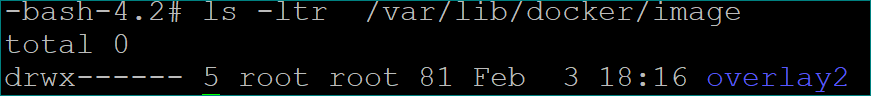
**Docker Images**

* **Image is Layers** of multiple files
* **Docker images** are templates used to create Docker containers
* **Docker image is source of docker container**
* Docker can build images automatically by reading the instructions from a dockerfile
* **Dockerhub:** dockerhub is public registry
* All image related information will be maintained ***docker local registry***

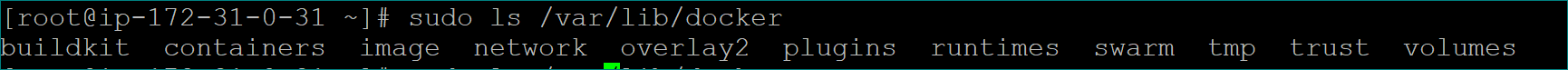
***(i.e var/lib/docker)***

* Will have only image details i.e ***/var/lib/docker/image***

****

* And layers of image information will be ***/var/lib/docker/overlay2***
* Each image will have multiple layers
* Whenever we are pushing or pulling the images, all the layers of images will be pushing into the registry and all the layers of images will pulling into the docker server
* **How to check (list) the layers of an image?**
* docker history <Image\_Name:tag/Id>
* docker image inspect <ImageID/ImageName:tag>
* docker image inspect venki22/maven-web-application

**Working folder of Docker:**

****

**Containers:**  is running instance of images

A single image can be used to create multiple containers

**To start the docker**

systemctl start docker

**To check the status of docker**

systemctl status docker

**To stop the docker**

**systemctl status docker**

**To stop the docker completely**

**systemctl stop docker.socket**

**To download (Pull) an image or a repository from a registry**

**Registry:** Collection of one or more repository is called registry

**Repository:** Collection of same type of versions is called repository

**docker pull <image\_name>**

**(or)**

**docker pull <image\_name>:<latest>**

**Note:**

* If we don’t mentioned anything it will download latest version

docker pull <image\_name>:<version\_number>

* If we want to specific version, we need to mentioned version number

at last

* docker pull mysql:5

**To see the list of docker images**

docker image ls

(or)

docker images

**To delete a docker image from docker host**

docker rmi <image\_name **|** image\_id>

docker rmi <image\_name-1> <image\_name-2> ..etc

**docker rmi -f $(docker images –q) // Delete all images with stopped container**

**docker rmi -f venki22/maven-web-application:11**

**Note: We can‘t delete images with running container.**

**To upload a docker image into docker hub**

docker push <registryURL/repository:tag>

docker push venki22/maven-web-application:1

**To build an image from a customised container**

docker commit container\_name/container\_id new\_image\_name

**To create an image from docker file**

docker build -t <registry\_Name/<Repository\_Name>:<version> <build-context>

docker build -t <ImageName:tag> .

docker build -t <ImageName> . #default tag will be latest

docker build -t venki22/maven-web-application .

docker build -f <customdocker-filename> -t <registryURL/repository:tag> <build-context>

docker build -f Dockerfilecust -t venki22/tomctimg:1 .

**Note:**

* If we don't mention version information. By default it will use '***latest***' as version
* Image name should have repository details along with name and version.
* We can give custom file name as well for that we have to use ***-f option*** to mention the dockerfile name and path
* We need to maintain all versions of docker images using tag
* While doing pull/push we have to use proper tag

