**Get docker image from internet. (Default image registry is @ hub.docker.com)**

# docker pull centos:8

8: Pulling from library/centos

8a29a15cefae: Pull complete

Digest: sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31982106ac204f6700

Status: Downloaded newer image for centos:8

**List the available images**

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

centos 8 470671670cac 7 weeks ago 237MB

ubuntu 10.04 e21dbcc7c9de 5 years ago 183MB

$ docker image ls --filter dangling=true

REPOSITORY                 TAG                         IMAGE ID                         CREATED                         SIZE

<none>                 <none>                 4fd34165afe0                 7 days ago                         14.5MB

A dangling image is an image that is no longer tagged, and appears in listings as <none>:<none>. We have also applied "filter" feature here!

You can delete all dangling images on a system with the **docker image prune** command. If you add the -a flag, Docker will also remove all unused images (those not in use by any containers).

List the full length image IDs.

$ docker images --no-trunc

**Searching Docker Hub from CLI**

$ docker search nileshjosh

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

nileshjoshi1983/webserver 0

nileshjoshi1983/pi 0

nileshjoshi2799/test test repo 0

$ docker search centos --filter "is-official=true"

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

centos The official build of CentOS. 6014 [OK]

**Docker search command will by default display 25 lines of result however that limit can be easily extended using "--limit" flag to maximum 100 lines.**

$ docker search busybox --limit 100

**Run the container**

# docker run -t -i centos:8

*<You're already inside container>*

[root@89d53068fe70 /]# uname -a

Linux 89d53068fe70 4.18.0-80.4.2.el8\_0.x86\_64 #1 SMP Fri Jun 14 13:20:24 UTC 2019 x86\_64 x86\_64 x86\_64 GNU/Linux

***Where,***

***-i is short for --interactive. Keep STDIN open even if unattached.***

***-t is short for --tty. Allocates a pseudo terminal that connects your terminal with the container’s STDIN and STDOUT.***

**List running containers**

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

d277542cf8f2 centos:8 "/bin/bash" 26 seconds ago Up 25 seconds friendly\_shaw

**List running as well shutdown containers**

# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

3e655dd18032 centos:8 "/bin/bash" 5 minutes ago Exited (0) 4 minutes ago sharp\_franklin

e11501ae9e8e centos:8 "/bin/bash" 5 minutes ago Exited (0) 5 minutes ago hopeful\_pascal

d277542cf8f2 centos:8 "/bin/bash" 13 minutes ago Exited (0) 5 minutes ago friendly\_shaw

89d53068fe70 centos:8 "/bin/bash" 14 minutes ago Exited (0) 13 minutes ago youthful\_minsky

b04a201351bf centos:8 "/bin/bash" 15 minutes ago Exited (0) 15 minutes ago quirky\_ride

**What's the use of container ID or names?**

Docker engine always provide a container ID and associated friendly name. This is useful to refer back to a specific container using it's ID or name to perform specific operation like start, attach etc. You can use your custom unique name to container.

# docker run -it --name web\_farm1 centos:8

[root@5cb8cc72b659 /]#

# docker start web\_farm1

web\_farm1

# docker attach web\_farm1

[root@5cb8cc72b659 /]#

# docker stop web\_farm1

web\_farm1

# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

5cb8cc72b659 centos:8 "/bin/bash" 19 minutes ago Exited (127) 11 minutes ago web\_farm1

3e655dd18032 centos:8 "/bin/bash" 29 minutes ago Exited (0) 28 minutes ago sharp\_franklin

e11501ae9e8e centos:8 "/bin/bash" 29 minutes ago Exited (0) 29 minutes ago hopeful\_pascal

d277542cf8f2 centos:8 "/bin/bash" 37 minutes ago Exited (0) 30 minutes ago friendly\_shaw

89d53068fe70 centos:8 "/bin/bash" 38 minutes ago Exited (0) 37 minutes ago youthful\_minsky

b04a201351bf centos:8 "/bin/bash" 39 minutes ago Exited (0) 39 minutes ago quirky\_ride

**To delete the image**

# docker rmi e21dbcc7c9de

Untagged: ubuntu:10.04

Untagged: ubuntu@sha256:f6695b2d24dd2e1da0a79fa72459e33505da79939c13ce50e90675c32988ab64

Deleted: sha256:e21dbcc7c9de73a19fc19187e8189bbe43617a08bc44f5a9ab124ed442ace155

Deleted: sha256:f500c3a7dec437bf271921d67a6d240c574a1aa186b7fa211818e7564f255da1

Deleted: sha256:170b376f64fb30995c140276be3d71dfb256b308d86183ca3b22aa93a79ad548

Deleted: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef

NOTE: You will not able to delete the image if that image used to run a container (doesn't matter container running or stopped). Hence to delete such image you must first get rid of container launched using that image.

