Lab 2 – Container and Images

June 2019

Document Control

Version	1.0
Created by	Priyanka Gawada
Updated by	Priyanka Gawada
Reviewed by	Chand Ali
Date	June, 2019
Earlier versions	

Table of Contents

1. Lab Environment

Environment will look as follows:

AWS
Linux VM

2. Lab Overview

- In this Lab, you will learn how to start containers based on existing images available on Docker hub. You will also learn how to run containers in various modes such as interactive mode, detached mode, and so forth and lastly how to package and run a custom app using Docker.
- A container is another core concept in Docker. Containers run applications or services, almost always just one per container. Containers run on top of an image. In terms of storage, a container is like another layer on top of the image, but the layer is writable instead of read-only. You can have many containers using the same image. Each will use the same read-only image and have its own writable layer on top. Containers can be used to create images using the commit command, essentially converting the writable layer to a read-only layer in an image.
- There are different ways to use containers. These include:

To run a single task	This could be a shell script or a custom app.
Interactively	This connects you to the container similar to the way you SSH into a remote server.
In the background	For long-running services like websites and databases

3. Lab Objectives

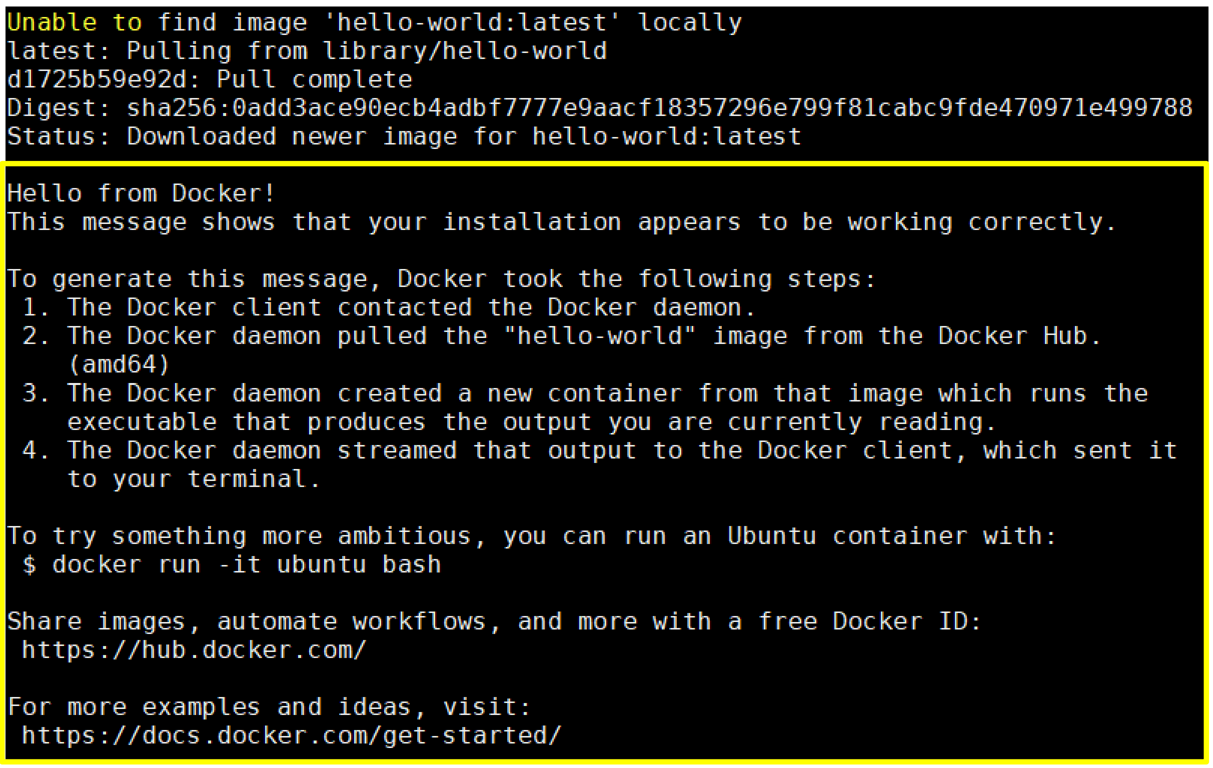
- Run hello-world Container
- Run More Docker Container
- Run an interactive Ubuntu container
- Run a Background Container
- Build a Container from Custom Image
- Cleaning Up Your Docker Containers and Images

4. Run Hello-World Container

4.1 Assignment 1: We will run hello-world container and list them.

- Enter the following command to see how easy it is to get a container running:

```
[root@ip-172-31-13-238 ec2-user]# docker run hello-world
```



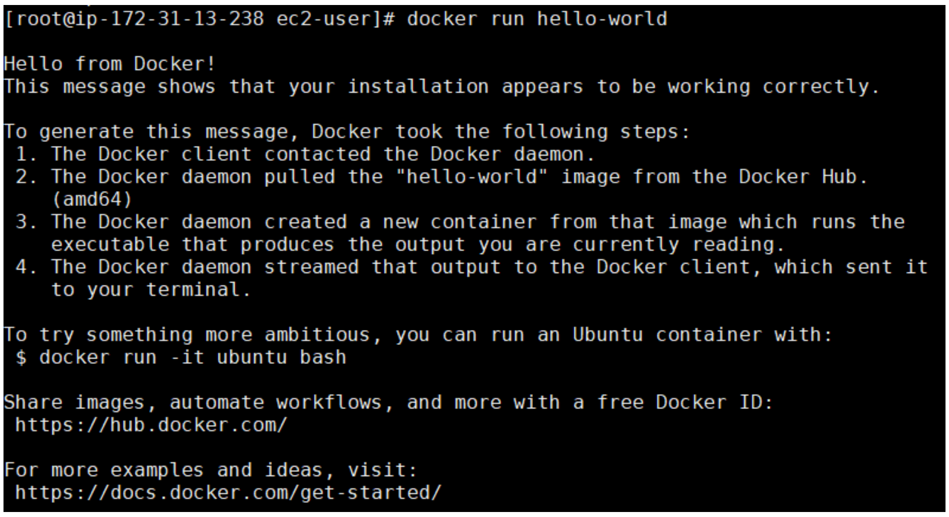
```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

- There are two sections to the output: output from Docker as it prepares to run the container and the output from the command running in the container.
- Take a look at the Docker output first:
  - In the first line, Docker is telling you that it couldn't find the image you specified, hello-world, on the Docker Daemon's local host. The latest portion after the colon (:) is a tag. The tag identifies which version of the image to use. By default, it looks for the latest version.
  - In the next line, it notifies you that it automatically pulled the image. The library/hello-world is the repository it's pulling from inside the Docker Hub registry.
    - *library* is the account name for official Docker images. In general, images will come from repositories identified using the pattern account/repository.
  - The last three lines confirm the pull completed and the image has been downloaded.

Note: The container output is the output that gets written by the command that runs in the container.

- Re-run the same command:

