**BEFORE DOCKER:-**

Before docker when any software is developed then it sent to testing team for testing before deployment of software.

So, when testing team tests the software then it sends feedback to development team that software has some issue & it’s not working properly, issue occurs due to some dependency, but at the same time developed software still works fine in development department.

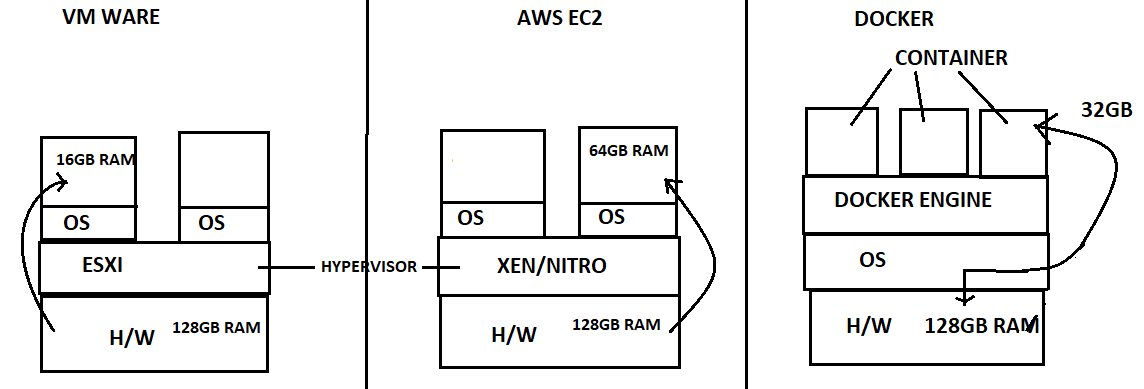
So, this is the conflict occurs in between development & testing team due to some OS or dependencies incompatibility.

Docker is the solution to resolve this type of conflicts.

Docker is an advance version of virtualization because virtualization has some limitations i.e. If I have a machine of 8 core CPU, 16 GB RAM & 1TB HDD & I need VM more than 4 VMs of 4 GB RAM, so, all RAM will be used in 4GB RAM VM. I can’t make 5th VM because I don’t have any RAM left or I can’t use more than 4GB of RAM if it’s required to any application. So, due to this limitation I have stop my work or I’ve to expense more money to buy higher configuration machine.

This problem is solved by docker engine because I don’t need to allocate RAM individually or to each & every container, it uses the required memory from the host(hardware) & release after using. So, we don’t need a very higher configuration or costly machine.

**Differences between VM Ware, AWS Ec2 & Docker**



Docker is a tool which creates a virtual machine but docker is an engine, but it’s very popular as docker engine.

**WHAT IS DOCKER?**

* Docker was first released in march 2013. It’s developed by **SOLOMAN HYKES** & **SEBASTIAN PAUL**.
* Docker is an open source centralized platform design to create, deploy & run applications.
* Docker is a set of platform as a service that uses OS level virtualization whereas VM ware uses hardware level virtualization.
* Docker uses container on the host OS to run application. It allows applications to use the same linux kernel as a system on the host computer rather than creating a whole virtual OS.
* We can install docker on any OS but docker engine runs natively on linux distribution.
* Docker is written in “go” language.
* Docker is a tool that performs OS level virtualization also known as containerization.
* Before docker, many user faces the problem that a particular code running in the developer’s system but not in the user’s system.

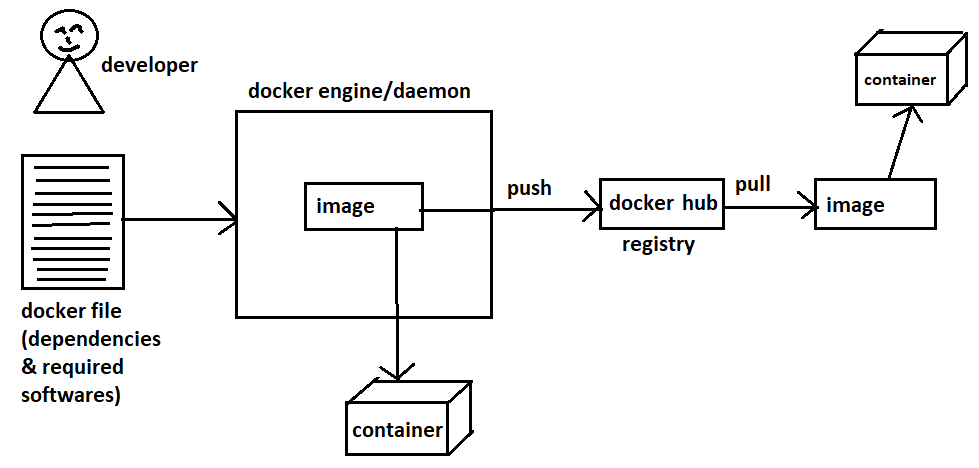
**ADVANTAGES OF DOCKER:**

* No pre-allocation of RAM.
* CI Efficiency-> Docker enables you to build a container image & use that same image across every step of the deployment process.
* Less cost.
* It’s light weight.
* It can run on a physical hardware/virtual hardware or on cloud.
* You can re-use the image.
* It takes very less time to create container.

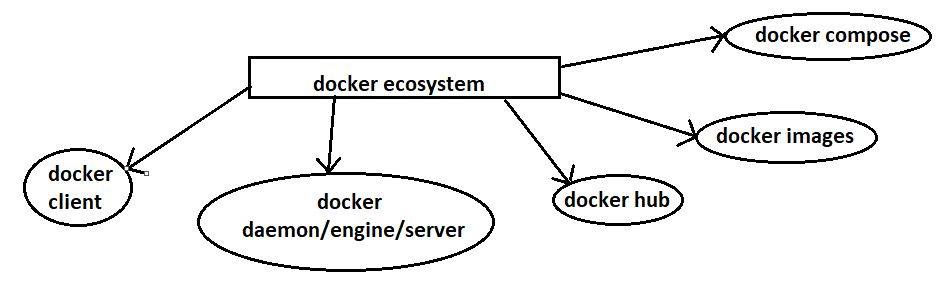
**DISADVANTAGES OF DOCKER:**

* Docker is not a good solution for application that requires rich GUI.
* Difficult to manage large amount of containers.
* Docker doesn’t provide cross-platform compatibility means if an application is designed to run in a docker container on windows, then it can’t run on linux or vice-versa.
* Docker is suitable when the development OS & testing OS are same, if the OS is different, we should use VM.
* No solution for data recovery & backup.

**ARCHITECTURE OF DOCKER:**



**Docker Architecture**



Docker Ecosystem

**DOCKER DAEMON/ENGINE:**

* Docker daemon runs on the host OS.
* It’s responsible for running containers to manage docker service.
* Docker daemon can communicate with other daemons.

**DOCKER CLEINT:**

* Docker user can interact with docker daemon through a client.
* Docker client uses commands & rest API to communicate with the docker daemon.
* When a client runs any server command on the docker client terminal, the client terminal sends these docker commands to the docker daemon.
* It’s possible for docker client to communicate with more than one daemon.

**DOCKER HOST:**

Docker host is used to provide an environment to execute & run application. It contains the docker daemon, images, containers, networks & storages.

**DOCKER HUB/REGISTRY:**

Docker registry manages & stores the docker images.

There are two types of registry in the docker:

1. **Public Registry:** It’s also called as docker hub.
2. **Private Registry:** It’s used to share image within the enterprise.

**DOCKER IMAGES:**

Docker images are the read only binary templates used to create docker containers. OR Single file with dependencies & configuration required to run a program.

WAYS TO CREATE AN IMAGES:

1. Take an image from docker hub.
2. Create an image from docker file.
3. Create an image from existing docker containers.

But in this all the existing files & software will also come in image.

**DOCKER CONTAINER:**

* Container holds the entire packages that is needed to run the application.

OR

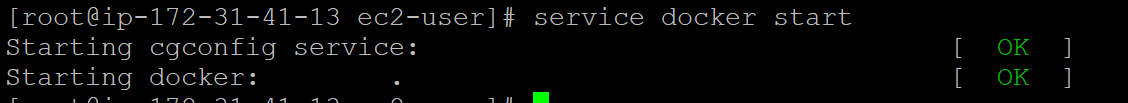
In other words, we can say that the image is a template & the container is a copy of that template

* Container is like a virtual machine.
* Images become container when they run on docker engine.

**BASIC COMMANDS IN DOCKER:**

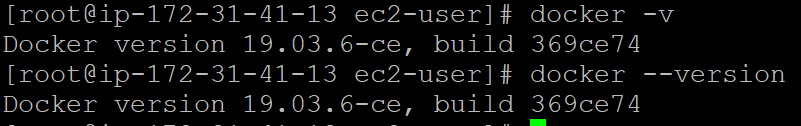
To start docker daemon.

