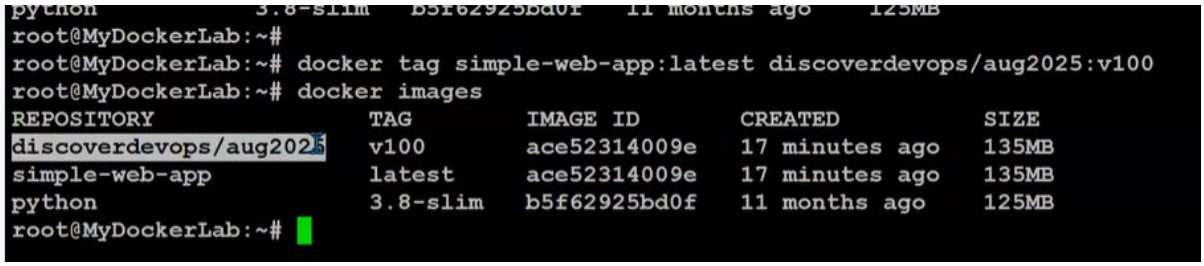
docker push new-repo:tagname

user_name/repo_name(discoverdevops/aug2025)

docker tag simple-web-app:latest discoverdevops/aug2025:v100

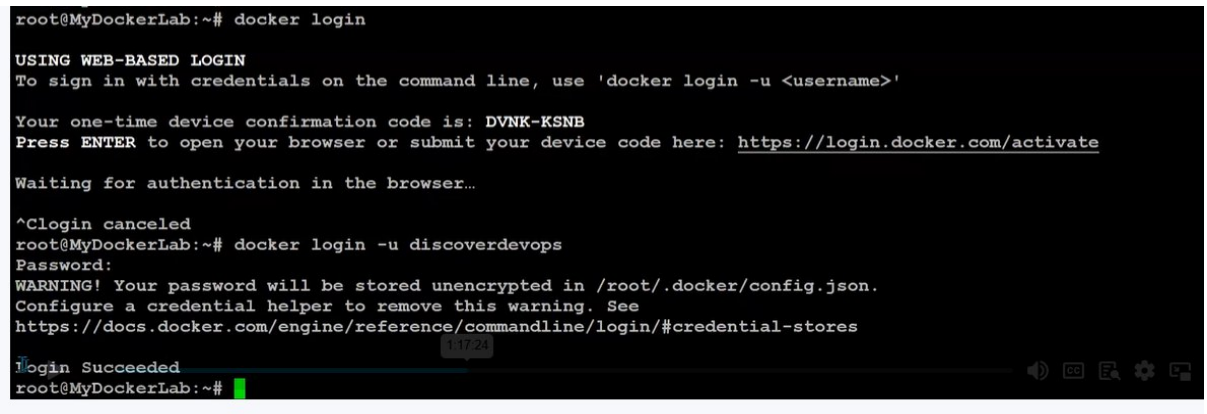
docker push discoverdevops/aug2025:v100

- So we will change the tag first



Screen clipping taken: 11-08-2025 09:56 PM

- Here the image ID is same is just tagging is changed
- To login to docker hub we need to use **docker login -u username**

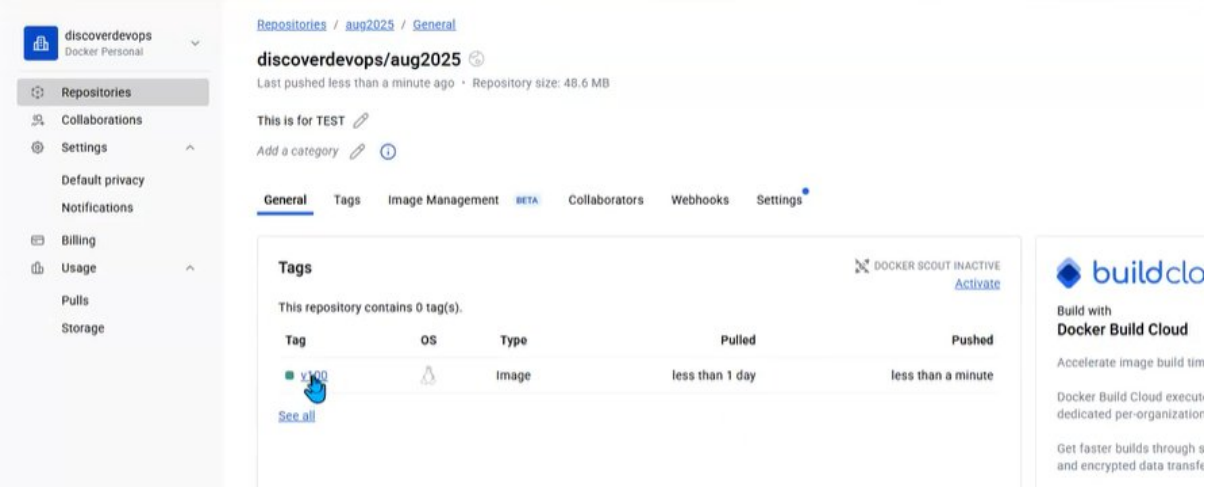


Screen clipping taken: 11-08-2025 09:57 PM

- Now we will push the image using **docker push new-repo:tagname**

Screen clipping taken: 11-08-2025 10:00 PM

- Once we refresh on the docker hub we wills ee the image

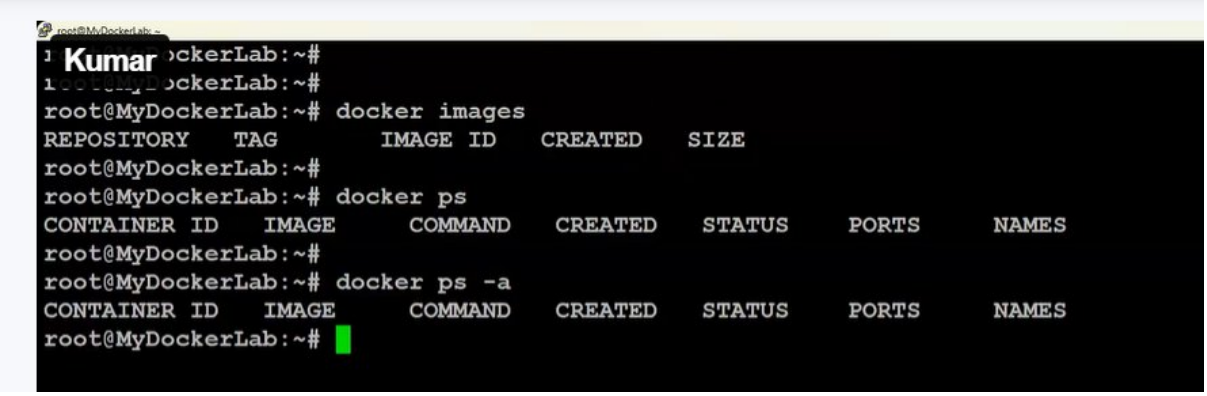


Screen clipping taken: 11-08-2025 10:00 PM

- So to push any image we will use

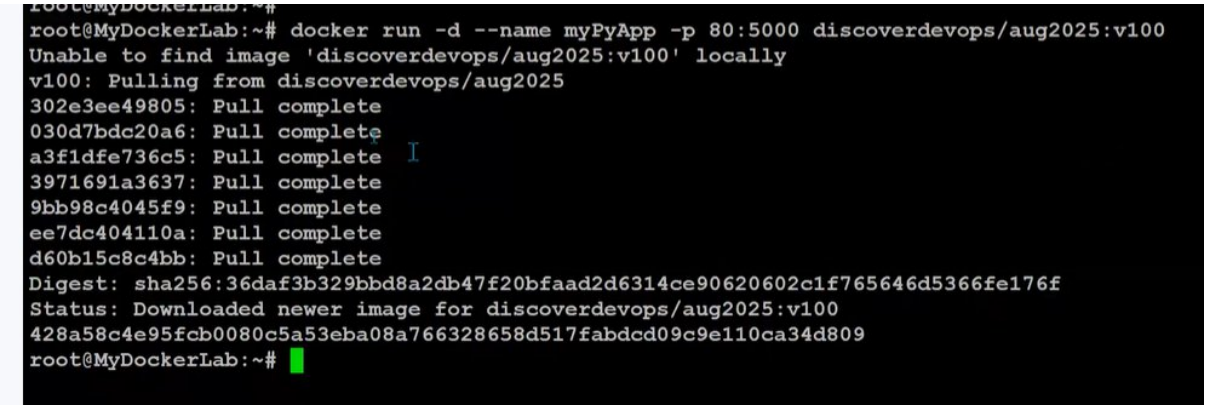
Docker push user_name/repo_name:TAG

- Now we will download the image from docker hub
- So right now we have nothing in the machine



Screen clipping taken: 11-08-2025 10:03 PM

- So to download we will use run command



Screen clipping taken: 11-08-2025 10:05 PM

- Now we will check the same

Screen clipping taken: 11-08-2025 10:05 PM

- Now we will run this app by using IP of the system
- Now we have a scenario where we will create a lab
- So for this we will clone the GitHub repository on the laptop

```
Docker $ cd
Docker $
Docker $
Docker $ git clone https://github.com/discover-devops/my-node-app.git
Cloning into 'my-node-app'...
remote: Enumerating objects: 587, done.
remote: Counting objects: 100% (587/587), done.
remote: Compressing objects: 100% (466/466), done.
remote: Total 587 (delta 110), reused 575 (delta 104), pack-reused 0 (from 0)
Receiving objects: 100% (587/587), 693.38 KiB | 19.26 MiB/s, done.
Resolving deltas: 100% (110/110), done.
Docker $ ls
my-node-app  my_py_App
```

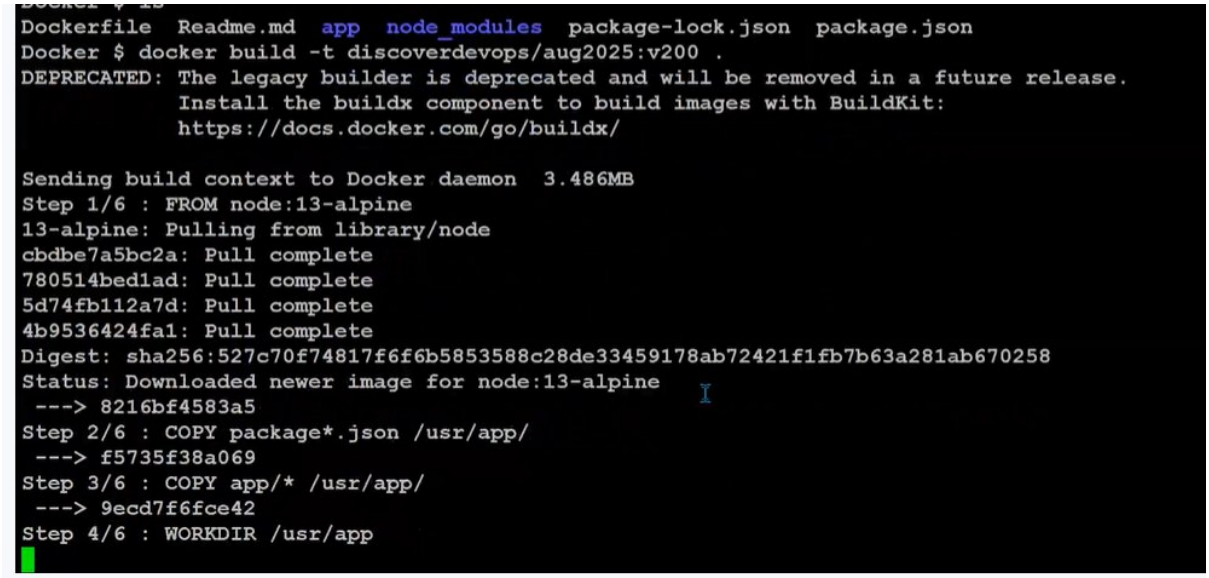
Screen clipping taken: 11-08-2025 10:14 PM

- Now we will go to my-node-app

```
Docker $ ls
my-node-app  my_py_App
Docker $ cd my-node-app
Docker $ ls
Dockerfile  Readme.md  app  deployment.yaml  node_modules  package-lock.json  package.json
Docker $ rm -rf deployment.yaml
Docker $
Docker $ ls
Dockerfile  Readme.md  app  node_modules  package-lock.json  package.json
Docker $
```

Screen clipping taken: 11-08-2025 10:15 PM

- Now we have everything here so we will create an image in the same repository which we just created on docker hub but tagging here as v200



Screen clipping taken: 11-08-2025 10:17 PM

- Now we will check the images

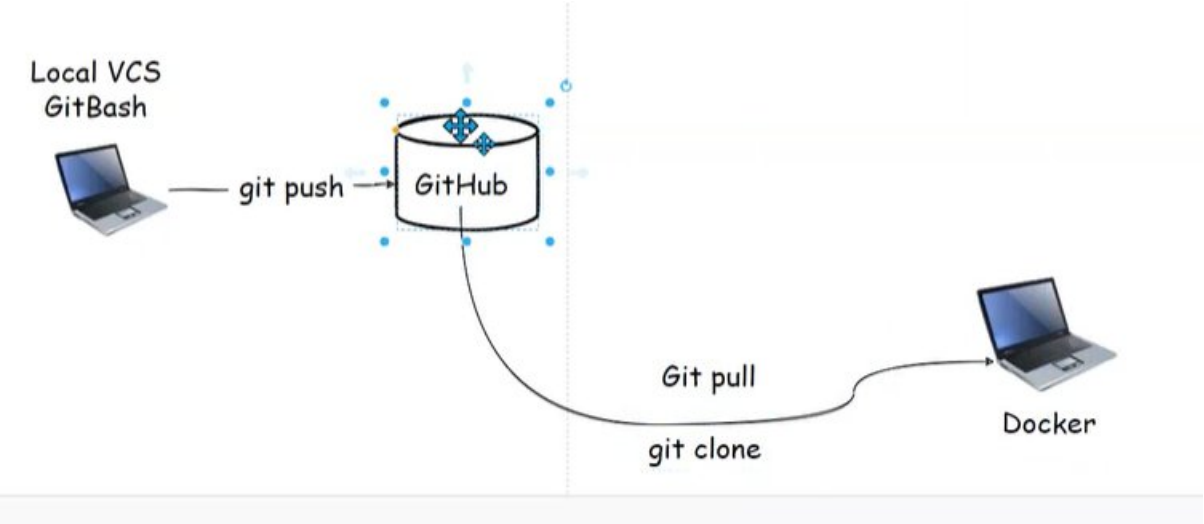
Screen clipping taken: 11-08-2025 10:17 PM

