**Install docker on Red Hat Linux.**

The first one bundled with RHEL/CentOS 7 distro and can be installed with the yum.

The second version distributed by the Docker project called docker-ce (community free version).

The third version distributed by the Docker project called docker-ee (Enterprise paid version).

**Repo for community free version:**

yum-config-manager --add-repo <https://download.docker.com/linux/centos/docker-ce.repo>

yum install docker-ce

systemctl enable docker.service | systemctl start docker.service

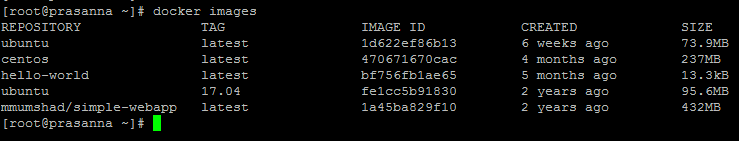
docker ps 🡪 To check running process.



docker ps -a 🡪To list all the process (Running and Exited).



docker images 🡪To list the images available.

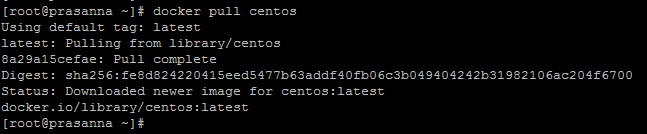


The images are by default pulled from docker hub.

To pull from any specific registry 🡪 docker pull myregistry.local/testing/test-image

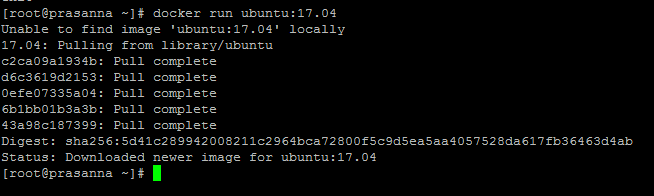
docker pull <image> 🡪To pull the image.

eg: docker pull centos

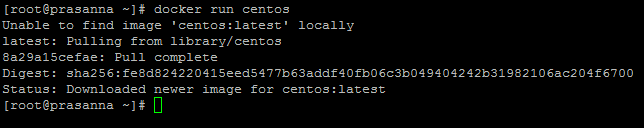


docker pull <image>:<tag> 🡪To pull a specific version on an image.

eg: docker pull ubuntu:17.04

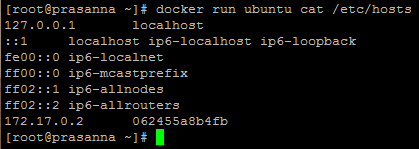


docker run <image> 🡪To run a image. If the image is not present locally, it will pull from docker hub and run it.



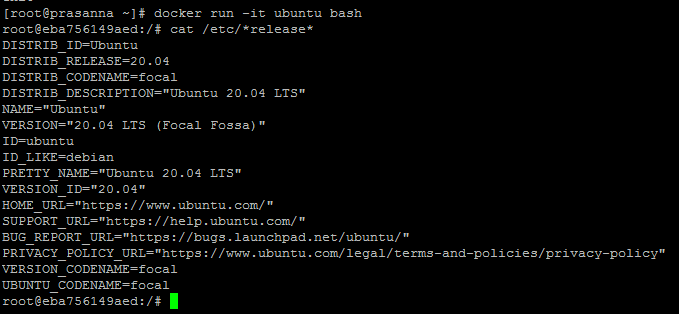
docker run <image> <process> 🡪To run any command in that image and that creates a container (ID and Name).

eg: docker run ubuntu cat /etc/hosts



docekr run -it <image> <process> 🡪To login to the image and creates a container (ID and Name).

eg: docker run -it ubuntu bash

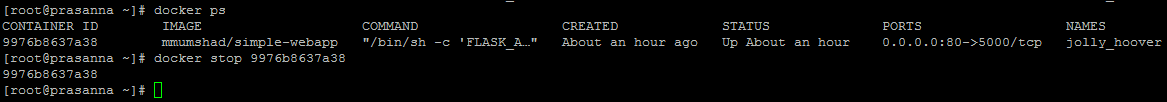


docker run -d <image> <command> 🡪To run a process in background.

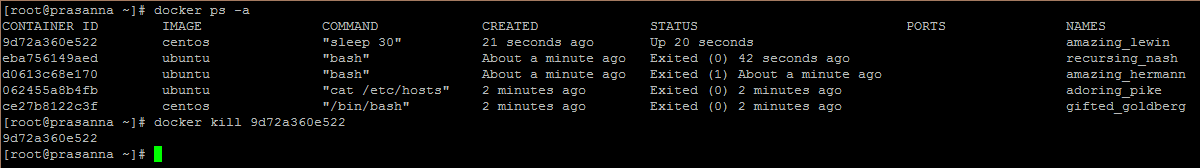
eg: docker run -d centos sleep 10



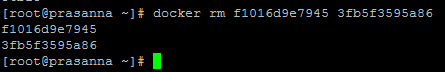
docker stop <container ID/Name> 🡪To stop a process.



docker kill <container ID/Name> 🡪To kill a process

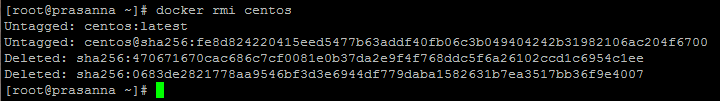


docker rm <container ID/Name> 🡪To remove the container which are Exited.



docker rmi <image> 🡪To remove an image.

eg: docker rmi centos



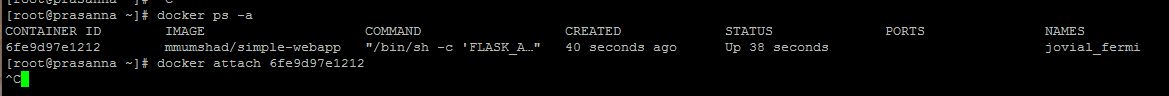
docekr exec <container ID> <command> 🡪To execute command using the current running container.

eg: docker exec f2d8492e6c5e cat /etc/\*release\*



docker attach <containerID/Name> 🡪To run a process in foreground.