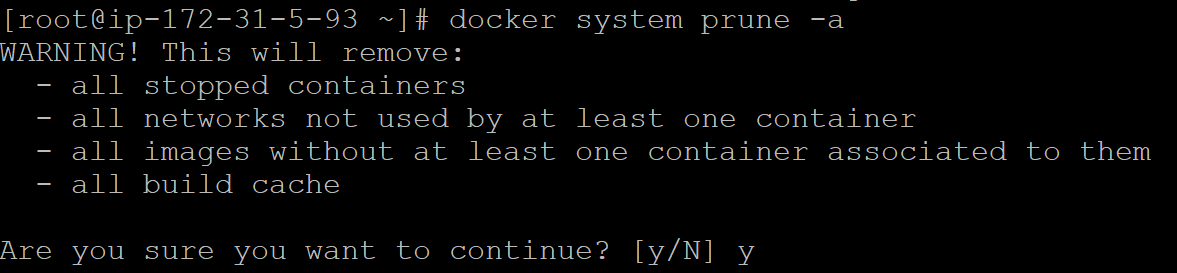
**To search for a docker image**

docker search <image\_name>

**To delete all images those are not attached to containers**

To clean the docker environment, removing all the containers and images.

docker system prune -a



**#Delete all image ids using below commands**

docker rmi -f $(docker images -q) ***//forcefully***

docker rmi $(docker images –q)

**Dangling Images:**

Dangling images are layers that have no relationship to any tagged images. They no longer serve a purpose and consume disk space. They can be

flag, -f with a value of ***dangling=true*** to the docker images command.

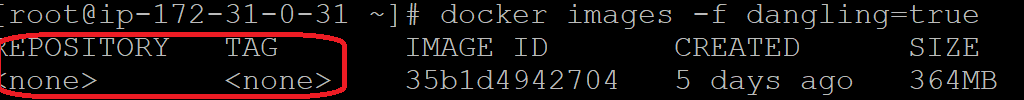
Which doesn’t have repository and un-tagged.

docker rmi -f <registry/repository:tag>

docker system prune **// clean up activity**

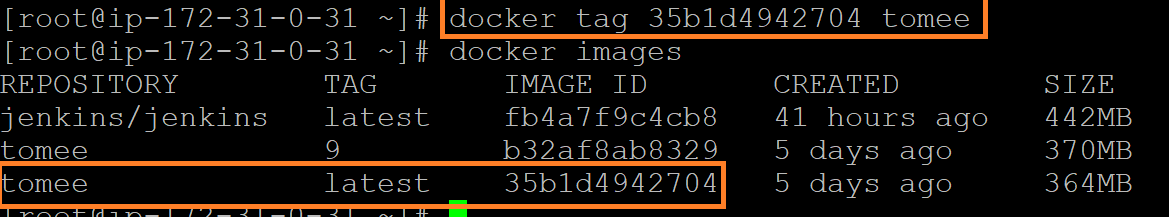
do

docker images -q -f dangling=true **//return only dangling images ids**

****

docker tag <old-image-id> <image-name> **//re-assign**

docker tag 35bd4942704 tomee

****

**//it will remove all dangling images**

docker rmi -f $(docker images -q -f dangling=true)

docker container prune

docker volume prune

docker network prune

**docker inspect image <imageId | image\_name>**

docker inspect image tomee

docker inspect tomee

**docker info**

**#To see docker server details**

*Working directory of docker is /var/lib/docker*

**Docker Root Dir:** /var/lib/docker

**docker --version**

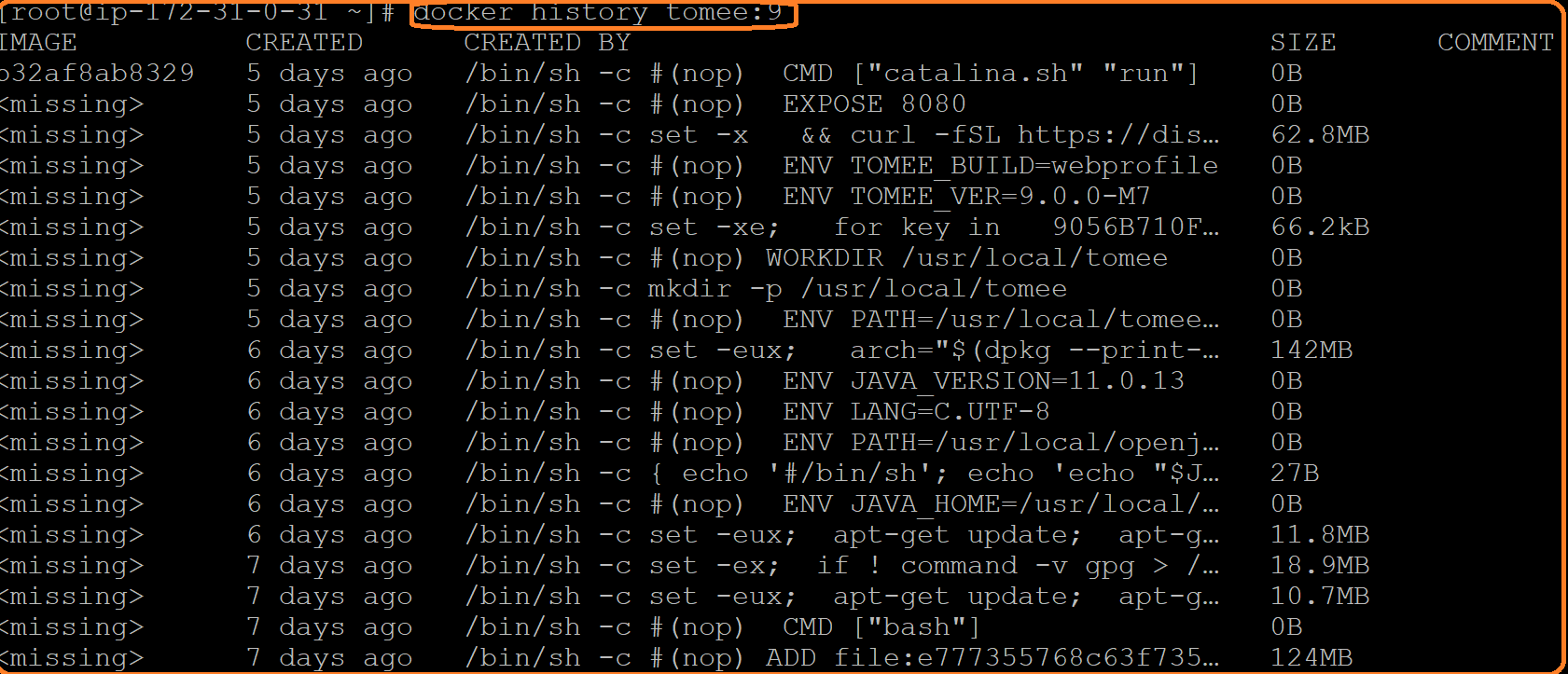
**docker version**

#Docker version 20.10.7, build f0df350

**#To see the list of layers in image**

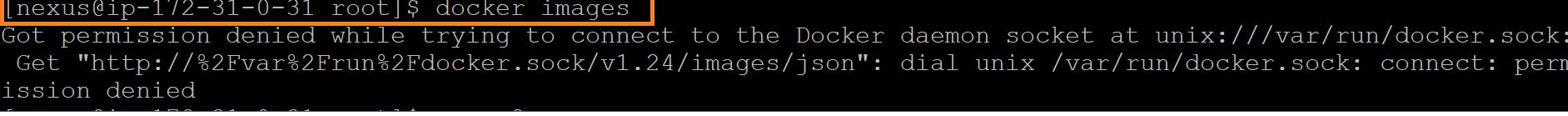
**docker history <imagename:tag>**

docker history tomee:9

docker history tome:9

**Adding user in docker Group**

We have install docker in root user from root user we can run all docker related command once, we switch to other user we can’t run docker related command and we will get below message

****

If we want to run docker commands with other users we have 2 options

1.we have to add sudo keyword for each docker command from other user

2.if we don’t want to use sudo keyword, in this case we can add that user in **docker** group using below command