**# docker rmi busybox**

**Error response from daemon: conflict: unable to remove repository reference "busybox" (must force) - container 4e925c94e87f is using its referenced image 78096d0a5478**

So first you must delete the container:

# docker rm 4e925c94e87f

4e925c94e87f

# docker rmi busybox

Untagged: busybox:latest

Untagged: busybox@sha256:836945da1f3afe2cfff376d379852bbb82e0237cb2925d53a13f53d6e8a8c48c

Deleted: sha256:78096d0a54788961ca68393e5f8038704b97d8af374249dc5c8faec1b8045e42

Deleted: sha256:1079c30efc82c9ad970f01c9a732115b34156fcfd77daf69d65e914bb1de8196

Delete all the images

# docker rmi -f $(docker images -q)

**To launch container and to see list of all container**

# docker container run -it --name os\_general centos:8

# docker container ls

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

3af31ee177fe centos:8 "/bin/bash" 58 seconds ago Up 57 seconds os\_general

**List running or stopped container just with container ID.**

# docker container ls -a -q

**Stop and remove containers**

# docker container stop $(docker container ls -aq)  
# docker container rm $(docker container ls -aq)

**Now let's see if you want to remove/terminate all the containers with single command however without using any loop. Then following command will help.**

**Remove containers in a non-graceful way.**

# docker rm -f $(docker container ls -a -q)

3af31ee177fe

90bda39a5f12

efdd0077a348

5cb8cc72b659

3e655dd18032

e11501ae9e8e

d277542cf8f2

89d53068fe70

B04a201351bf

# docker container ls -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

**Requirement: Launch container, run the program and shutdown or terminate the container once program execution completed.**

# docker run -it --name time\_bound centos:8 date

# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

38e6cbf27349 centos:8 "date" 13 seconds ago Exited (0) 11 seconds ago time\_bound *>>>>>>> here's the container in exited state after running program date.*

c835485fb798 centos:8 "/bin/bash" 14 minutes ago Up 14 minutes os1

# docker run -it --name time\_bound1 --rm centos:8 date

Sat Mar 7 06:28:04 UTC 2020

# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

38e6cbf27349 centos:8 "date" 2 minutes ago Exited (0) 2 minutes ago time\_bound

c835485fb798 centos:8 "/bin/bash" 17 minutes ago Up 16 minutes os1

If you note, you have no entry in above output because we have deleted the container after executing date program.

**Launch container named "web" and check the details of container.**

# docker container run -it --name web centos:8

# docker inspect web

<shows details about status, container name, StartedAt, FinishedAt, Image container booted off from, CPU, Memory, Mounts, Networking and lot more in JSON format>

**To get a specific information from inspect output**

# docker container inspect --format "{{ .NetworkSettings.IPAddress }}" web

172.17.0.2

***(This is JSON parsing, Under inspect output NetworkSettings is an array and key IPAddress is part of that array hence to find out IP address you use a combination of NetworkSettings.IPAddress and then if you look carefully there is a dot prefixed to NetworkSettings, so that dot indicates NetworkSettings is not part of any other array and no parent associated with NetworkSettings)***

**Save and Load Vs Import and Export**

Imagine a scenario where you have built Docker images and containers that you would be interested to keep and share it with your other collaborators or colleagues. The below methods shall help you achieve it. Four basic Docker CLI comes into action:

$ docker --help | grep -E "(export|import|load|save)"

**export Export a container\'s filesystem as a tar archive**

**import Import the contents from a tarball to create a filesystem image**

**load Load an image from a tar archive or STDIN**

**save Save one or more images to a tar archive (streamed to STDOUT by default)**

Let's dig a little deeper into this:

$ cat Dockerfile

FROM busybox

cmd echo $((40 \*2))

Build an image and run a container.

$ docker build -t calc .

