**Docker**

**Introduction**

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications

There are two big pieces to Docker: **The Docker Engine**, which is the Docker binary that’s running on your local machine and servers and does the work to run your software. **The Docker Hub**, which is a website and cloud service that makes it easy for everyone to share their docker images.

Docker is available in two editions: **Community Edition (CE)** and **Enterprise Edition (EE)**. Docker Community Edition (CE) is ideal for developers and small teams looking to get started with Docker and experimenting with container-based apps. Docker CE has two update

channels, **stable** and **edge**:

**Stable** gives you reliable updates every quarter

**Edge** gives you new features every month

Docker Enterprise Edition (EE) is designed for enterprise development and IT teams who build, ship, and run business critical applications in production at scale.

## Dockerfile (Docker builder):

* Dockerfile is a file, it will describe the steps to create an image quickly.
* The Dockerfile uses a basic DSL (Domain Specific Language) with instructions for building Docker images.
* The Docker daemon runs the instructions in the Dockerfile are processed from the top to down, so you should order them accordingly.

## The Dockerfile is the source code of the Image.

## Docker image

* **An image is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files**.

**Container**

* **A container is a runtime instance of an image**—what the image becomes in memory when actually executed. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.

**Docker Commands**

**#docker --version (OR) #docker version** Docker version 17.03.1-ce, build c6d412e **#docker-compose --version**

docker-compose version 1.11.2, build dfed245

## # docker-machine --version

docker-machine version 0.10.0, build 76ed2a6

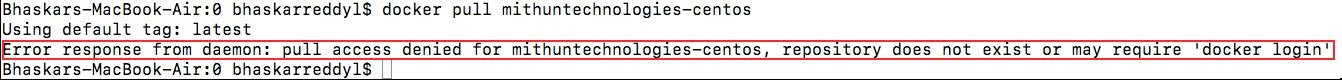
**#docker info:** It will give status of weather docker is running or not will display the detailed information about docker service.

Text

Description automatically generated

**#docker run <Docker Image Id / Docker Image Name>:** To run any docker image, we will use run command.

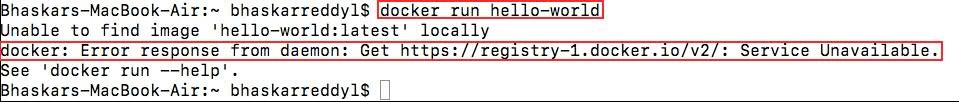
**Note:** If you give wrong image name, you will get the below error.



**#docker run hello-world:** This command will download the hello-world image, if it is not already present, and run the hello-world as a container.

Here hello-world is image name.

**Note:** While executing above command, if hello-world image is not there in local docker engine, and if your machine is not connected to internet, you will get below error.



Text, letter

Description automatically generated

**#docker images**: List all images that are locally stored with the Docker engine (OR)

## #docker image ls

Table

Description automatically generated with medium confidence

**#docker images <<Image Name>>:** It will display the information about image.



**#docker images -q:** It will display only the images IDs.

Graphical user interface, text, application

Description automatically generated

**#docker rmi << Image ID/Image Name>> (OR) docker image rm << Image ID/Image Name>>:** It will delete an image from the local image store.

## #docker rmi nging

Text, letter

Description automatically generated

**#docker rmi -f 27d783d4d74d:** With image id also we can remove docker image.

Text

Description automatically generated

**#docker rmi $(docker images -q) (OR) docker rmi -f $(docker images -q) :** It will remove all the images from docker engine.

**#docker run --name “hello-world-container” helloworld:** Start the hello-world image with “**hello- world-container**” container name.

**#docker create “hello-world-cont” helloworld :** It will create a container called “hello-world-cont” from the image hello-world and it won’t start the container.

**Note:** If the image is not found locally, Docker will pull it from Docker Hub.

**#docker ps (OR) docker container ls:** Lists running containers (It will not display the stopped containers)

**#docker ps –a (OR) docker container ls --all (OR) docker container -a:** Lists all containers (It will display the stopped containers along with running containers.)