- Now we will push this image to docker hub

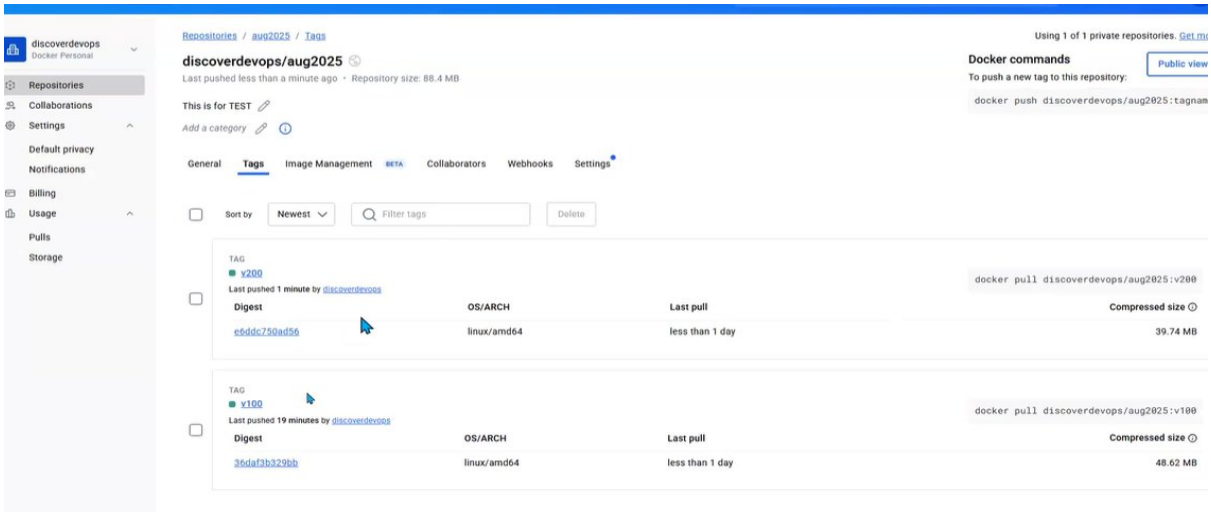
```
Docker $ docker images
REPOSITORY                TAG         IMAGE ID       CREATED          SIZE
discoverdevops/aug2025    v200        ddc51e20e803   7 seconds ago    117MB
discoverdevops/aug2025    v100        ace52314009e   37 minutes ago   135MB
node                      13-alpine   8216bf4583a5   5 years ago      114MB
Docker $ docker push discoverdevops/aug2025:v200
The push refers to repository [docker.io/discoverdevops/aug2025]
15c3436a93cf: Pushed
814e47e6587b: Pushed
d3e57e80f96c: Pushed
629960860aca: Mounted from library/node
f019278bad8b: Mounted from library/node
8ca4f4055a70: Mounted from library/node
3e207b409db3: Mounted from library/node
v200: digest: sha256:e6ddc750ad56d3025a4e4e9806ba36babf72a311c76464be6ad6e34dd477ea8f size: 1784
Docker $
```

Screen clipping taken: 11-08-2025 10:18 PM

- Now we will see another example where we will use code from docker hub

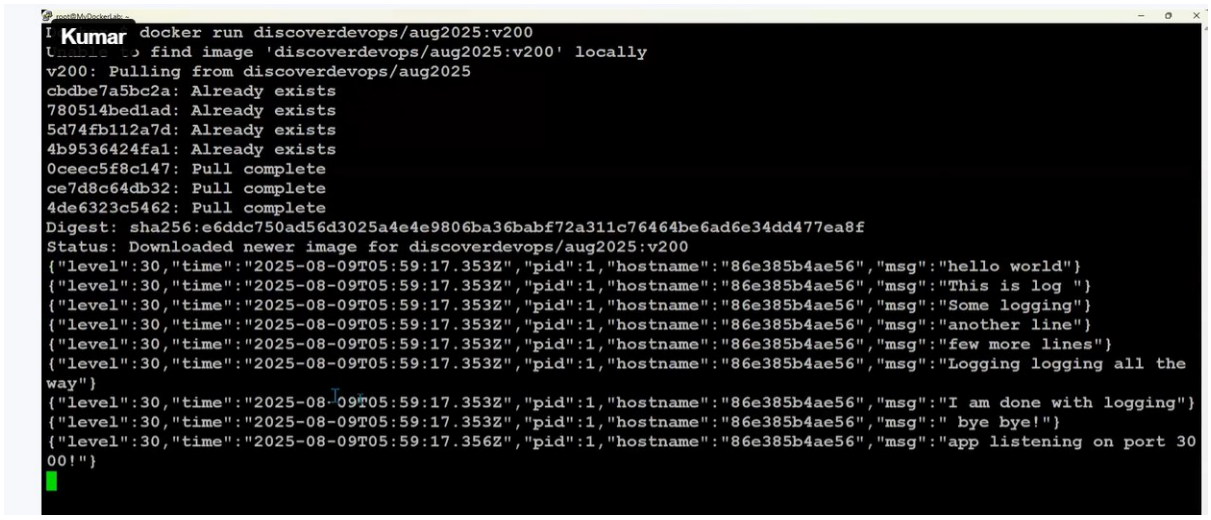


Screen clipping taken: 11-08-2025 10:21 PM



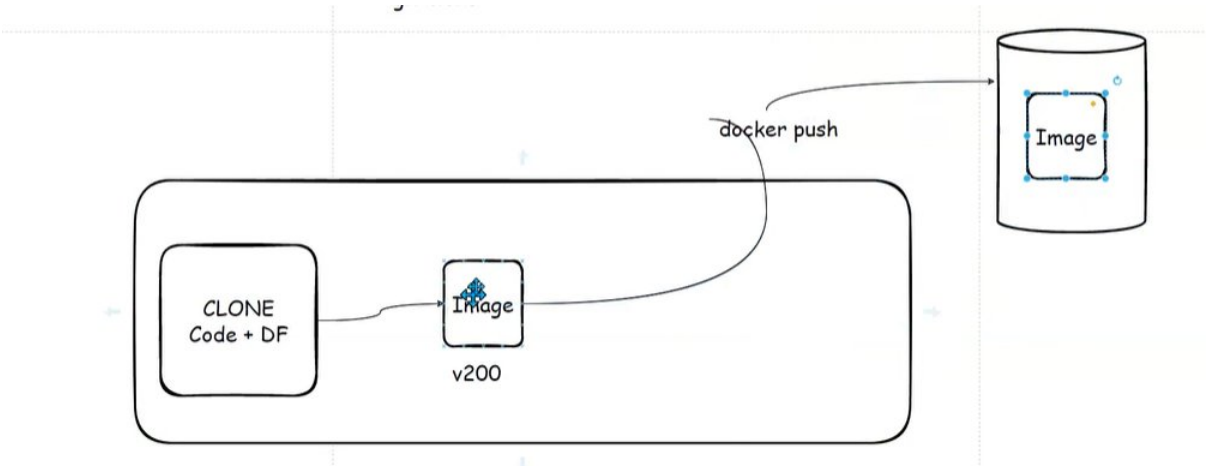
Screen clipping taken: 11-08-2025 10:19 PM

- We can create the container using the same image



Screen clipping taken: 11-08-2025 10:20 PM

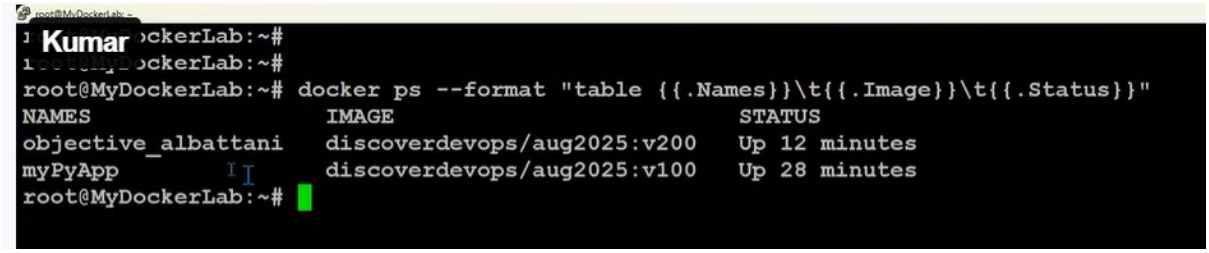
- So here what we did we got the code from GitHub which contains code, docker file and everything which is required to create an image, once it is created we uploaded it to the docker hub



Screen clipping taken: 11-08-2025 10:24 PM

- Then we created container using the same image
- To check if app is running fine or not inside the container

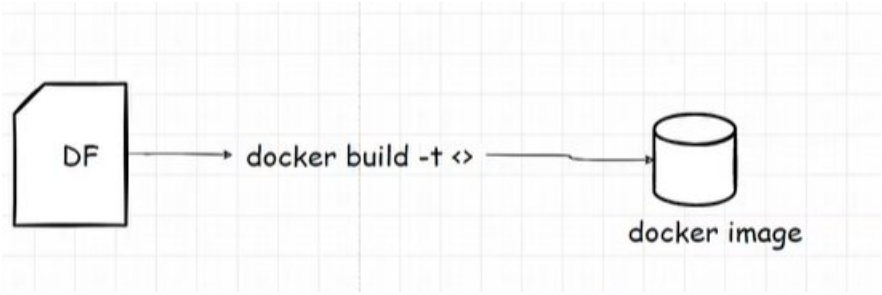
docker ps --format "table {{.Names}}\t{{.Image}}\t{{.Status}}"



Screen clipping taken: 11-08-2025 10:32 PM

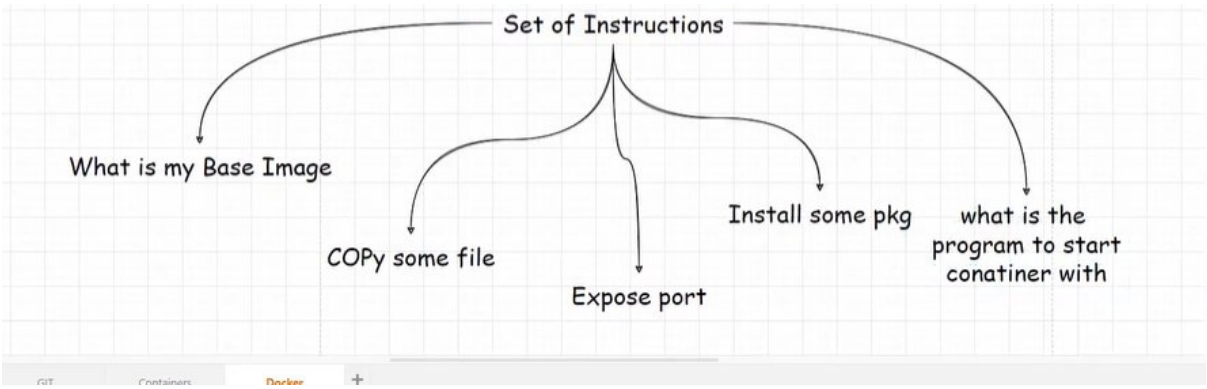
What happens when we run docker build command

- Basically this command create the image on the basis of info which is present in docker file



Screen clipping taken: 12-08-2025 01:16 PM

- Docker file is nothing but the set of instructions which provides the info below:



Screen clipping taken: 12-08-2025 01:17 PM

- In order to install the machine should be up and running
- So we will read about the instructions which we get when we run build command

$ docker build -t simple-web-app .

REPOSITORY TAG IMAGE ID CREATED SIZE

Docker $ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

Docker $ docker build -t simple-web-app .

Sending build context to Docker daemon 4.096kB

Step 1/6(the base image got downloaded) : FROM python:3.8-slim

3.8-slim: Pulling from library/python

302e3ee49805: Pull complete

030d7bdc20a6: Pull complete

a3f1dfe736c5: Pull complete

3971691a3637: Pull complete

Digest: sha256:1d52838af602b4b5a831beb13a0e4d073280665ea7be7f69ce2382f29c5a613f

Status: Downloaded newer image for python:3.8-slim

---> b5f62925bd0f

Step 2/6 (in this we can't create any directory on image so the docker is smart and creating a container using this image which it would delete afterwards): WORKDIR /app

---> Running in ff2c45939cc4 (this is nothing but the container where it created the directory)

---> Removed intermediate container ff2c45939cc4

---> afc6328239c5

Step 3/6 : COPY . /app

---> 2f4bd4eee353

Step 4/6 : RUN pip install --no-cache-dir -r requirements.txt

---> Running in 06a6edecd547

Collecting Flask==2.0.1

Downloading Flask-2.0.1-py3-none-any.whl (94 kB)

━━

━ 94.8/94.8 kB 1.1 MB/s eta 0:00:00

Collecting Werkzeug==2.0.1

Downloading Werkzeug-2.0.1-py3-none-any.whl (288 kB)

━━

288.2/288.2 kB 4.0 MB/s eta 0:00:00

Collecting click>=7.1.2

Downloading click-8.1.8-py3-none-any.whl (98 kB)

━━

━ 98.2/98.2 kB 26.4 MB/s eta 0:00:00

Collecting Jinja2>=3.0

Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)

━━━

134.9/134.9 kB 21.5 MB/s eta 0:00:00

Collecting itsdangerous>=2.0

Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)

Collecting MarkupSafe>=2.0

Downloading MarkupSafe-2.1.5-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (26 kB)

Installing collected packages: Werkzeug, MarkupSafe, itsdangerous, click, Jinja2, Flask

Successfully installed Flask-2.0.1 Jinja2-3.1.6 MarkupSafe-2.1.5 Werkzeug-2.0.1 click-8.1.8 itsdangerous-2.2.0

WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1

[notice] To update, run: pip install --upgrade pip

---> Removed intermediate container 06a6edecd547**(after installing everything it removed the temporary container)**

---> a2be1abcf1ae

Step 5/6 : EXPOSE 5000

---> Running in 0daebafab0e9

---> Removed intermediate container 0daebafab0e9

---> 3b7470f982c4

Step 6/6 : CMD ["python", "app.py"]

---> Running in f87a7dda72da

---> Removed intermediate container f87a7dda72da

---> ace52314009e

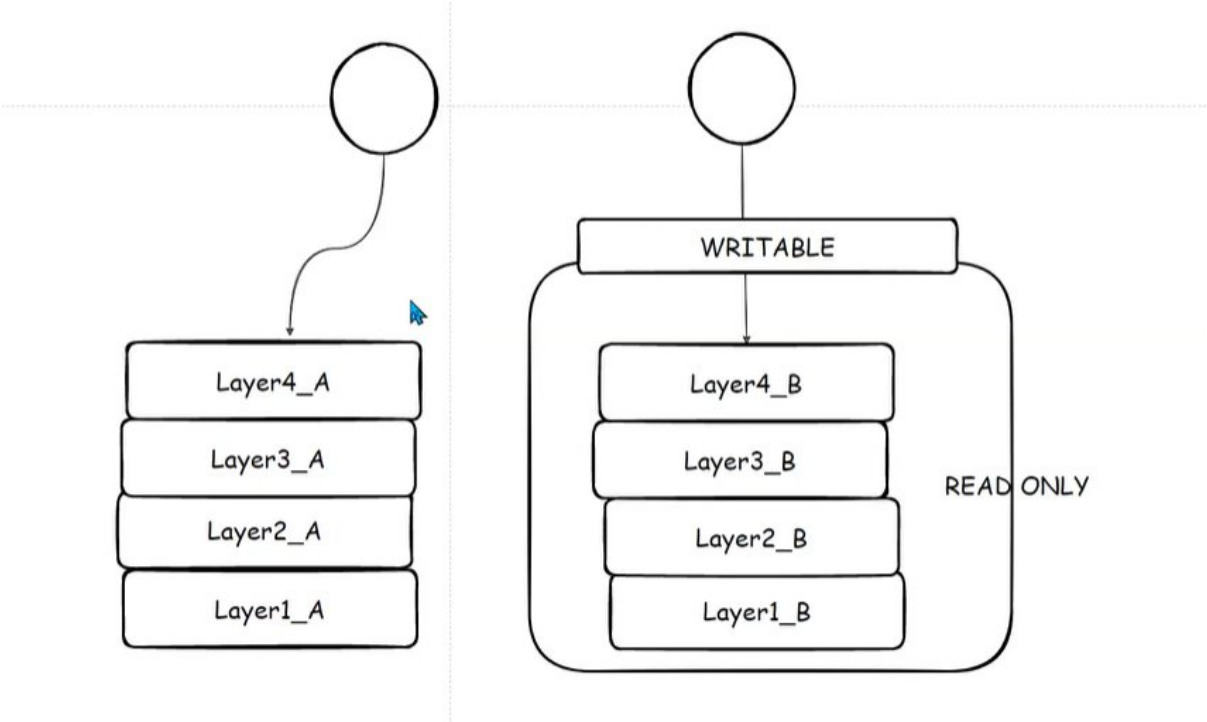
Successfully built ace52314009e

Successfully tagged simple-web-app:latest

- So here we ha**ve 6 steps as we had 6 instructions in the docker file**
- **In this for each instructions we need up and running machine to perform the steps, which is why it is creating and deleting the temp container on each step**
- **So the image is nothing but the collection of layers as per the steps which we mentioned in docker file**
- If we inspect the image we can see we have these many layers