```
[root@ip-172-31-13-238 ec2-user]# docker run hello-world
```

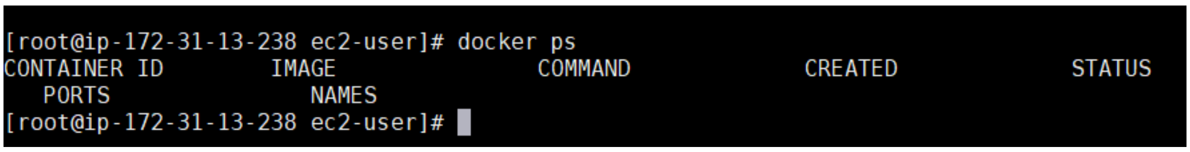


```
[root@ip-172-31-13-238 ec2-user]# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Notice this time the Docker output is not included because the specific version of the image was found locally and there is no need to pull the image again.

- To list all running containers, enter:

```
[root@ip-172-31-13-238 ec2-user]# docker ps
```

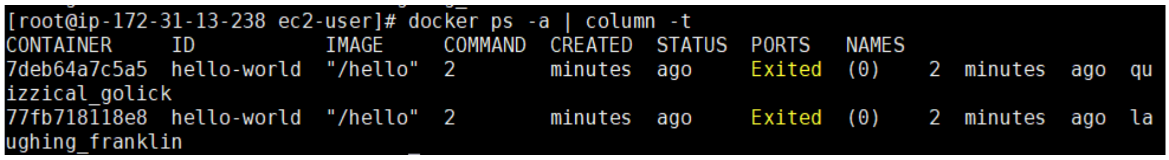


```
[root@ip-172-31-13-238 ec2-user]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
   PORTS               NAMES
[root@ip-172-31-13-238 ec2-user]# 
```

We cannot see anything in the list, as there is no running container.

- Enter the following to see a list of all running and stopped containers:

```
[root@ip-172-31-13-238 ec2-user]# docker ps -a
```

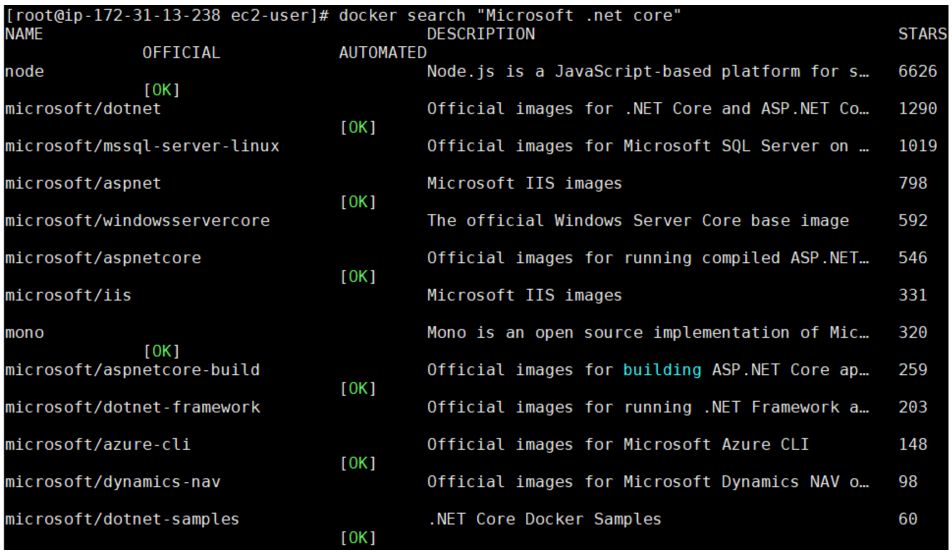


```
[root@ip-172-31-13-238 ec2-user]# docker ps -a | column -t
CONTAINER     ID           IMAGE     COMMAND  CREATED  STATUS  PORTS   NAMES
7deb64a7c5a5  hello-world  "/hello"  2        minutes  ago     Exited  (0)    2  minutes  ago  qu
izzical_golick
77fb718118e8  hello-world  "/hello"  2        minutes  ago     Exited  (0)    2  minutes  ago  la
ughing_franklin
```

This time you can see the two hello-world containers. They simply wrote a message and then stopped when the command finished. Notice, that Docker automatically assigned random friendly names for the hello-world containers, quizzical_golick, and laughing_franklin. These names are useful if you need to reference a container that you didn't assign a name.