A picture containing calendar

Description automatically generated

docker ps -a

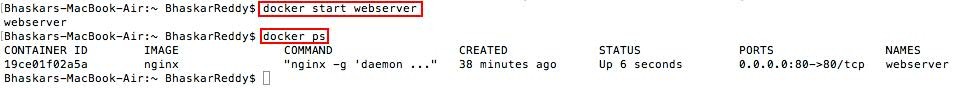
**#docker stop <container name|id> (OR) docker container stop <container name|id>:** It will stop the docker container.

**#docker stop webserver:** It will stop the container called webserver.



## #docker start <Container name|id>

**#docker start webserver:** It will start the webserver.



**#docker pause CID/CNAME:** It will pause the container.

**#docker unpause CID/CNAME:** It will unpause the container.

## Docker Container status are One of below

created, restarting, running, removing, paused, exited, or dead

**#docker ps -a --filter "name=vinc16thjune":** It will display all the containers with name vinc16thjune name.

## #docker ps -a --filter 'exited=0' :

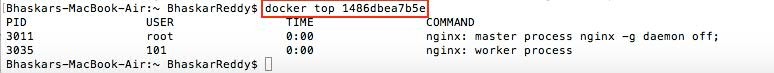
**#docker ps --filter status=running**:

**#docker ps --filter status=paused:**

**#docker logs <container name>:** It will display the logs for that container.

**#docker logs --tail 100 <<Container Name>>:** Print the last 100 lines of a container’s logs.

**#docker top <<Container ID>>:** This will shows the top processes in within in a container. #**docker top 1486dbea7b5e**



**#docker reanme <<Container Old Name>> <<Container New Name>>:** It will rename the conatiner.

**#docker rm -f <<Container Name>>:** It will remove the container.

**#docker rm -f webserver:** It will stop and remove the running container with a single command.



**#docker stop:** It will restrt the container.

**#docker stop $(docker ps -a -q):** It will stop all the containers.

**#docker rm -f $(docker ps -aq):** Delete all running and stopped containers.

**#docker kill <<CID/C Name>>:** It will kill the containe.

**#docker container prune:** Delete all stopped containers.

Text

Description automatically generated

**#docker ps -a|awk '$2=="hello-world" {print $1}' |xargs docker rm :** It will delete all the containers related hello-world image.

## #docker ps --filter "status=paused"

**#docker search <<Image Name>>:** It will search all of the publicly available images on Docker Hub(https://hub.docker.com).

**Note:** This command will not work if your machine is not connect to internet.



**#docker search puppet:** It will search puppet images from Dockerhub.

Text

Description automatically generated

This command will returns the below details.

* Repository names
* Image descriptions
* Stars - these measure the popularity of an image
* Official - an image managed by the upstream developer (e.g., the ubuntu image managed by the Ubuntu team)
* Automated - an image built by the Docker Hub’s Automated Build process

## #docker search --no-trunc puppet

Graphical user interface, text

Description automatically generated

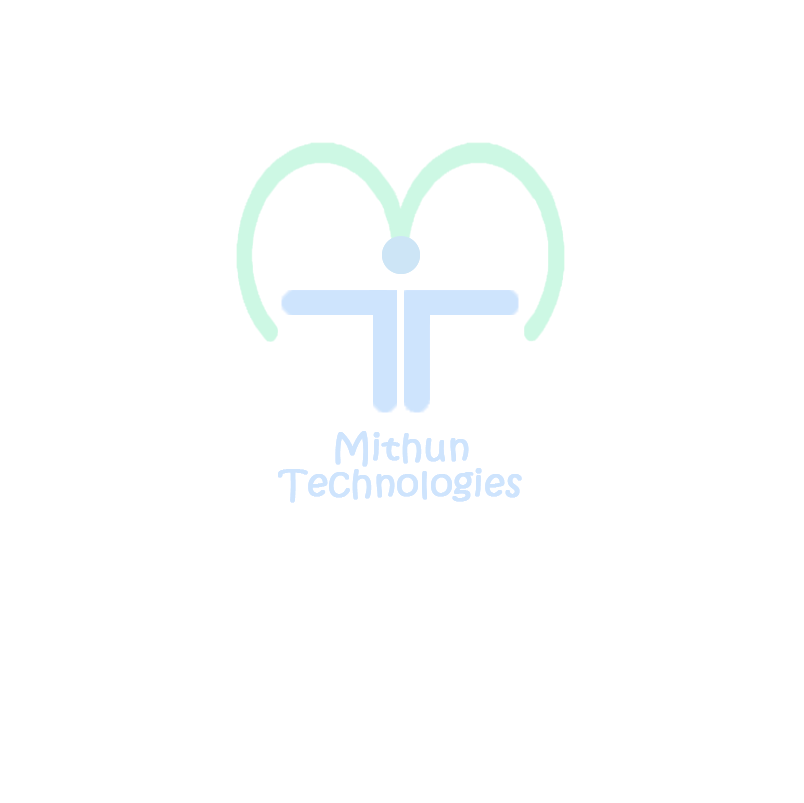
**#docker search --filter=stars=10 puppet**