**sudo usermod -aG docker <user\_name>**

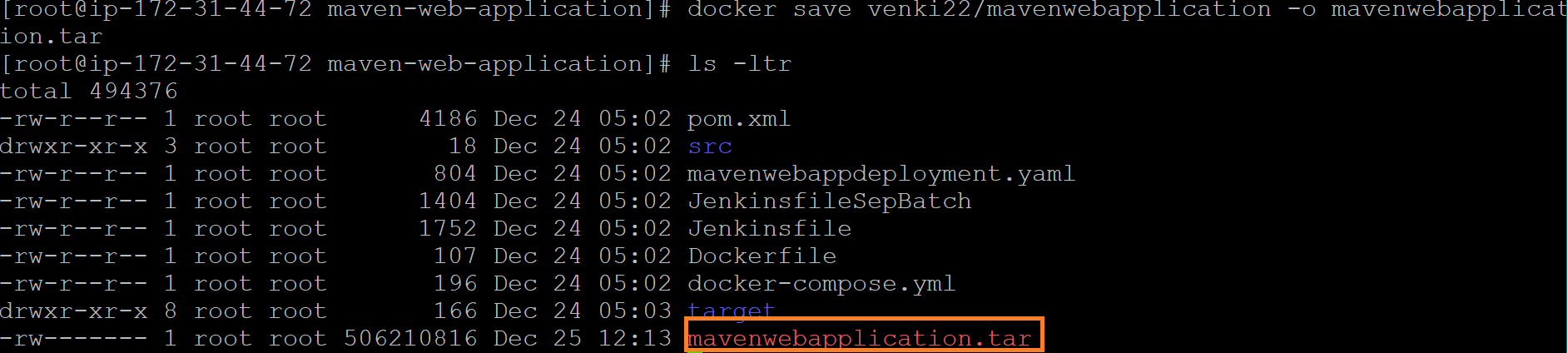
**Moving Image from one sever to another server without using registry hub.**

We can use below command to move image from one server to another server

**docker pull venki22/maven-web-application:1**

**docker save <Image\_name> -o <File\_Name>.tar**

**docker save venki22/maven-web-application:1 -o mavenwebapplication.tar**

****

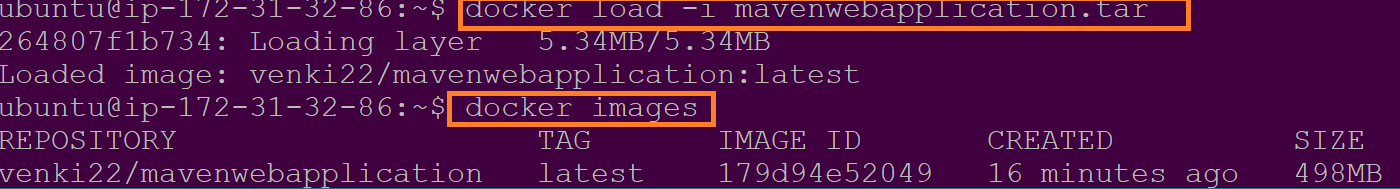
We can move .tar file to other server using SCP command

**scp -i devops.pem /root/maven-web-application/mavenwebapplication.tar ubuntu@3.108.60.146:/home/ubuntu/**

**(Another Server)**

**docker load -i <Imange\_name>.tar**

**docker load -i mavenwebapplication.tar**

****

**#In case ECR(ElasticContaienrRegistry)**

We will use ECR to maintain the docker images

docker build -t <registryURL/repository:tag> <buid-context>

docker build -t 485430697198.dkr.ecr.ap-south-1.amazonaws.com/maven-web-application:1

docker login -u <username> -p <password> 485430697198.dkr.ecr.ap-south-1.amazonaws.com

**#In case Nexus**

docker build -t <nexusserverDomainname or IP:Port>/<contextroot:tag>

docker build -t 172.31.89.32:8083/maven-web-application:1 #with IP Address

docker build -t nexus.tcs.com/maven-web-application:1 #with domain name

**#using pipeline**

def imagetag = BUILD\_NUMBER

sh docker build -t venki22/maven-web-application:${imagetag} .

sh docker push venki22/maven-web-application:${imagetag}

**dockerhub login command:**

**#login docker hub**

docker login -u <username> -p <password> <registryURL/repository:tag>

docker loign -u <username>

**#In case of Jfrog or nexus**

docker login <nexus/jfrog endpointurl>

docker login 172.31.89.32:8083

docker login 172.31.89.32:8083 -u <username> -p <password>

docker login nexus.tcs.com