- Search for an image whose exact name is not known, say an image for **Microsoft .NET Core**, by entering:

```
[root@ip-172-31-13-238 ec2-user]# docker search "Microsoft .net core"
```



```
[root@ip-172-31-13-238 ec2-user]# docker search "Microsoft .net core"
NAME                                        DESCRIPTION                                     STARS
              OFFICIAL            AUTOMATED
node                                        Node.js is a JavaScript-based platform for s…   6626
              [OK]
microsoft/dotnet                            Official images for .NET Core and ASP.NET Co…   1290
                                  [OK]
microsoft/mssql-server-linux                Official images for Microsoft SQL Server on …   1019

microsoft/aspnet                            Microsoft IIS images                            798
                                  [OK]
microsoft/windowsservercore                 The official Windows Server Core base image     592

microsoft/aspnetcore                        Official images for running compiled ASP.NET…   546
                                  [OK]
microsoft/iis                               Microsoft IIS images                            331

mono                                        Mono is an open source implementation of Mic…   320
              [OK]
microsoft/aspnetcore-build                  Official images for building ASP.NET Core ap…   259
                                  [OK]
microsoft/dotnet-framework                  Official images for running .NET Framework a…   203

microsoft/azure-cli                         Official images for Microsoft Azure CLI         148
                                  [OK]
microsoft/dynamics-nav                      Official images for Microsoft Dynamics NAV o…   98

microsoft/dotnet-samples                    .NET Core Docker Samples                        60
                                  [OK]
```

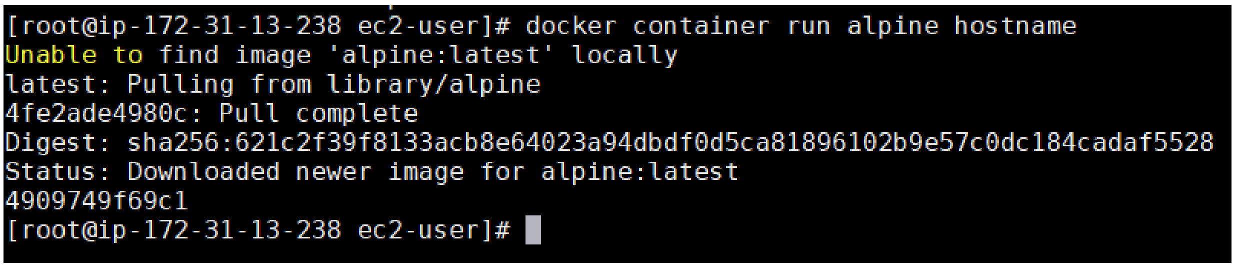
5. Run More Docker Containers

5.1 Assignment 2: Run a single task in an Alpine Linux container

- Start a new container and run the hostname command
  Run the following command in your Linux console.

```
[root@ip-172-31-13-238 ec2-user]# docker container run alpine hostname
```

The output below shows that the alpine:latest image could not be found locally. When this happens, Docker automatically pulls it from Docker Hub.



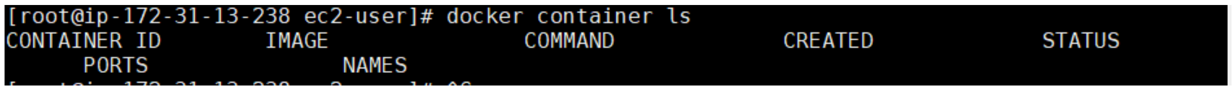
```
[root@ip-172-31-13-238 ec2-user]# docker container run alpine hostname
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
4fe2ade4980c: Pull complete
Digest: sha256:621c2f39f8133acb8e64023a94dbdf0d5ca81896102b9e57c0dc184cadaf5528
Status: Downloaded newer image for alpine:latest
4909749f69c1
[root@ip-172-31-13-238 ec2-user]#
```

- View the details of running container.

  Run the following command in your Linux console.

```
[root@ip-172-31-13-238 ec2-user]# docker container ls
```

The command does not list any existing containers as shown below.



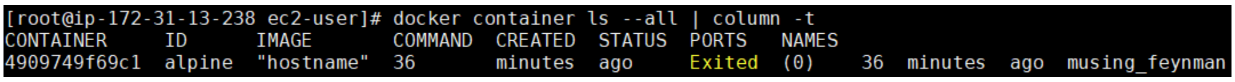
```
[root@ip-172-31-13-238 ec2-user]# docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
      PORTS               NAMES
```

Docker keeps the running instances as long as the process it started inside the container is still running. In the case, **docker container run alpine hostname**

- To view all containers running as well as exited.
  Run the following command in your Linux console.

```
[root@ip-172-31-13-238 ec2-user]# docker container ls –all
```

Here, as shown in the output, the hostname process exits as soon as the output is written. This means the container stops. However, Docker doesn't delete resources by default, so the container still exists in the Exited state.



```
[root@ip-172-31-13-238 ec2-user]# docker container ls --all | column -t
CONTAINER     ID      IMAGE       COMMAND  CREATED  STATUS  PORTS   NAMES
4909749f69c1  alpine  "hostname"  36       minutes  ago     Exited  (0)    36  minutes  ago  musing_feynman
```

6. Run an Interactive Ubuntu Container

6.1 Assignment 3: We want to setup Ubuntu Linux container on top of the Docker host.

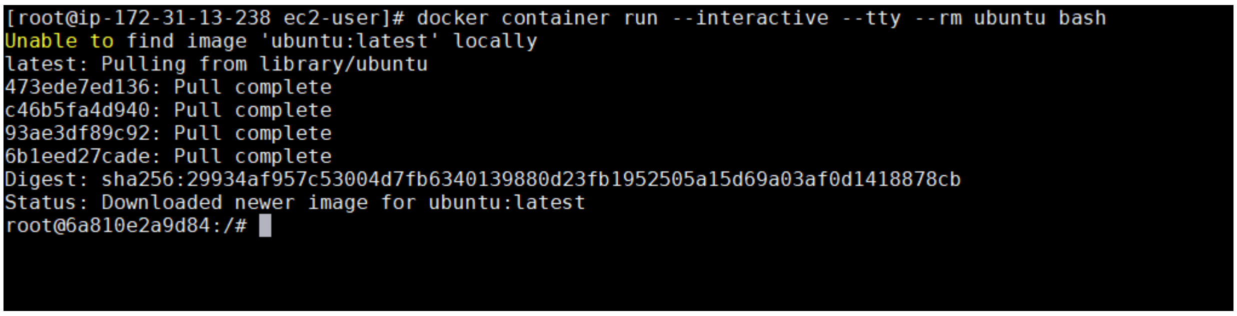
- Run a Docker container and access its shell.

```
[root@ip-172-31-13-238 ec2-user]# docker container run --interactive --tty --rm ubuntu bash
```