Screen clipping taken: 12-08-2025 01:30 PM

- So image nothing but a collection of layers and **these layers can be shared between containers**
- These layers are in **only read only format, there is one tiny writable layer,** these top circles are container A and Container B



Screen clipping taken: 12-08-2025 01:35 PM

- So we create a container using alpine image

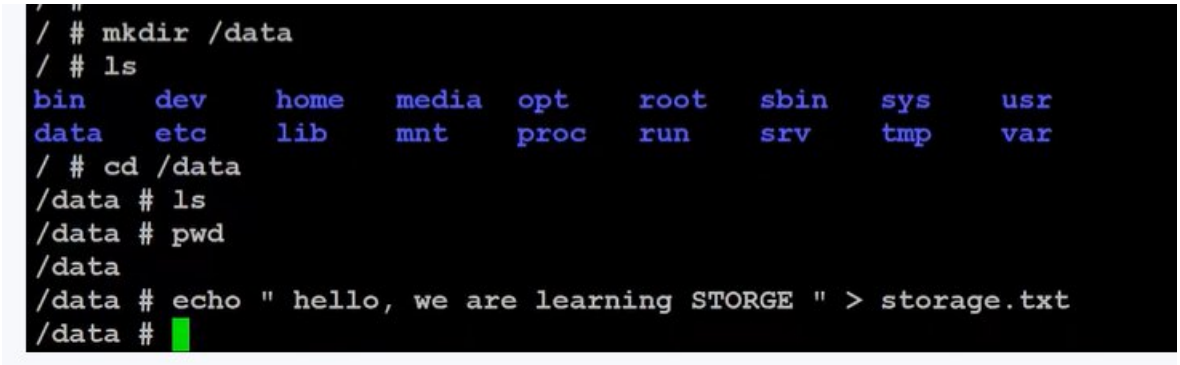
```
Docker $
Docker $ docker run -it --name demo-container alpine:3.20 sh^C
Docker $ ^C
Docker $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
Docker $
Docker $
Docker $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
Docker $  docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
Docker $
Docker $
Docker $ docker run -it --name demo-container alpine:3.20 sh
Unable to find image 'alpine:3.20' locally
3.20: Pulling from library/alpine
01d036902a3c: Pull complete
Digest: sha256:b3119ef930faabb6b7b976780c0c7a9c1aa24d0c75e9179ac10e6bc9ac080d0d
Status: Downloaded newer image for alpine:3.20
/ #
```

Screen clipping taken: 12-08-2025 01:36 PM

- Now we are inside container and will create directory

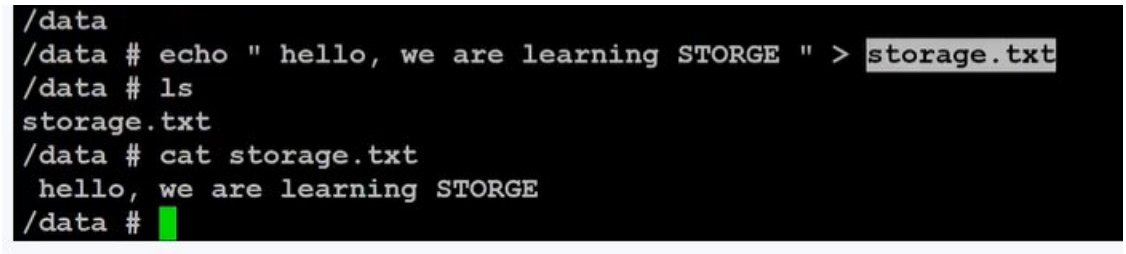
Screen clipping taken: 12-08-2025 01:38 PM

- Will cd to directory and will create a file



Screen clipping taken: 12-08-2025 01:38 PM

- Will check the output of the file

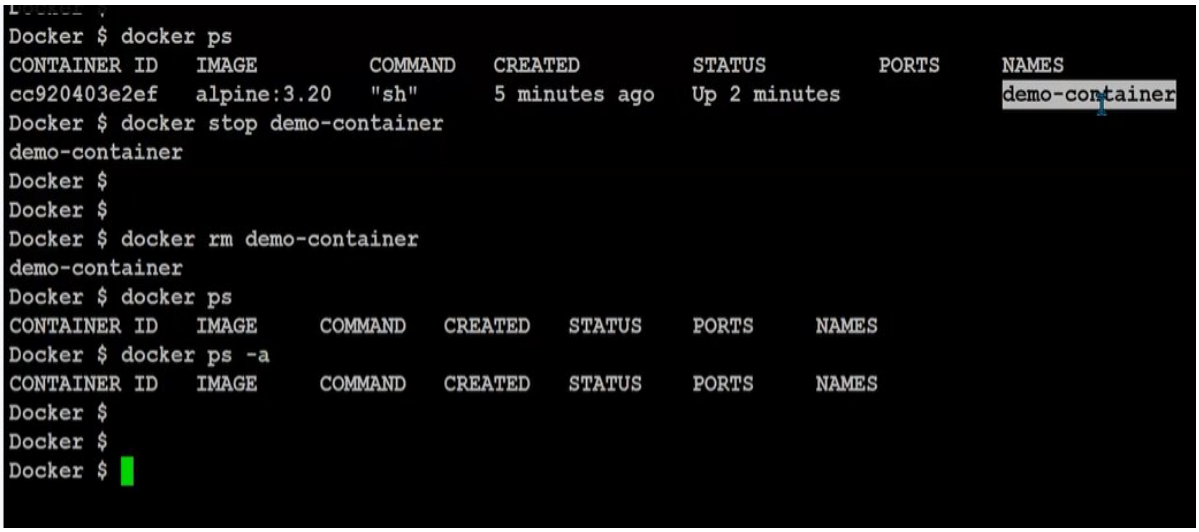


Screen clipping taken: 12-08-2025 01:39 PM

- We will exit using (ctrl+P+Q) from the container

Screen clipping taken: 12-08-2025 01:41 PM

- So whatever **we are creating in container it is happening inside the writable layer**
- So whenever **we stop/start the container the data is there inside the writable layer**
- Just in case if our container is deleted, so here we deleted the container



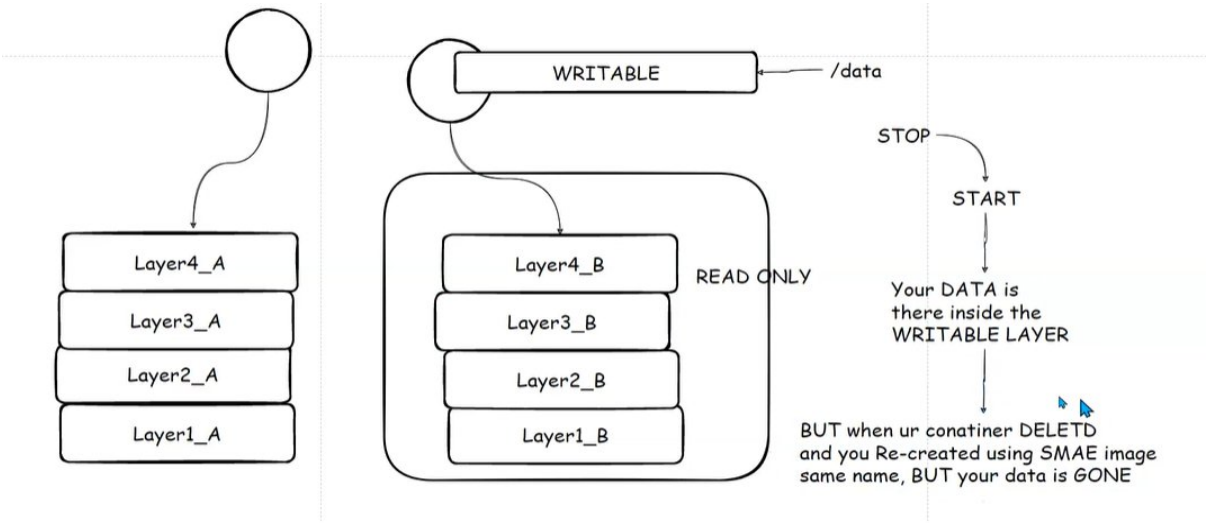
Screen clipping taken: 12-08-2025 01:43 PM

- If we see the image, it will be there

```
Docker $
Docker $ docker  images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
alpine        3.20      411bc66a7adb   3 weeks ago    7.79MB
```

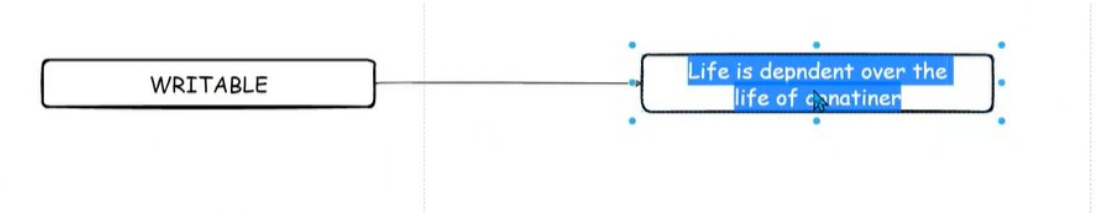
Screen clipping taken: 12-08-2025 01:44 PM

- Now we will create the container using the same image with same name and everything
- So if we check the data will not be there



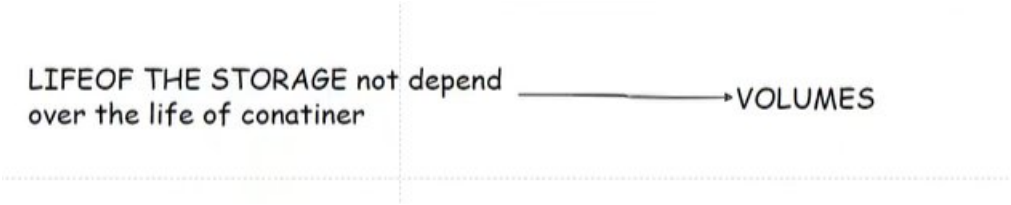
Screen clipping taken: 12-08-2025 01:46 PM

- **Writable layer is dependent on life of container which is not good**



Screen clipping taken: 12-08-2025 01:46 PM

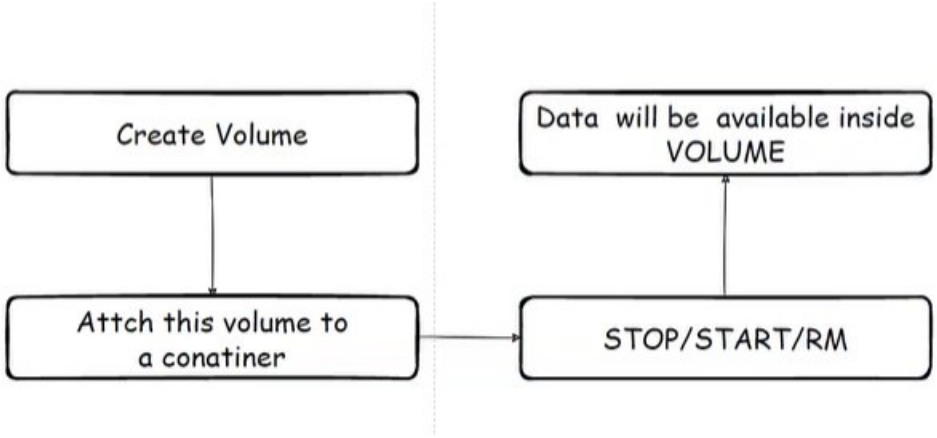
- Now we want to save our data so that it is not dependent on the life of container to overcome this volumes came into the picture



Screen clipping taken: 12-08-2025 01:48 PM

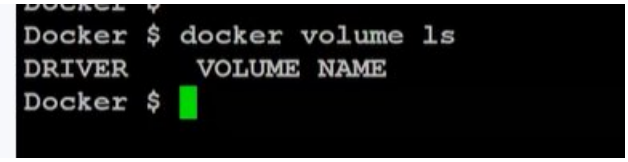
<u>Volumes</u>

- So in laptop we have hard disk, so if laptop is crashed , so there are chances we can lose the data , so if we store the data in pen drive, then we can recover from it, so pen drive is like a volume
- So what we do we create a volume and then we attach it to the container



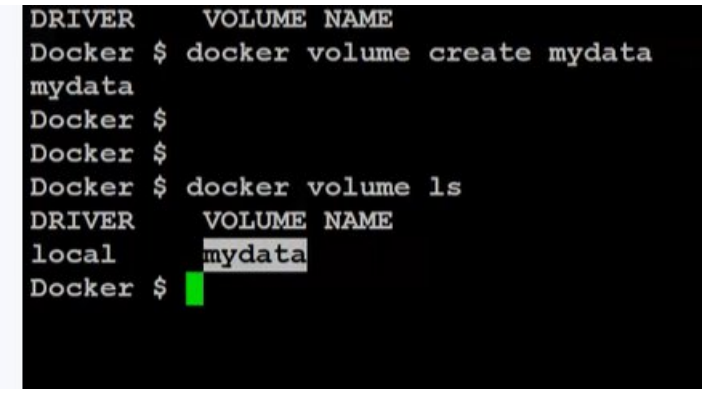
Screen clipping taken: 12-08-2025 01:53 PM

- To check volume we will use docker volume ls



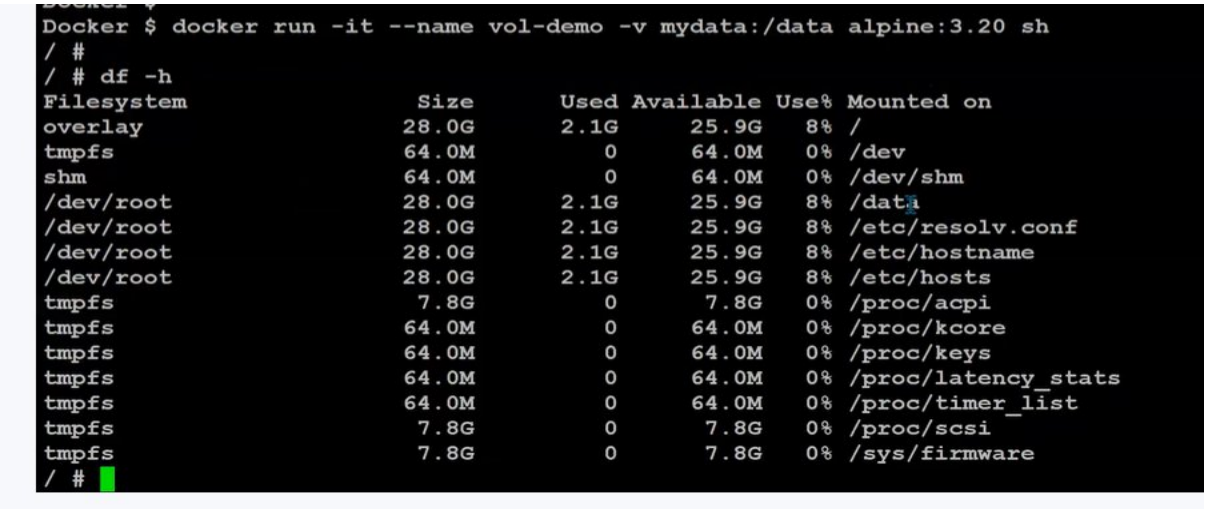
Screen clipping taken: 12-08-2025 01:55 PM