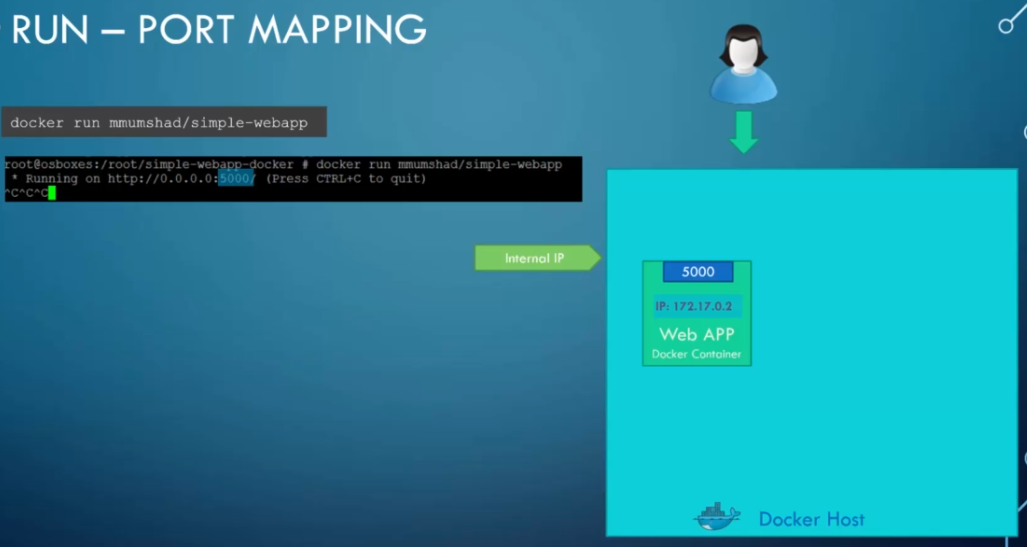
eg: docker attach d5f8dc8d7a1c

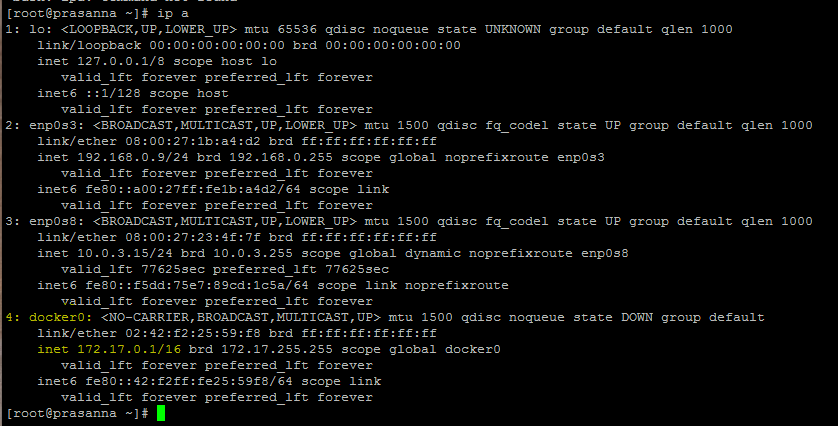


docker data are stored under - /var/lib/docker

**Port Mapping:**

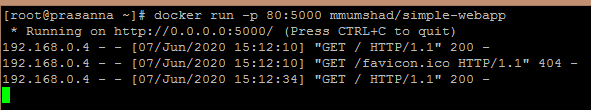
Every container has its own private IP, which will be accessible only from the host. The IP can be found from host server.

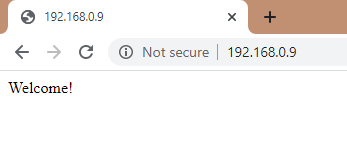




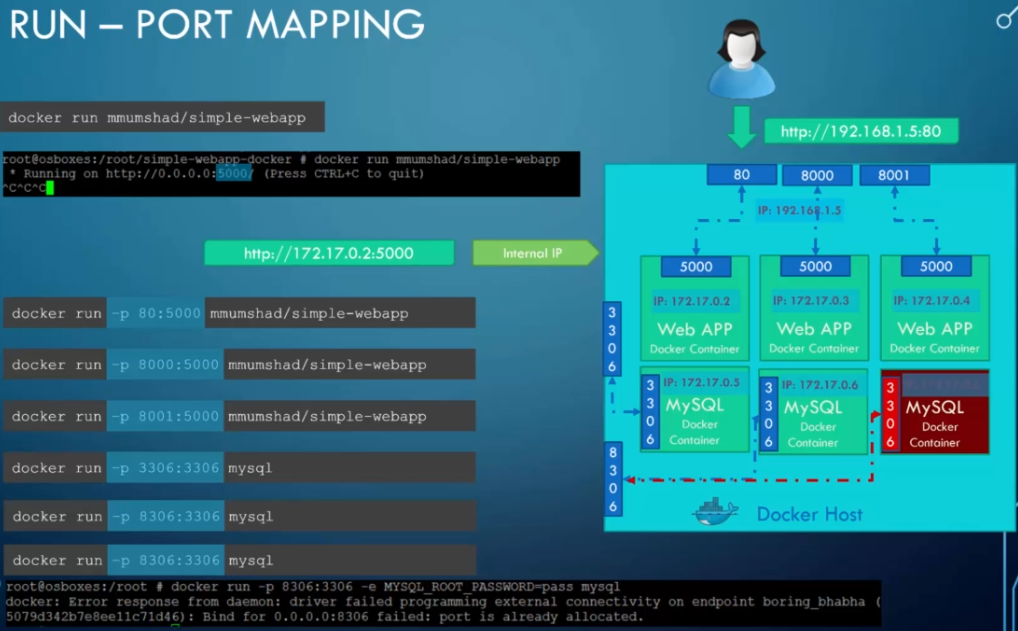
To allow the outer world to access the application, we need to port map the docker with host port.

eg: docker run –p <port>:<port> <image>





We can map multiple ports to a single application.



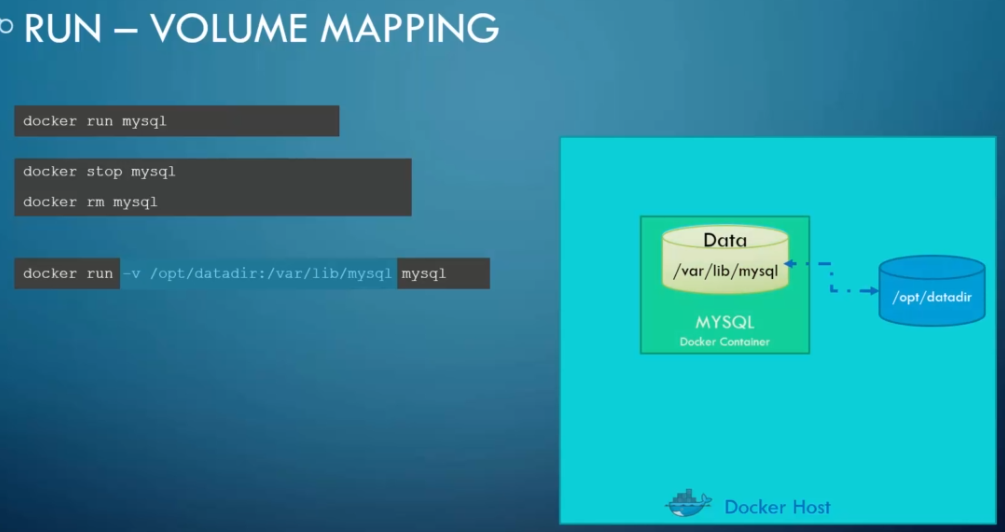
**Volume Mapping:**

docker run mysql 🡪 This is a database image, the data will be stored within the container under /var/lib/mysql.