# service docker start



To see docker version

# docker –v / --version



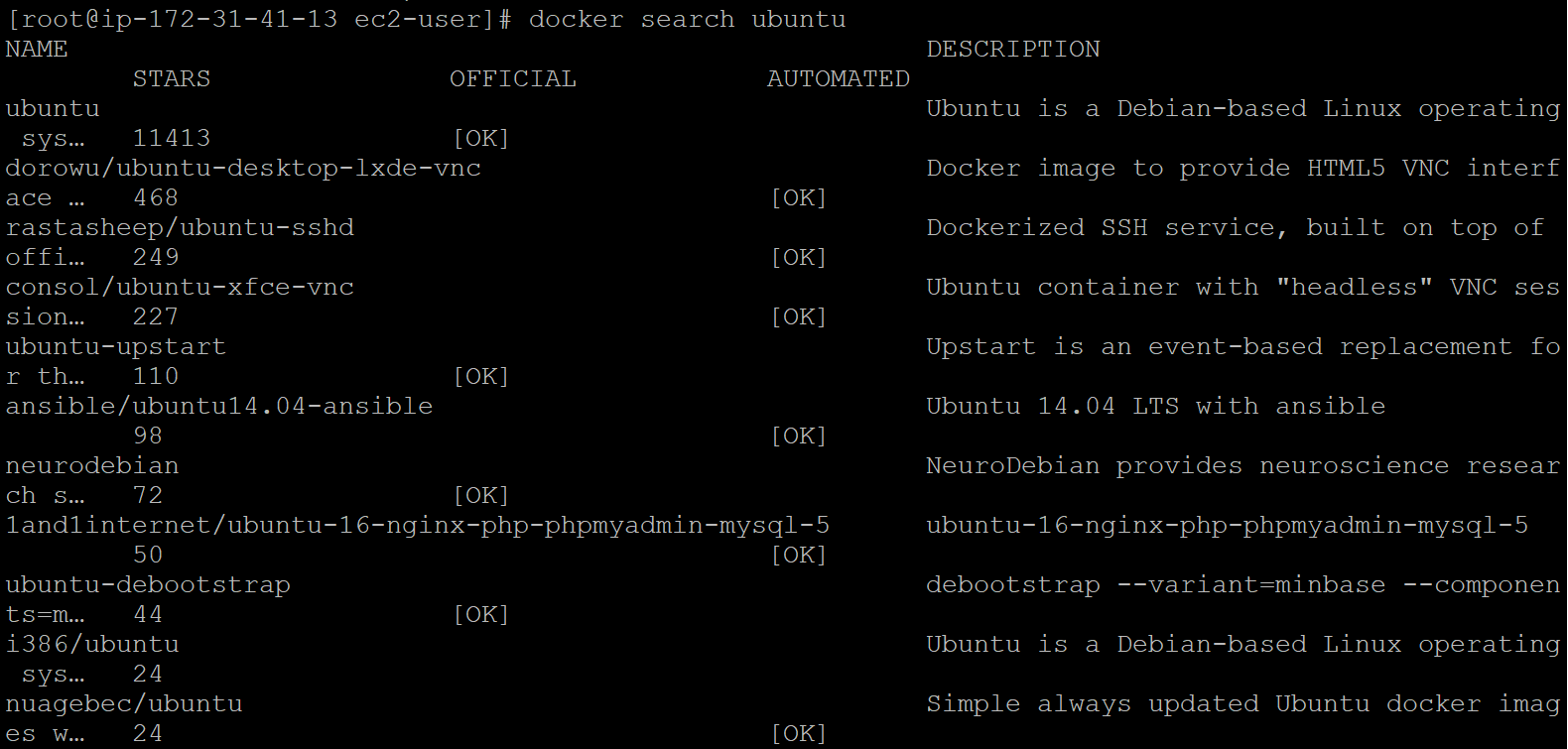
To see all images present in your local machine.

# docker images



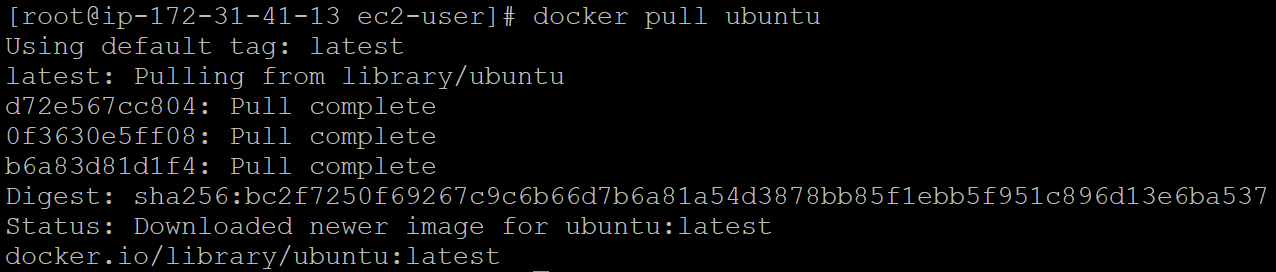
To find out images in docker hub.

# docker search <image-name>



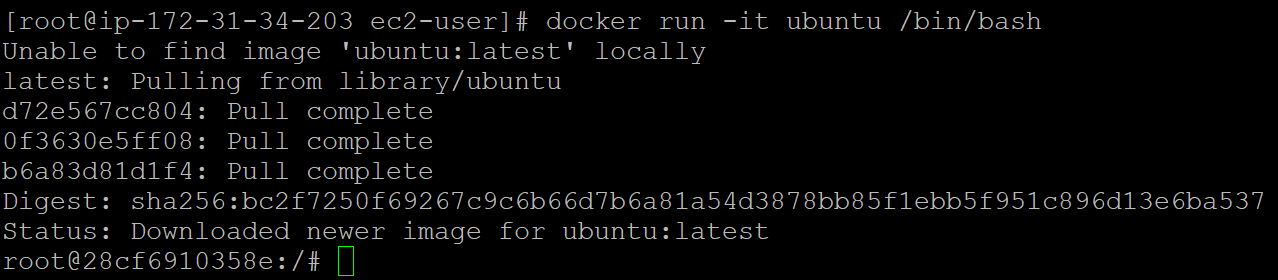
To download image from docker hub to local machine.

# docker pull <image-name>



To download image from docker hub & create container.

# docker run –it ubuntu /bin/bash



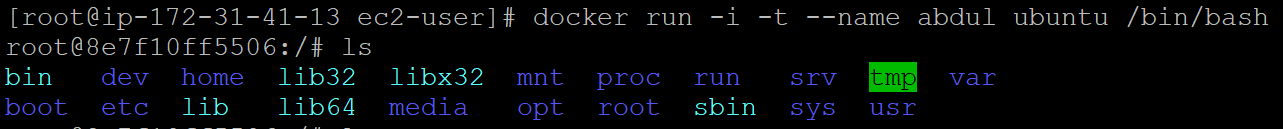
To give name to container.

# docker run –it(-i –t) –name <container-name> <image-name> /bin/bash

“run” is used in above command to create & enter at same time.

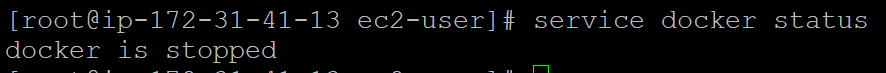
-i is termed as interactive mode.

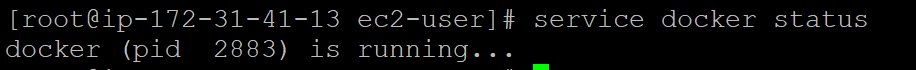
-t is termed as terminal.



To check, service is start or not.

# service docker status





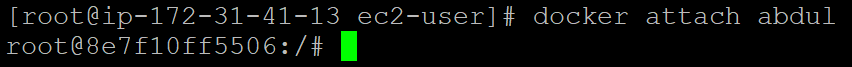
To start container.

# docker start <container-name>



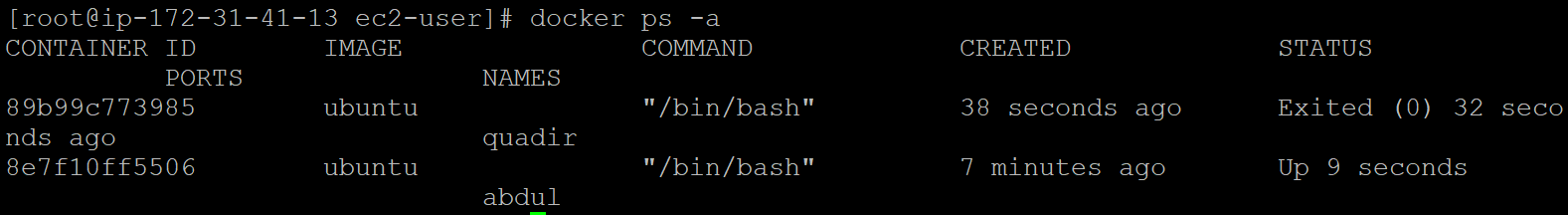
To go inside container.

# docker attach <container-name>



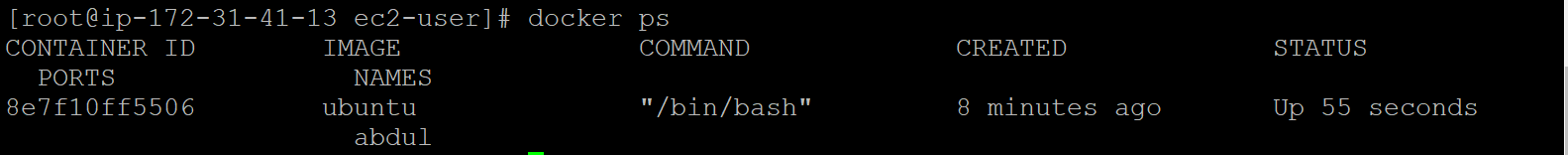
To see all containers

# docker ps –a



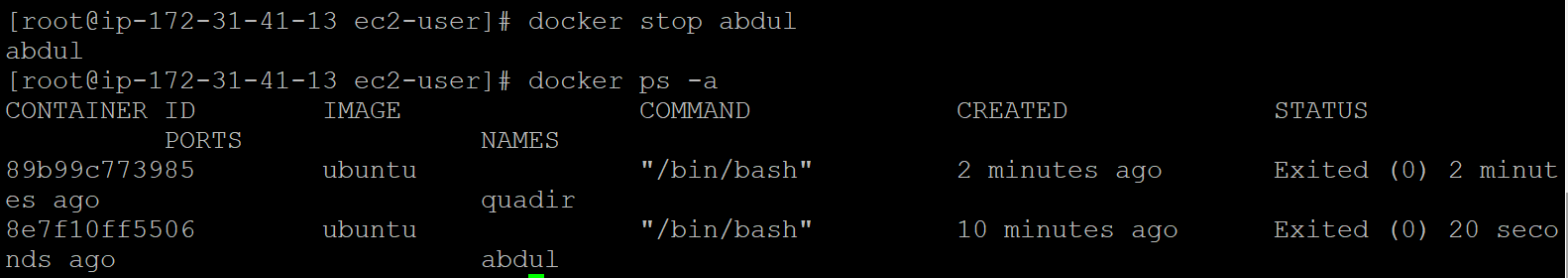
To see only running containers.