- To create we will use docker volume create <volume name>



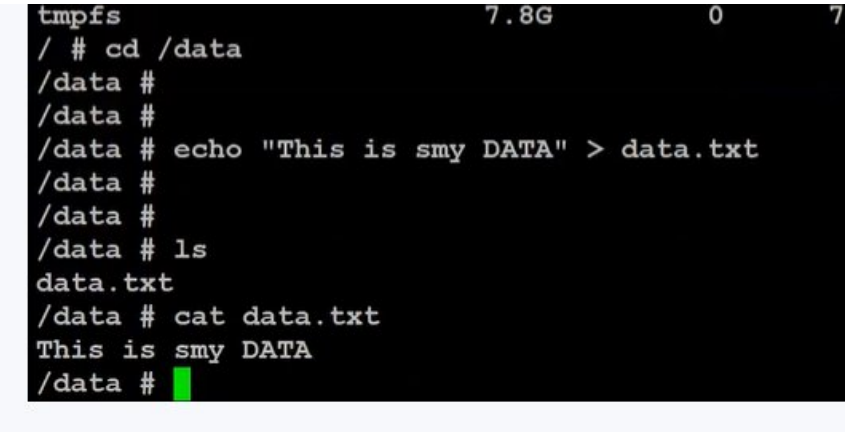
Screen clipping taken: 12-08-2025 01:56 PM

- Now we will create a container, in here with this command we create data folder in the volume



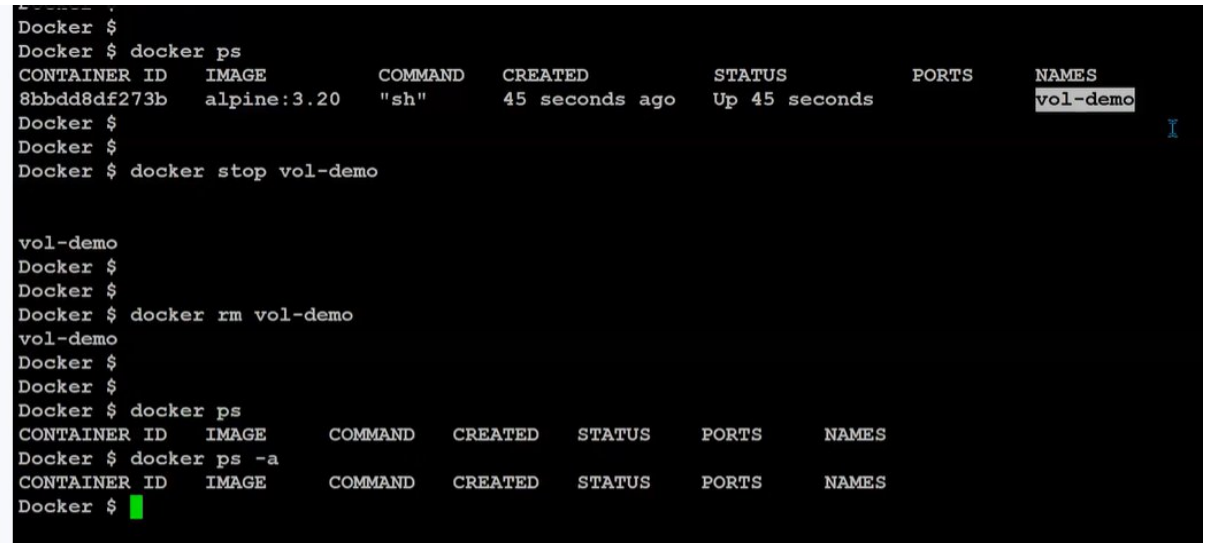
Screen clipping taken: 12-08-2025 01:57 PM

- We will got data folder and will create something



Screen clipping taken: 12-08-2025 01:58 PM

- Now we will stop and delete the container



Screen clipping taken: 12-08-2025 01:59 PM

- Now we will create the container again with same name and everything, and we can check we will have data in it

```
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
Docker $ docker run -it --name vol-demo -v mydata:/data alpine:3.20 sh
/ #
/ # cd /data/
/data # ls
data.txt
/data # cat data.txt
This is smy DATA
/data #
```

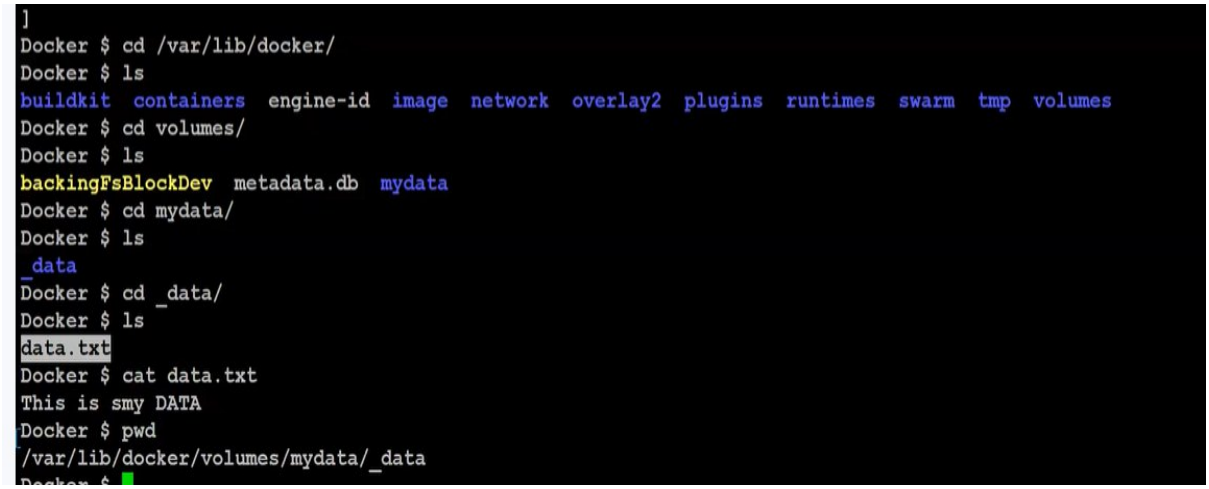
Screen clipping taken: 12-08-2025 02:00 PM

- If we inspect the volume

```
local     mydata
Docker $ docker inspect mydata
[
    {
        "CreatedAt": "2025-08-09T07:16:07Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/mydata/_data",
        "Name": "mydata",
        "Options": null,
        "Scope": "local"
    }
]
Docker $
```

Screen clipping taken: 12-08-2025 02:04 PM

- So this file is located in this folder locally

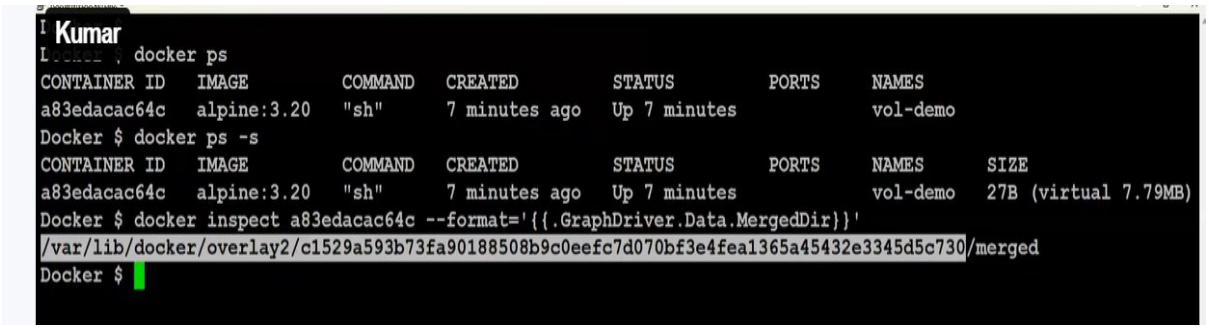


Screen clipping taken: 12-08-2025 02:05 PM

- But in case if the local machine gets crashed, then the volume will get crashed as well, this we will learn in Kubernetes
- To fetch the storage path of the container we can use the command below

docker inspect <container_name_or_id> --format='{{.GraphDriver.Data.MergedDir}}'

docker inspect a83edacac64c --format='{{.GraphDriver.Data.MergedDir}}'



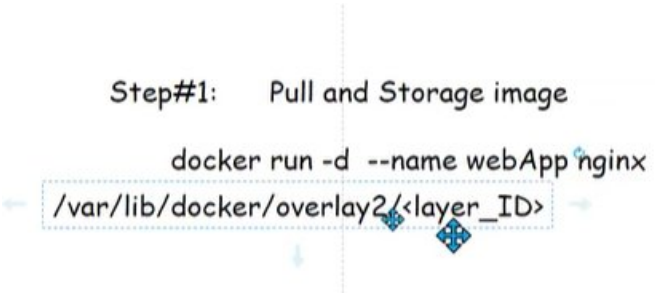
Screen clipping taken: 12-08-2025 02:49 PM

==========

Docker Storage Internals

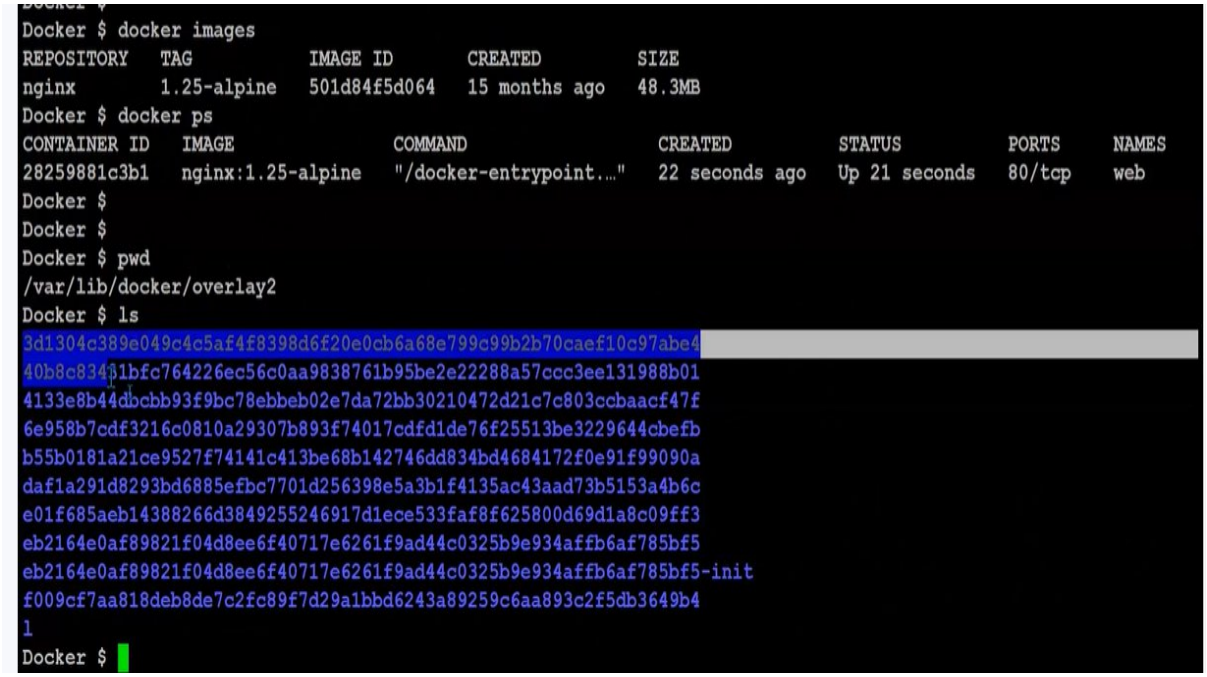
So when we create container there are several steps which we follow

Step#1: Pull and storage image---- so here we pull the image and then store it locally in a TAR format



Screen clipping taken: 12-08-2025 10:18 PM

- So whenever we pull an image it downloads all the layers
- To check these layers we can go to **/var/lib/docker/overlay2**



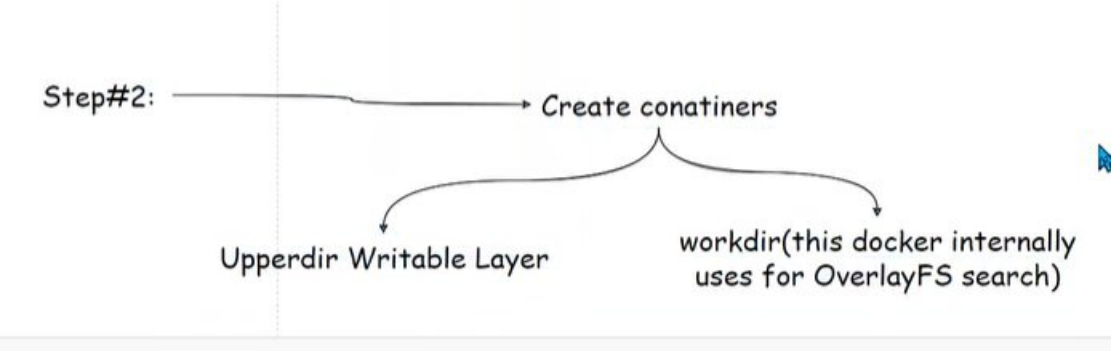
Screen clipping taken: 12-08-2025 10:15 PM

- Docker unpacks each layer into its folder
- So if we go to any of the layers there is this diff folder which belongs to that particular layer and it is a read only file

Screen clipping taken: 12-08-2025 10:16 PM

Step#2:----- creating a container

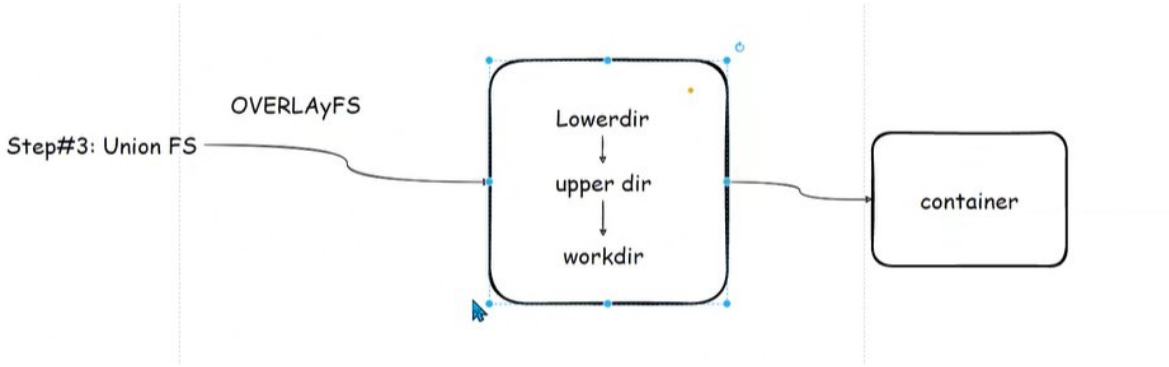
- So when we create a container it creates first upperdir which is nothing but the writable layer where we create all the changes and second it creates workdir which is used by docker for overlayFS search



Screen clipping taken: 12-08-2025 10:20 PM

Step#3: Union FS

- Docker uses union file system (it is a Linux concept)that is known as **overlayFS**
- So this will c**ombine lowerdir, upperdir and workdir and create a single file system** to run the container