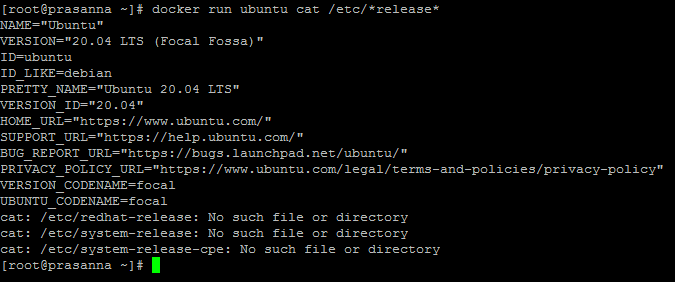
When “docker stop/rm mysql” is given, the data in it will also get deleted.

We can map the external volume (host) to the docker, the data would store on the external volume. So that when the docker is deleted, the data would persist in host itself. The data can retrieve by simply running the mysql mapping to the external volume.

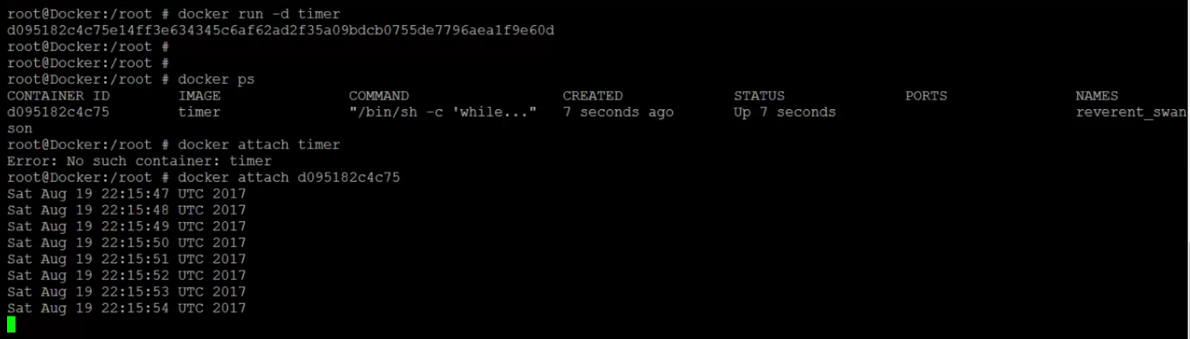
docker run -v /opt/datadir:/var/lib/mysql mysql



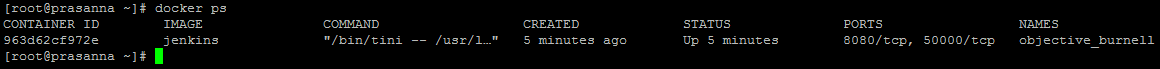
Before an image is run and exit, we can provide command to that image to display.



Attach and detach. Timer displays time every second.

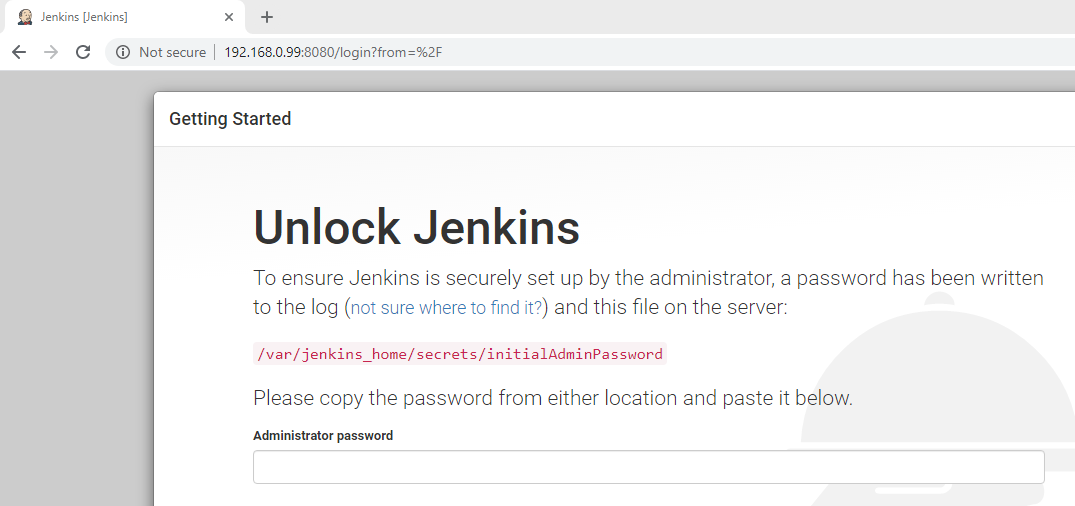


docker run jenkins 🡪 To run Jenkins images.

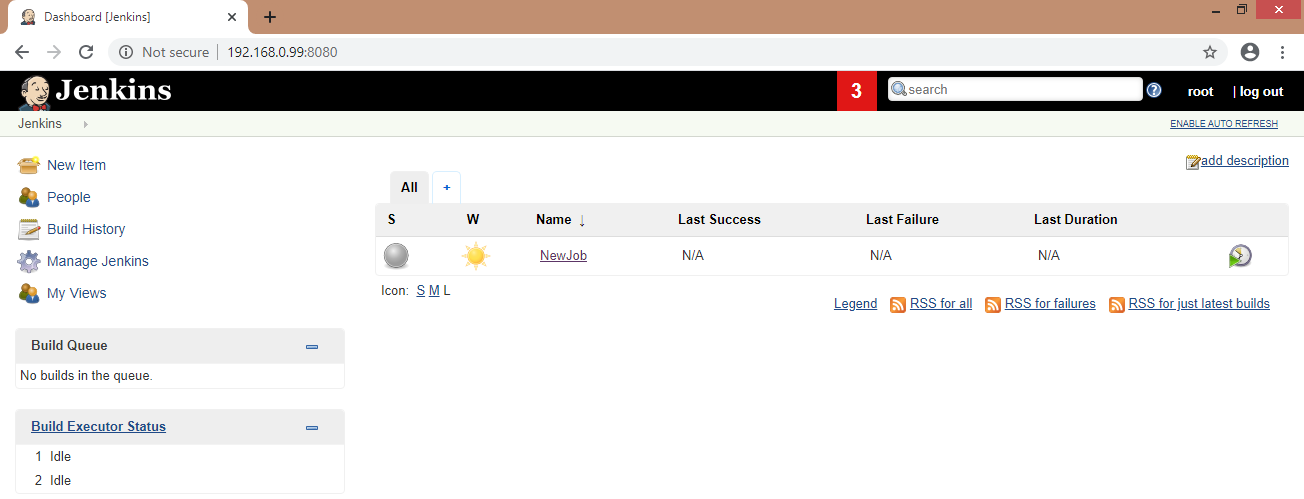


docker inspect <container ID> 🡪 To find the details of running container like Version, Volume, port, IP and so on.