In this example, we're giving three parameters of Docker:

- --interactive says you want an interactive session.
- --tty allocates a pseudo-tty.
- --rm tells Docker to go ahead and remove the container when it has completed executing.

We're also telling the container to run bash as its main process (PID 1). When the container starts, you'll drop into the bash shell with the default prompt as shown below.



```
[root@ip-172-31-13-238 ec2-user]# docker container run --interactive --tty --rm ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
473ede7ed136: Pull complete
c46b5fa4d940: Pull complete
93ae3df89c92: Pull complete
6b1eed27cade: Pull complete
Digest: sha256:29934af957c53004d7fb6340139880d23fb1952505a15d69a03af0d1418878cb
Status: Downloaded newer image for ubuntu:latest
root@6a810e2a9d84:/#
```

Docker has attached to the shell in the container, relaying input and output between your local session and the shell session in the container.

- Try few commands in the container.
  List contents of the root directory in the container

```
root@6a810e2a9d84:/# ls
```

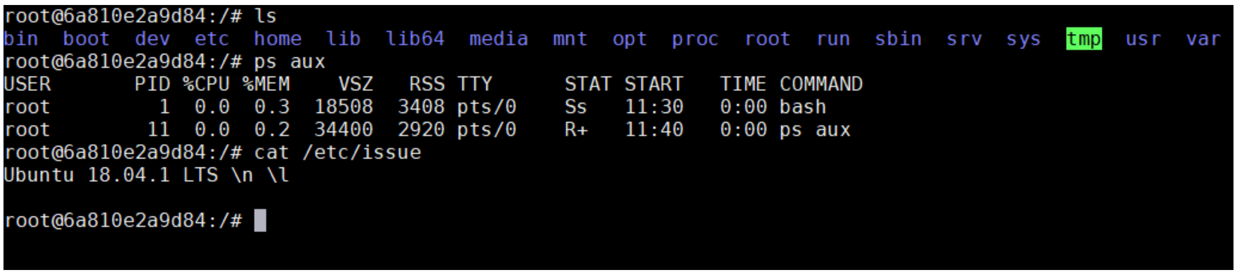
- Show all running processes in the container.

```
root@6a810e2a9d84:/# ps aux
```

- Show which Linux distro the container is running

```
root@6a810e2a9d84:/# cat /etc/issue
```

Here, is the output of all the commands executed within the bash shell of the running container.



```
root@6a810e2a9d84:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@6a810e2a9d84:/# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.3  18508  3408 pts/0    Ss   11:30   0:00 bash
root        11  0.0  0.2  34400  2920 pts/0    R+   11:40   0:00 ps aux
root@6a810e2a9d84:/# cat /etc/issue
Ubuntu 18.04.1 LTS \n \l

root@6a810e2a9d84:/#
```

- Type **exit** to leave the shell session. This will terminate the bash process, causing the container to exit.

```
root@6a810e2a9d84:/# exit
exit
[root@ip-172-31-13-238 ec2-user]#
```

Note: As we used the --rm flag when we started the container, Docker removed the container when it stopped.

So, you won't see the Ubuntu container if you run another docker container ls –all

7. Run a Background Container

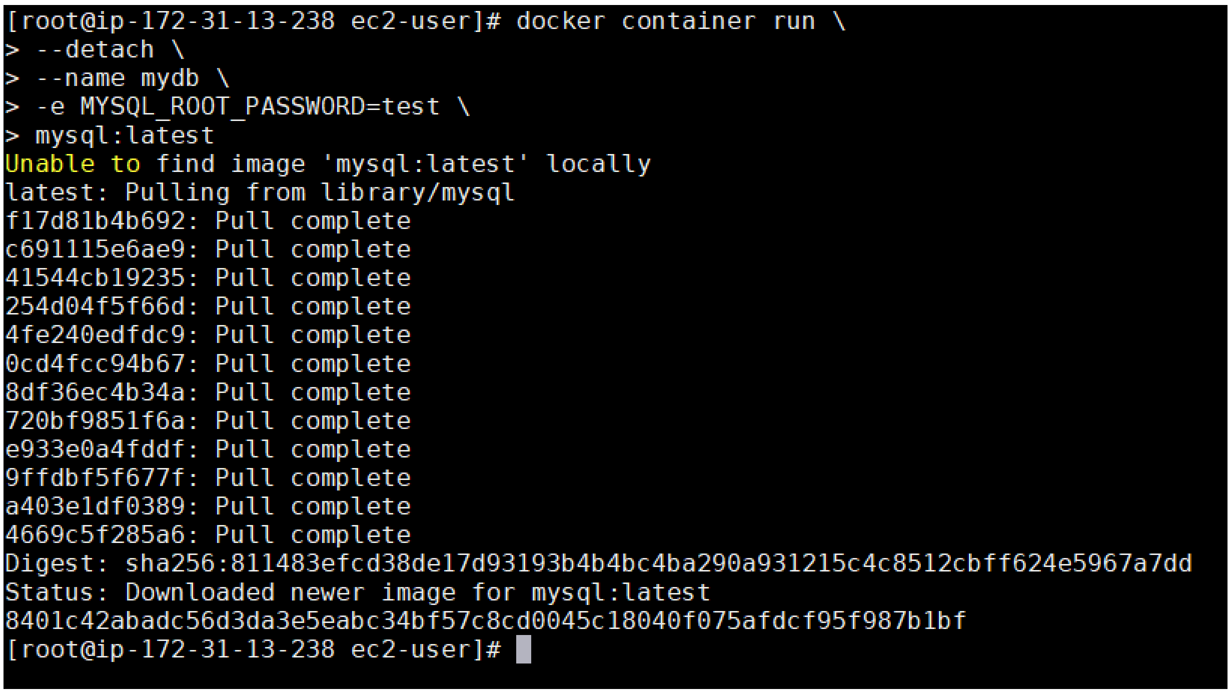
7.1 Assignment 4: We will setup a MySQL container that will keep on running in the detached mode. The detach mode runs the container in the background.