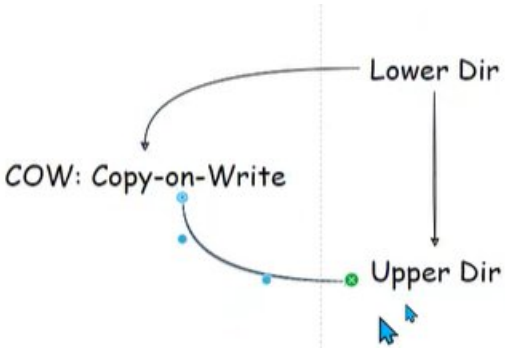


Screen clipping taken: 12-08-2025 10:25 PM

<u>COW: Copy on write</u>

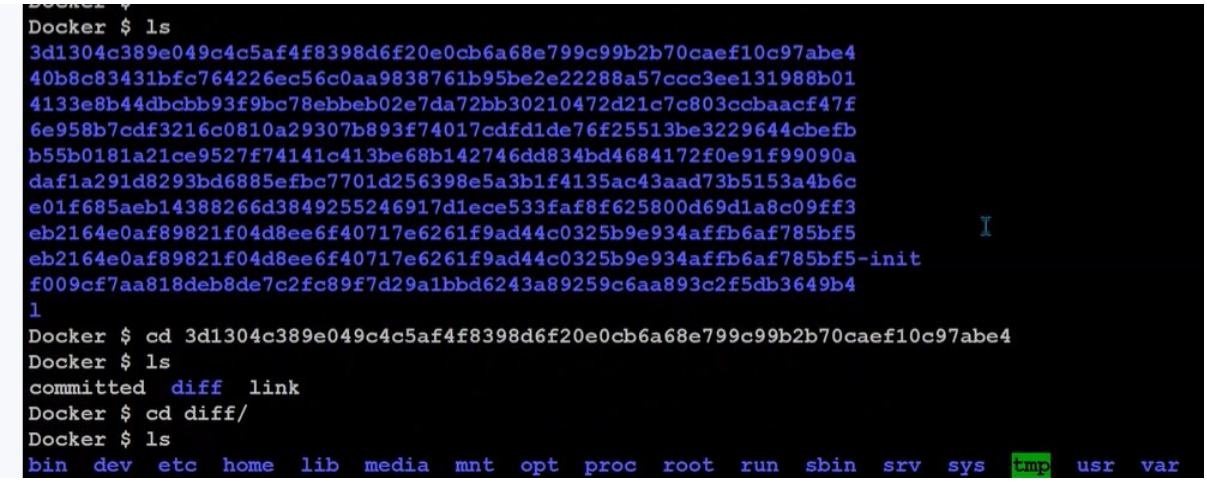
- So whenever we create nginx the config file is saved in **/etc/nginx/nginx.conf** (this is **stored in lowerdir which is ready only layer**).. So this will display a msg, but we want to display the msg of our choice we will change this file but this is a read only in image layer

- So to modify this file **COW comes in the picture, so what it does it copies the file from lower dir to upper dir which is writable laye**r



Screen clipping taken: 12-08-2025 10:32 PM

- **Tille the moment container is live the our webpage will be visible the moment container dies , the writable layer is gone so the webpage**
- To see the upper dir contents the path is shown below



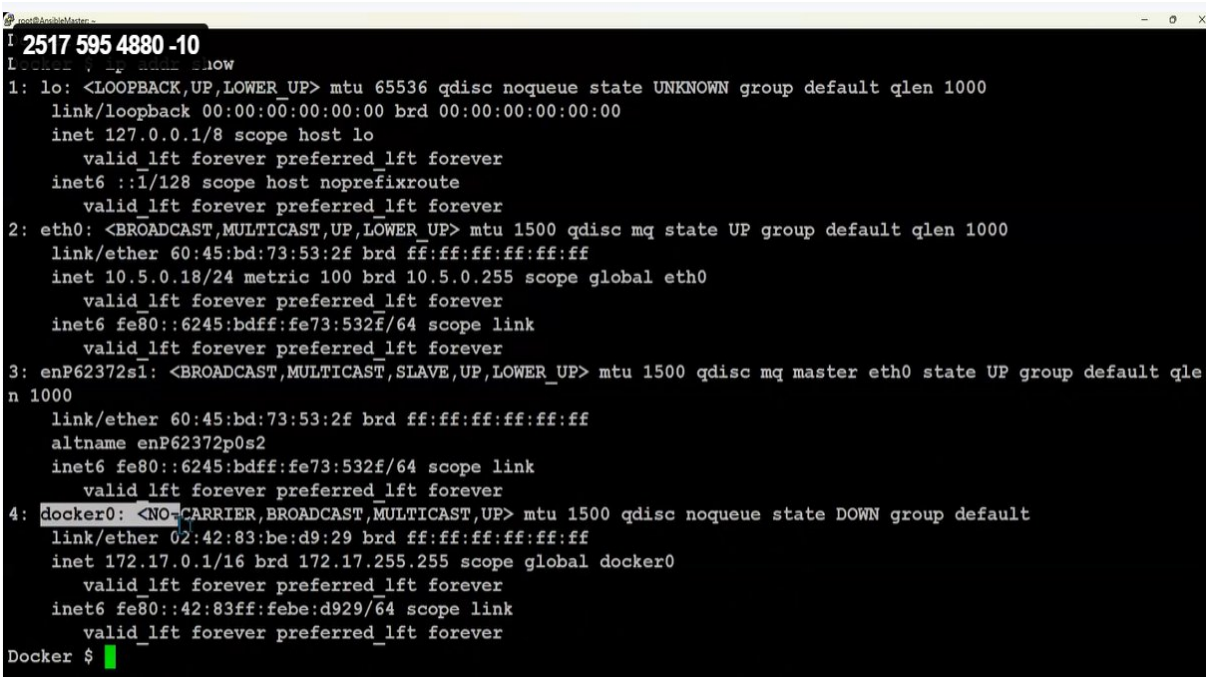
Screen clipping taken: 12-08-2025 10:36 PM

- So now we have a linux machine where docker is not installed
- To check IP in docker use **ip addr show,** it will show machine interface ETH0

```
630 Docker $ ip addr show
631 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
632     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
633     inet 127.0.0.1/8 scope host lo
634        valid_lft forever preferred_lft forever
635     inet6 ::1/128 scope host noprefixroute
636        valid_lft forever preferred_lft forever
637 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
638     link/ether 7c:1e:52:33:0d:f3 brd ff:ff:ff:ff:ff:ff
639     inet 10.5.0.11/24 metric 100 brd 10.5.0.255 scope global eth0
640        valid_lft forever preferred_lft forever
641     inet6 fe80::7e1e:52ff:fe33:df3/64 scope link
642        valid_lft forever preferred_lft forever
643 Docker $
```

Screen clipping taken: 13-08-2025 10:42 AM

- Now we will install the docker
- Now if we check the IP it will show docker interface as well



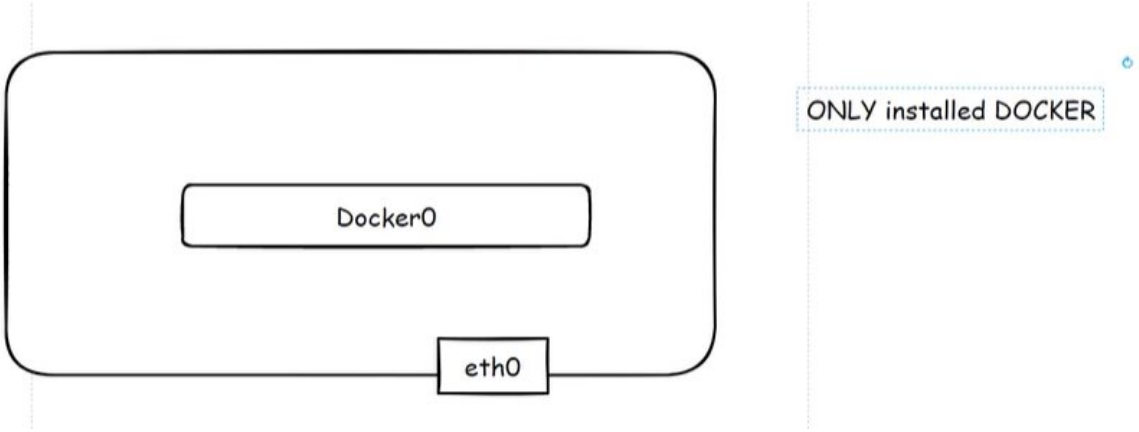
Screen clipping taken: 13-08-2025 10:46 AM

- So whenever we install docker this interface get installed, **so the docker0 is a bridge driver,**
- To check drivers in the docker use **docker network ls**

```
Docker $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
3a5107b8516b   bridge    bridge    local
fb9ac4474563   host      host      local
65342fb1cbae   none      null      local
Docker $
```

Screen clipping taken: 13-08-2025 10:50 AM

- It is **a virtual ethernet bridge which is created by docker** just like shown in the diagram



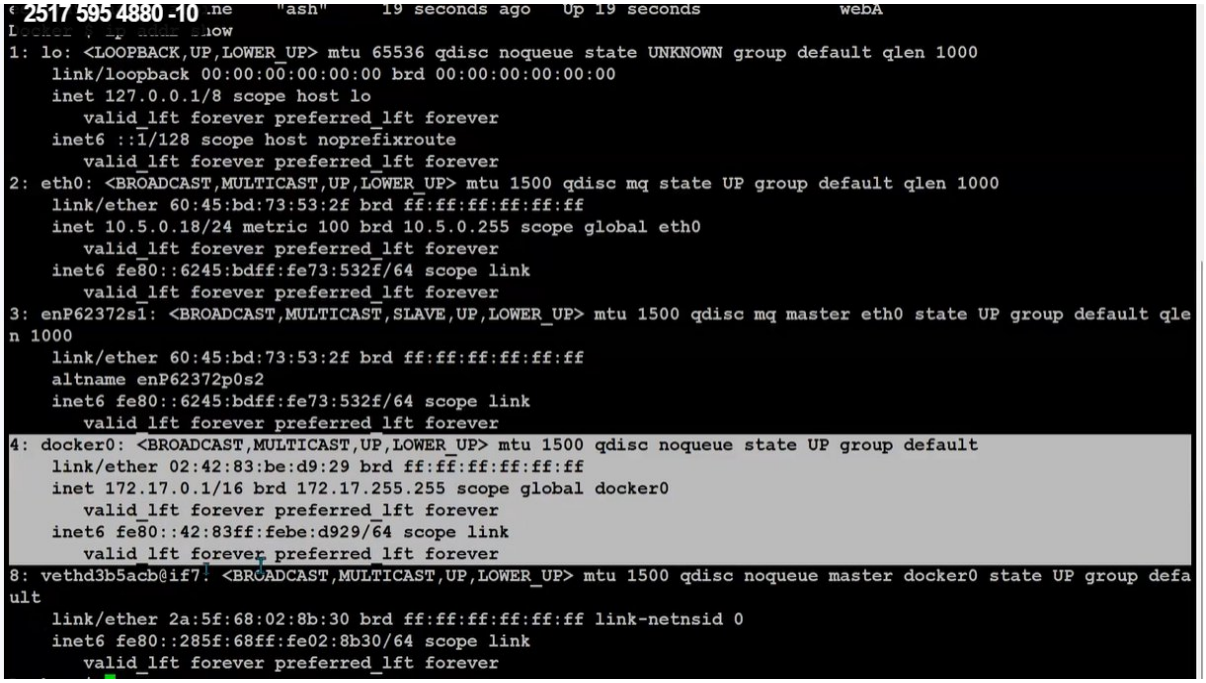
Screen clipping taken: 13-08-2025 02:12 PM

- Now we will create a container

```
65342fb1cbae   none      null      local
Docker $ docker run -dit --name webA alpine ash
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
9824c27679d3: Pull complete
Digest: sha256:4bcff63911fcb4448bd4fdacec207030997caf25e9bea4045fa6c8c44de311d1
Status: Downloaded newer image for alpine:latest
eece7a95baa2ff3e38f3a99bd010b8580cd1d353d92872473ec799a00391dd85
Docker $
```

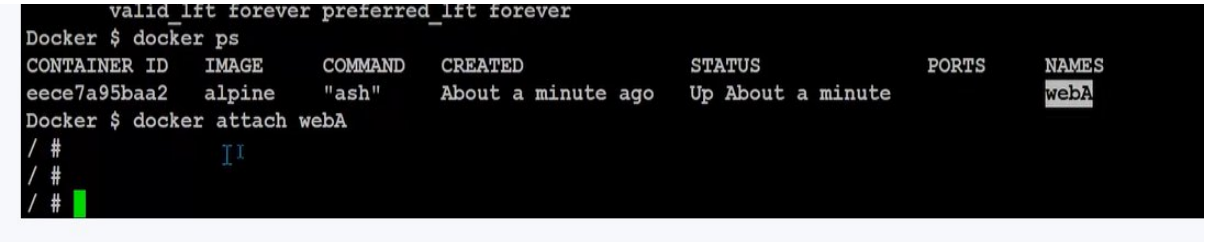
Screen clipping taken: 13-08-2025 02:13 PM

- Now we will check the IP and will get this result



Screen clipping taken: 13-08-2025 02:14 PM

- So when we create container this will also have a ethernet card known as **vethd3**
- Now to check the same we will go inside the container



Screen clipping taken: 13-08-2025 02:15 PM

- Now if we check the IP the result will be this **docker attach webA**

```
Docker $ docker attach webA
/ #
/ #
/ #
/ #
/ #
/ # ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ #
```

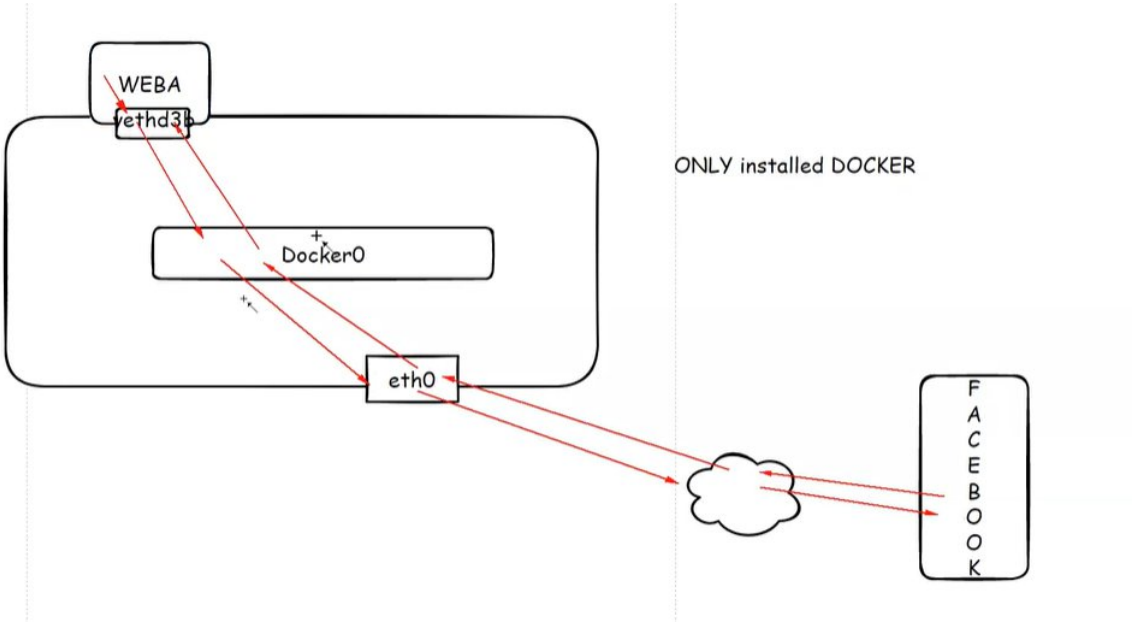
Screen clipping taken: 13-08-2025 02:16 PM

- Now we can ping facebook and google from the container

```
2517 595 4880 -10
/ # ping facebook.com
PING facebook.com (157.240.237.35): 56 data bytes
64 bytes from 157.240.237.35: seq=0 ttl=46 time=7.076 ms
64 bytes from 157.240.237.35: seq=1 ttl=46 time=9.197 ms
64 bytes from 157.240.237.35: seq=2 ttl=46 time=7.116 ms
64 bytes from 157.240.237.35: seq=3 ttl=46 time=7.062 ms
64 bytes from 157.240.237.35: seq=4 ttl=46 time=7.415 ms
64 bytes from 157.240.237.35: seq=5 ttl=46 time=7.257 ms
^C
--- facebook.com ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 7.062/7.520/9.197 ms
/ # ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=112 time=4.069 ms
64 bytes from 8.8.8.8: seq=1 ttl=112 time=4.002 ms
64 bytes from 8.8.8.8: seq=2 ttl=112 time=4.126 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 4.002/4.065/4.126 ms
/ #
```