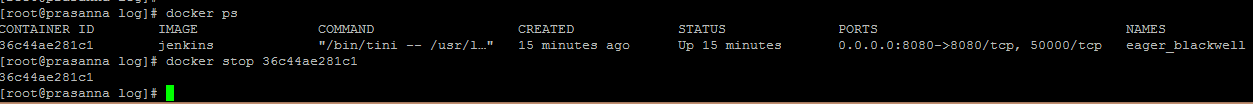
docker run -p 8080:8080 jenkins 🡪 To access the Jenkins with external IP.



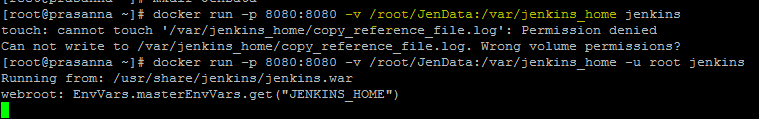
New job created in the name of “NewJob”.



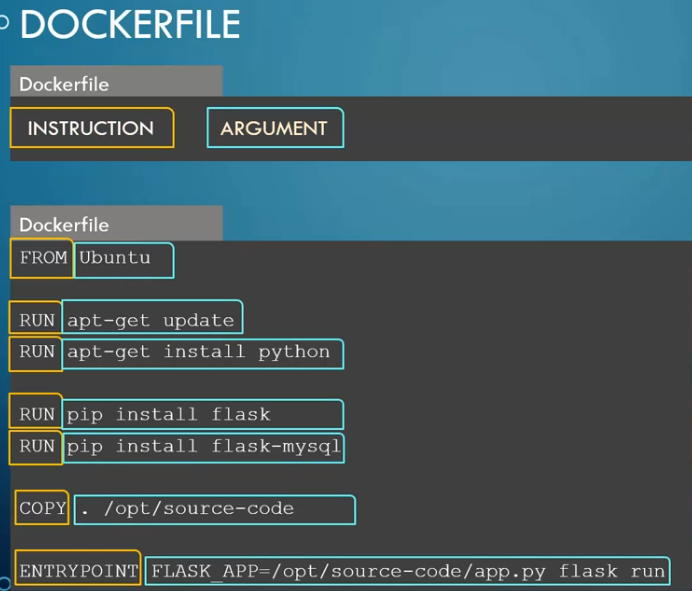
If the container is stopped and re-run the Jenkins, the job that is created would disappear.



To make the data persist, we need to use volume mapping. The data now store in host server - /root/JenData. If the container is stopped and re-run, the data would persist.

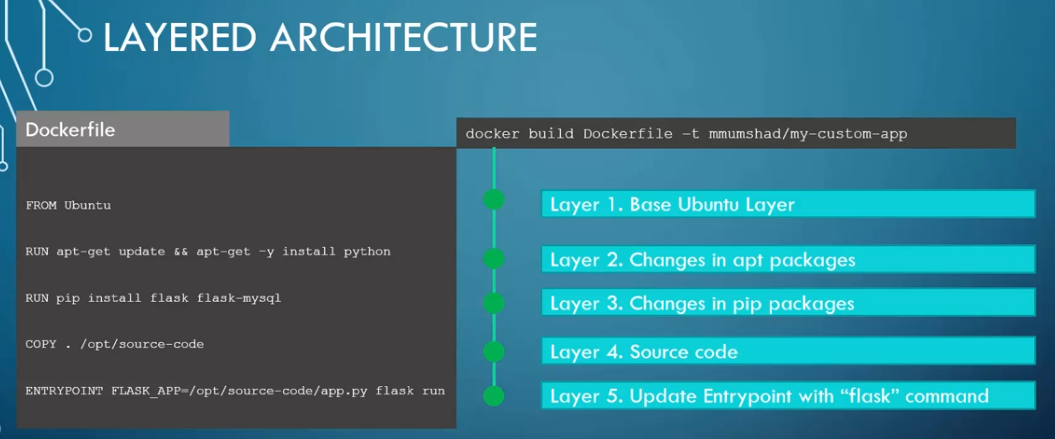


**Building an image:**

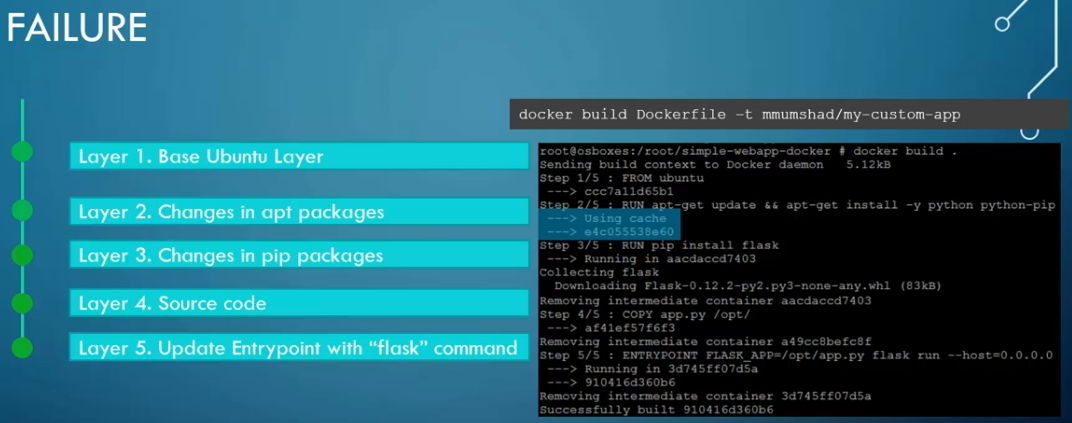


FROM 🡪 must be a base OS or another image that has base OS.