# docker ps



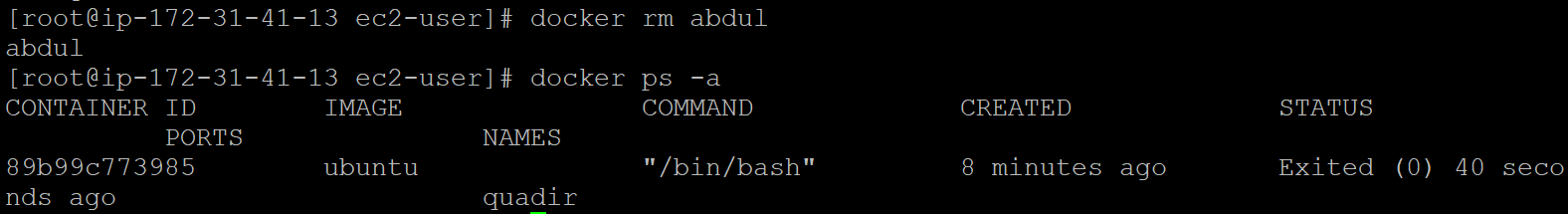
To stop container.

# docker stop <container-name>



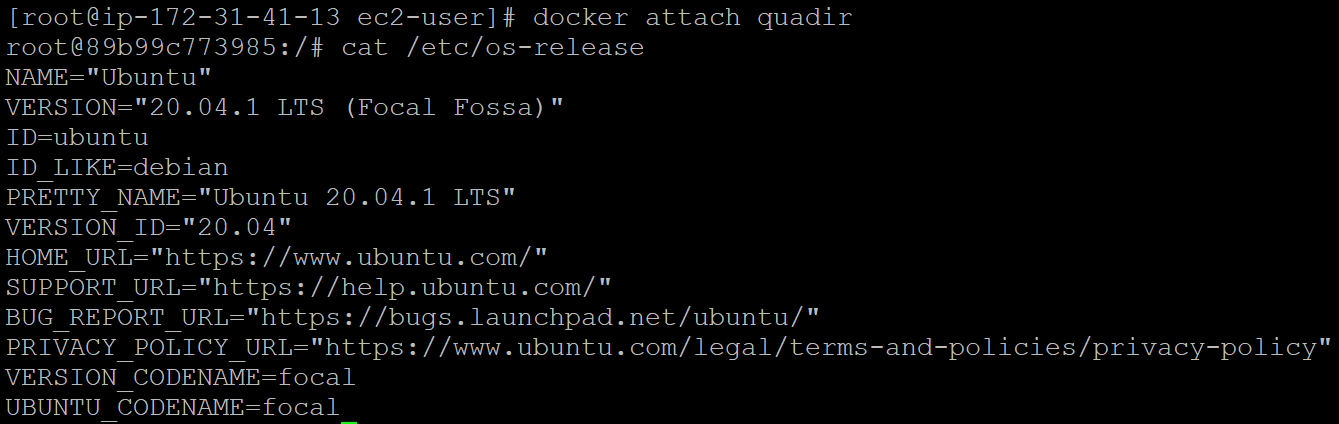
To delete container.

# docker rm <container-name>



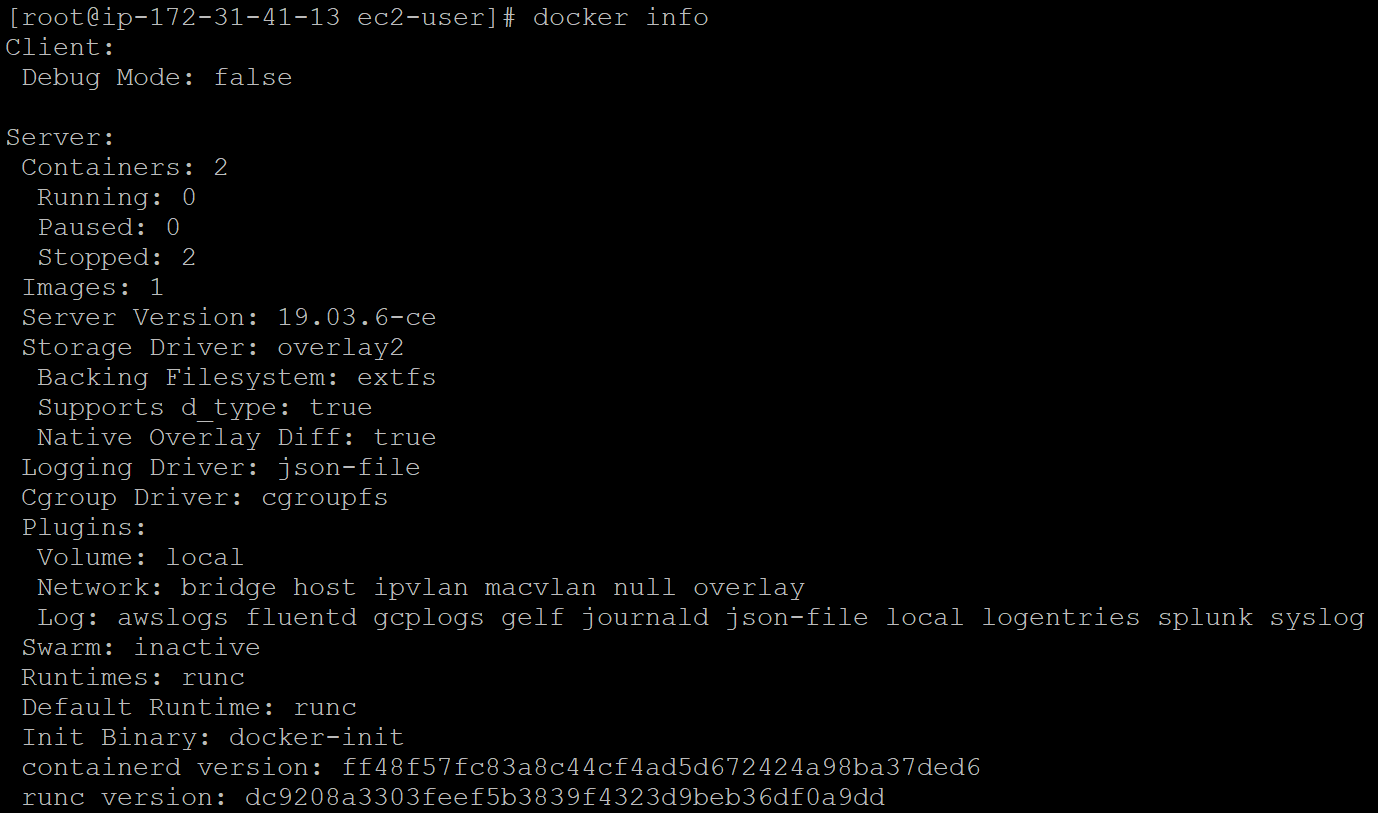
To see container info inside container.

# cat /etc/os-release



To see docker information.

# docker info



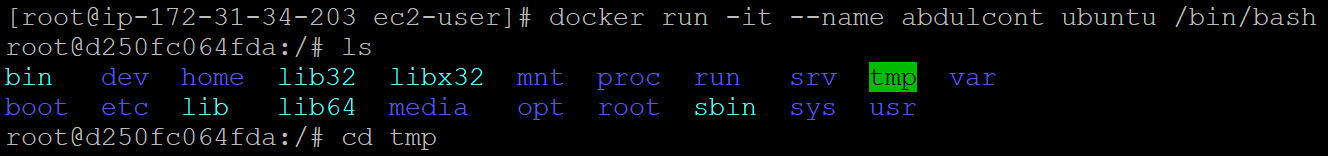
**DOCKER COMPONENTS & ‘diff’ COMMAND:**

Now, we have to create container from our own image.

Therefore, create one container first.

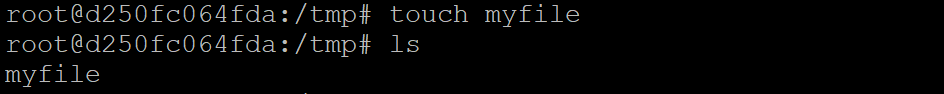
# docker run –it --name abdulcont ubuntu /bin/bash.

# cd tmp/



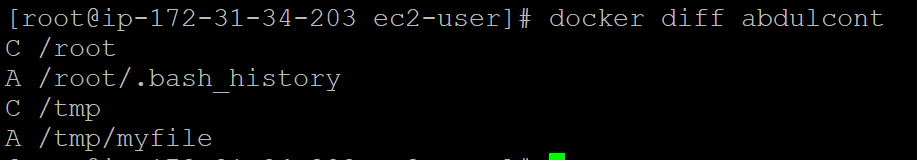
Now, create one file inside this tmp directory.

# touch myfile



Now, if you want to see the difference between the base image & changes on it then

# docker diff abdulcont



C-change

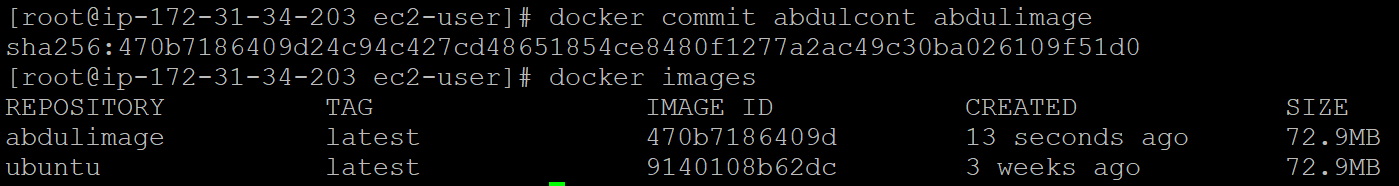
D-deletion

A-add/append

Now, create image of this container.