Screen clipping taken: 13-08-2025 02:18 PM

- So the traffic will flow from **vethd3h to docker0 and from there it will go to eth0 and from eth0 it will go to facebook**



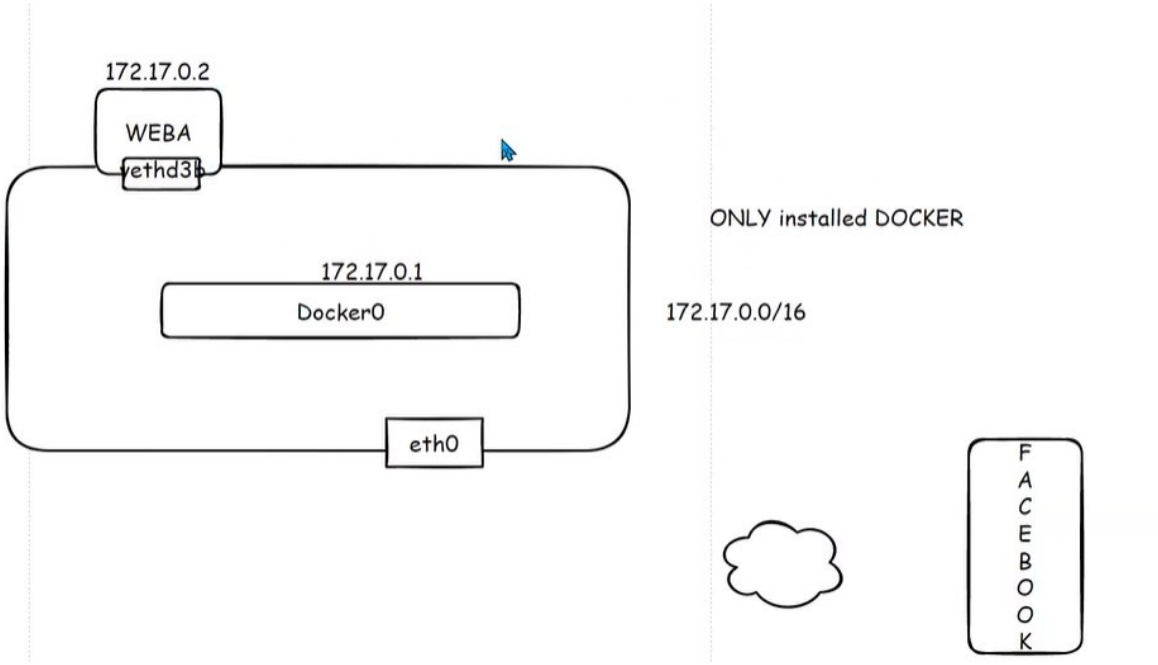
Screen clipping taken: 13-08-2025 02:19 PM

- Now we will inspect the bridge

```
Docker $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
3a5107b8516b   bridge    bridge    local
fb9ac4474563   host      host      local
65342fb1cbae   none      null      local
Docker $ docker network inspect bridge^[[D
[]
Error response from daemon: network bridg not found
Docker $ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "3a5107b8516bd68cf7ca793bf10369e29a0872ef18fe15183ad513f6ea4fc4dc",
        "Created": "2025-08-10T04:48:53.133444135Z",
        "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,
```

Screen clipping taken: 13-08-2025 02:21 PM

- So from here the IP distribution will be like this



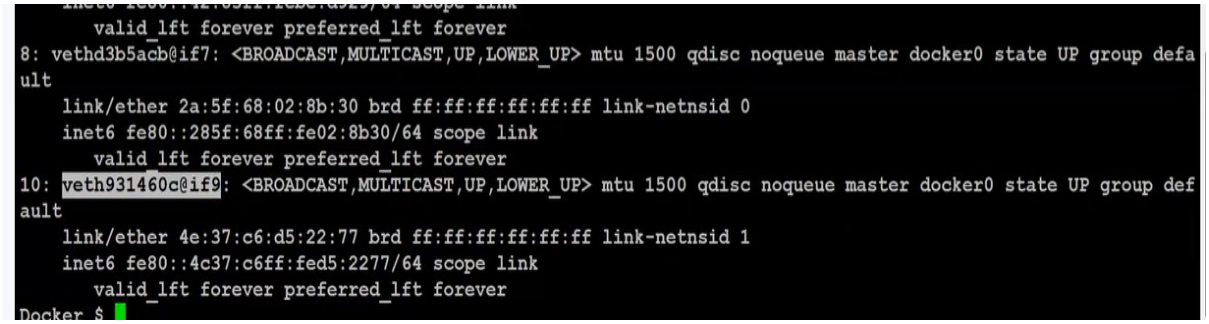
Screen clipping taken: 13-08-2025 02:22 PM

- Now **we will create another container**

```
Docker $ docker run -dit --name WebB alpine ash
3f14e28f2e9dc220437e05135bc11f2c1147bc9c57e87f4ea0d94c5ff4d037ec
Docker $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
3f14e28f2e9d   alpine    "ash"     4 seconds ago    Up 3 seconds              WebB
eece7a95baa2   alpine    "ash"     7 minutes ago    Up 7 minutes              webA
Docker $
```

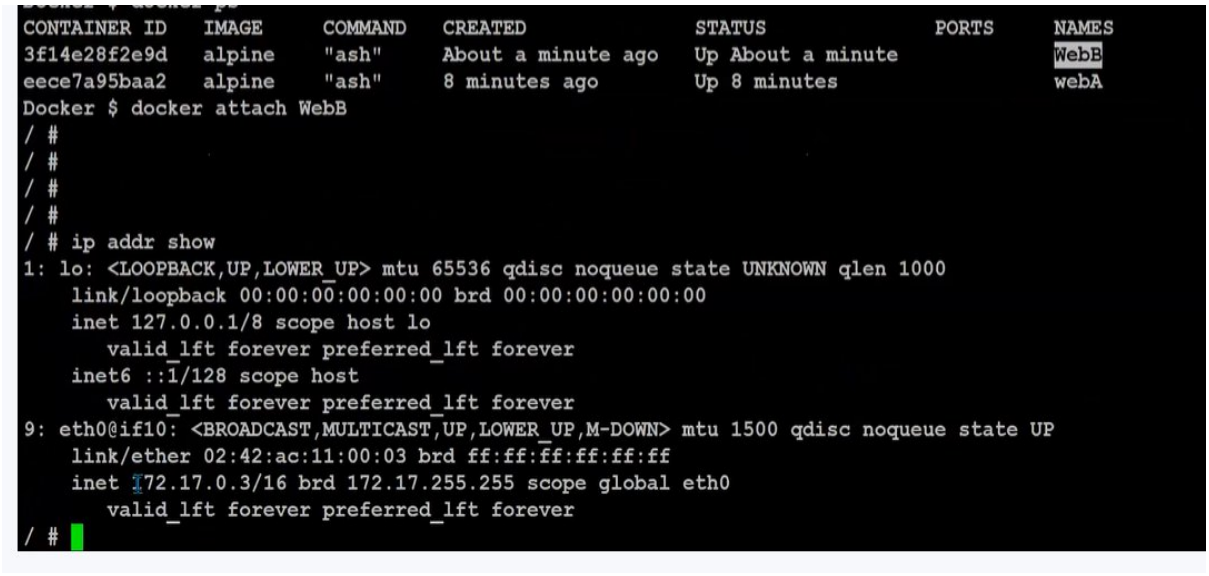
Screen clipping taken: 13-08-2025 02:23 PM

- So **if we check the IP it will show another veth**



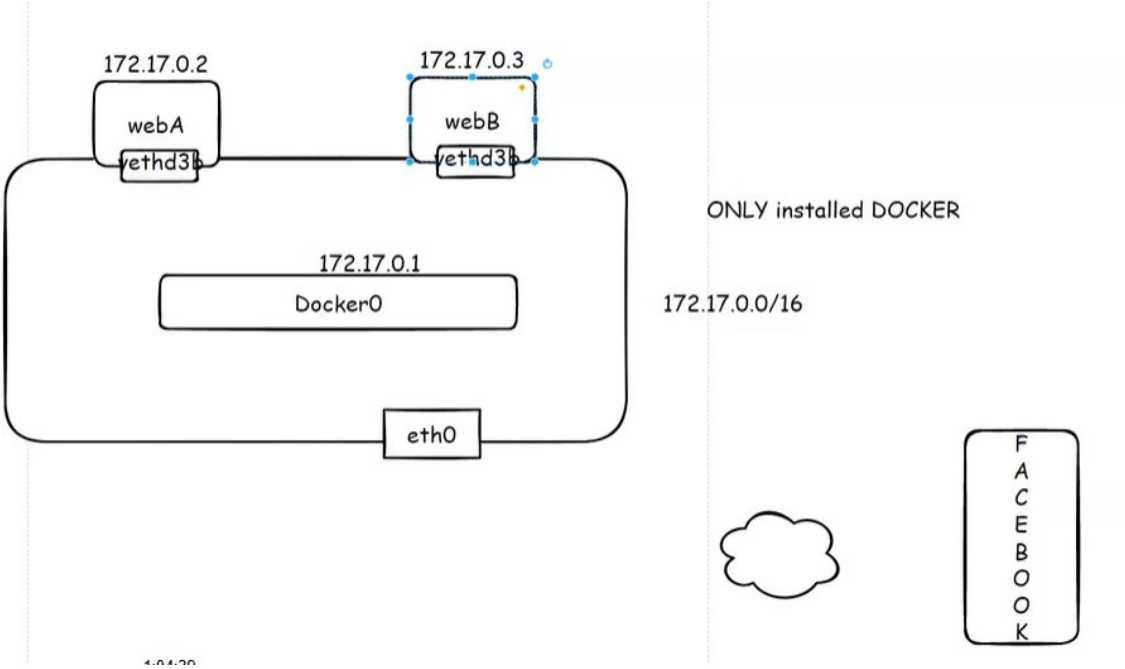
Screen clipping taken: 13-08-2025 02:24 PM

- Now we will check the IP of this container by going inside the docker



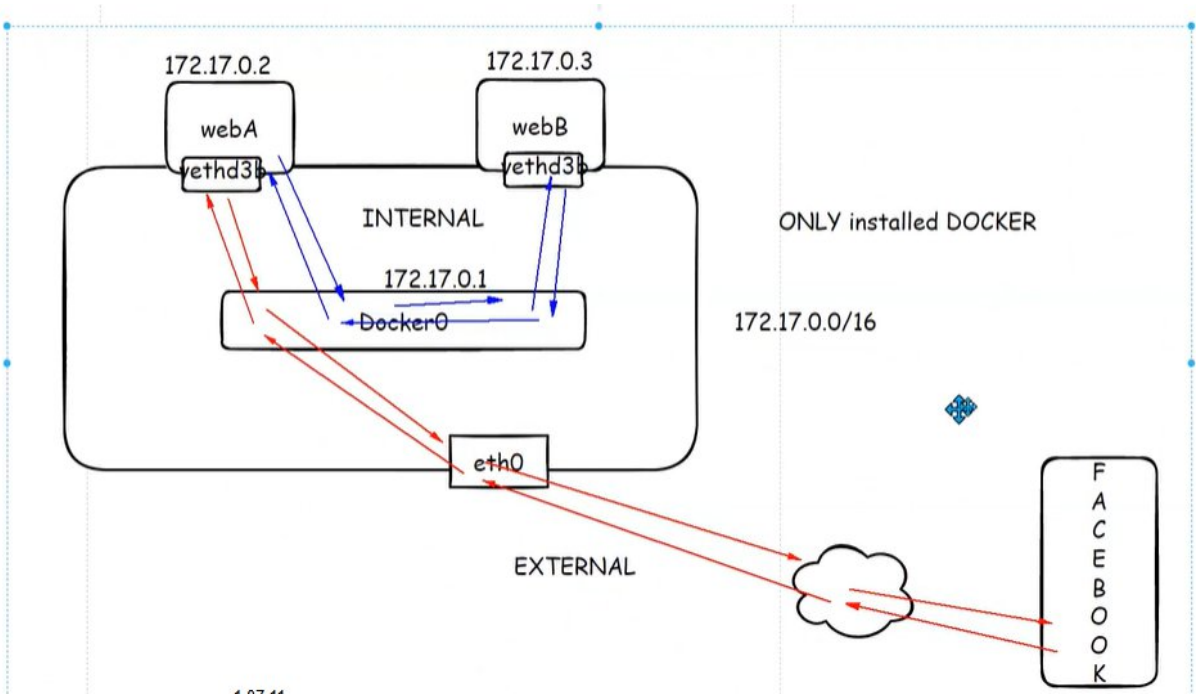
Screen clipping taken: 13-08-2025 02:25 PM

- Now the Ips will be like this



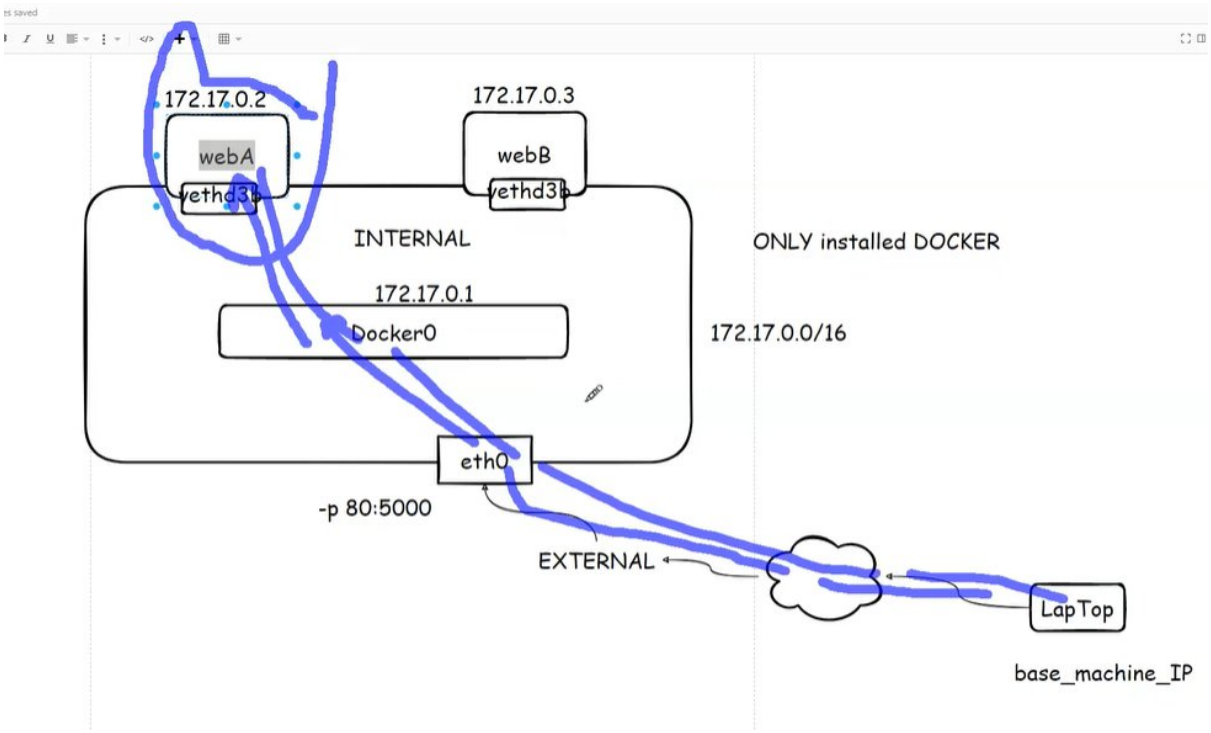
Screen clipping taken: 13-08-2025 02:26 PM

- And we would be able to ping google from this container as well
- And we can ping from one container to another container
- So for containers the traffic will be like this