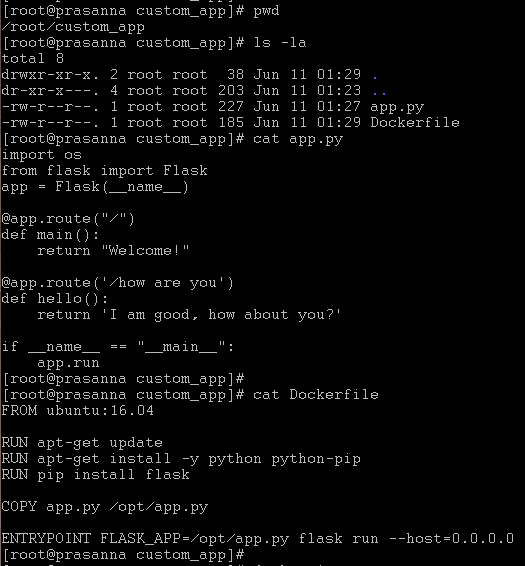
Docker image is a layered architecture. Each layer consists of changes from previous layer.



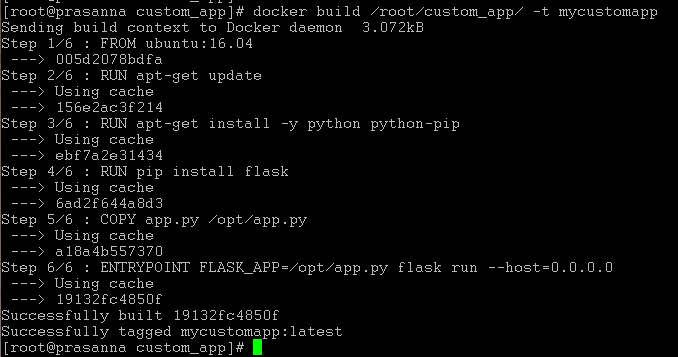
If the build failed for some reason, you can modify the file and re-run it. In that case, the previous successful executed step would store in cache and when you re-run it the successfully executed layer pulled from cache and start run from the failed layer.

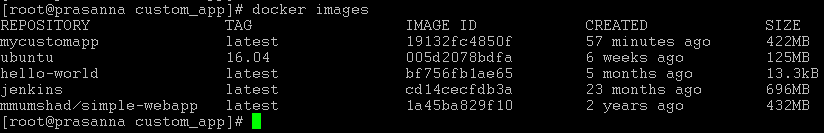


**Building an own image:**

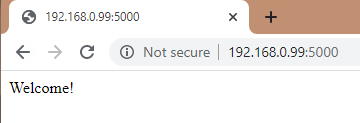


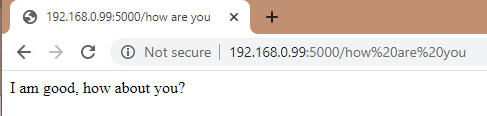
docker build <path\_of\_Dockerfile> –t <ImageName>





Map the image port with host to access the webpage from external world.

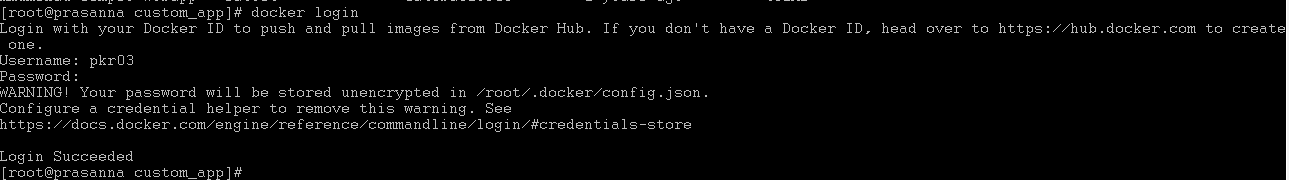




**Pushing the image to docker hub:**

docker login

docker push mycustomapp



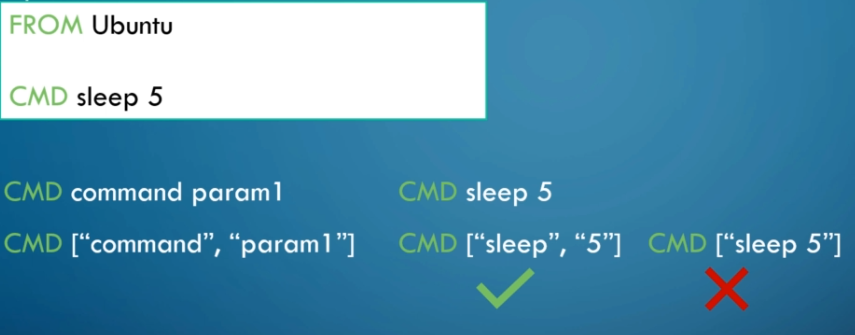
**Command VS Entrypoint**

**CMD** – we should specify the command with parameter in Dockerfile.

The command can be in:

shell format (CMD command parameter) or

JSON format (CMD [“command”, “parameter”]).

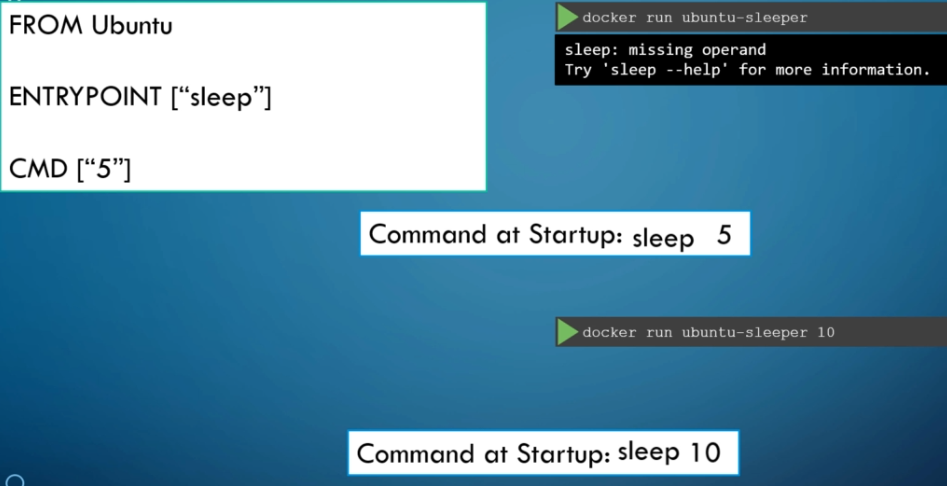


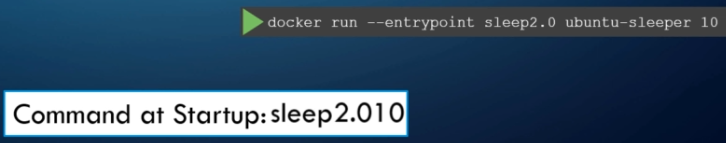
**Entrypoint** – we should specify only the command in Dockerfile, the parameter can be insert while run the image. If no parameter mentioned while running the image, it will throw error.