- Run a new MySQL container with the following command.

```
[root@ip-172-31-13-238 ec2-user]# docker container run \
>   --detach \
>   --name mydb \
>   -e MYSQL_ROOT_PASSWORD=test \
>   mysql:latest
```

- --detach will run the container in the background.
- --name will name it mydb.
- -e will use an environment variable to specify the root password (NOTE: This should never be done in production).

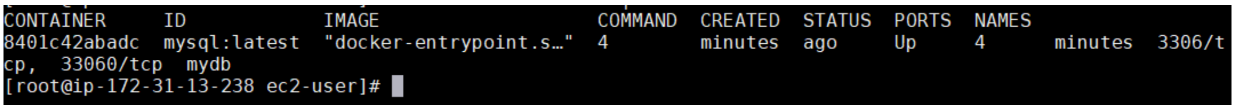
As shown below, the MySQL image was not available locally, Docker automatically pulled it from Docker Hub.



```
[root@ip-172-31-13-238 ec2-user]# docker container run \
> --detach \
> --name mydb \
> -e MYSQL_ROOT_PASSWORD=test \
> mysql:latest
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
f17d81b4b692: Pull complete
c691115e6ae9: Pull complete
41544cb19235: Pull complete
254d04f5f66d: Pull complete
4fe240edfdc9: Pull complete
0cd4fcc94b67: Pull complete
8df36ec4b34a: Pull complete
720bf9851f6a: Pull complete
e933e0a4fddf: Pull complete
9ffdbf5f677f: Pull complete
a403e1df0389: Pull complete
4669c5f285a6: Pull complete
Digest: sha256:811483efcd38de17d93193b4b4bc4ba290a931215c4c8512cbff624e5967a7dd
Status: Downloaded newer image for mysql:latest
8401c42abadc56d3da3e5eabc34bf57c8cd0045c18040f075afdcf95f987b1bf
[root@ip-172-31-13-238 ec2-user]#
```

As long as the MySQL process is running, Docker will keep the container running in the background.

So now, you would see the Ubuntu container if you run another **docker container ls**

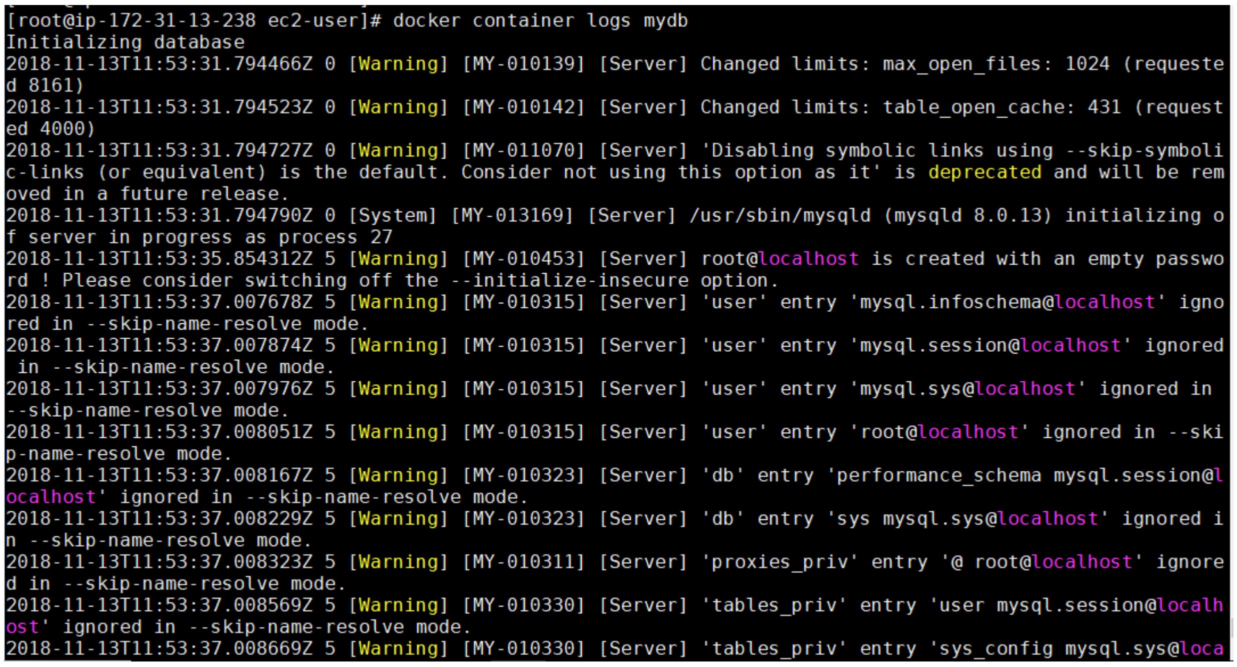


```
CONTAINER     ID            IMAGE                   COMMAND   CREATED   STATUS   PORTS   NAMES
8401c42abadc  mysql:latest  "docker-entrypoint.s…"  4         minutes   ago      Up      4       minutes   3306/t
cp,  33060/tcp  mydb
[root@ip-172-31-13-238 ec2-user]#
```