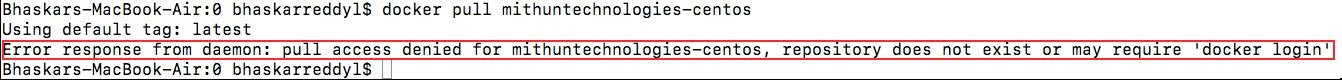
A screenshot of a computer

Description automatically generated with medium confidence

**#docker pull <<Image Name>:** Pull an image from Docker Hub

**Note:** If you give wrong image name, you will get the below error.





**#docker pull ubuntu:** It will download the ubuntu image from docker hub. If no tag is provided, Docker Engine uses the :latest tag as a default.

This command pulls the ubuntu:latest image.

**#docker pull ubuntu:16.04 :** It will download the ubuntu 16.04 version.

**#docker run -i -t ubuntu:16.04 /bin/bash:** It will conenct to the ubuntu. (By default user is root)



**#docker inspect <<CID>>** : It will give information for container.

**Note:** You can use “docker inspect “ command to get information for containers, netwroks.., not only for containers.

## #docker inspect --format '{{ .State.Status }}' <<CID>> :

**#docker inspect --format '{{ .State.Pid }}' <<CID>> :**

**#docker inspect --format '{{ .NetworkSettings.IPAddress }}' <<CID>> :**

**For More info:** https://docs.docker.com/engine/reference/commandline/inspect/#get-an-instances- mac-address

**#docker attach <<CID>> :** It will conenct to running container.

**Note:** If the container is in stop/pause state while executing docker attach command, you will get below error.





**#docker exec <<CID>> :** Run a linux command in a running container.

## #docker exec <<CID>> ls :

**#docker exec <<CID>> hostname :**

**#docker exec <<CID>> touch /tmp/vinctechnologies.txt :**

**#docker stats <<CID>> :** It will display a live stream of container resource usage statics.

**Note:** If you want to come out form console, type Ctrl + C.

Execute docker stats and then execte below command in another command prompt and see the result.

## #docker exec <<CID>> yum update

If the container is paused, then the docker exec command will fail with an error:

## #docker pause <<CID|CNAME>>

**#docker exec <<CID>> ls :** It will give error, because the container is in paused state.

**#docker network --help :** It will display the all options for network command.

**#docker network ls:** List the networks.

Network is a group of two or more devices that can communicate.

In practice, a network is comprised of a number of different computer systems connected by physical and/or wireless connections.

A picture containing text, microscope, computer

Description automatically generated

Text

Description automatically generated

**#docker network create vinctechnologies :** It will create the network with name called as vinctechnologies with default type is bridge.