We can set default parameter in Dockerfile but also can override that with different parameter while we run the image.

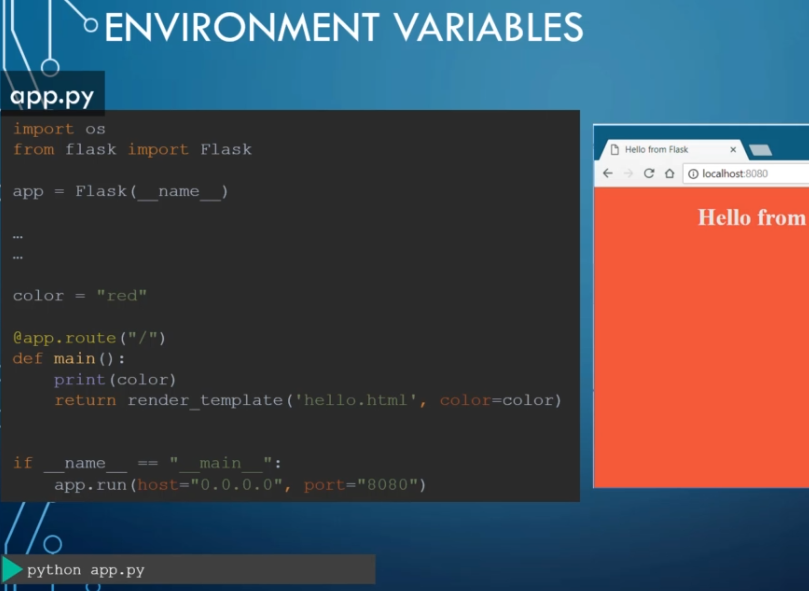
The below example: ENTRYPOINT as “sleep” and CMD as “5”. If no parameter specified when running the image, it takes “sleep 5” as default value. If you specify parameter (10) while run the image, the default value (5) would get override with new value (10).



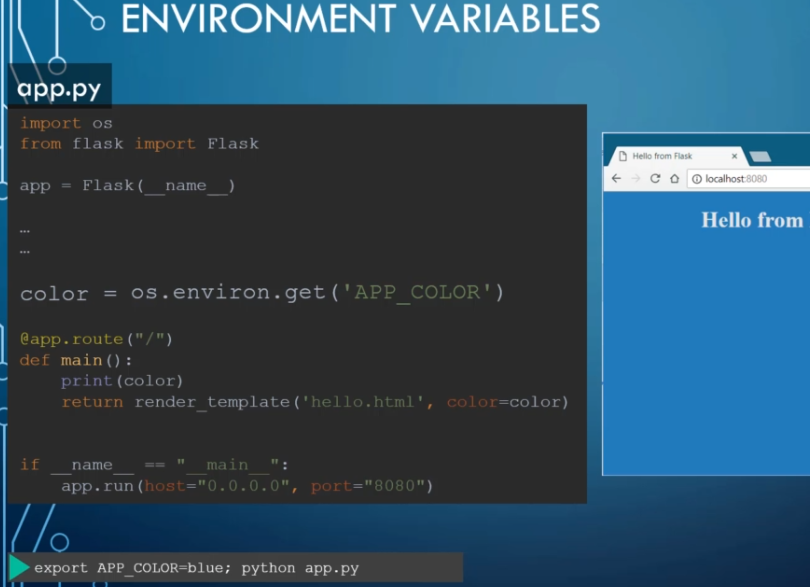
The same way, we can override ENTRYPOINT as well.

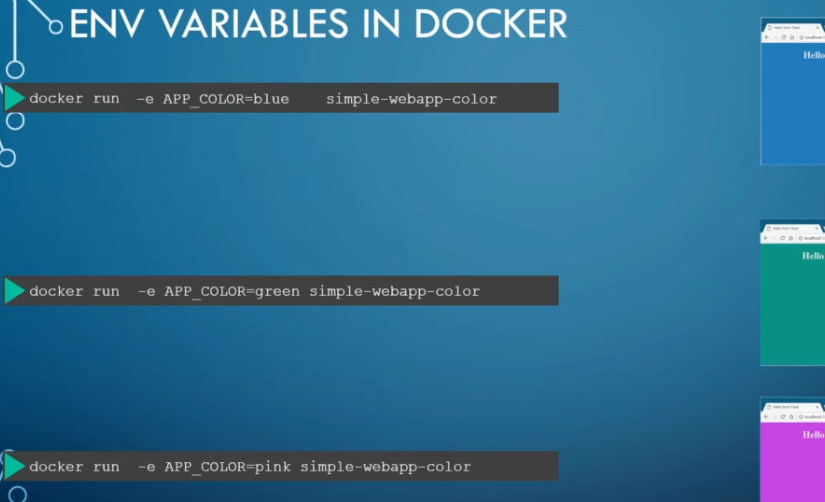
**Environment variable:**

The code has line “color = red”. Whenever this docker is run, the webpage display in red color. If you want to change the line to “color = blue” in future, you will have to modify the code.



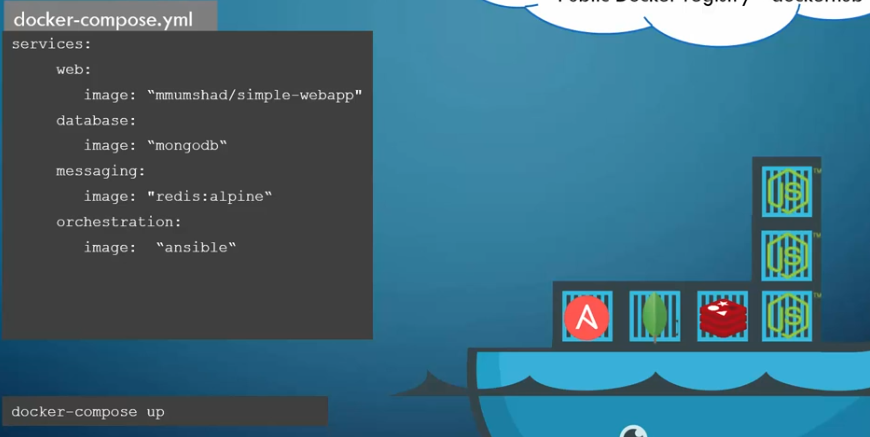
Instead, we can move such information outside of the code by setting environment variable. We can also mention the environment variable while performing the docker run with the option –e.