- While mydb container is running in the background, you want to view logs present at the time of execution.
  Run the following command.

```
[root@ip-172-31-13-238 ec2-user]# docker container logs mydb
```

Here is the output.


```
[root@ip-172-31-13-238 ec2-user]# docker container logs mydb
Initializing database
2018-11-13T11:53:31.794466Z 0 [Warning] [MY-010139] [Server] Changed limits: max_open_files: 1024 (requeste
d 8161)
2018-11-13T11:53:31.794523Z 0 [Warning] [MY-010142] [Server] Changed limits: table_open_cache: 431 (request
ed 4000)
2018-11-13T11:53:31.794727Z 0 [Warning] [MY-011070] [Server] 'Disabling symbolic links using --skip-symboli
c-links (or equivalent) is the default. Consider not using this option as it' is deprecated and will be rem
oved in a future release.
2018-11-13T11:53:31.794790Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 8.0.13) initializing o
f server in progress as process 27
2018-11-13T11:53:35.854312Z 5 [Warning] [MY-010453] [Server] root@localhost is created with an empty passwo
rd ! Please consider switching off the --initialize-insecure option.
2018-11-13T11:53:37.007678Z 5 [Warning] [MY-010315] [Server] 'user' entry 'mysql.infoschema@localhost' igno
red in --skip-name-resolve mode.
2018-11-13T11:53:37.007874Z 5 [Warning] [MY-010315] [Server] 'user' entry 'mysql.session@localhost' ignored
 in --skip-name-resolve mode.
2018-11-13T11:53:37.007976Z 5 [Warning] [MY-010315] [Server] 'user' entry 'mysql.sys@localhost' ignored in
--skip-name-resolve mode.
2018-11-13T11:53:37.008051Z 5 [Warning] [MY-010315] [Server] 'user' entry 'root@localhost' ignored in --ski
p-name-resolve mode.
2018-11-13T11:53:37.008167Z 5 [Warning] [MY-010323] [Server] 'db' entry 'performance_schema mysql.session@l
ocalhost' ignored in --skip-name-resolve mode.
2018-11-13T11:53:37.008229Z 5 [Warning] [MY-010323] [Server] 'db' entry 'sys mysql.sys@localhost' ignored i
n --skip-name-resolve mode.
2018-11-13T11:53:37.008323Z 5 [Warning] [MY-010311] [Server] 'proxies_priv' entry '@ root@localhost' ignore
d in --skip-name-resolve mode.
2018-11-13T11:53:37.008569Z 5 [Warning] [MY-010330] [Server] 'tables_priv' entry 'user mysql.session@localh
ost' ignored in --skip-name-resolve mode.
2018-11-13T11:53:37.008669Z 5 [Warning] [MY-010330] [Server] 'tables_priv' entry 'sys_config mysql.sys@loca
```

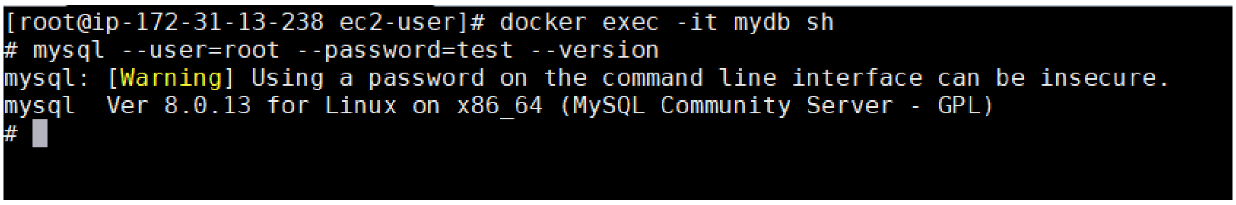
- Connect to detached container.
  Run the following command

```
[root@ip-172-31-13-238 ec2-user]# docker exec -it mydb sh
#
```

The docker container exec connects to a new shell process inside an already-running container. It gives an interactive shell (sh) inside existing MySQL container.

- Now, as we are in MySQL container, let's check the version number by running the following command.

```
[root@ip-172-31-13-238 ec2-user]# docker exec -it mydb sh
# mysql --user=root --password=test --version
```


```
[root@ip-172-31-13-238 ec2-user]# docker exec -it mydb sh
# mysql --user=root --password=test --version
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql  Ver 8.0.13 for Linux on x86_64 (MySQL Community Server - GPL)
#
```

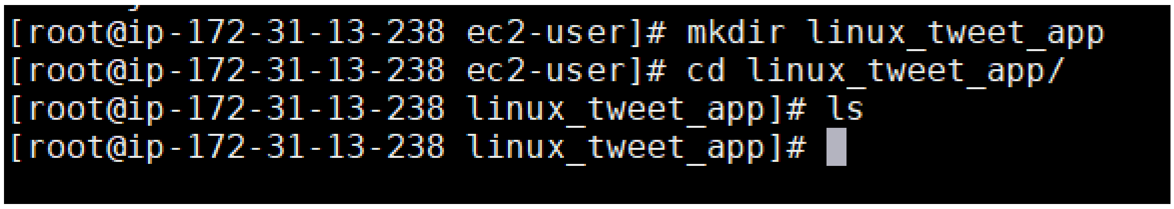
- Type exit to leave the interactive shell session.

```
# exit
[root@ip-172-31-13-238 ec2-user]#
```

