Trouble Shooting: Push Docker Images to Docker Hub

If you are able to run docker login , but encountered the following **unauthorized: authentication required** error while running docker push

1. [root@terry ~]# docker login --username=1972
2. Password:
3. Login Succeeded
4. [root@terry ~]#
5. [root@terry ~]# docker push asiye\_yigit\_tutorial/debian:1.01
6. The push refers to a repository [docker.io/asiye\_yigit\_tutorial/debian]
7. 29303f03b719: Preparing
8. 77a77cd4826d: Preparing
9. fe4c16cbf7a4: Preparing
10. unauthorized: authentication require

**Solution:**

*Creating the repository on Docker hub before running docker push.*

Take a look at

http://stackoverflow.com/questions/36663742/docker-unauthorized-authentication-required-upon-push-with-successful-login

My Docker account

Docker id : pintu12345

Password : Pintu12345&&&&

docker login -u pintu12345 -p Pintu12345&&&&

Git username: p1i2n3t4u5

Password: Pintu\_12345

C:\Users\nirpanig>docker info

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 0

Server Version: 18.09.2

Storage Driver: overlay2

Backing Filesystem: extfs

Supports d\_type: true

Native Overlay Diff: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

Plugins:

Volume: local

Network: bridge host macvlan null overlay

Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog

Swarm: inactive

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version: 9754871865f7fe2f4e74d43e2fc7ccd237edcbce

runc version: 09c8266bf2fcf9519a651b04ae54c967b9ab86ec

init version: fec3683

Security Options:

seccomp

Profile: default

Kernel Version: 4.9.125-linuxkit

Operating System: Docker for Windows

OSType: linux

Architecture: x86\_64

CPUs: 2

Total Memory: 1.934GiB

Name: linuxkit-00155d380103

ID: KSQO:N5HY:NTTR:RXRK:X6LC:VRRF:U7HM:CKZD:OLOL:RKVO:A3GK:S7YB

Docker Root Dir: /var/lib/docker

Debug Mode (client): false

Debug Mode (server): true

File Descriptors: 22

Goroutines: 47

System Time: 2019-07-05T07:26:16.5253078Z

EventsListeners: 1

Registry: https://index.docker.io/v1/

Labels:

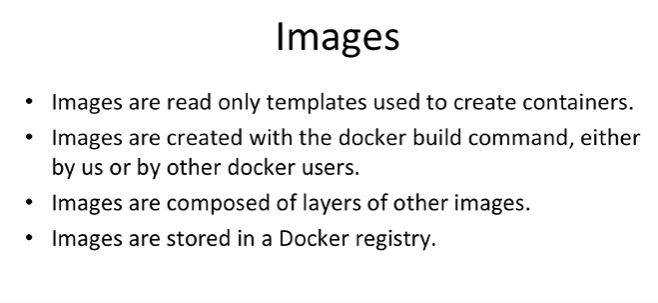
Experimental: false

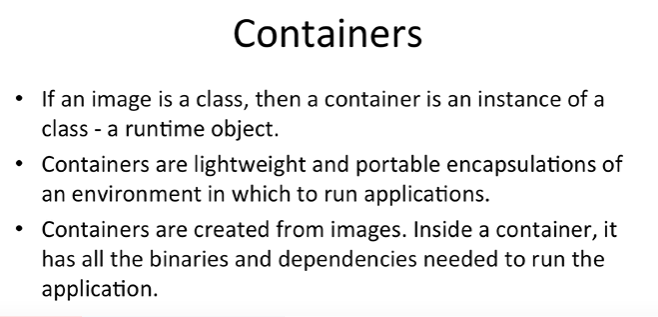
Insecure Registries:

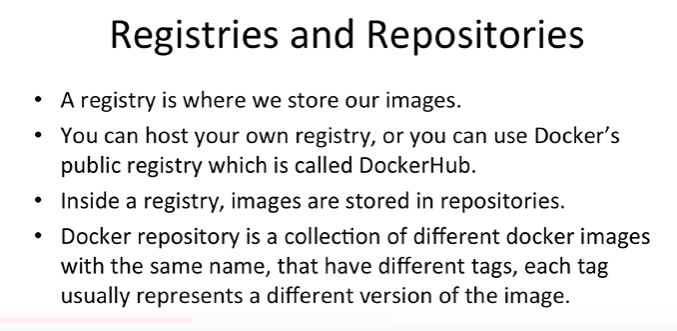
127.0.0.0/8

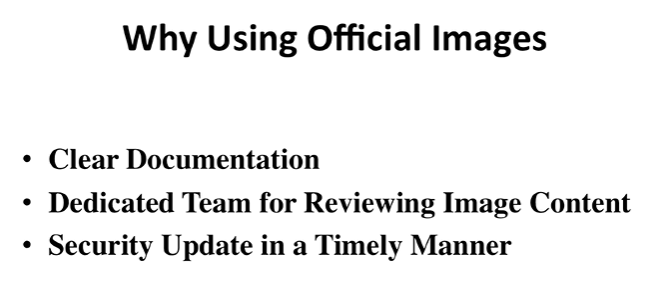
Live Restore Enabled: false

Product License: Community Engine





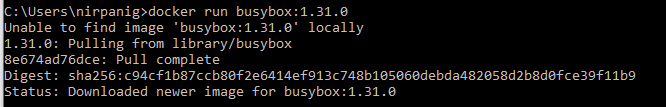




<https://hub.docker.com/>

Docker images command is used to see existing images

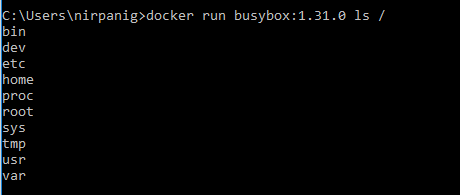


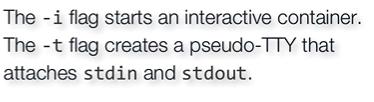


In above command busybox image name 1.31.0 is the tag.

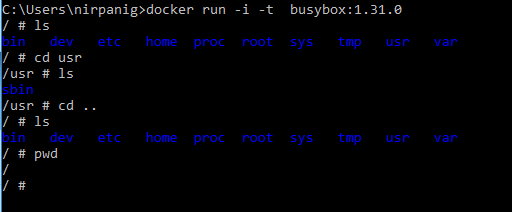
Syntax : docker run imagename:tagname command <argument>

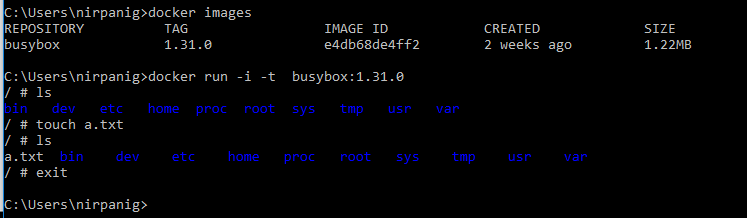






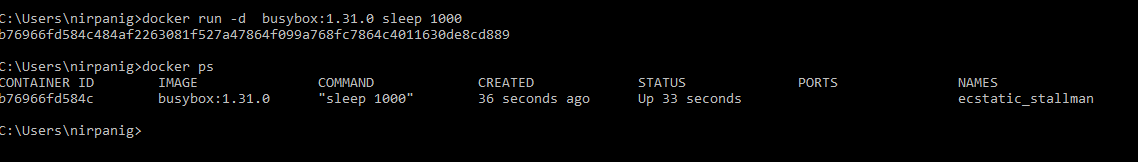
Opens interactive container



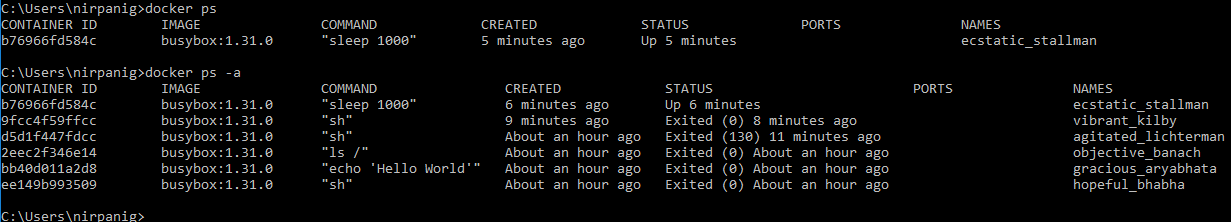


Docker ps command lists all the running docker images

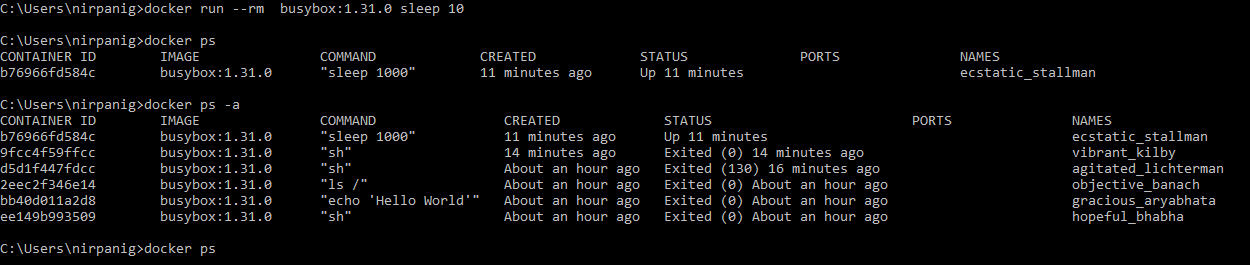
-d option returns long container id



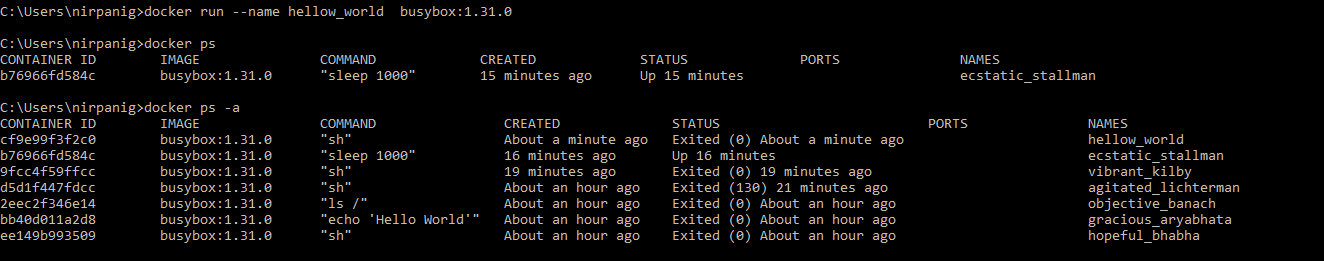
Docker ps –a to list all the previously ran docker images and command

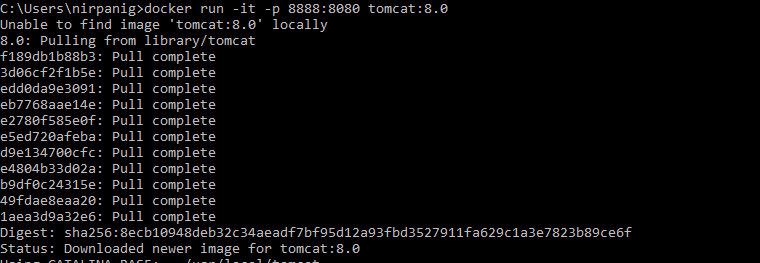


--rm removes the docker container after it exits



--name for changing the container name

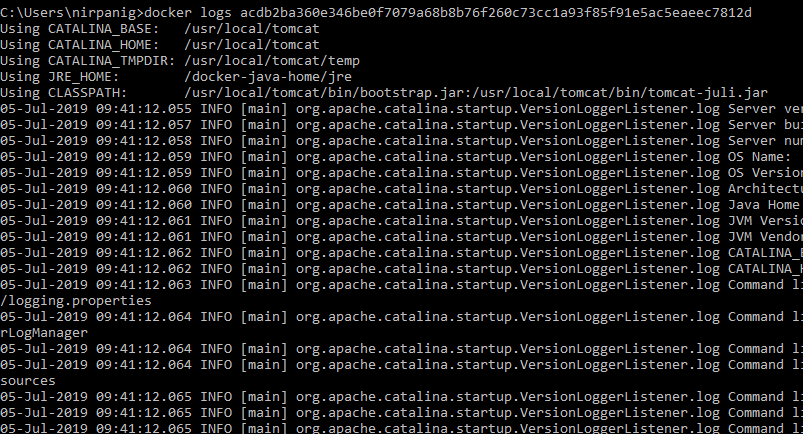


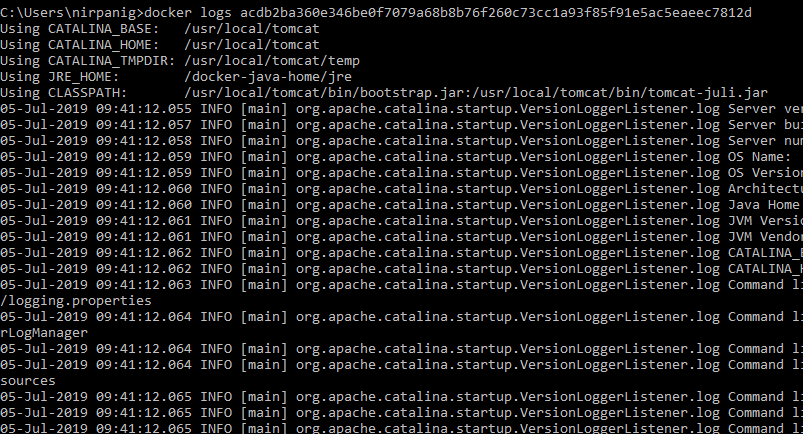




Docker logs container\_id

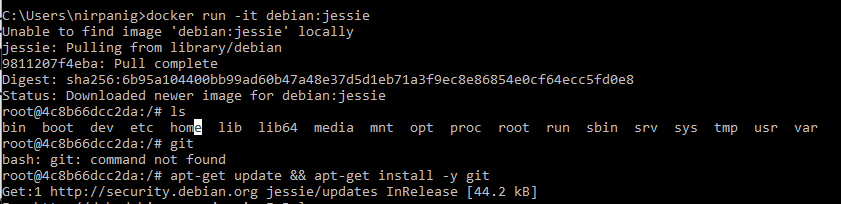






-it is short for --interactive + --tty when you docker run with this command

C:\Users\nirpanig>docker run -it debian:Jessie



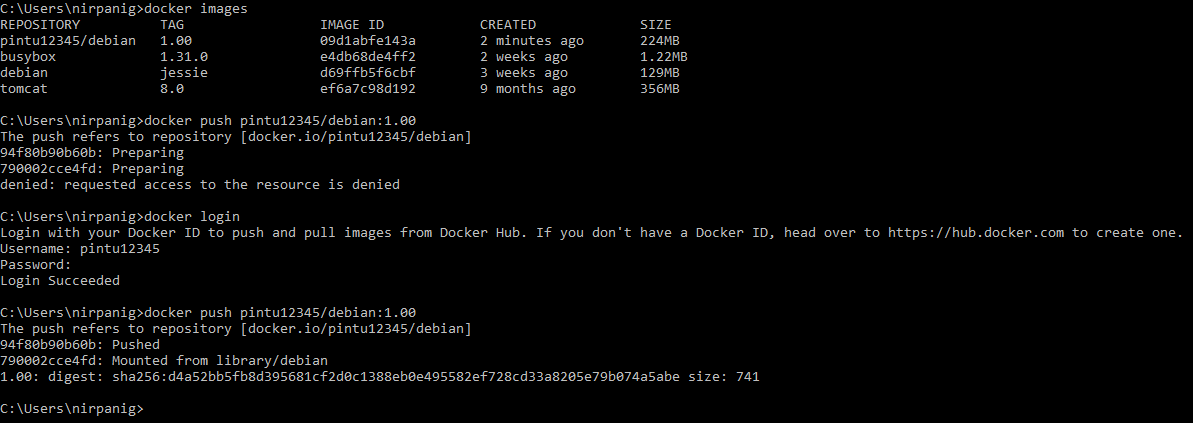
-y option is used to accept automatically prompt for continue

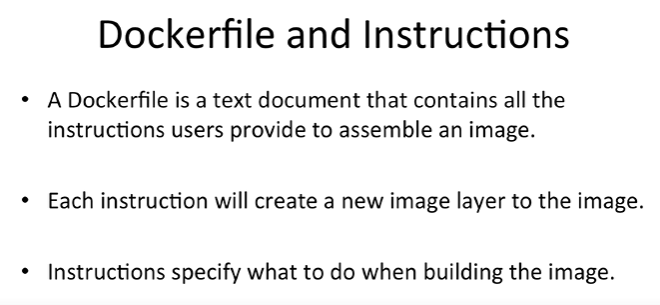
To create an image of our own we have to take an image upon that we have to install other stuff.