**Docker Compose:**

Docker compose is used to setup/run a large number of service/images in server. All the service information will be written in docker compose file. Docker compose should be in .yml (YAML) format. This file has service (image) that needs to be run. In the below example, we are running multiple images using docker-compose up command.

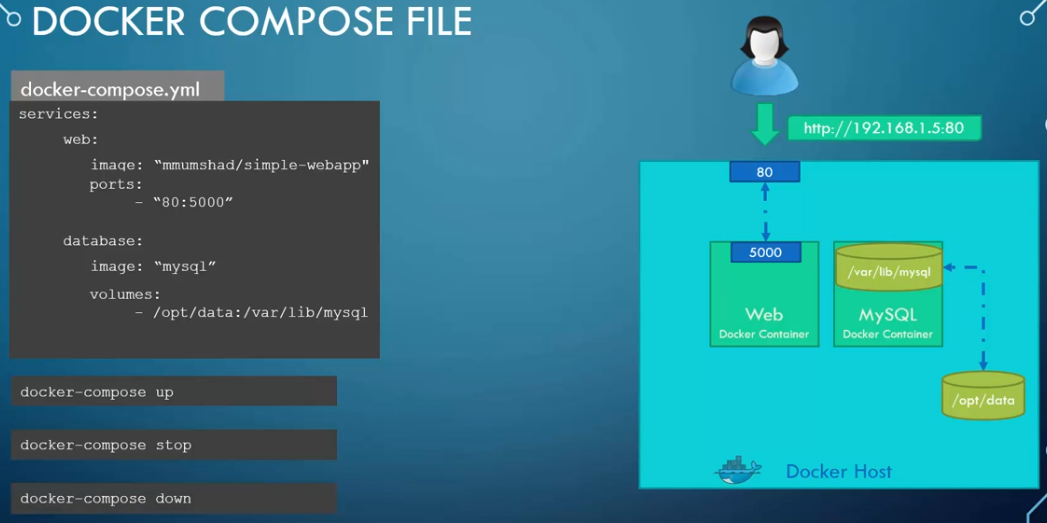


We can also do port mapping, volume mapping in the .yml file.

docker-compose up 🡪 Bring the containers up.

docker-compose stop 🡪 Stop the containers.

docker-compose down 🡪 Stop the containers and remove it.

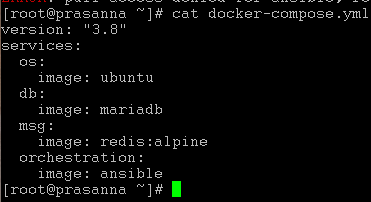


**Hands-on on docker compose:**

Before running docker-compose up



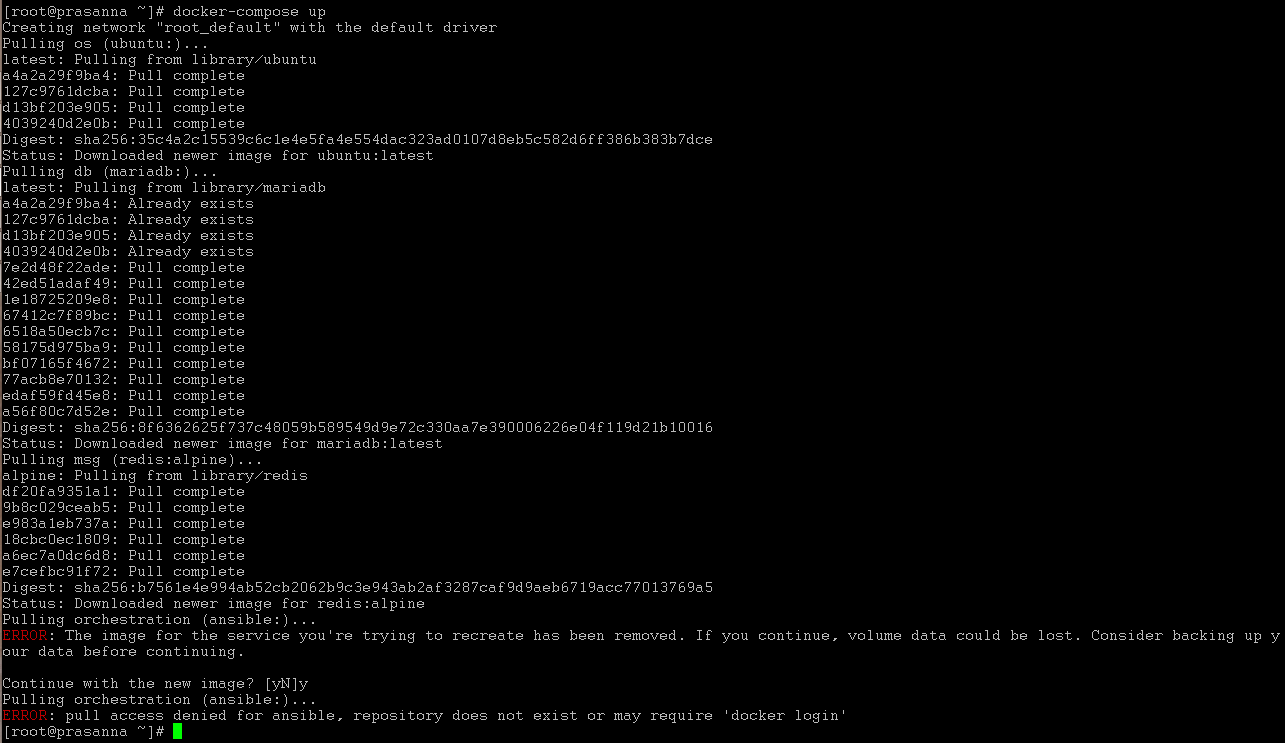
This way different images can be put in docker-compose file and run all the service at a time.