8. Build a Container from Custom Image

8.1 Assignment 5: Here, we will create a simple NGINX Website from a Dockerfile.

- Create a new directory named linux_tweet_app directory and open it.



```
[root@ip-172-31-13-238 ec2-user]# mkdir linux_tweet_app
[root@ip-172-31-13-238 ec2-user]# cd linux_tweet_app/
[root@ip-172-31-13-238 linux_tweet_app]# ls
[root@ip-172-31-13-238 linux_tweet_app]#
```

- Create a Dockerfile using the command, **vim Dockerfile** and enter following instructions in it.

```
FROM nginx:latest

COPY index.html /usr/share/nginx/html
COPY linux.png /usr/share/nginx/html

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
```

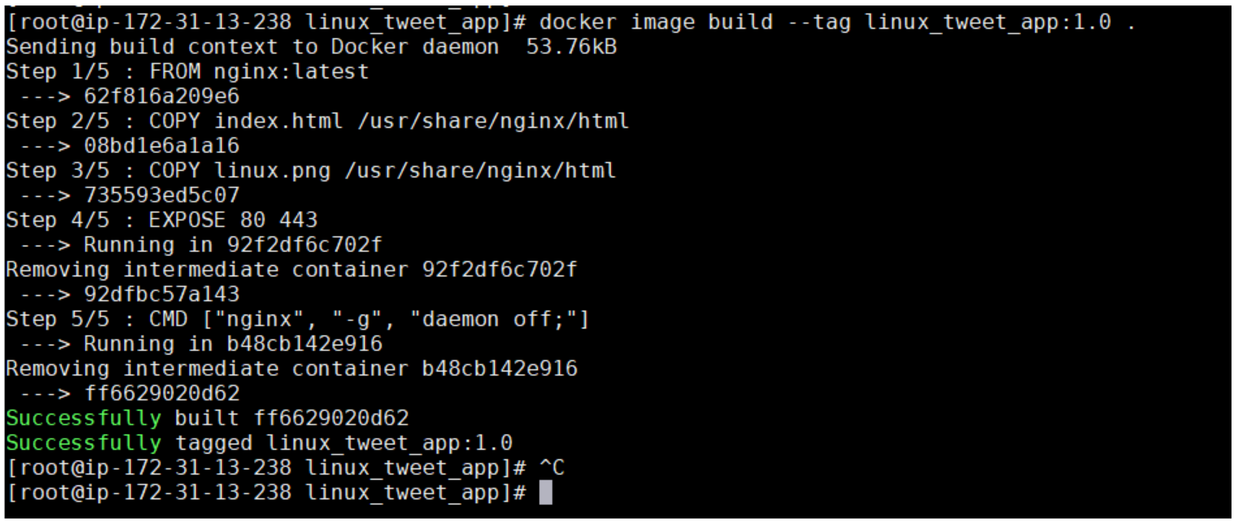
Each of these lines in the Dockerfile does the following:

- **FROM** specifies the base image to use as the starting point for this new image that is created. For this example we're starting from nginx:latest.
- **COPY** copies files from the Docker host into the image, at a known location. In this example, COPY is used to copy two files into the image: index.html. and a graphic that will be used on our webpage.
- **EXPOSE** ports which application will use.
- **CMD** specifies what command to run when a container is started from the image. Notice that we can specify the command, as well as run-time arguments.

- Build an image using the instructions in the Dockerfile.
  Run the following command.

```
[root@ip-172-31-13-238 linux_tweet_app]# docker image build --tag linux_tweet_app:1.0 .
```

- --tag allows us to give the image a custom name.
- . tells Docker to use the current directory as the build context.



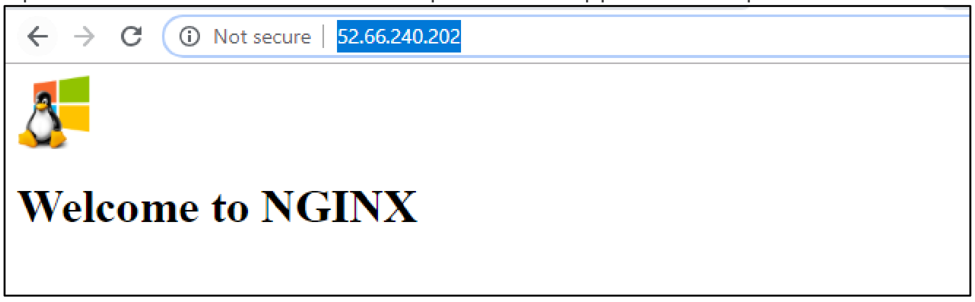
```
[root@ip-172-31-13-238 linux_tweet_app]# docker image build --tag linux_tweet_app:1.0 .
Sending build context to Docker daemon  53.76kB
Step 1/5 : FROM nginx:latest
 ---> 62f816a209e6
Step 2/5 : COPY index.html /usr/share/nginx/html
 ---> 08bd1e6a1a16
Step 3/5 : COPY linux.png /usr/share/nginx/html
 ---> 735593ed5c07
Step 4/5 : EXPOSE 80 443
 ---> Running in 92f2df6c702f
Removing intermediate container 92f2df6c702f
 ---> 92dfbc57a143
Step 5/5 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in b48cb142e916
Removing intermediate container b48cb142e916
 ---> ff6629020d62
Successfully built ff6629020d62
Successfully tagged linux_tweet_app:1.0
[root@ip-172-31-13-238 linux_tweet_app]# ^C
[root@ip-172-31-13-238 linux_tweet_app]#
```

- Use the **docker container run command** to start a new container from the image created in the last step.

```
[root@ip-172-31-13-238 ec2-user]# docker container run \
        --detach \
        --publish 80:80\
        --name tweet \
        linux_tweet_app:1.0
```

As this container will be running an NGINX web server, we'll use the --publish flag to publish port 80 inside the container onto port 80 on the host. This will allow traffic coming in to the Docker host on port 80 to be directed to port 80 in the container.

- Open the Web browser and send request to the application on port 80.