We have to create an repository in docker hub to push the new image

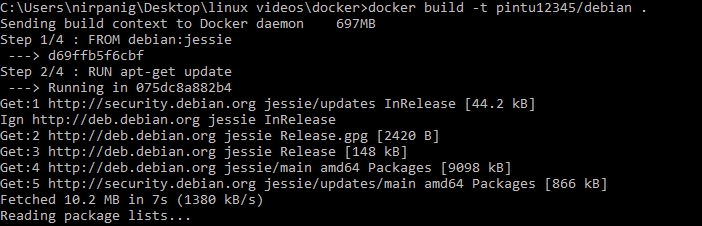


Before pushing the new image to docker hub we mush login from command line

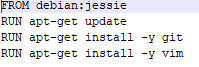




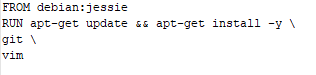
The below command will read the DockerFile from current directory



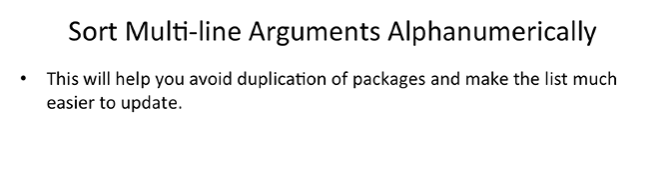
**Multi Line Docker file**

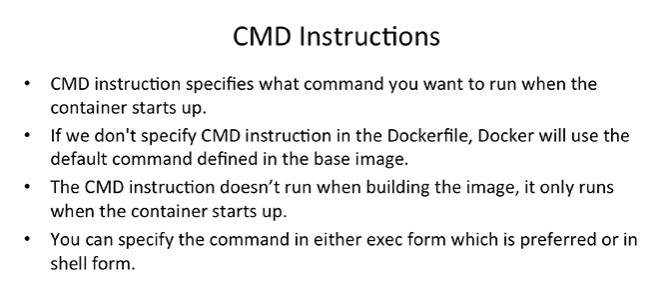


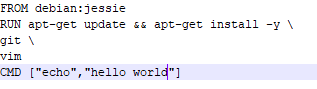
**Single Line Docker file**

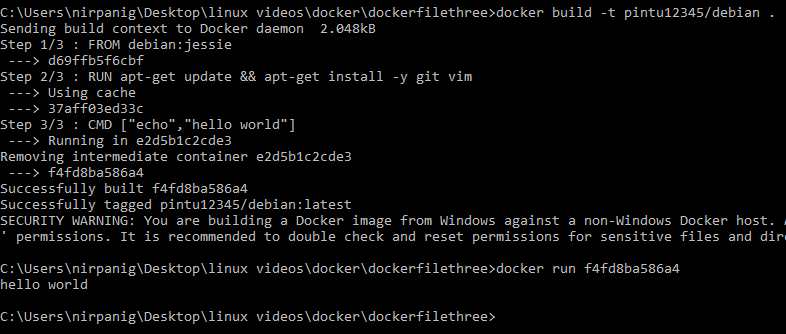


Single line docker file takes lessor number of steps to create an image.

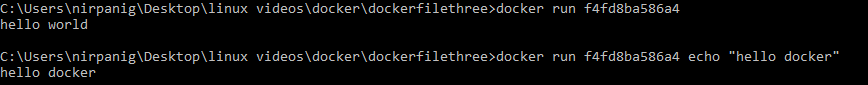


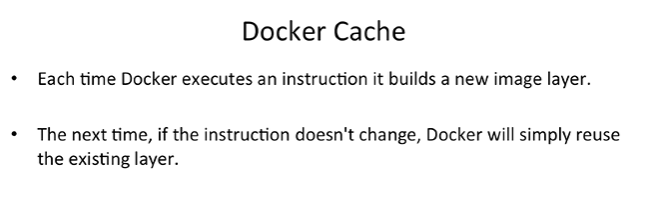


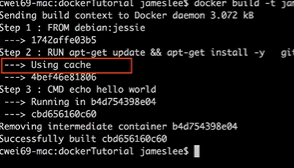


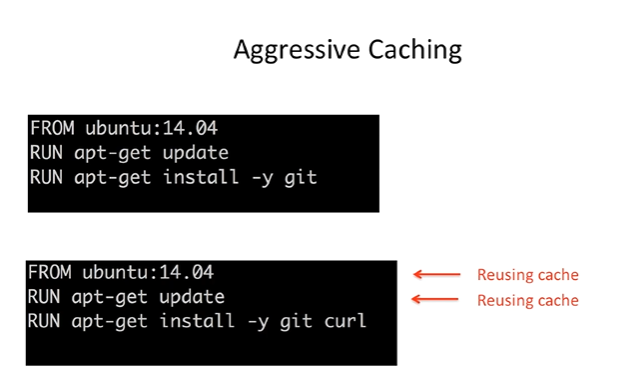


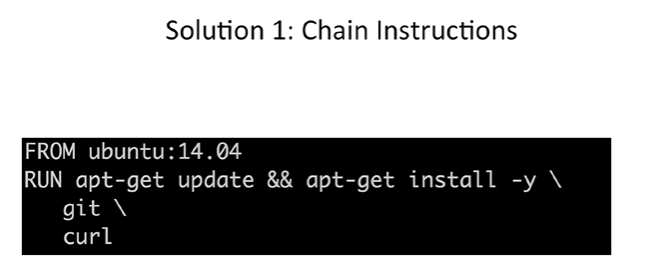
We can override the default command given in DockerFile at run time

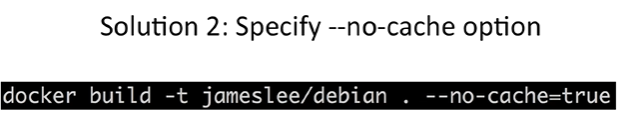


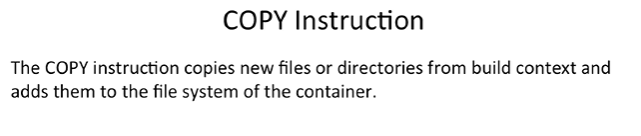


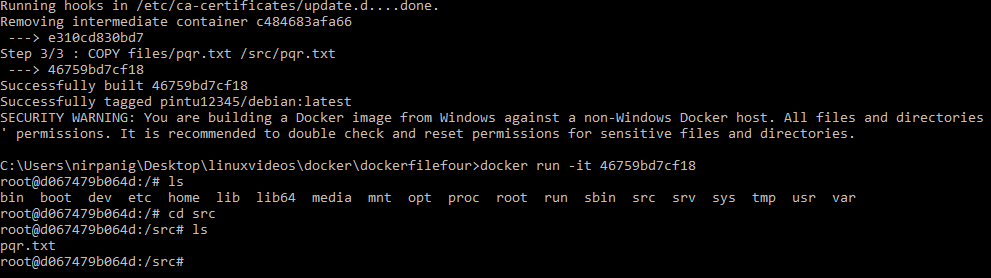


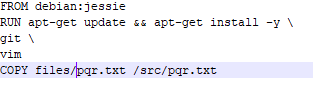




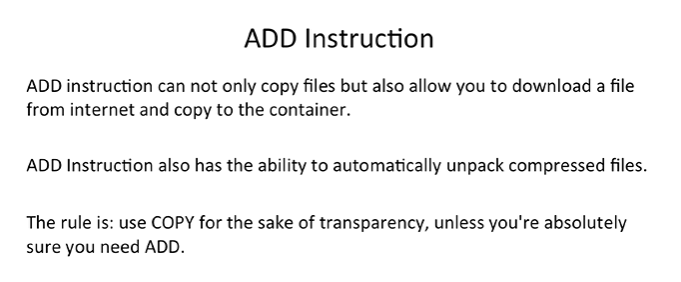


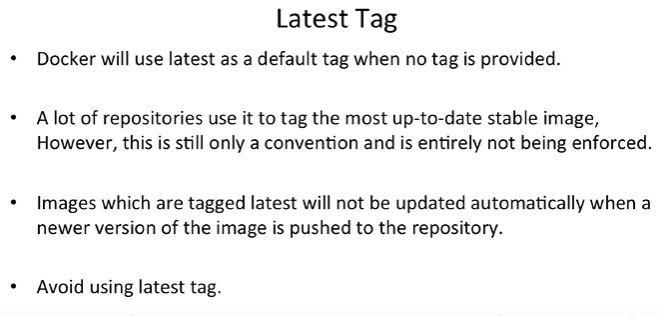




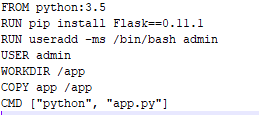


There is some issue with keeping text file directly in the build context.