After running docker-compose up



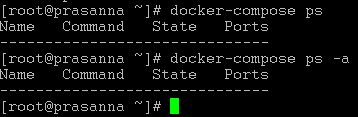
Output of docker-compose up



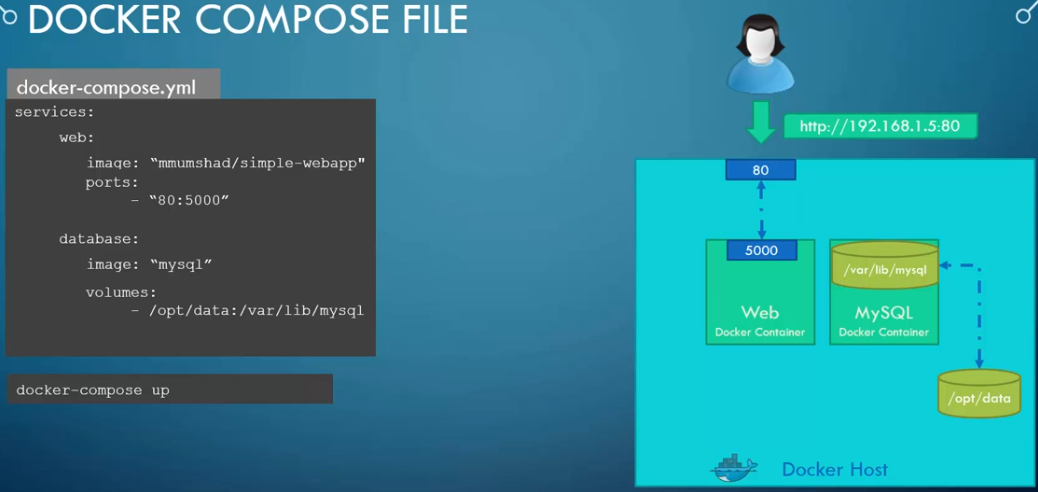
Removing images using docker-compose down



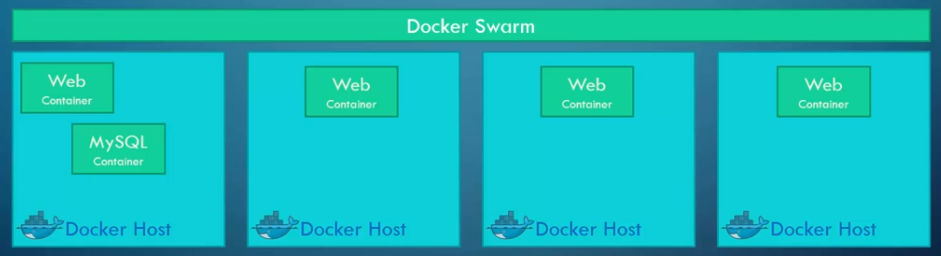
docker-compose ps –a will show the current/exited running process



Ports and volumes can also be mentioned in docker-compose file.



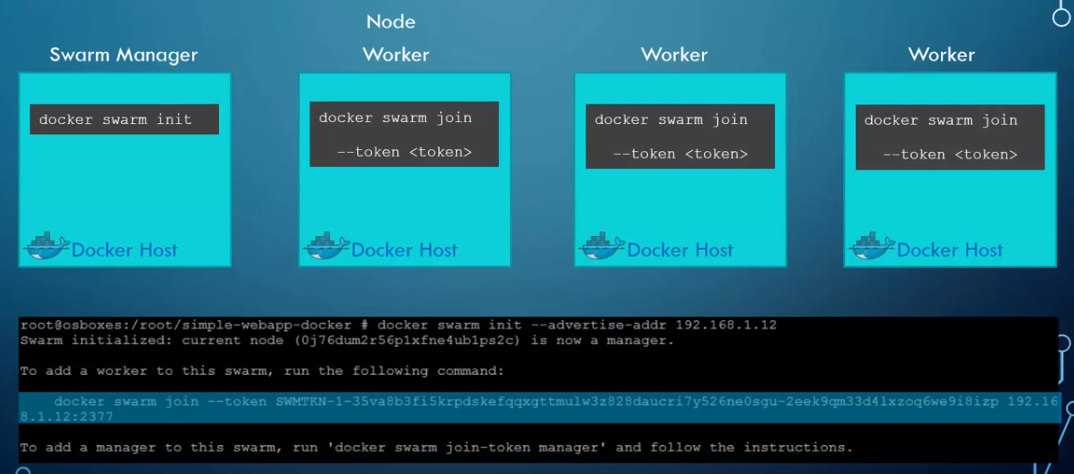
**Docker Swarm:**



One manager and multiple worker/Node.

docker swarm init 🡪 To initiate docker swarm (cluster)

docker swarm join --token <token> 🡪 To join the node to the Swarm Manager



**Docker Networking:**

When docker in installed, it will create three networks - Bridge, none, host.

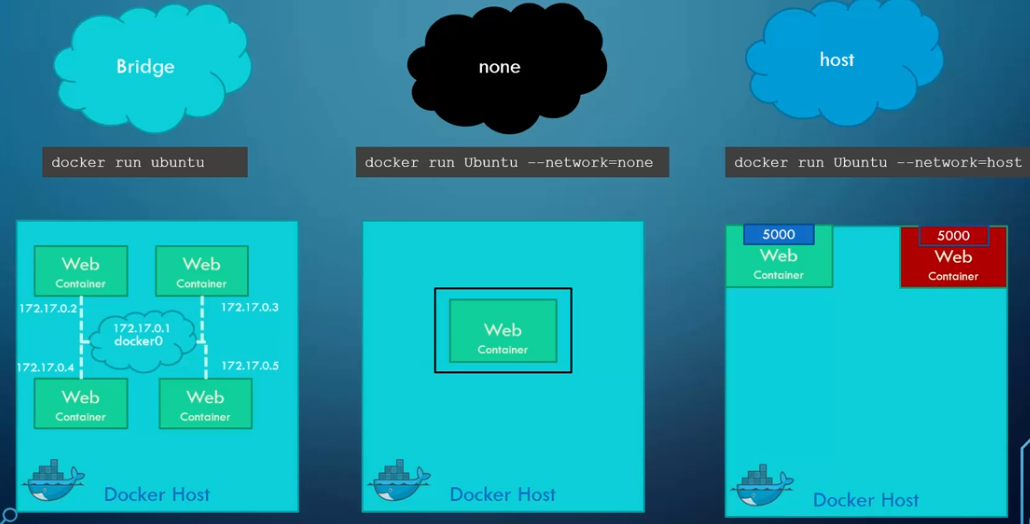
By default a service/image would get network attached to Bridge. If the image you run wants to associate with none or host network, then you have to mention is as “docker run Ubuntu –network=none”



**Bridge** – By default the container would get associated with it and it will have a private IP. The container can access each other using the internal IP. To access the container externally, you need to port map with host port.

**None** – This means the container is not attached with any network and won’t be accessible to external network.

**Host** – If a container is associated with host network, then it would be accessible to external network, no need of port map with host server, because the container is already using host network. If a port (eg: 5000) is allocated to any container, the same port can’t be used to other container.



**User-defined network:**

Since the containers are associated with Bridge network by default, they can access other container using internal private IP. But to isolate the containers, we can create a custom network. So we can assign different networks to containers.