Screen clipping taken: 13-08-2025 02:28 PM

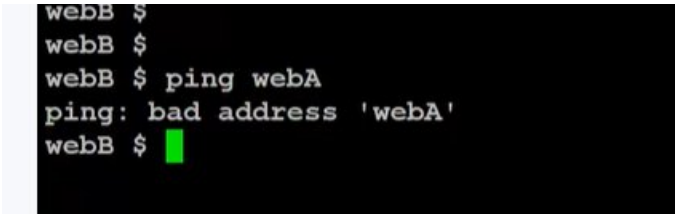
- So now port mapping traffic will flow like this, here once we hit port 80 it will take the traffic to 5000 port (WEB A) and we would be able to see our webpage



Screen clipping taken: 13-08-2025 02:31 PM

User Defined driver

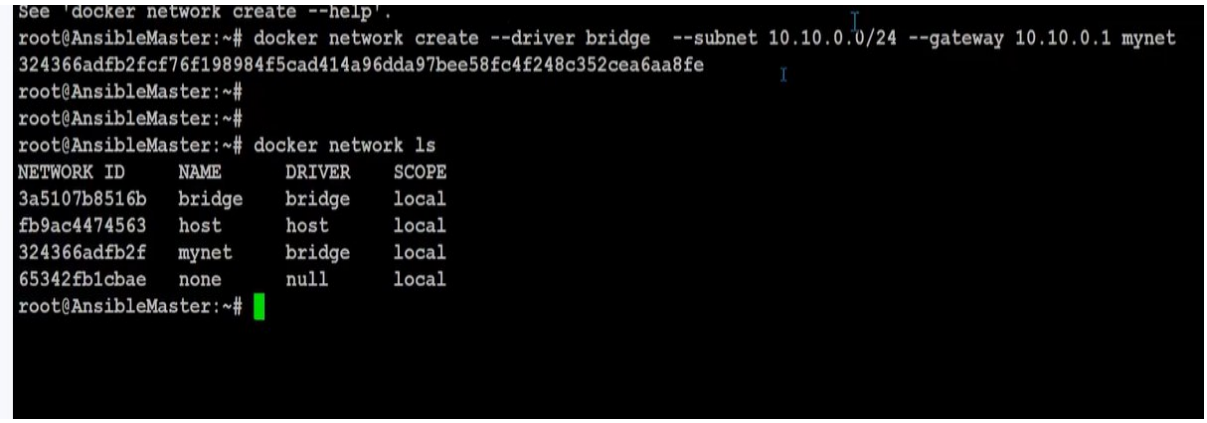
- **Limitation of bridge driver is we can ping the other container with IP but not with container name, it means DNS is not configured**



Screen clipping taken: 13-08-2025 02:35 PM

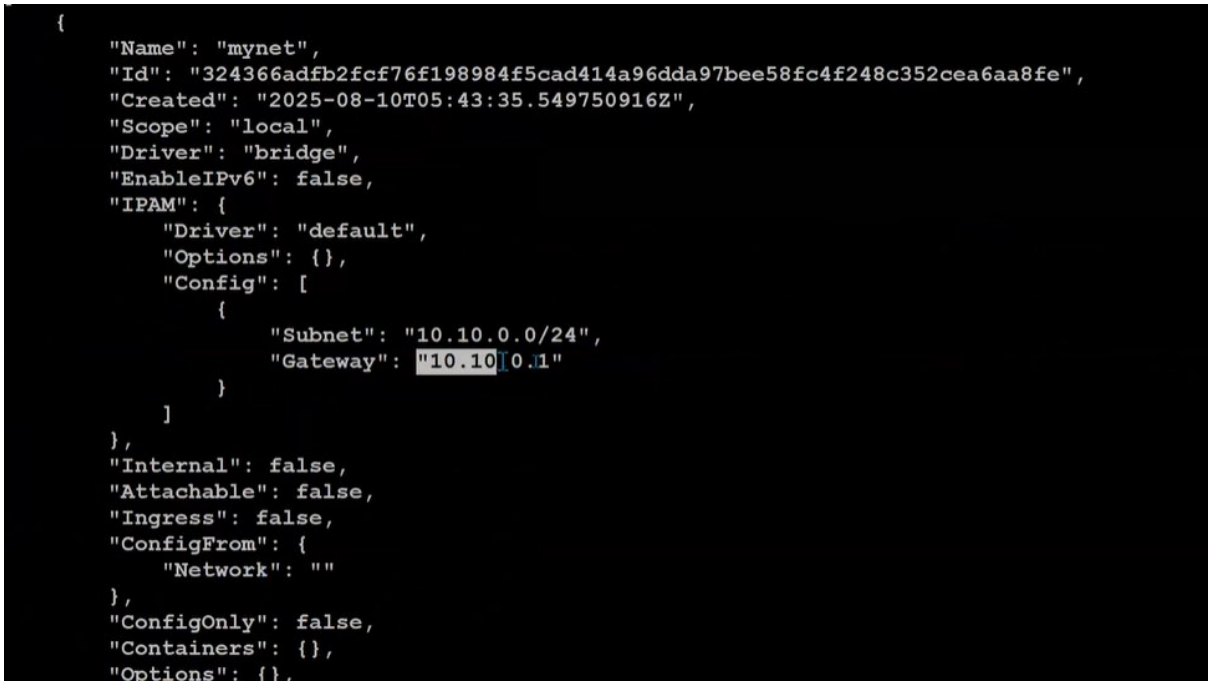
- **Docker CIDR range is fixed which is 172.0.0.0/16**
- Which is why we create our own driver to overcome this issue
- We can create it using **docker network create --driver bridge mynet (name)** and it will choose its own IP
- But if we want to choose our own IP we will use this

Docker network create --driver bridge --subnet 10.10.0.0/24 --gateway 10.10.0.1 mynet



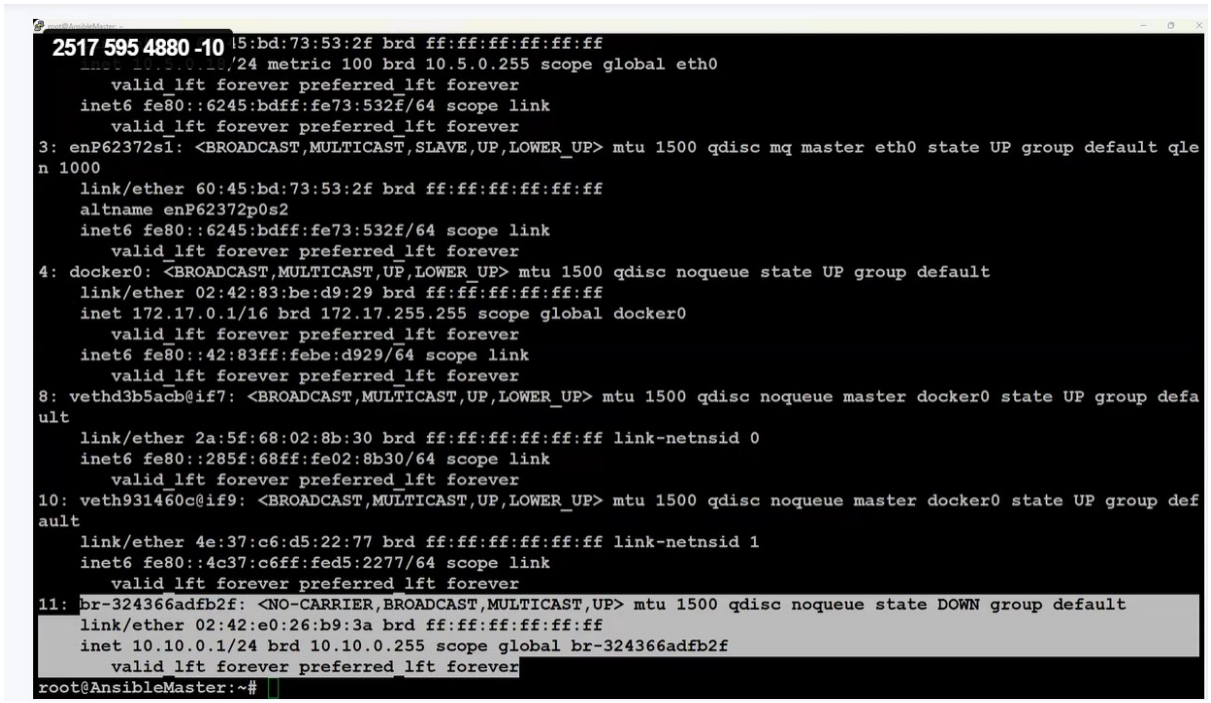
Screen clipping taken: 13-08-2025 02:41 PM

- If we inspect the driver we will see the given IP



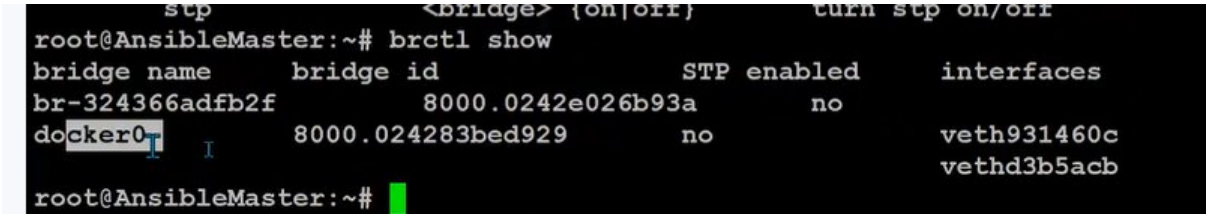
Screen clipping taken: 13-08-2025 02:42 PM

- If we check the IP address of the new driver we will see new ethernet



Screen clipping taken: 13-08-2025 02:44 PM

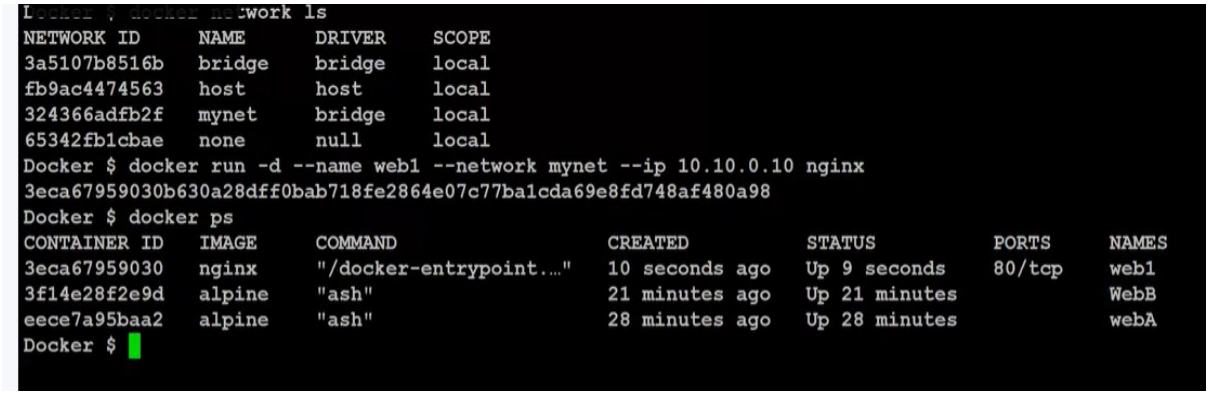
- To check all the bridges which was created in the machine we can use **brctl show**, the above is created by us and the docker0 was created by docker



Screen clipping taken: 13-08-2025 02:45 PM

- Now we want container to use this driver we will use this command, we can give our own IP as well

docker run -d --name web1 --network mynet --ip 10.10.0.10 nginx

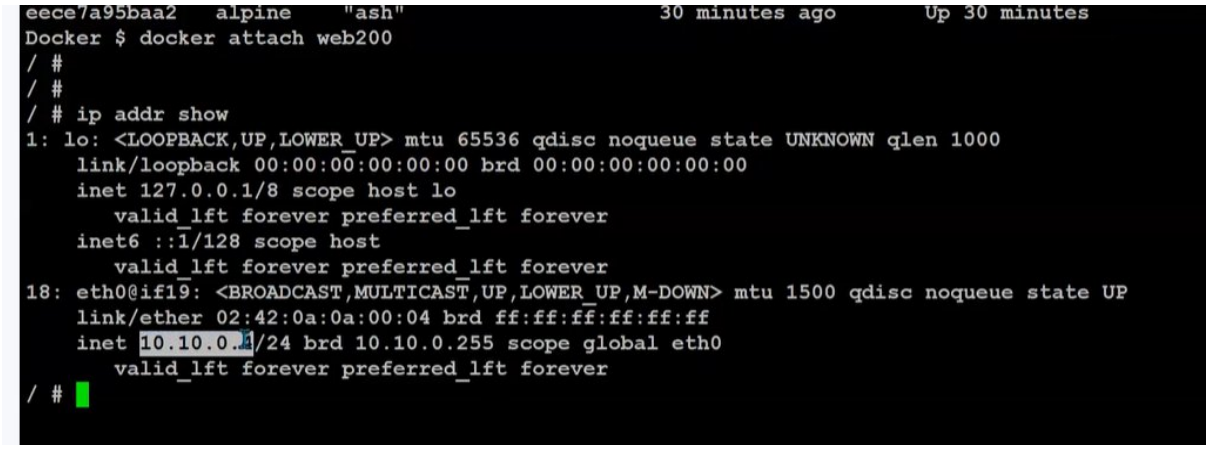


Screen clipping taken: 13-08-2025 02:48 PM

- We created one more container but didn't give any IP

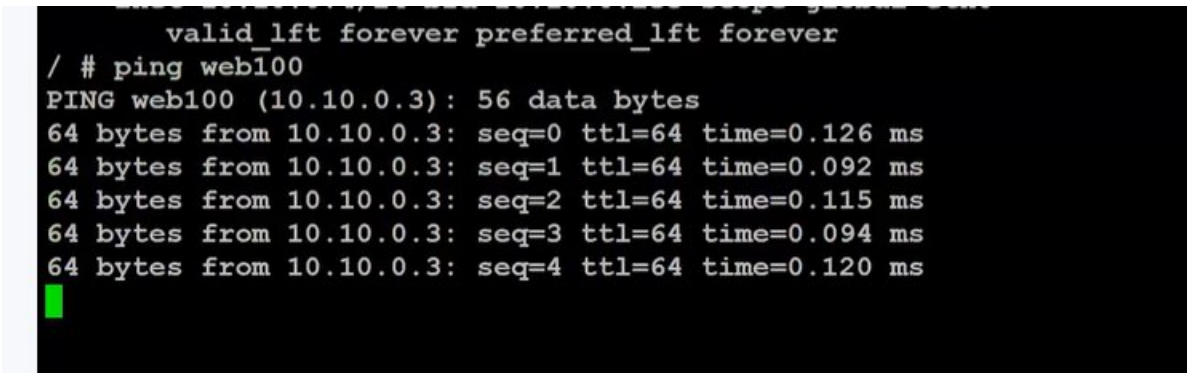
Screen clipping taken: 13-08-2025 02:51 PM