**First web application**



Line1-> creates image from base image of python

Line2-> installs flask using python pip install

Line3-> creates an user with name admin –ms option creates a home directory for admin user

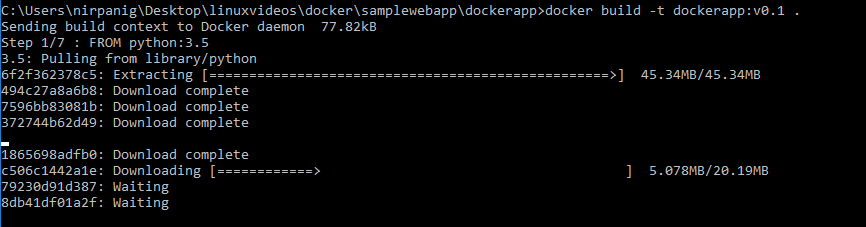
And setting default shell as bash.

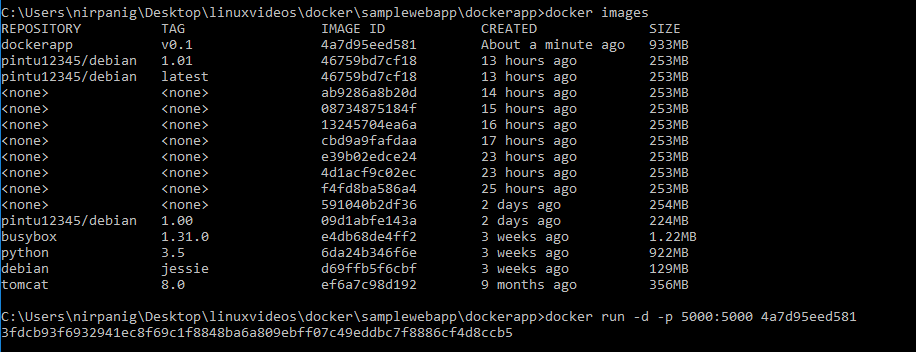
Line4-> this line ensures that the python application and app server will run under admin user.(if not mentioned it will run under root user)

Line5->sets the working directory for RUN, CMD, COPY and entry point application(when we enter the running container it will land in this directory)

Line6-> copy the files

Line7-> run the command as we are already in app folder no need to mention app/app.py

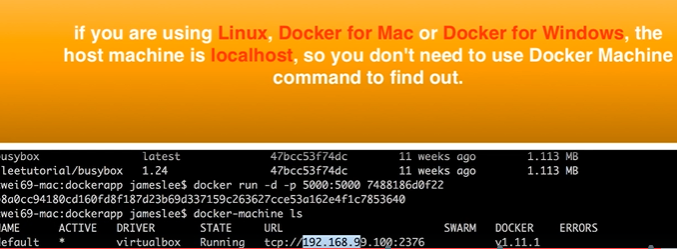




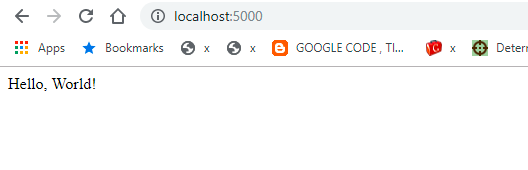
-d option is to run the in the background of container.

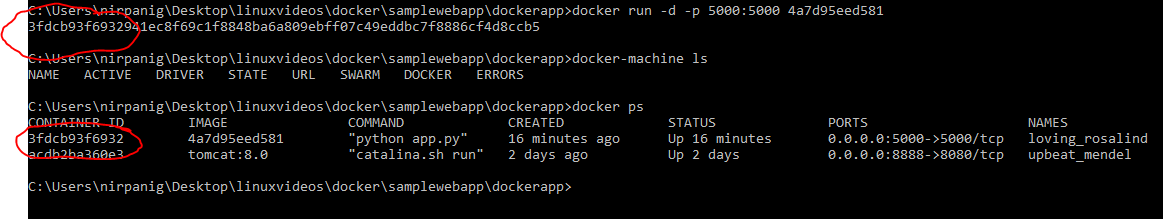
-p option maps container port to host machine port

Smaller alphanumeric value is image id and after starting the image in a container it returns the container id.



Docker-machine ls is used to get the virtual machine ip address where this docker container is running.

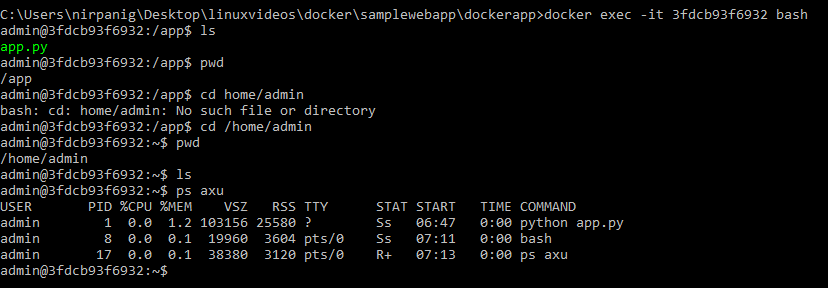




These are long and short container id and short container id.

Using docker exec command we can get in to container running in background and run the bash program

docker exec -it 3fdcb93f6932 bash



Ps axu -> this command will list down all the processes running inside this container.

**Check out source code:**

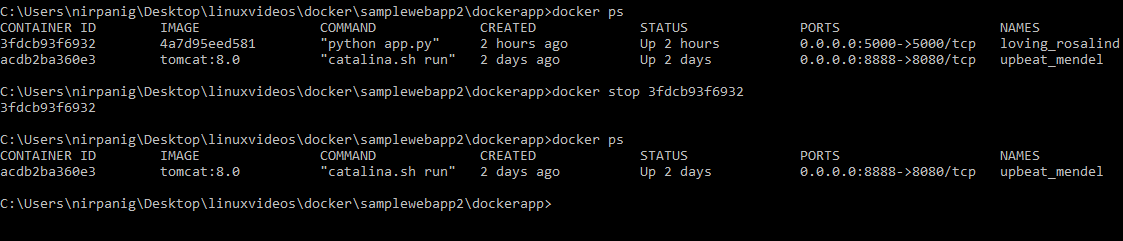
**git clone -b v0.1** <https://github.com/jleetutorial/dockerapp.git>

**Second web application**

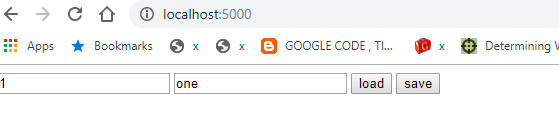
**git clone -b v0.2 https://github.com/jleetutorial/dockerapp.git**



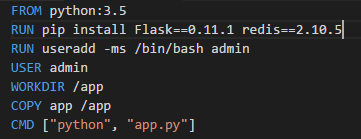
**Stop running container**

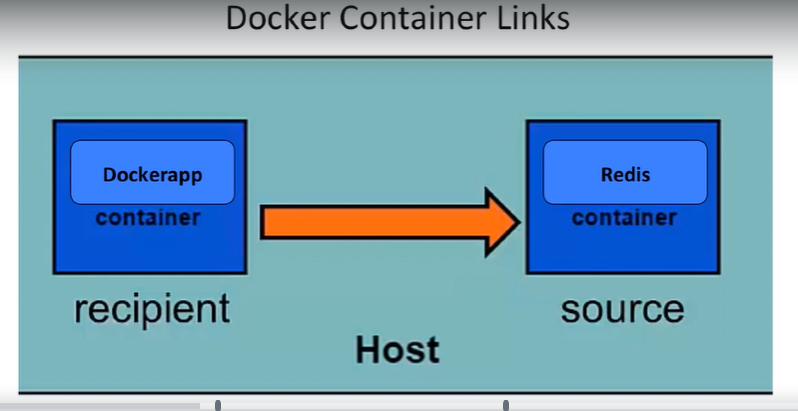






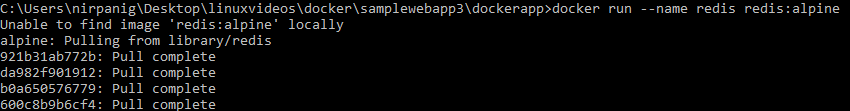






Step 1:

Start the redis container



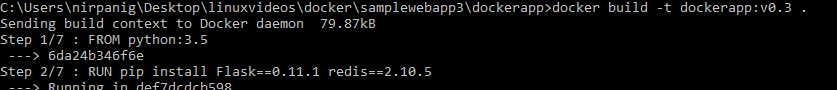
Step 2:

Check whether redis container is running or not



Step 3:

Build the dockerapp



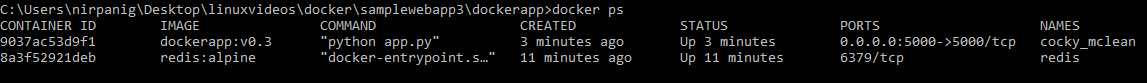
Step 4:

Step run the dockerapp image



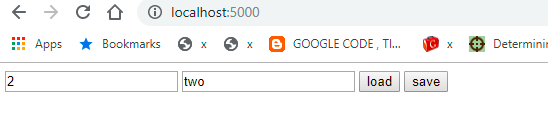
Step 5:

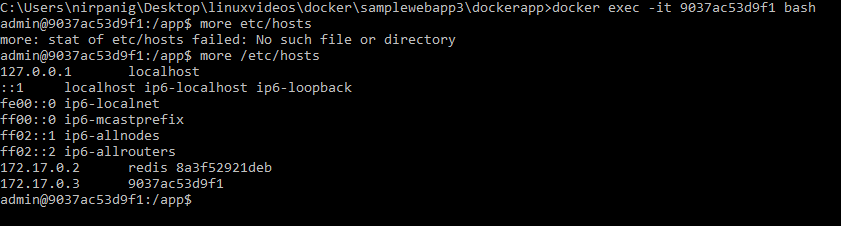
Check both containers are running or not

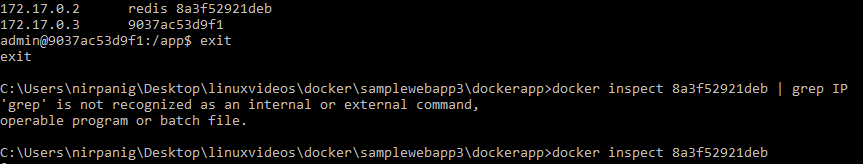


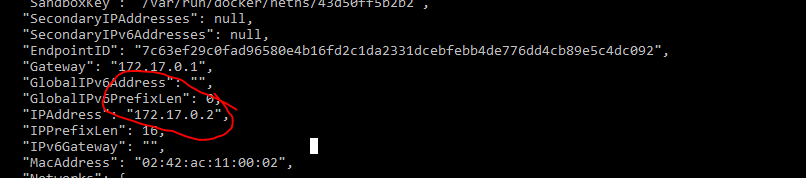
Step 6:

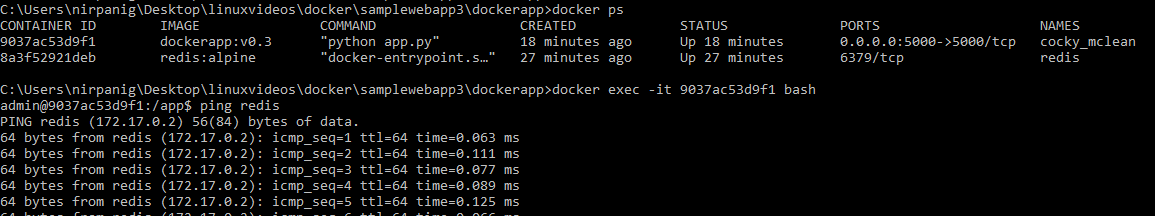
Check app running fine



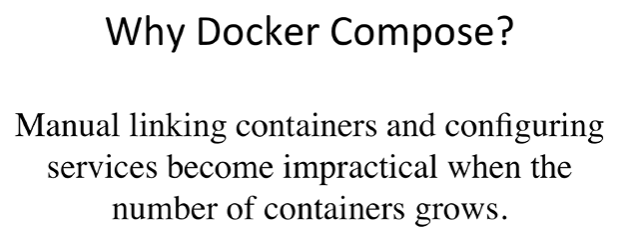




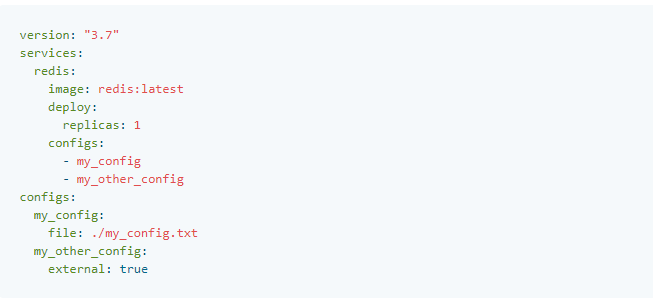


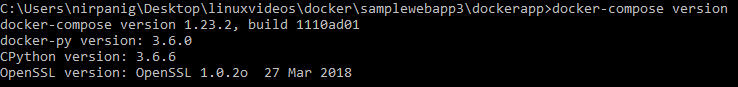


In above example we are connecting from one container to another container with the help container name.



<https://docs.docker.com/compose/compose-file/>

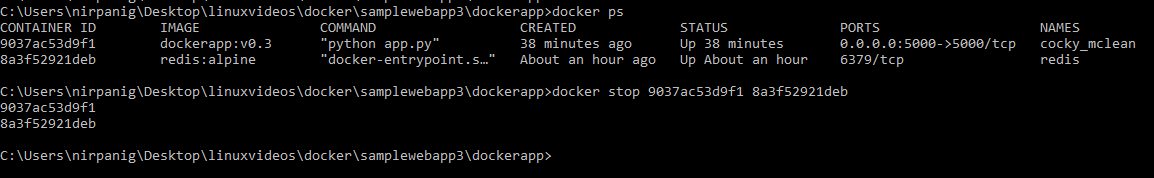




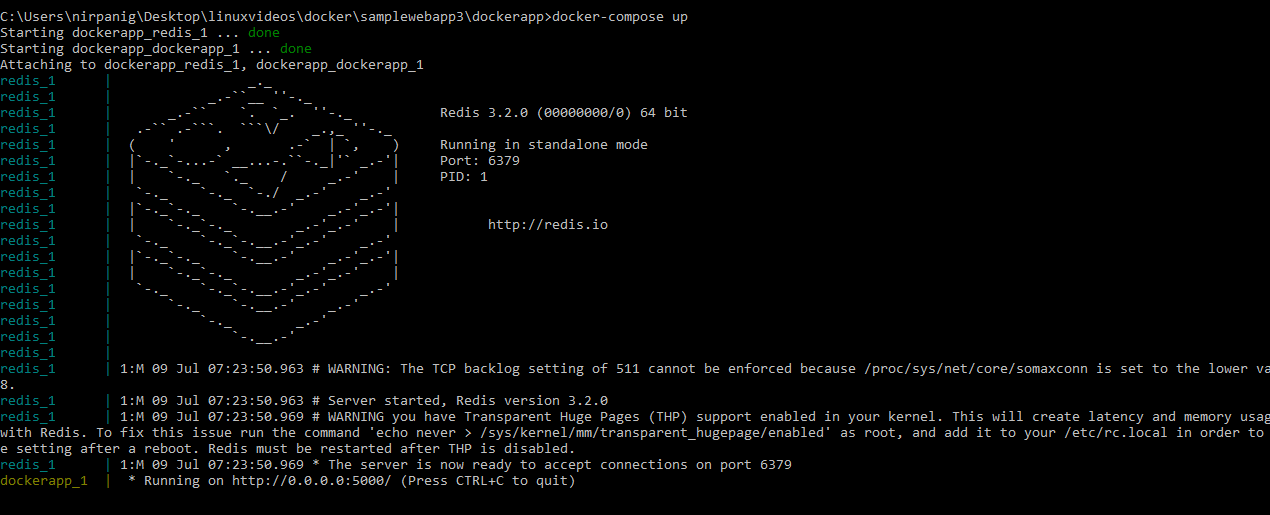


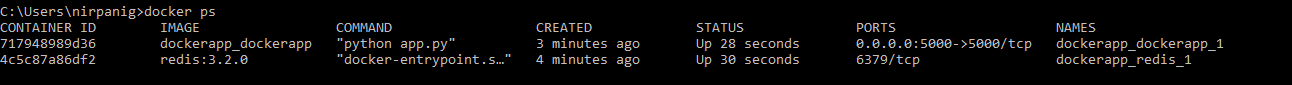
Using docker compose we can create a container from docker file using build step or we can create a container from an image.

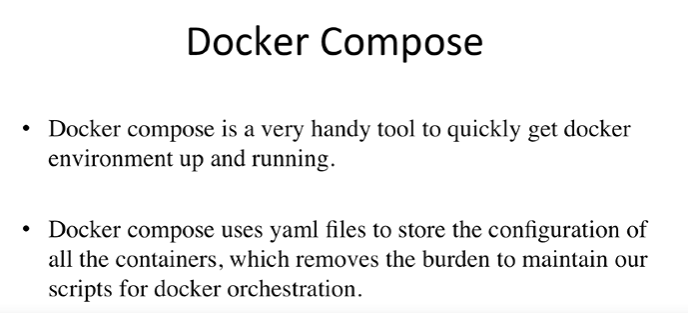
**To stop all running containers**



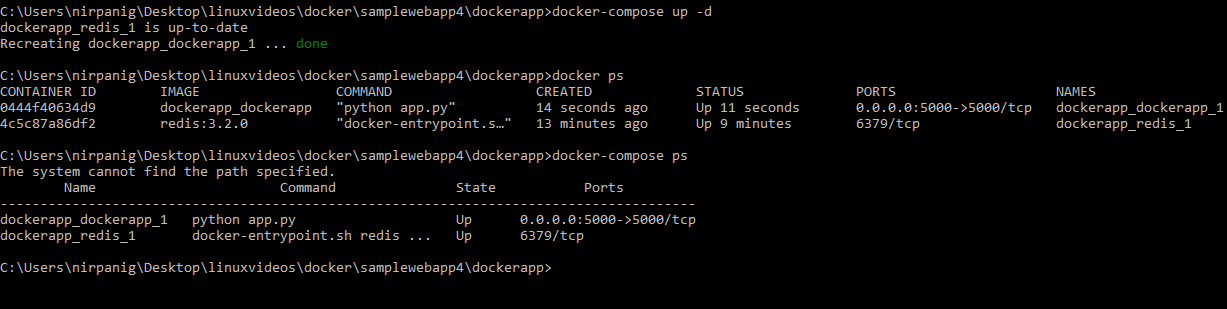
To run docker compose file use command docker-compose up where docker-compose.yml file present





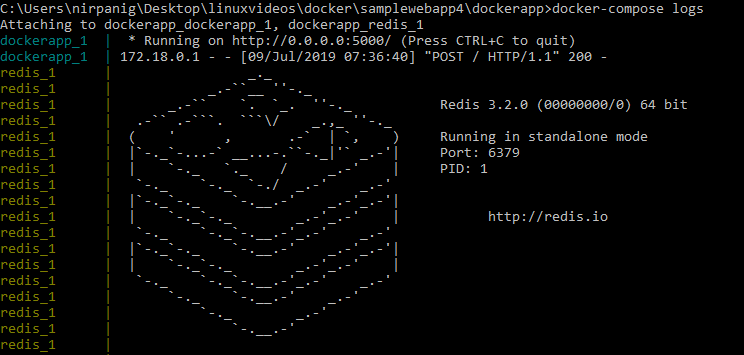


**git clone -b v0.4** <https://github.com/jleetutorial/dockerapp.git>

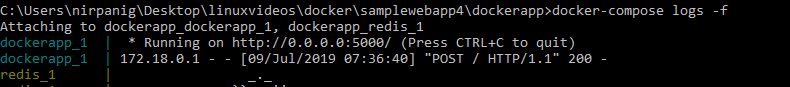


To run the docker compose in background –d option is used

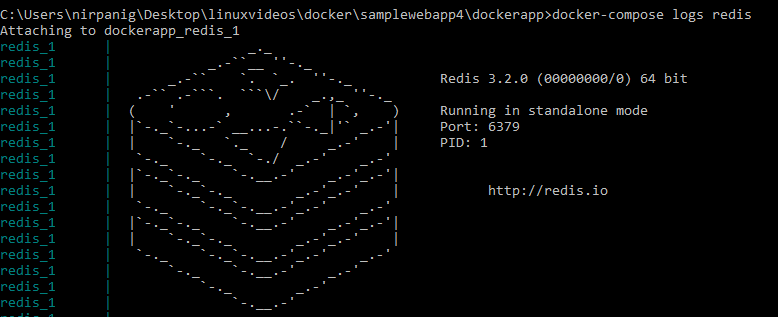
Docker-compose ps -> to see the status of container managed by compose



**docker-compose logs** to see the logs



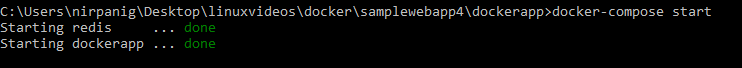
For running logs - f option



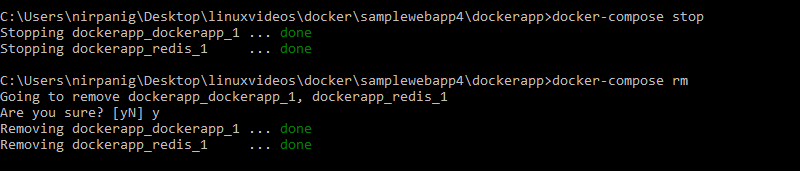
To see the logs of a specific container **docker-compose logs container\_name** is used



**docker-compose stop** to stop all running containers.



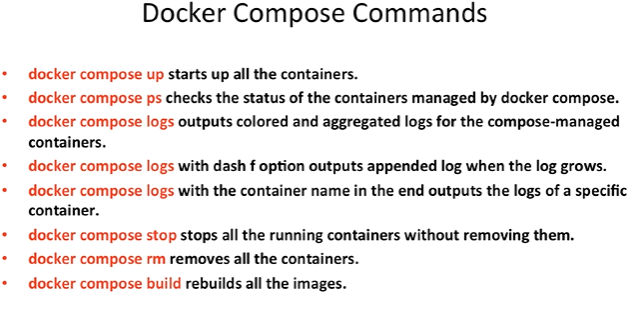
**docker-compose start** to start all stopped containers

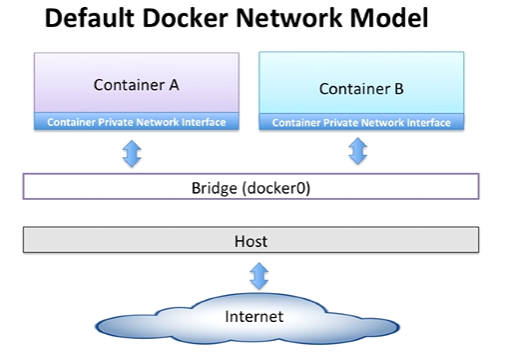


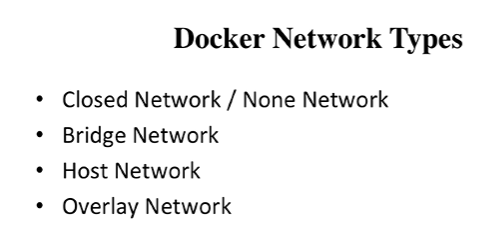
**docker-compose rm** removes stopped containers

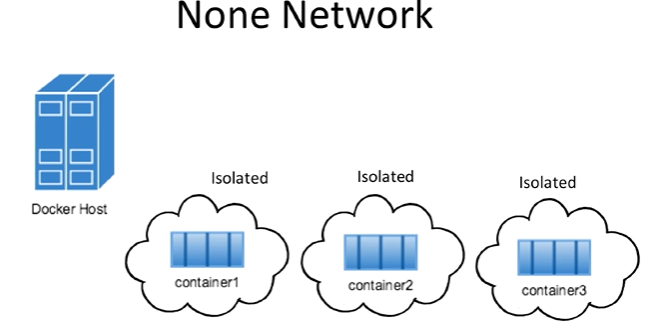
docker-compose up will not rebuild the images

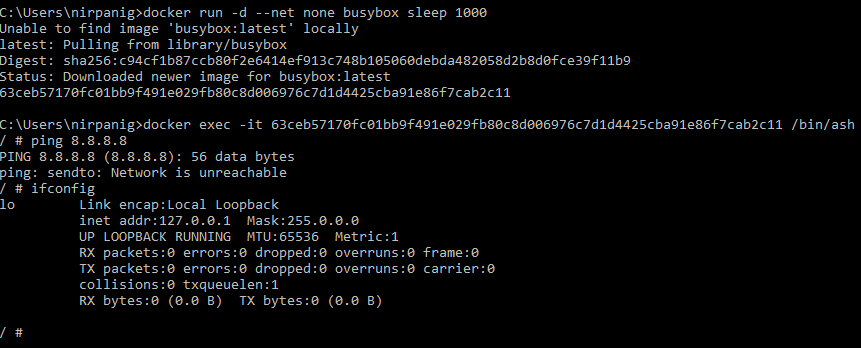
docker-compose build will force for rebuild the images

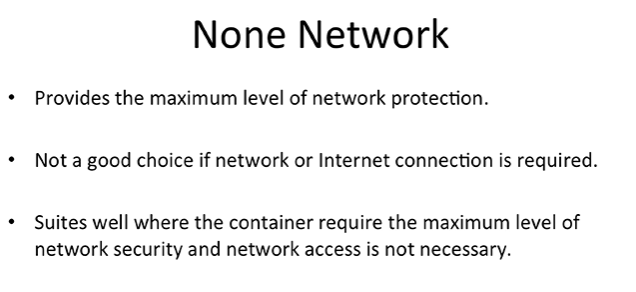


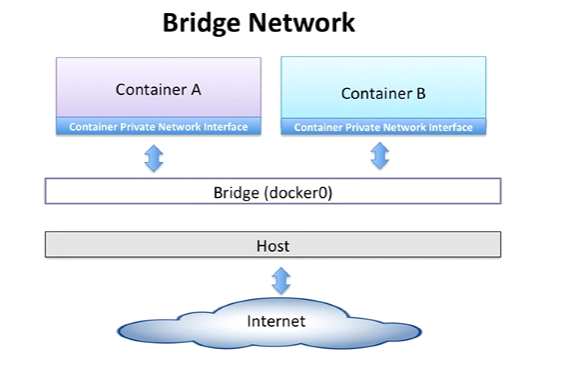


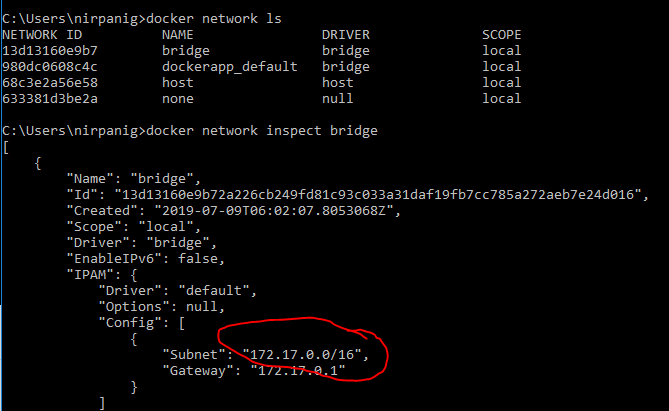




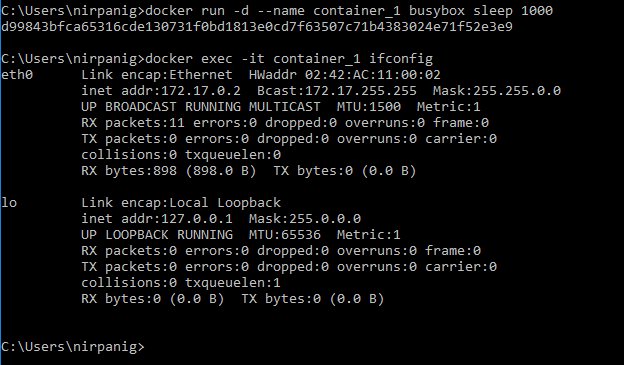




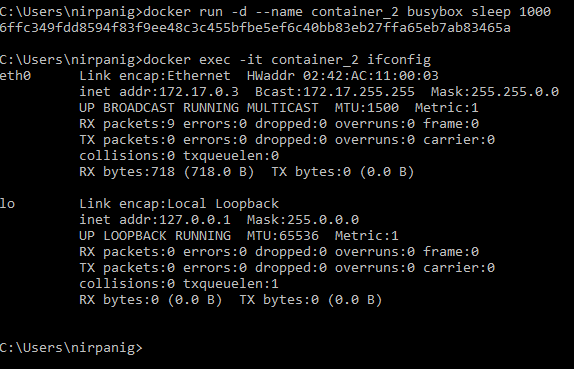




**Container 1**

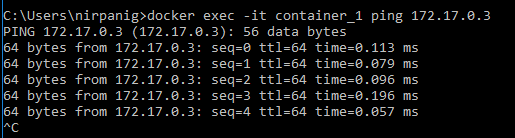


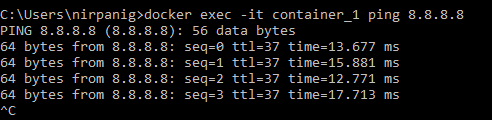
**Container 2**



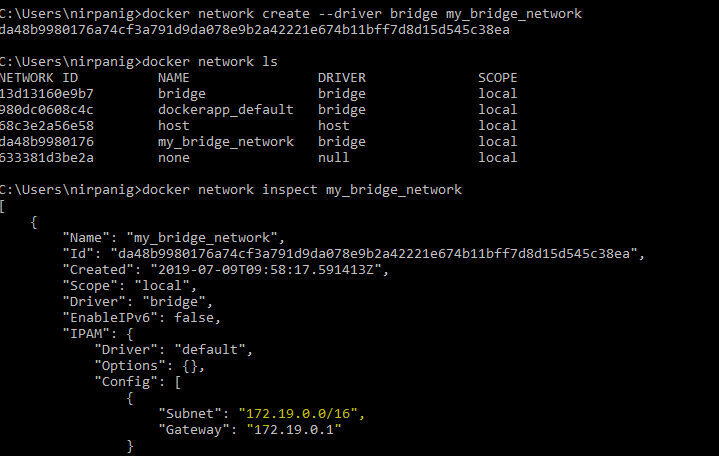
Pinging container\_2 from container\_1 using private network interfaces.

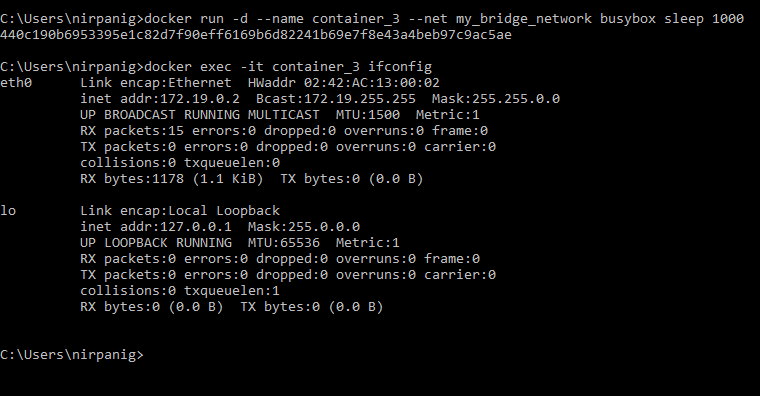
Containers can connect each other using private network interface. They can connect to external world using bridge network interface. 8.8.8.8 is Google’s public IP.



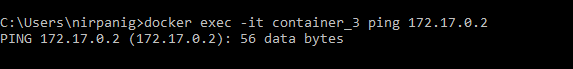


By default different bridge networks are isolated from each other .

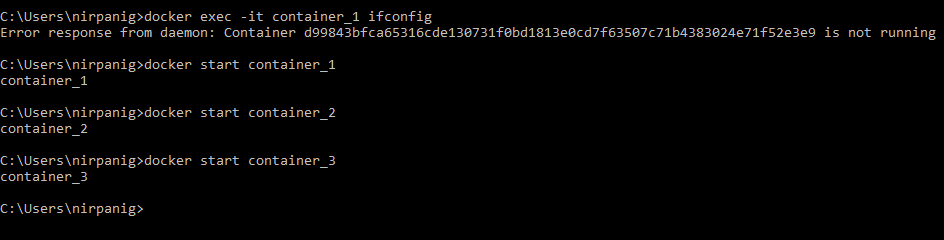




Trying to connect container\_1 from container\_3. it will connect because both are two different bridge network.

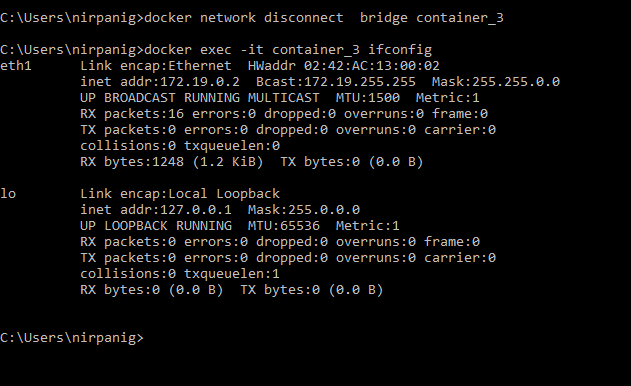


Running again the exited containers

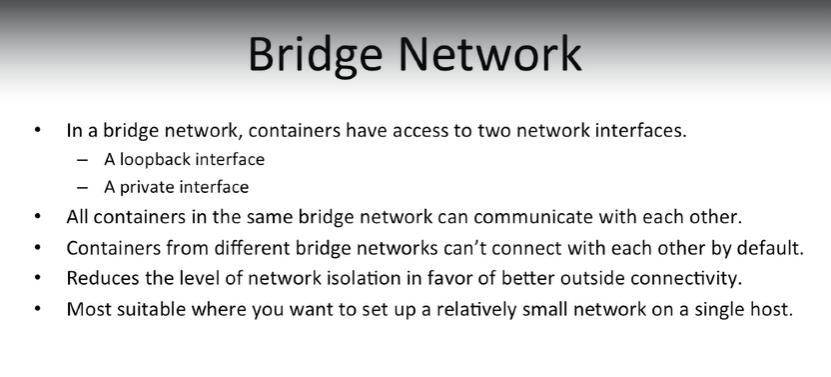


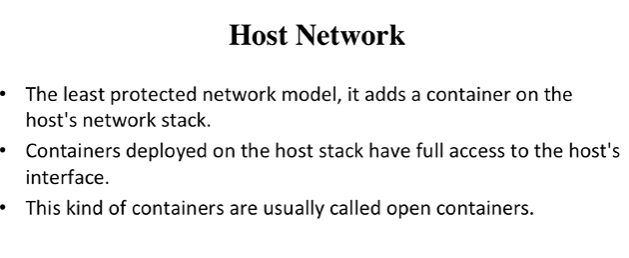


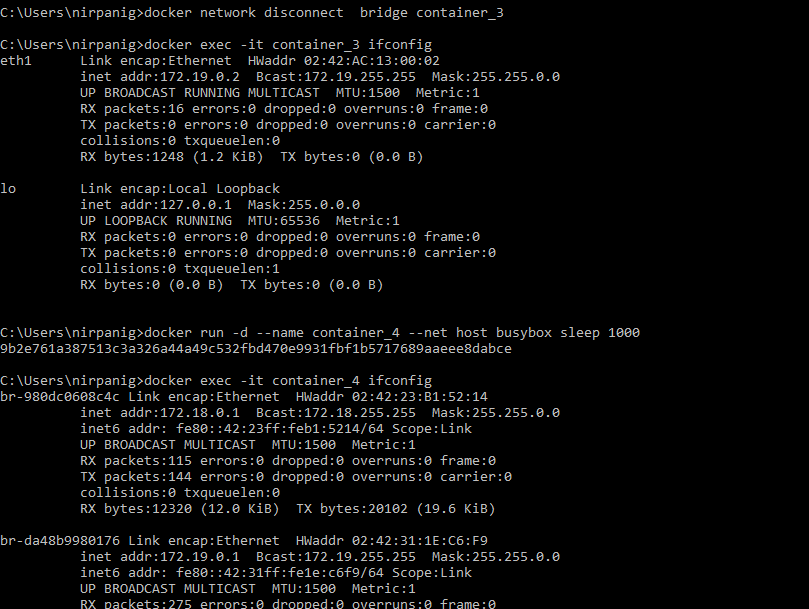
Connecting the default **bridge** network interface to container\_3 ‘s network interface.



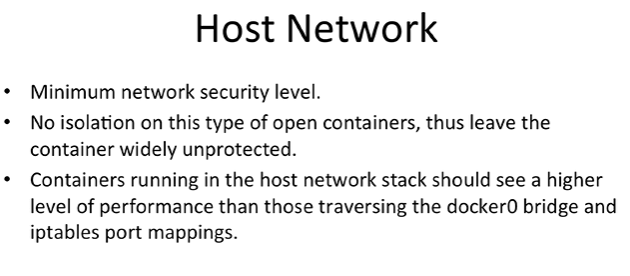
Here we are disconnecting bridge network interface from container\_3 ‘s network interface.

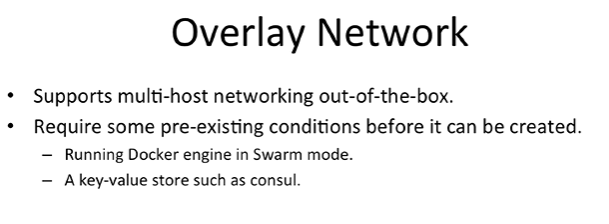


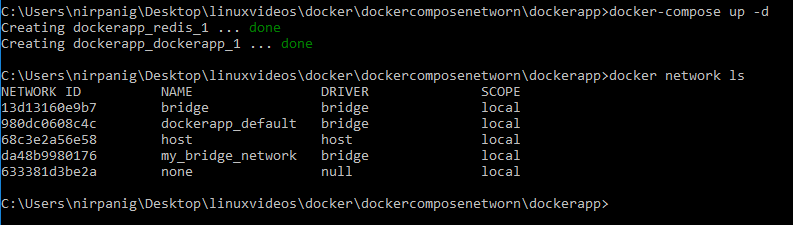




All the available network interface in the host machine are available to the container in this network model.



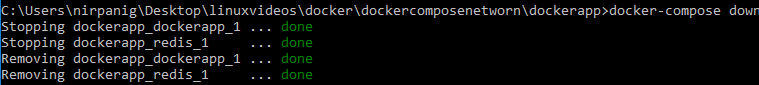




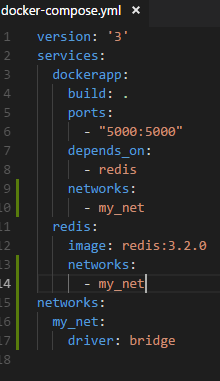
By default docker compose uses default driver to create a bridge network among all the containers.

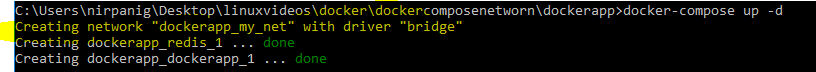
Here dockerapp\_default is the name of the bridge network where dockerapp is the name of the folder where the yml file is present.

Docker-compose down is used to stop the containers and remove all network interface.



Below is the changes to **docker-compose.yml** file to assign user defined network to the containers





**Text Lecture: Overlay Network**

<https://docs.docker.com/engine/userguide/networking/overlay-standalone-swarm/#create-a-swarm-cluster>

Text Direction: Deploy Docker Application to the Cloud with Docker Machine

**Docker Machine Create command**

docker-machine create --driver digitalocean --digitalocean-access-token <xxxxx> docker-app-machine

Extra learning Material: Dockers Monitoring Tools

We didn't cover much about Docker monitoring strategy in this course. However, monitoring tools are handy option for a smooth functioning of your ops. If properly configured, logs and visual data can make a huge difference during troubleshooting.  
We have written a post to summarize some of the popular Docker monitoring tools that you can choose for your environment depending upon your needs.

[Check this up if you are interested.](https://www.level-up.one/dockers-monitoring-tools/)

<https://www.level-up.one/dockers-monitoring-tools/>

**Spotify docker-maven-plugin can’t connect to localhost:2375**

Follow following steps for window 10  
Step 1: Right click on "Dcoker Desktop is running icon "  
Step 2: click on Settings  
Step 3: In “General Tab” you must enable checkbox “Expose Demon on tcp://localhost:2375 without TLS”  
Reference screen attach

**For unix**

export DOCKET\_HOST=unix:///private/var/tmp/docker.sock

COPY and ADD are both Dockerfile instructions that serve similar purposes. They let you copy files from a specific location into a Docker image.

COPY takes in a *src* and *destination*. It only lets you copy in a local file or directory from your host (the machine building the Docker image) into the Docker image itself.

ADD lets you do that too, but it also supports 2 other sources. First, you can use a URL instead of a local file / directory. Secondly, you can extract a tar file from the source directly into the destination.

gcloud auth application-default login

gcloud auth configure-docker

gcloud auth application-default print-access-token

{

"auths": {

"asia.gcr.io": {},

"eu.gcr.io": {},

"gcr.io": {},

"https://index.docker.io/v1/": {},

"marketplace.gcr.io": {},

"staging-k8s.gcr.io": {},

"us.gcr.io": {}

},

"HttpHeaders": {

"User-Agent": "Docker-Client/18.09.2 (windows)"

},

"credsStore": "wincred",

"credHelpers": {

"asia.gcr.io": "gcr",

"eu.gcr.io": "gcr",

"gcr.io": "gcr",

"marketplace.gcr.io": "gcr",

"staging-k8s.gcr.io": "gcr",

"us.gcr.io": "gcr"

},

"stackOrchestrator": "swarm"