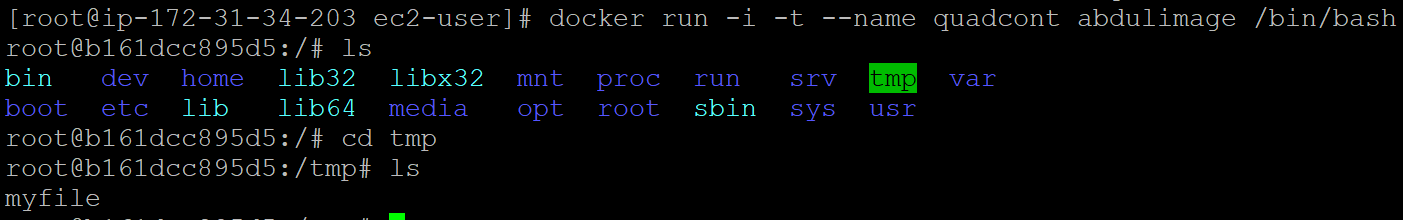
# docker commit abdulcont abdulimage

# docker images



Now, create container from this image.

# docker run –it --name quadcont abdulimage /bin/bash



**CREATING A DOCKER IMAGE FROM FILE:**

* “Dockerfile” is basically a text file. It contains some set of instruction.
* Automation of docker image creation.

**Dockerfile Components:**

FROM: For basic image, this command must be on top of the Dockerfile.

RUN: To execute commands, it’ll create a layer in image.

MAINTAINER: Author/owner/description.

COPY: Copy files from local system(docker vm) we need to provide source, destination(we can’t download file from interner & any remote repo).

ADD: Similar to copy but it provides a feature to download files from internet, also we extract file at docker image side.

EXPOSE: To expose ports, such as port 8080 for tomcat, port 80 for nginx etc.

WORKDIR: To set working directory for a container.

CMD: Execute commands but during container creation.

ENTRYPOINT: Similar to CMD, but has higher priority over CMD, first commands will be executed by ENTRYPOINT only.

ENV: Environment Variables.

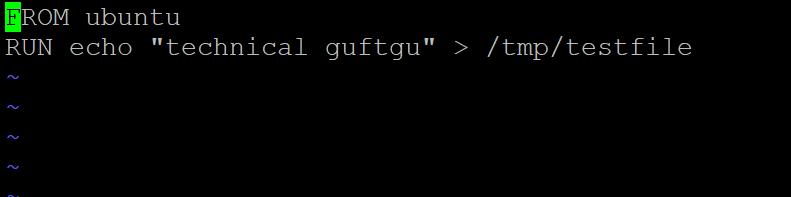
**HOW TO CREATE Dockerfile?**

1. Create a file named “Dockerfile”.
2. Add instructions in Dockerfile.
3. Build Dockerfile to create image.
4. Run image to create container.

Open vi editor

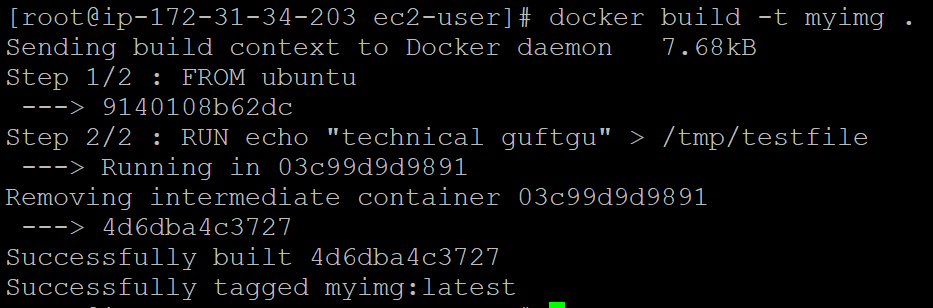
# vi Dockerfile

Write below codes in vi editor.

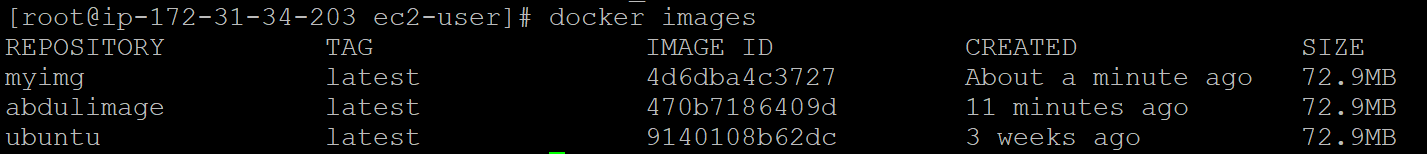


To create image run below command

# docker build –t myimg .



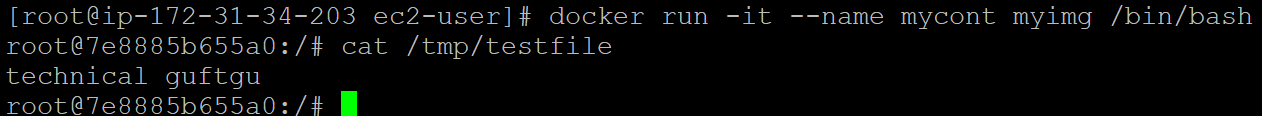
# docker images



Now, create container from the above image.

# docker run –it –name mycont myimage /bin/bash

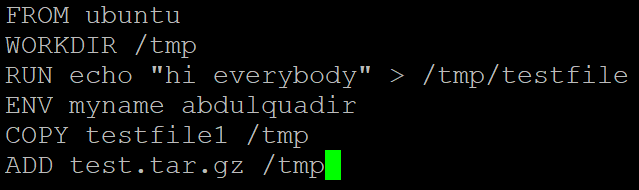
# cat /tmp/testfile



CREATING 2nd DOCKER IMAGE:

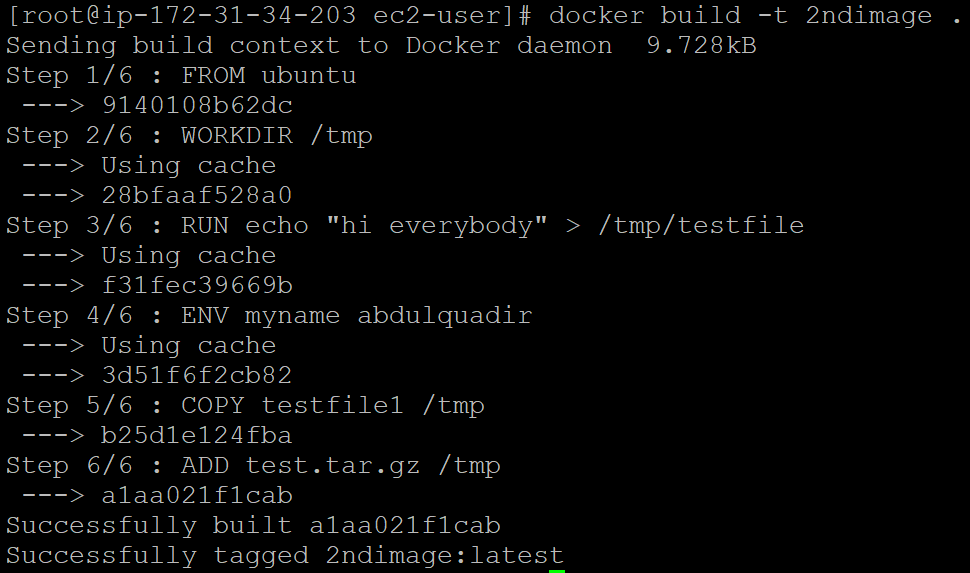
# vi Dockerfile

Write below instructions in vi editor.



Create some file like test.tar.gz & testfile1

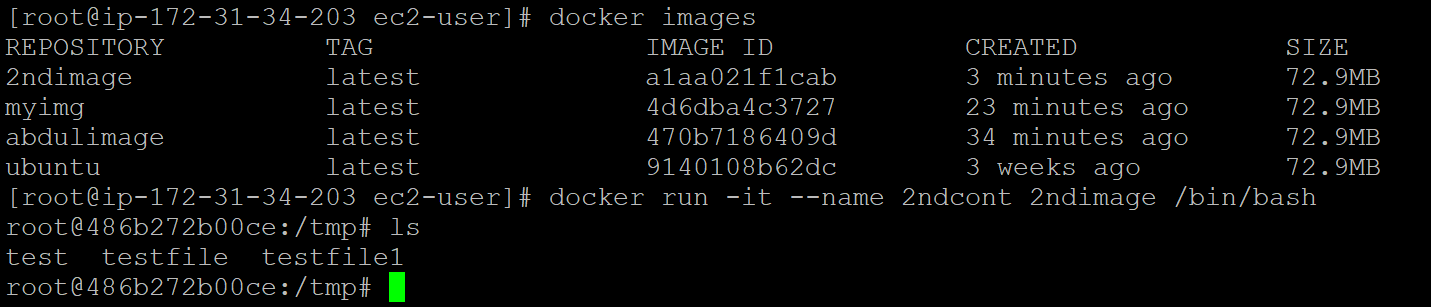
# docker build –t 2ndimage



# docker images

# docker run –it --name 2ndcont 2ndimage /bin/bash

As you can see file copies and unzipd automatically.



**DOCKER VOLUME & HOW TO SHERE IT?**

**DOCKER VOLUME:**

* Volume is simply a directory inside our container.
* Firstly, we have to declare this directory as a volume & then share volume.
* Even if we stop container, still we can access volume.
* Volume will be created in one container.
* You can declare a directory as a volume, only while creating containers.
* You can’t create volume from existing container.
* You can share one volume across any number of containers.
* Volume will not be included when you update an image.

You can map volume in two ways:

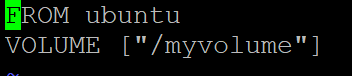
1. Container to container.
2. Host to container.

**BENEFITS OF VOLUME:**

1. Decoupling container from storage.
2. Share volume among different containers.
3. Attach volume to containers.
4. On deleting container volume doesn’t delete.