- Now we will go inside web200



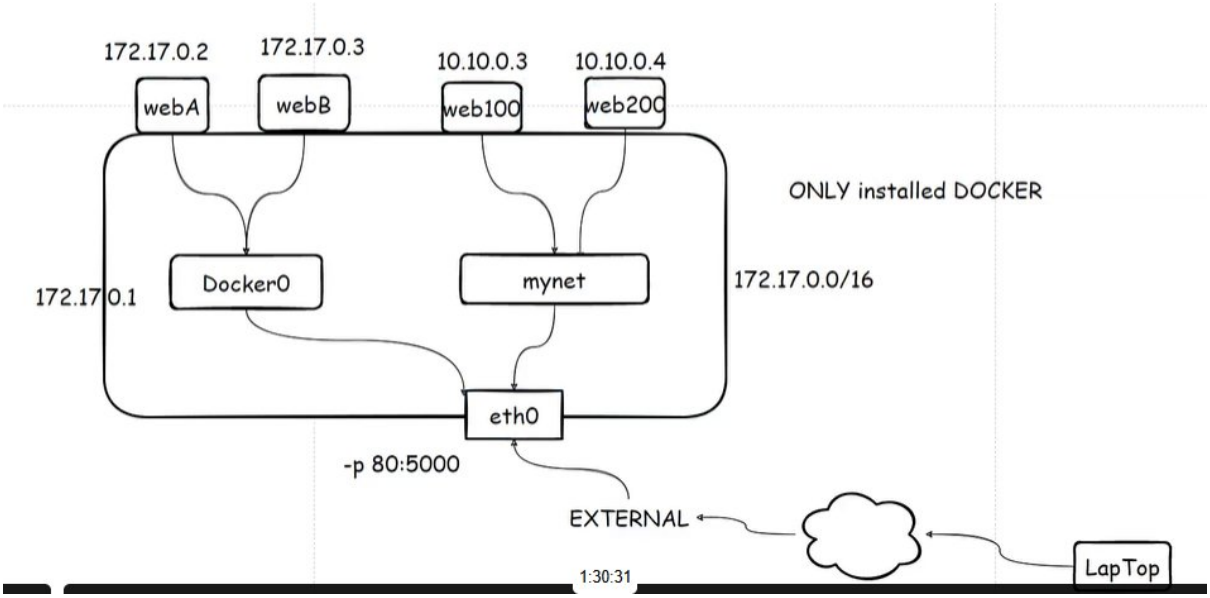
Screen clipping taken: 13-08-2025 02:51 PM

- So here **if we ping web100 by name from web200 it is working**



Screen clipping taken: 13-08-2025 02:52 PM

- Here in the diagram below we will see default driver and our own created driver

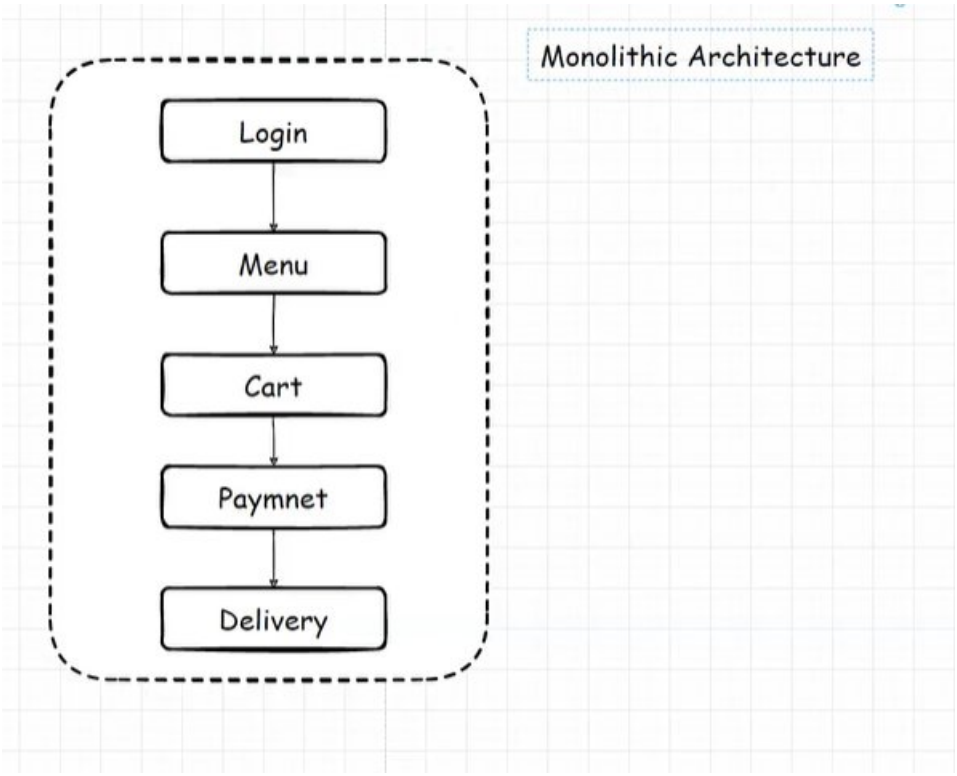


Screen clipping taken: 13-08-2025 02:55 PM

- Now if we want to connect WebB to web100 we will need to connect docker0 to mynet driver

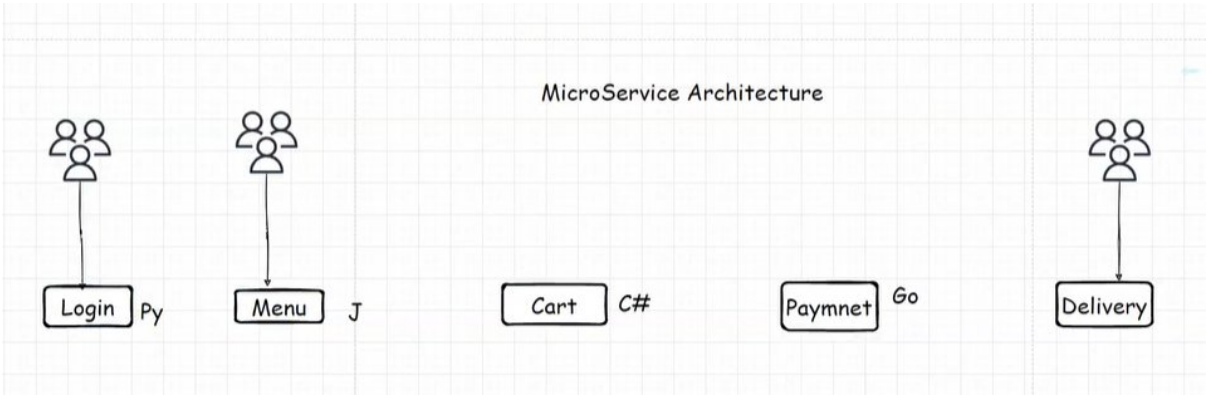
Container Orchestration Tool

- so for example these are the steps which we follow to order the pizza, **if we are writing a single program for all the steps is known as monolithic architecture**



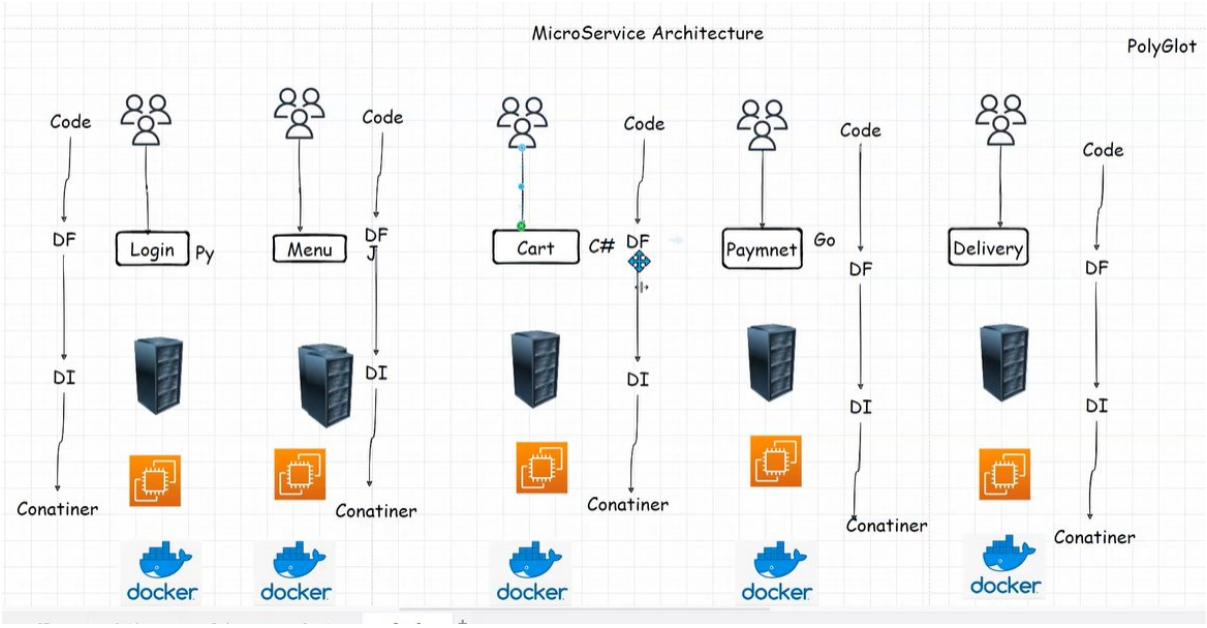
Screen clipping taken: 13-08-2025 03:01 PM

- **Monolithic means it is written in one single language , scaling of application as single unit we can't scale the single unit (like login, menu etc)we can only scale the whole server**
- Pros --- simple and easy to deploy
- Cons --- can't code in different language for different business logic
  ---can't scale each unit as per the requirement can scale the whole server
- Here these steps are but independent applications means it is written by different users and these are written in different languages and this is known as **microservice architecture**



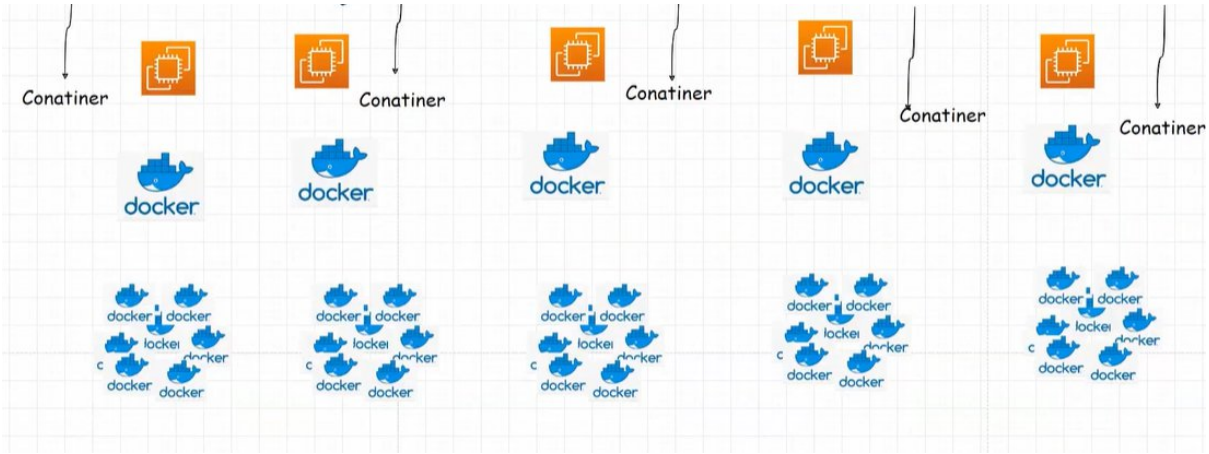
Screen clipping taken: 13-08-2025 03:36 PM

- In this we can use multiple languages known as **polyglot**
- Running these services on physical machines (**BareMetal**) is not good idea as this will be very expensive
- Yes we can go for VM for these services but still this is not a better choice
- So containers are the best to run these services



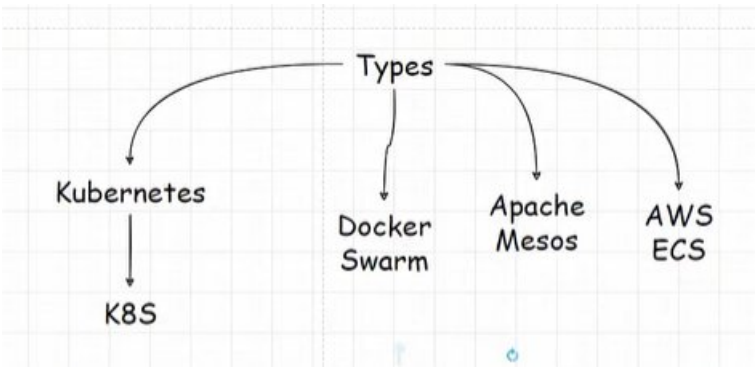
Screen clipping taken: 13-08-2025 05:50 PM

- So just in case if container goes down so users will not be able to use that service for an instance Netflix, so in this we will not install service on one container instead on multiple containers so that we don't lose the access to Netflix



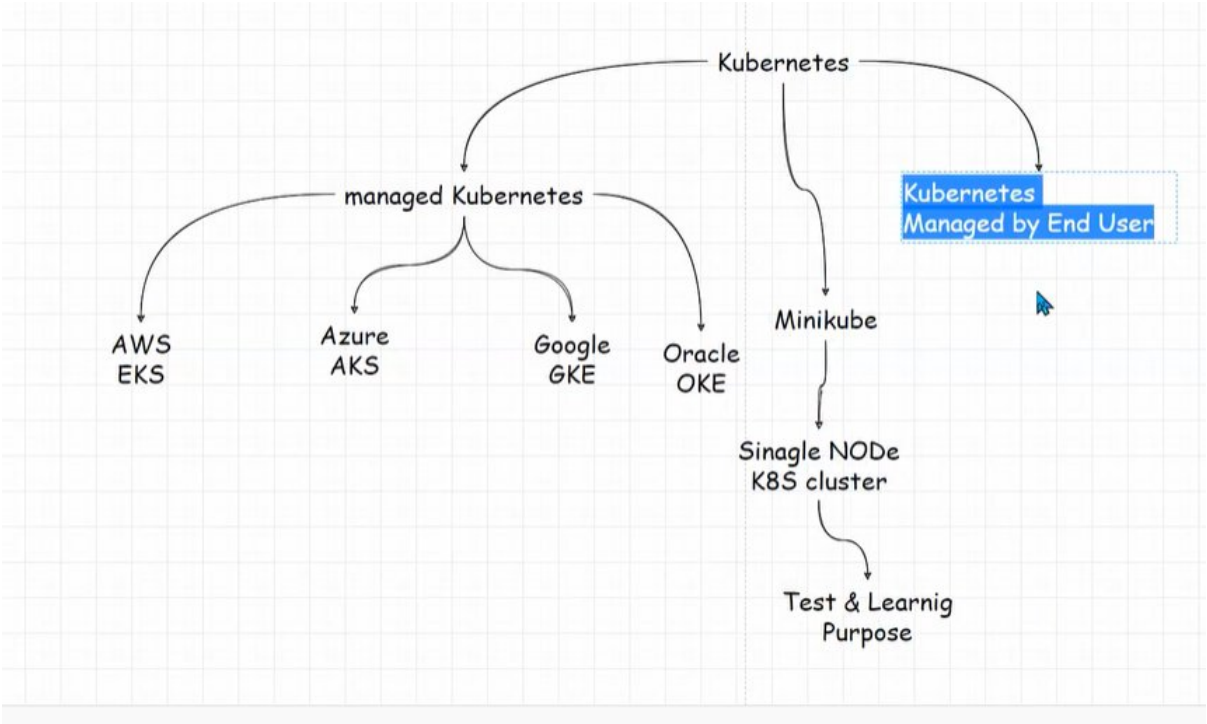
Screen clipping taken: 13-08-2025 05:52 PM

- So to manage these many containers we use **container orchestration tool**
- It is a tool which automate lot of things **like deployment, management(if any container goes down it will take care of it), Scaling(it will scale up and down automatically as per traffic)**
- So we have types of this tool



Screen clipping taken: 13-08-2025 06:00 PM

- Kubernetes itself has 2 types



Screen clipping taken: 13-08-2025 06:04 PM

- Managed Kubernetes are managed by cloud service providers

Why we need Container Orchestration

- For multiple containers will need more machines, the collection of machines called cluster
- Manual management is impossible
- Scaling is difficult when traffic is high, automated scaling should be there
- Networking gets complex
- Recovery is slow, when we have a lot containers