**#docker network rm vinctechnologies :** It will delete the network with name called as vinctechnologies.

**#docker network inspect bridge :** Display detailed information on one or more networks.

**#docker network connect :** Connect a container to a network

## docker network connect [OPTIONS] NETWORK CONTAINER

Options:

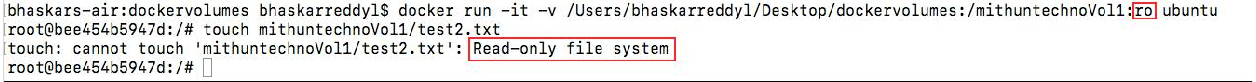
--alias strings Add network-scoped alias for the container

--ip string IPv4 address (e.g., 172.30.100.104)

--ip6 string IPv6 address (e.g., 2001:db8::33)

--link list Add link to another container

--link-local-ip strings Add a link-local address for the container



**#docker network prune:** Remove all unused networks.

**Docker Volumes**

By default, all files created inside a container.

The data doesn’t persist when that container is no longer running, and it can be difficult to get the data out of the container if another process needs it.

Volume is a specially designed directory.

Volumes are created and managed by Docker, if you don’t create explicitly and set. Volumes are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux).

We can create a volume explicitly using the **'docker volume create**' command docker voulume --help:

## Create a container with volume vinctechnoVol1

#docker run -it -v /Users/bhaskarreddyl/Desktop/dockervolumes:/vinctechnoVol1 ubuntu

## Create the Read only Volume

**docker run -it -v /Users/bhaskarreddyl/Desktop/dockervolumes:/vinctechnoVol1:ro --name ubuntucontainer16 ubuntu**

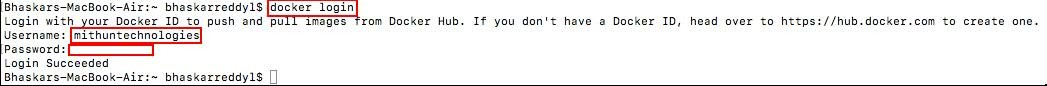
**#docker run -it --volumes-from ubuntucontainer16 --name ubuntucontainer1604 ubuntu:16.04:**

Create a container ubuntucontainer1604 that uses the same volumes as ubuntucontainer16

## Can I mount same volume to multiple docker containers?

Ans) Yes you can add same location as a volume to many docker containers.

**#docker login:** To sign into the Docker Hub.



If no server is specified, then the default is used **Default is** https://index.docker.io/v1/ https://registry-1.docker.io/v2/

**#docker logout:** To logout from the Docker Hub. If no server is specified, then the default is used.



**Note:** In docker hub we can create only one private repository. We can create many public repositories.

Dockerfile have various keywords.

**FROM:** This keyword indicates the base image from which the container is built.

# **MAINTAINER:** One of the commands that can be set anywhere in the file - although it would be better if it was declared on top - is MAINTAINER. This non-executing command declares the author, hence setting the author field of the images. It should come nonetheless after FROM.

**RUN:** This keyword indicates the command that needs to be executed on the image.

**EXPOSE**: The EXPOSE command is used to associate a specified port to enable networking between the running process inside the container and the outside world (i.e. the host).

# **ENV:** The ENV command is used to set the environment variables (one or more). These variables consist of “key = value” pairs which can be accessed within the container by scripts and applications alike.

**COPY**: Copying local files into container.

# **CMD:** The command CMD, similarly to RUN, can be used for executing a specific command. However, unlike RUN it is not executed during build, but when a container is instantiated using the image being built.

**ENTRYPOINT :** ENTRYPOINT has two forms:

ENTRYPOINT ["executable", "param1", "param2"] (exec form, preferred) ENTRYPOINT command param1 param2 (shell form)

An ENTRYPOINT allows you to configure a container that will run as an executable.

**VOLUME :** The VOLUME command is used to enable access from your container to a directory on the host machine (i.e. mounting it).

**WORKDIR :** The WORKDIR directive is used to set where the command defined with CMD is to be executed.

## What is the difference between ADD and COPY?

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 command will copy files/directories from your host machine to your image.

ADD command will copy files/directories from you host machine to your image and also it will copy tar files and will extract that file and remove the tar file once it extract. In this case if you use COPY command you need to use layers like below.