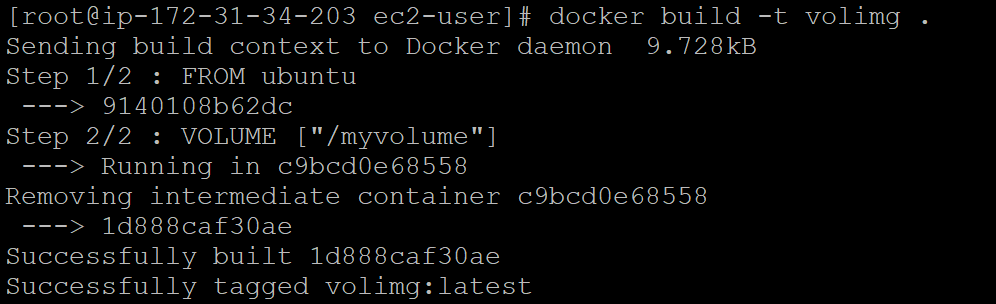
**CREATING VOLUMES FROM Dockerfile:**

Create a Dockerfile & write:



Then create image from this Dockerfile.

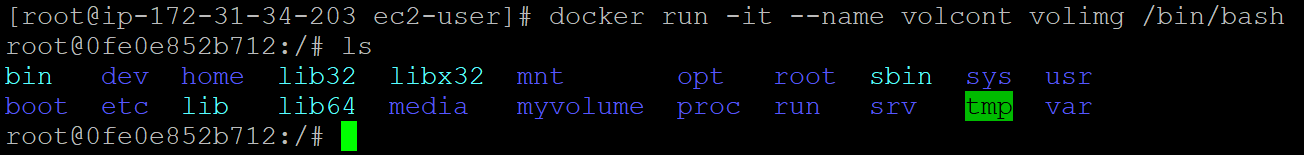
# docker build –t myimage



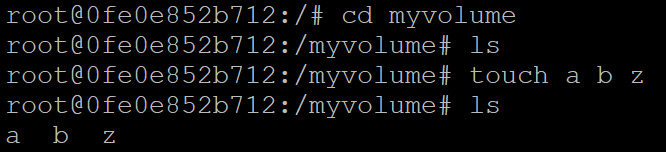
Now, create a container from the images & run container.

# docker run –it –name <container-name> <image-name> /bin/bash

Now, do ls you can see myvolume



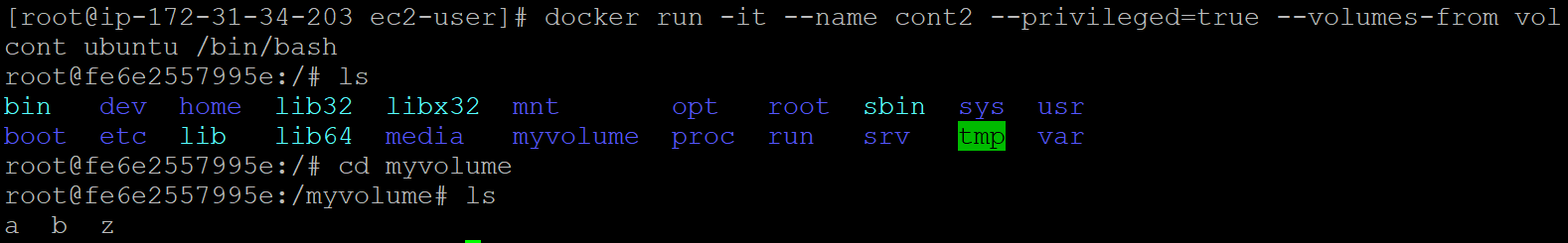
Now create some files in myvolume.



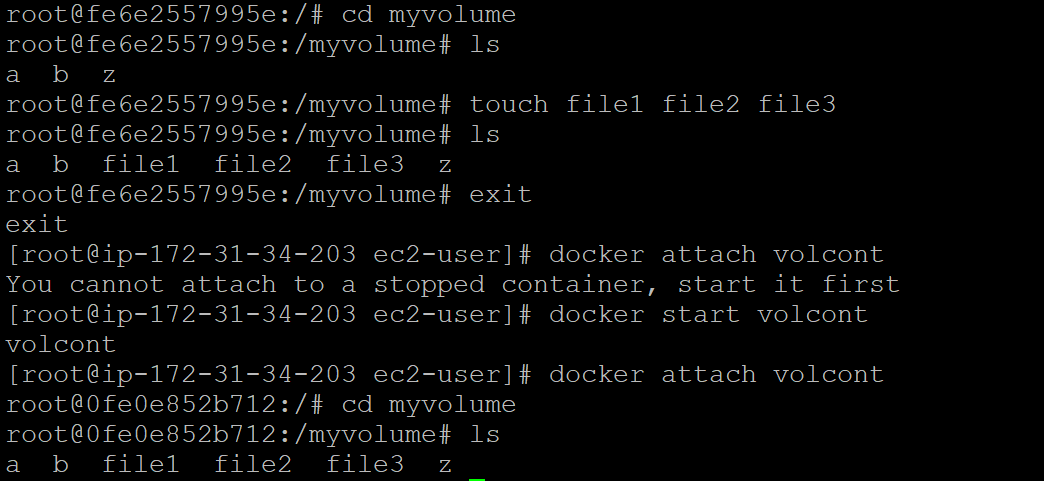
Now, share volume with another container.

Continer1 to container2

# docker run –it --name cont2 --privillege=true --volumes-from volcont ubuntu /bin/bash.



Now, after creating cont2, myvolume will be visible , whatever you do in one volume, can see from other container volume.



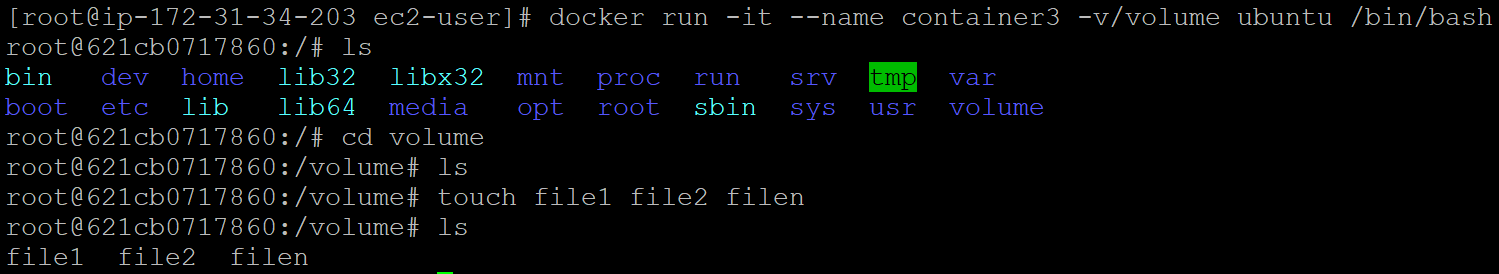
**CREATING VOLUME BY USING COMMAND:**

# docker run –it --name <container-name> –v/<volume-name> ubuntu /bin/bash

# ls

# cd /volume

Now, create one file cont3file & exit.

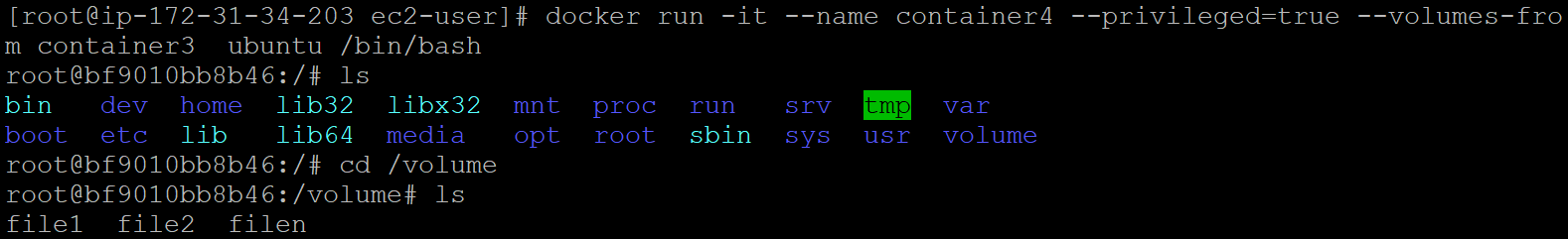


Now, create one more container & share volume2.

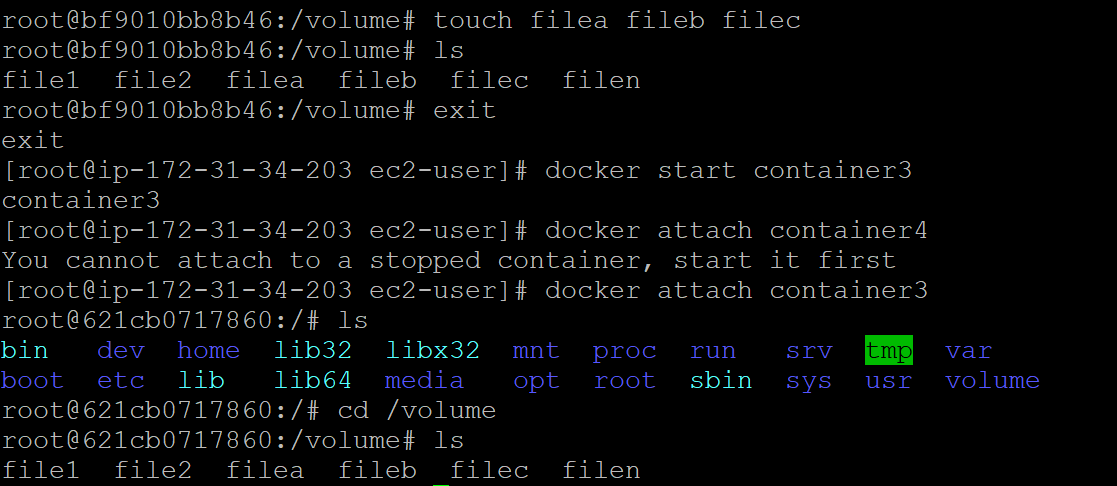
# docker run –it --name <container-name> --privilege=true --volumes-from <volume-container-name> ubuntu /bin/bash

Now, you are inside container

Do ls, you can see volume.



Now, create one file inside this volume & then check in container3, you can see that files.



**VOLUMES SHARING BETWEEN HOST & CONTAINER:**

Verify files in /home/ec2-user.

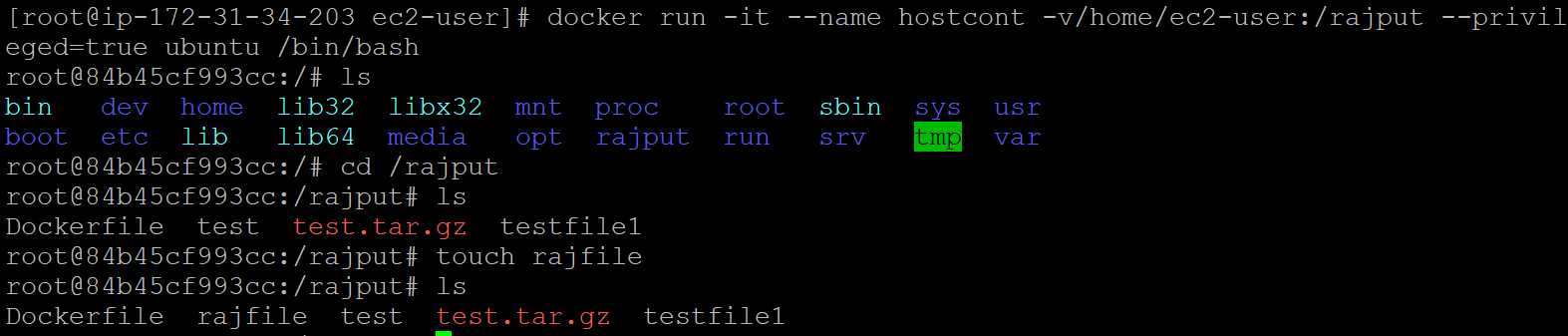
# docker run –it --name hostcont –v /home/ec2-user:/rajput --privilege=true ubuntu /bin/bash

# cd /rajput

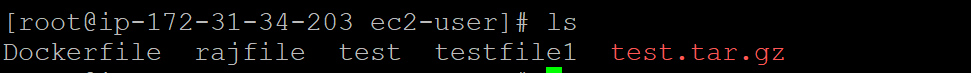
Do ls, now you can see all files of host machine.

# touch rajfile (in container)

Exit

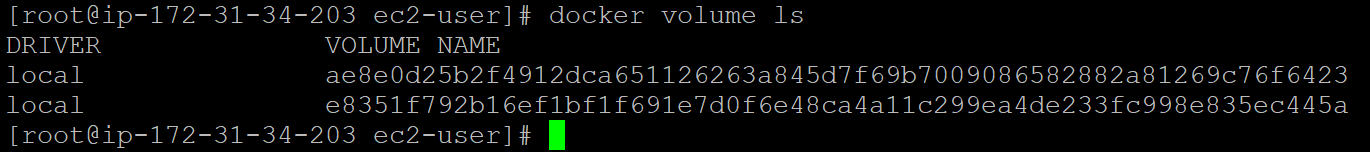


Now, check in ec2 machine, you can see this file.

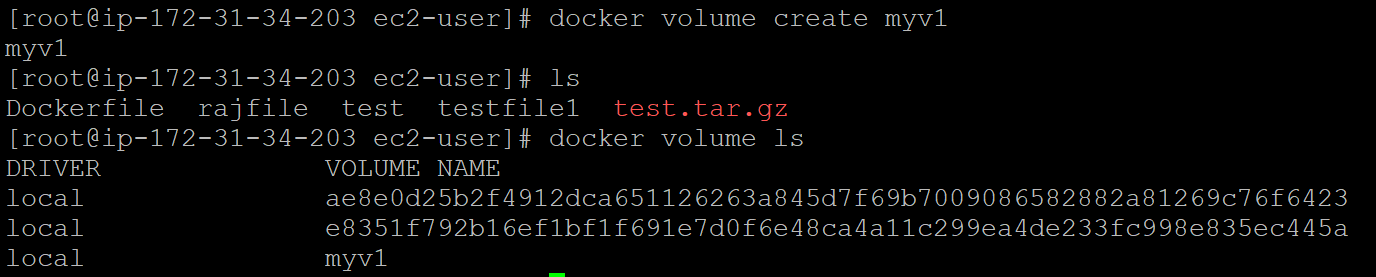


**SOME OTHER COMMANDS:-**

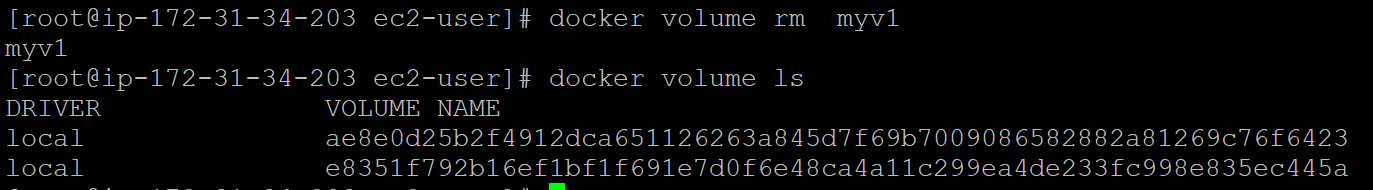
# docker volume ls



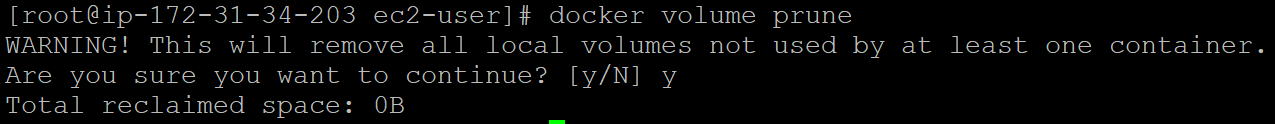
# docker volume create <volume-name>



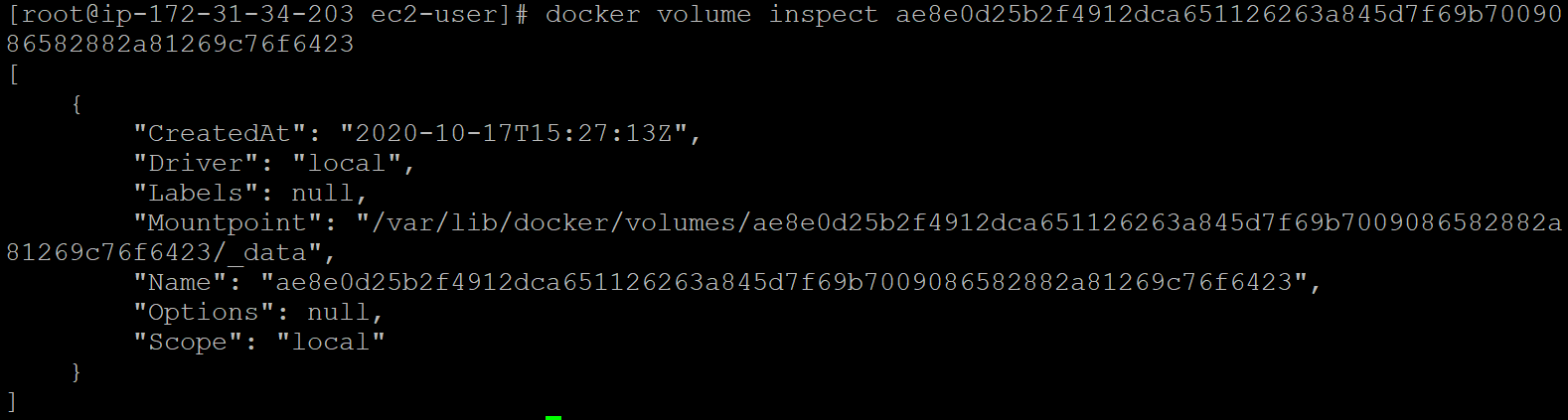
# docker volume rm <volume-name>



# docker volume prune {it removes all unused docker volumes}



# docker volume inspect <volume-name>



# docker container inspect <container-name>

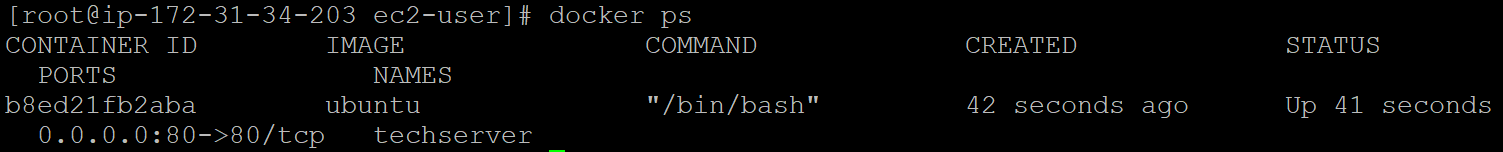
**DOCKER PORT EXPOSE:**

To map port

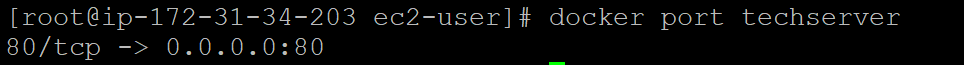
# docker run –td –name techserver –p 80:80 ubuntu