Sending build context to Docker daemon 38.15MB

Step 1/2 : FROM busybox

latest: Pulling from library/busybox

d9cbbca60e5f: Pull complete

Digest: sha256:836945da1f3afe2cfff376d379852bbb82e0237cb2925d53a13f53d6e8a8c48c

Status: Downloaded newer image for busybox:latest

---> 78096d0a5478

Step 2/2 : cmd echo $((40 \*2))

---> Running in c77cd58346fd

Removing intermediate container c77cd58346fd

---> 83e3ba62c1fe

Successfully built 83e3ba62c1fe

Successfully tagged calc:latest

$ docker run calc

80

Alright, this image and container works fine for me fine, I send this image to Hrihaan to try.

$ docker save calc > calc.tar

And when Hrihaan imports it and run.

$ docker run calc

**docker: Error response from daemon: No command specified.**

**See 'docker run --help'.**

FAILED!! What just happened here? Well, we saved our image with "docker save" and Hrihhan used "dcoker import" to extract the image. So, this is something incompatible happened here!

Critical **save works with Docker images. It saves everything needed to build a container from scratch (Contains all parent layers, and all tags + versions). Use this command if you want to share an image with others.**

Critical **load works with Docker images. Use this command if you want to run an image exported with save. Unlike pull, which requires connecting to a Docker registry, load can import from anywhere (e.g. a file system, URLs).**

Critical **export works with Docker containers, and it exports a snapshot of the container’s file system. Use this command if you want to share or back up the result of building an image.**

Critical **import works with the file system of an exported container, and it imports it as a Docker image. Use this command if you have an exported file system you want to explore or use as a layer for a new image.**

**Docker Registry**

Registry is place where we keep our Docker images. There are three types of registry:

* 1. Local registry
  2. Public registry
  3. **Private registry (widely used)**

# docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to <https://hub.docker.com> to create one.

Username: nileshjoshi1983

Password:

WARNING! Your password will be stored unencrypted in /root/.docker/config.json.

Configure a credential helper to remove this warning. See

<https://docs.docker.com/engine/reference/commandline/login/#credentials-store>

Login Succeeded

In order to upload/push image to docker hub you must tag image with your official ID.

# docker tag webserver:v1 nileshjoshi1983/webserver:v1

# docker push nileshjoshi1983/webserver:v1

The push refers to repository [docker.io/nileshjoshi1983/webserver]

dd3a8e7665a6: Pushed

0683de282177: Pushing [==================================================>] 244.9MB

0683de282177: Pushed

v1: digest: sha256:5e6ec992b27b781d31e268ce09bfb86b54878ad623d424beee8fc85003f5691f size: 741

**To see terminal logs or console logs of container.**

# docker logs <container\_name>

**To view runtime terminal of container.**

# docker logs -f <container\_name>

**If you want to run command within container from base OS.**

# docker exec <container\_name> ifconfig -a

**Run a specific program within container.**

# docker exec -it <container\_name> bash

**NOTE: If you exit on container prompt then it will not shutdown the container (as default behavior of exit command) but rather it will just kill the process you executed using docker exec command.**

# docker exec -it 9438ead4322c cat /etc/hosts

127.0.0.1 localhost

::1 localhost ip6-localhost ip6-loopback

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

172.17.0.2 9438ead4322c

**Launch container in detach mode. This basically runs the container in background.**

# docker container run -it -d --name <container\_name> centos:8v2

***Where,***

***-d is short for --detach. Run the container in the background. Allows you to use the terminal for other commands while your container runs.***

**Docker provides a very powerful command diff which lists the changes in the files and directories. The changes include addition, deletion and those represented by the A, D and C flags, respectively.**

# docker diff 9438ead4322c

**Copies file(s) from the containers filesystems to the specified path on host system.**

# docker cp 9438ead4322c:/test /var/tmp

Point to Ponder If you want to come out from running container which is attached to your base OS then you can use shortcut key - **Ctrl + p + q** and to go back to terminal use **# docker attach os1**

Point to Ponder SELinux plays a vital role, keep SELinux "Permissive" to have Docker operating seamlessly.

**Myth busters:**

* + Important Data is lost residing inside container…… It's a myth, data inside the container is permanent until you terminate/remove container. And if you use docker volumes then you can keep data persistent even container is terminated.
  + Important You can't run GUI program inside container…… It's a myth, you can run GUI program inside a container.

$ cat .mydochubpasswd | docker login -u nileshjoshi --password-stdin

WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.

Configure a credential helper to remove this warning. See

<https://docs.docker.com/engine/reference/commandline/login/#credentials-store>

Login Succeeded