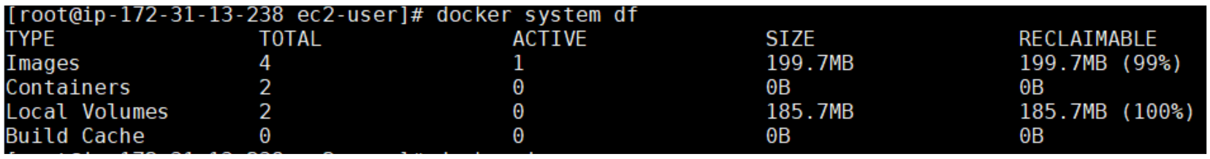
The url will be the IP address of the host machine and port 80. In our case, it is **52.66.240.202:80**

9. Cleaning Up Your Docker Containers and Images

9.1 Assignment 6: We will learn how to reclaim the space from unneeded images and clear out needed space.

- Get an understanding of how much space is being used with the following system management command:

```
[root@ip-172-31-13-238 ec2-user]# docker system df
```

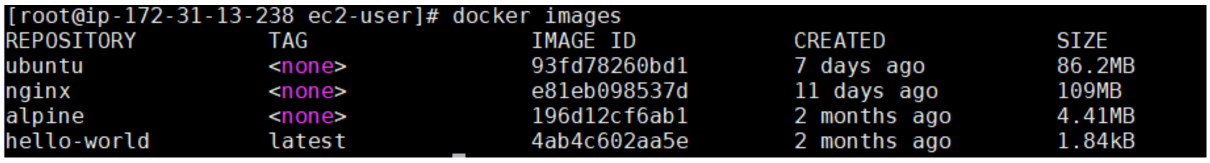


```
[root@ip-172-31-13-238 ec2-user]# docker system df
TYPE                TOTAL               ACTIVE              SIZE                RECLAIMABLE
Images              4                   1                   199.7MB             199.7MB (99%)
Containers          2                   0                   0B                  0B
Local Volumes       2                   0                   185.7MB             185.7MB (100%)
Build Cache         0                   0                   0B                  0B
```

This breaks down the disk usage into images and containers. Images are taking up majority of the space.

- To view images that are taking up most space, enter:

```
[root@ip-172-31-13-238 ec2-user]# docker images
```



```
[root@ip-172-31-13-238 ec2-user]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              <none>              93fd78260bd1        7 days ago          86.2MB
nginx               <none>              e81eb098537d        11 days ago         109MB
alpine              <none>              196d12cf6ab1        2 months ago        4.41MB
hello-world         latest              4ab4c602aa5e        2 months ago        1.84kB
```

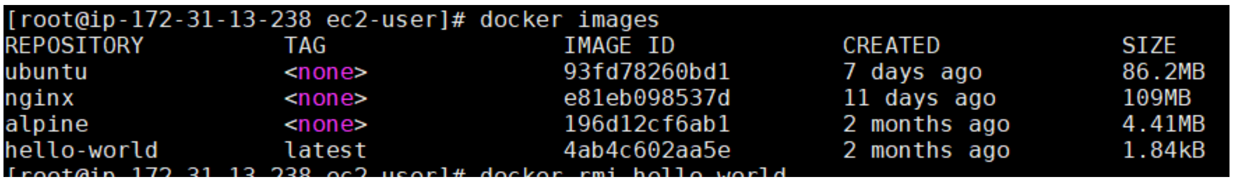
- To remove the stopped web-server container, enter:

```
[root@ip-172-31-13-238 ec2-user]# docker ps –a

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
7deb64a7c5a5        hello-world         "/hello"            37 minutes ago      Exited (0) 37 minutes
ago                 quizzical_golick
77fb718118e8        hello-world         "/hello"            37 minutes ago      Exited (0) 37 minutes
ago                 laughing_franklin

[root@ip-172-31-13-238 ec2-user]# docker rm 7deb6
7deb6
```

The container must be stopped for you to be able to remove it. Confirm it has been removed with docker **ps -a.**

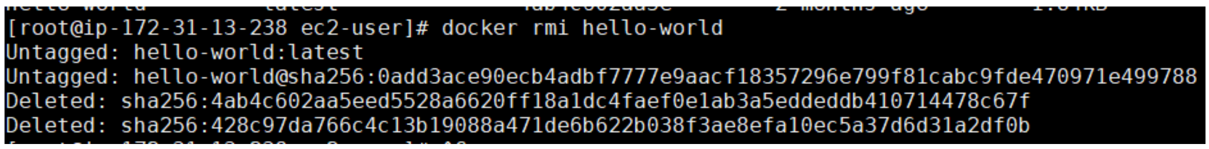
- The image can now be removed when there are no containers depending upon it:

```
[root@ip-172-31-13-238 ec2-user]# docker images
```



```
[root@ip-172-31-13-238 ec2-user]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              <none>              93fd78260bd1        7 days ago          86.2MB
nginx               <none>              e81eb098537d        11 days ago         109MB
alpine              <none>              196d12cf6ab1        2 months ago        4.41MB
hello-world         latest              4ab4c602aa5e        2 months ago        1.84kB
```

```
[root@ip-172-31-13-238 ec2-user]# docker rmi hello-world
```



```
[root@ip-172-31-13-238 ec2-user]# docker rmi hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Deleted: sha256:4ab4c602aa5eed5528a6620ff18a1dc4faef0e1ab3a5eddeddb410714478c67f
Deleted: sha256:428c97da766c4c13b19088a471de6b622b038f3ae8efa10ec5a37d6d31a2df0b
```

Each layer in the image is deleted. If there were image layers shared with other images that were used by containers, Docker would be smart enough to only remove the layers not being used.

10. Conclusion:

Containers are fast and boots quickly as it uses host operating system and shares the relevant libraries. Docker runs processes in isolated containers.