# docker ps

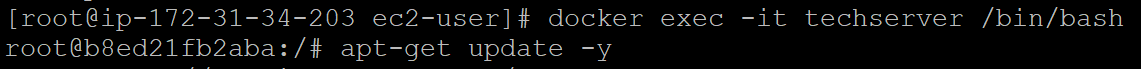


# docker port techserver



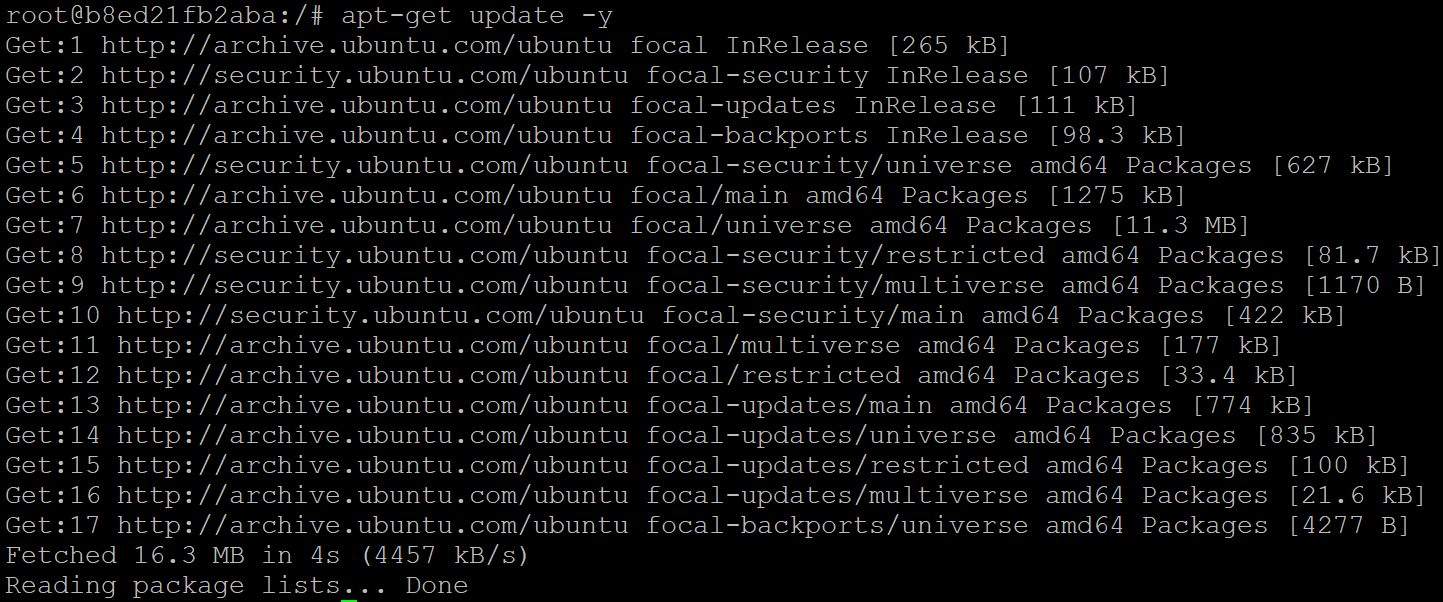
To enter in container by new process.

# docker exec –it techserver /bin/bash



Now, run this commands in container.

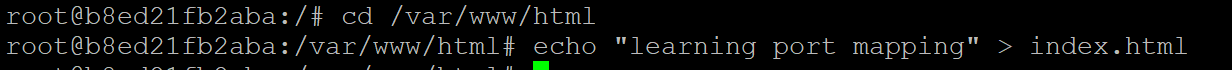
# apt-get update



# apt-get install apache2 –y

# cd /var/www/html

# echo “ subscribe techguftgu” > index.html

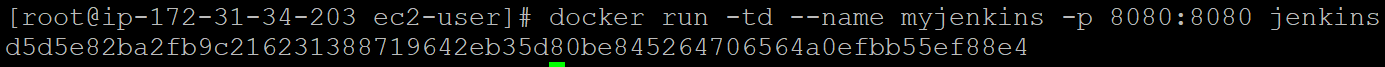


# service apache2 restart

Check public ip in browser of your linux machine.

Ex. 2

# docker run –td --name myjenkins –p 8080:8080 jenkins



Now check ip:8080 in web browser.



**DIFFERENCE BETWEEN DOCKER ATTACH & DOCKER EXEC:**

Docker **exec** create a new process in the container’s environment while **docker attach** just connect the standard input/output of the main process inside the container to corresponding standard input/output error of current terminal.

**docker exec** is specifically for running new things in an already started container be it a shell or some other process.

(pid = process id, ppid = parent process id)

**What is difference between expose & publish a docker?**

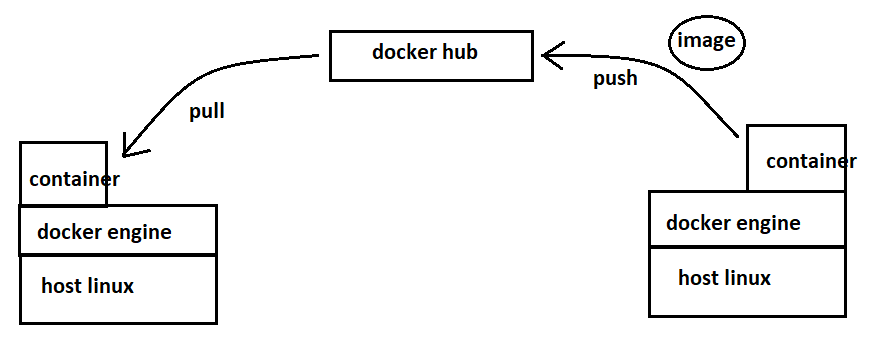
Basically you have three options:-

1. Neither specify **expose** nor **–p.**
2. Only specify **expose.**
3. Specify **expose** & **-p.**

* If you specify neither **expose** nor ­**–p**, the service in the container will only be accessible from inside the container itself.
* If you **expose** a port, the service in the container is not accessible from outside docker, but from inside other docker containers, so this is good for inter-container communication.
* If you **expose** & **-p** a port, the service in the container is accessible from anywhere, even outside docker.
* If you **–p** but don’t **expose** docker does an implicit expose. This is because if a port is open to the public, it is automatically also open to the other docker containers, hence **­–p** includes expose.

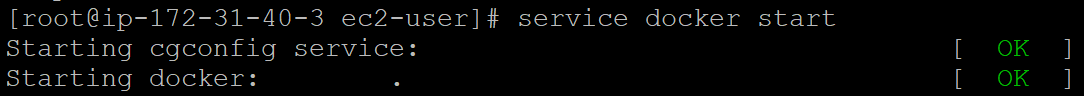
WHEN YOU STOP YOUR CONTAINERS IT UNMAPS ALL PORTS.

**HOW TO PUSH DOCKER IMAGE IN DOCKER HUB?**



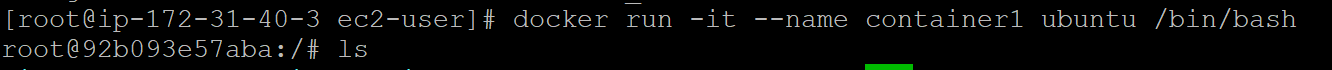
To start docker service.

# service docker start

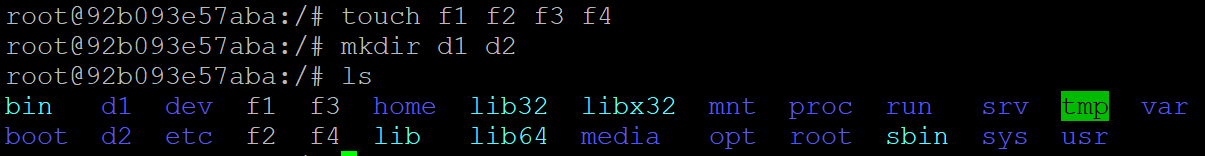


Create a container & enter in it.

# docker run –it --name container1 ubuntu /bin/bash

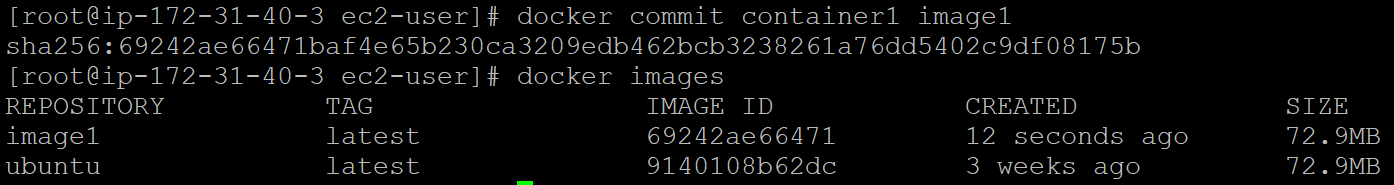


Now, create some files inside container.



Now, create image of the container.

# docker commit container1 image1

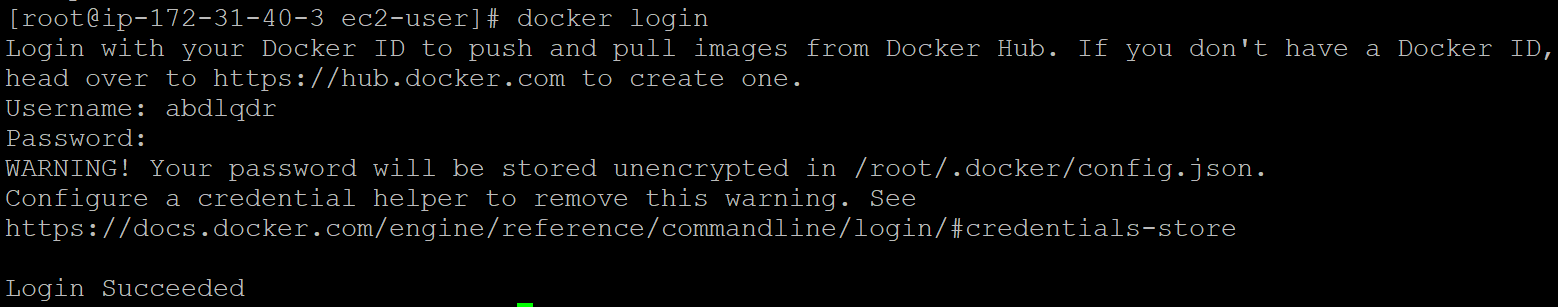


Now, create account in hub.docker.com

Now, go to aws linux.

# docker login

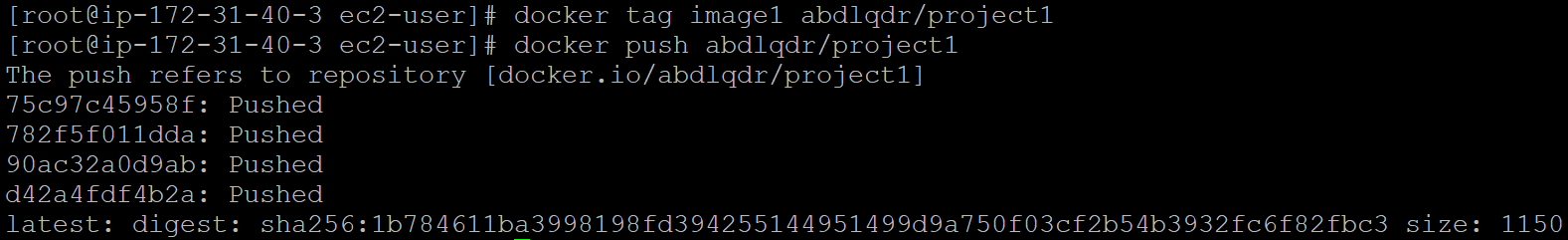
Enter username & password.



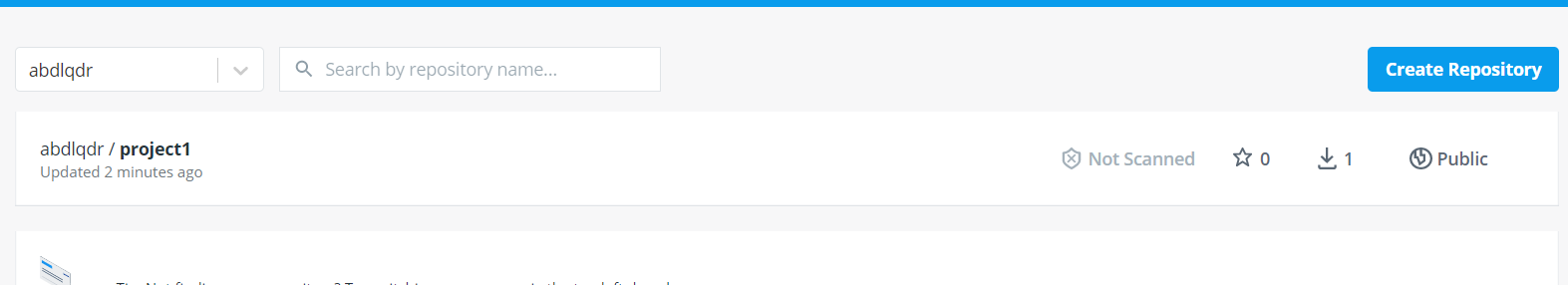
Now, give tag to your image.

# docker tag image1 <dockerhub-usrname>/<image-name>

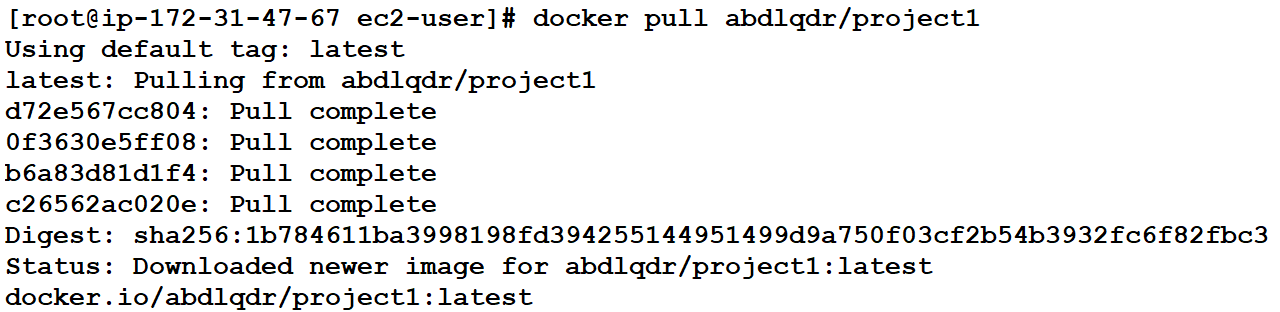
# docker push <dockerhub-usrname>/<image-name>



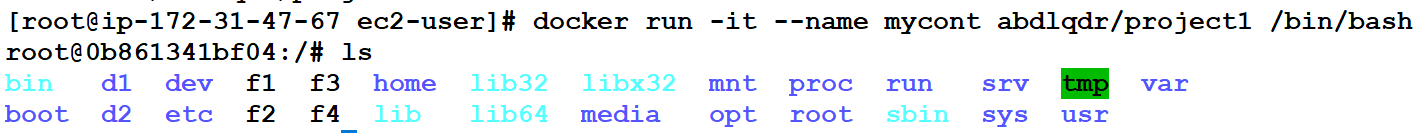
Now, you can see this image in docker hub account.



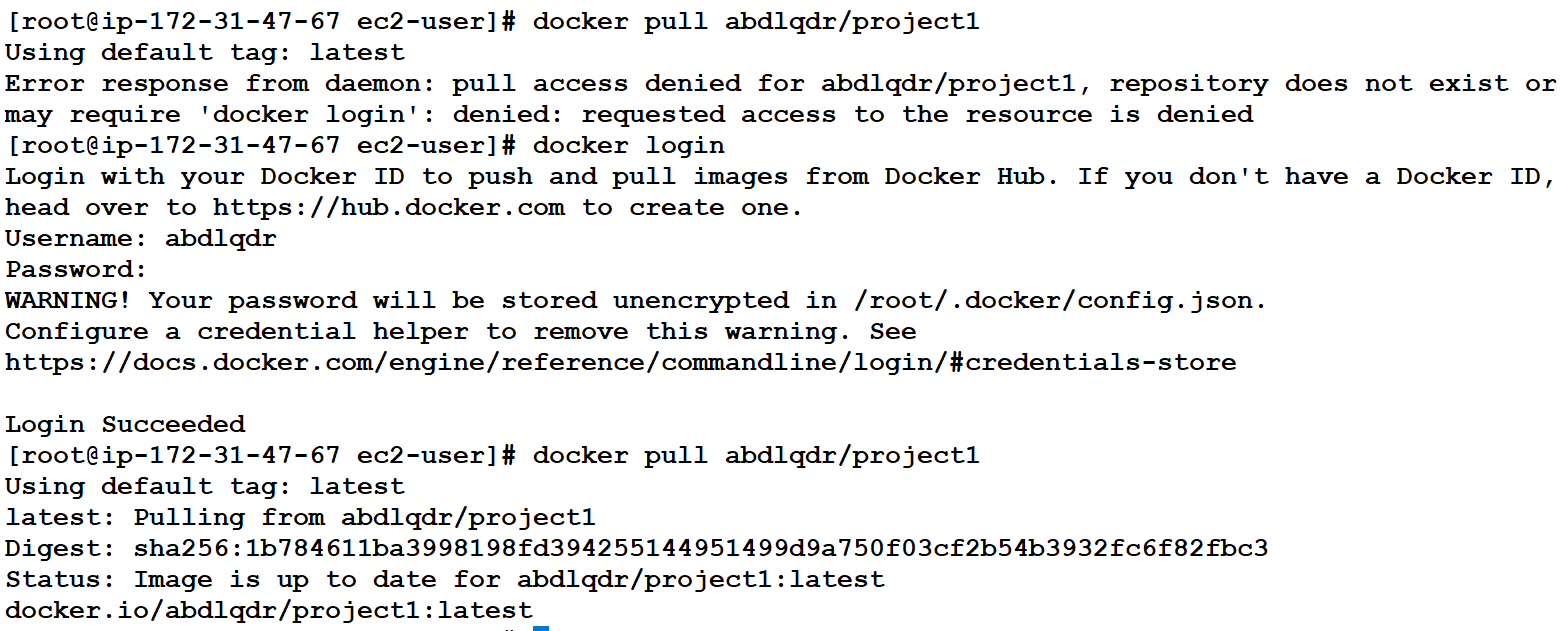
Now, create one instance in Tokyo region & pull image from hub.

# docker pull <dockerhub-usrname>/<image-name>

# docker run –it –name mycont <dockerhub-usrname>/<image-name> /bin/bash



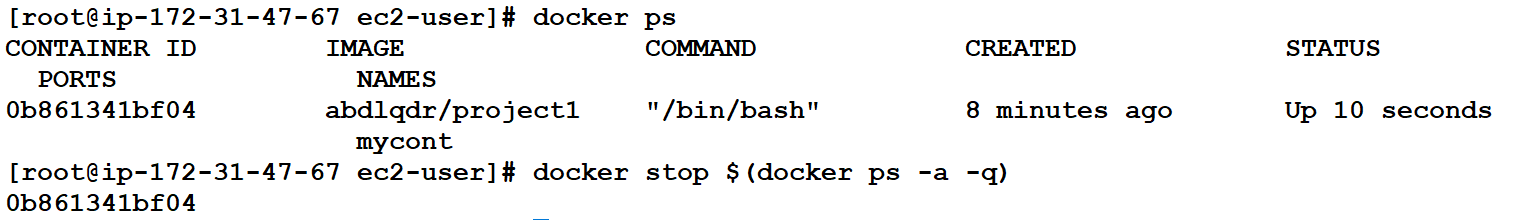
If i make this docker hub file private then we have to login first for pulling.



**SOME IMPORTANT COMMANDS:**

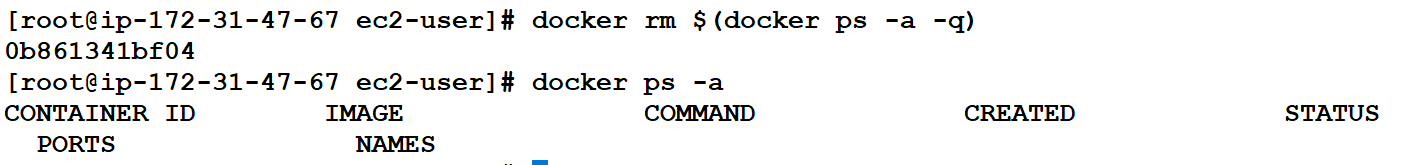
Stop all running containers.

# docker stop $(docker ps –a –q)



Delete all stopped containers.

# docker rm $(docker ps –a –q)



Delete all images

Docker rmi -f $(docker images –q) 