COPY vinctechnologies.tar /tmp/

RUN tar -xvf /tmp/vinctechnologies.tar -C /usr/local/ RUN rm /tmp/vinctechnologies.tar

If you user ADD

ADD vinctechnologies.tar /usr/local/

And also ADD is used for getting files from remotely and copy into image as follows. ADD [http://vinctechnologies.com/vinctechnologies.tar](http://mithuntechnologies.com/mithuntechnologies.tar) /usr/src/vinc/

## What is the difference between RUN and CMD?

RUN is an image build step, the state of the container after a RUN command will be committed to the docker image. A Dockerfile can have many RUN steps that layer on top of one another to build the image.

CMD is the command the container executes by default when you launch the built image. A Dockerfile can only have one CMD. The CMD can be overridden when starting a container with docker run $image $other\_command.

ENTRYPOINT is also closely related to CMD and can modify the way a container starts an image.

With help of below command we can create the docker image from Dockerfile.

docker build

Syntax: docker build -t ImageName:TagName dir Here

* -t is to mention a tag to the image
* ImageName – This is the name you want to give to your image
* TagName – This is the tag you want to give to your image
* Dir – The directory where the Dockerfile is present.

Example: docker build -t mongodbimage:1.0 .

In this example **mongodbimage** is the image name

**1.0** is the tag name

**.** is the current directory for the Dockerfile.

Since the Docker File is in the present working directory, we used "." at the end of the command to signify the present working directory.

Example: docker build -f /path/to/a/Dockerfile.

## Task: Create the one Dockerfile to install httpd and copy index.html file into

**/usr/local/apache2/htdocs/ directory.**

Step 1: Create the file name called Dockerfile and put the below contents. FROM httpd:2.4

MAINTAINER DevOps Training <[devopstrainingblr@gmail.com>](mailto:devopstrainingblr@gmail.com)

COPY ./index.html /usr/local/apache2/htdocs/ #EXPOSE 80

Step 2: docker build -t vinctechnohttpserver:1.0 .

Step 3: docker run -p 8081:80 httpd (**Note:** If you use EXPOSE 80 in Docker file, while running image no need to mention -p 80)

Once you ran the above command successfully, open the browser and type the localhost:8081, it will display the contents of index.html.

**Note:** If it will not work, use IP address instead of localhost Step 4: docker images

Step 5: Tag the image

docker tag <<Image ID>> vinctechnologies/httpdimage:tag

**Note:** Here vinctechnologies is username/organisation name and httpdimage is the repository name.

To find the image id use the **docker images** command.

Step 6: Publish the image to docker hub docker push vinctechnologies/httpdimage

Text

Description automatically generated

If you see this error you have to login into docker hub with below command.

## docker login

**docker diff <Container ID/Container Name> :** Inspect changes to files or directories on a container’s filesystem

A ---> A file or directory was added D ---> A file or directory was deleted

C ---> A file or directory was changed

**Task: Create the one Dockerfile to install one war on Tomcat server. Step 1:** Create the file name called **Dockerfile** and put the below contents. FROM tomcat:8.0.20-jre8

COPY ./SampleAntProject.war /usr/local/tomcat/webapps/SampleAntProject.war

**Note:** Make sure executing below commands , need to have SampleAntProject.war file and Dockefile in same directory.

Follow below commands.

docker build -t tomcatimage:8.0 . docker run -p 8088:8080 tomcatimage

Once it started open the browser and type http://localhost:8088/SampleAntProject

## # docker save -o ~/Desktop/backup\_vnc\_container.tar vnc\_container:

A picture containing text

Description automatically generated

**# docker load -i ~/Desktop/vnc\_container.tar**



What are the deployment strategies you followed in docker swarm? blue/green technique.

https://container-solutions.com/deployment-strategies/

## DockerHub URL

https://hub.docker.com

Create the Org Create the Repo Create Team

## Errors

1) Sending build context to Docker daemon 3.072kB

Error response from daemon: No build stage in current context

**Cause:** Issue with Dockerfile, something is wrong with Dockerfile syntax.

**Solution:** Check Dockerfile and fix it.

Before the Docker daemon runs the instructions in the Dockerfile, it performs a preliminary validation of the Dockerfile and returns an error if the syntax is incorrect: