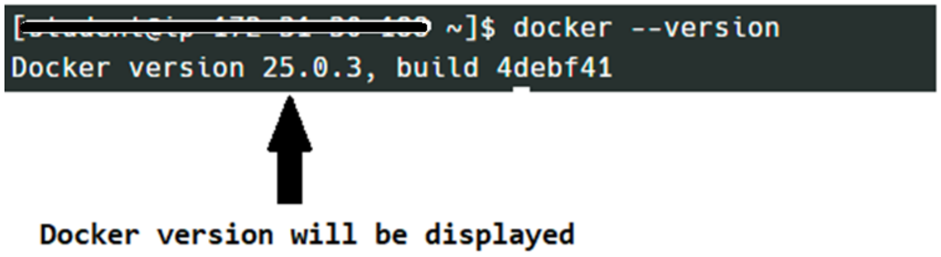
Docker Lab-Manual

Basics of Docker

In this section, you will learn some basics commands that are used in the docker.

1. Checking if the docker is installed and its version

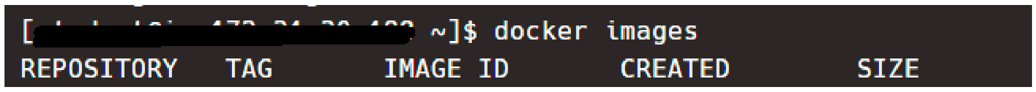
docker --version



```
[[redacted] ~]$ docker --version
Docker version 25.0.3, build 4debf41
```

2. Listing images available locally

docker images



```
[[redacted] ~]$ docker images
REPOSITORY    TAG         IMAGE ID       CREATED         SIZE
```

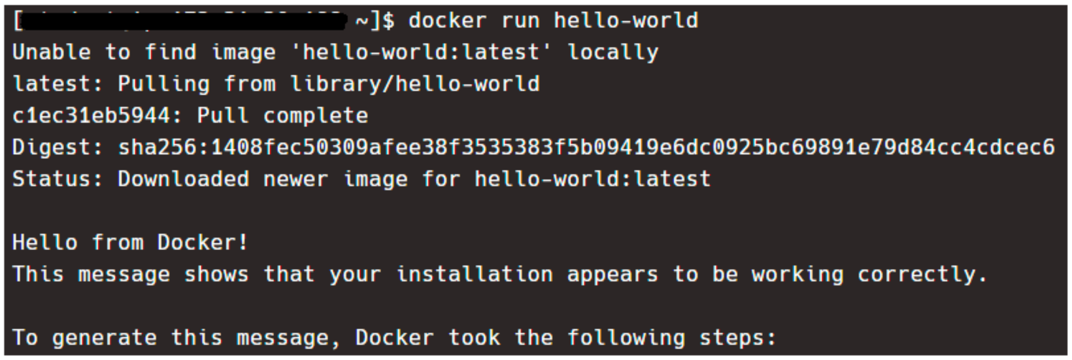
The output will not show any images as we don't have any images locally!

3. Running the first image

Syntax: docker run [image-name]

docker run hello-world

Note:
Here, *hello-world* is the name of the image.

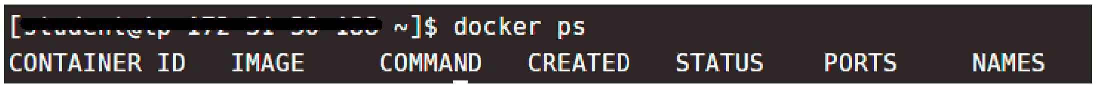


```
[[redacted] ~]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:1408fec50309afee38f3535383f5b09419e6dc0925bc69891e79d84cc4cdcec6
Status: Downloaded newer image for hello-world:latest

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

To generate this message, Docker took the following steps:
```

4. Searching the containers

a. Explore the running containers

docker ps

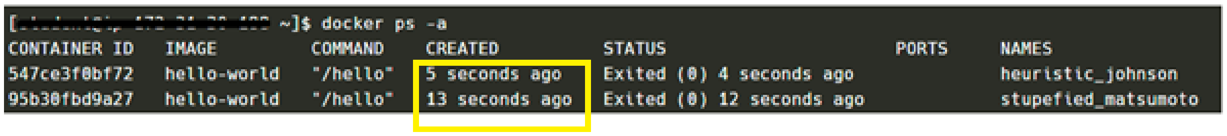


```
[[redacted] ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

The command displays the list of running containers.
We may have some running containers or some stopped containers. The command above will display only the available containers. We might be interested in finding stopped containers as well.

b. Exploring all the containers (running+stopped)

docker ps -a



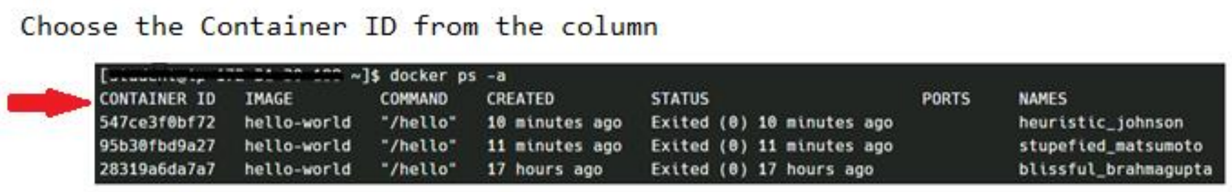
```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
547ce3f0bf72   hello-world   "/hello"   5 seconds ago    Exited (0) 4 seconds ago              heuristic_johnson
95b30fbd9a27   hello-world   "/hello"   13 seconds ago   Exited (0) 12 seconds ago             stupefied_matsumoto
```

Observe the "CREATED" column where it shows when the container is created.

Note:
The number of containers on your system depends on how many times you have executed '*run'* command. The container *IDs* and also *NAMES* will vary. The CREATED time also varies as per when the command had on the system.

5. Container Filtering (with ID or NAME)

We first need ID of the container which might be running or exited. Let us run the command *docker ps -a*



Choose the Container ID from the column

```
[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                     PORTS     NAMES
547ce3f0bf72   hello-world   "/hello"   10 minutes ago   Exited (0) 10 minutes ago            heuristic_johnson
95b30fbd9a27   hello-world   "/hello"   11 minutes ago   Exited (0) 11 minutes ago            stupefied_matsumoto
28319a6da7a7   hello-world   "/hello"   17 hours ago     Exited (0) 17 hours ago              blissful_brahmagupta
```

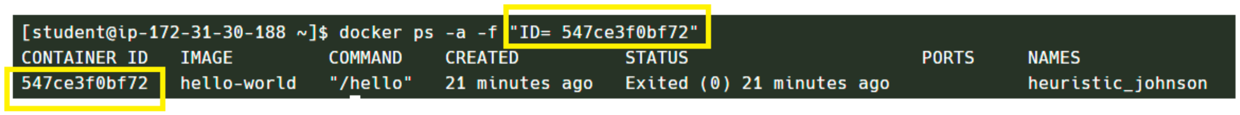
From the output, we will choose one of the ID.

a. Filter by ID

Let's use *547ce3f0bf72* from the above output. (Your ID may be different. Please use the one specified on your output)

docker ps –a –f "ID=547ce3f0bf72"

The above command will generate the following output:

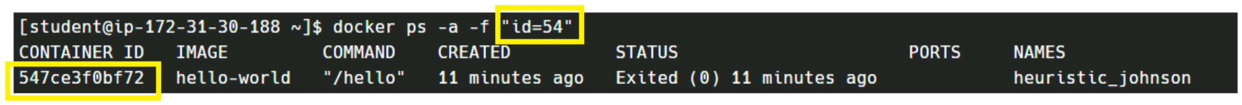


```
[student@ip-172-31-30-188 ~]$ docker ps -a -f "ID= 547ce3f0bf72"
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
547ce3f0bf72   hello-world   "/hello"   21 minutes ago   Exited (0) 21 minutes ago             heuristic_johnson
```

As you can observe, the ID's are very lengthy. Instead of that, we can also use the first two characters as well, as shown below:

docker ps –a –f "ID=54"

Observe that you will get the same output similar to earlier command as shown below:



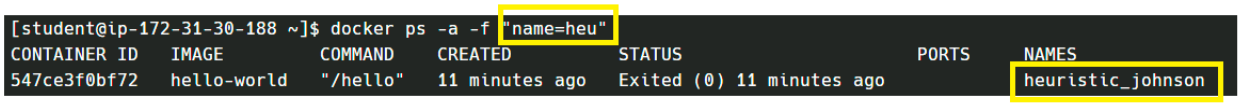
```
[student@ip-172-31-30-188 ~]$ docker ps -a -f "id=54"
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
547ce3f0bf72   hello-world   "/hello"   11 minutes ago   Exited (0) 11 minutes ago             heuristic_johnson
```

b. Filter by NAME

From *docker ps –a* command you will get the name of container. Observe the 'NAMES' column. And, use it in the command. E.g., *heuristic_johnson* or *heu* as a short hand as shown in the image above,

Syntax: docker ps –a –f "name=container name" *docker ps –a –f "name=heu"*

This will generate the following output:

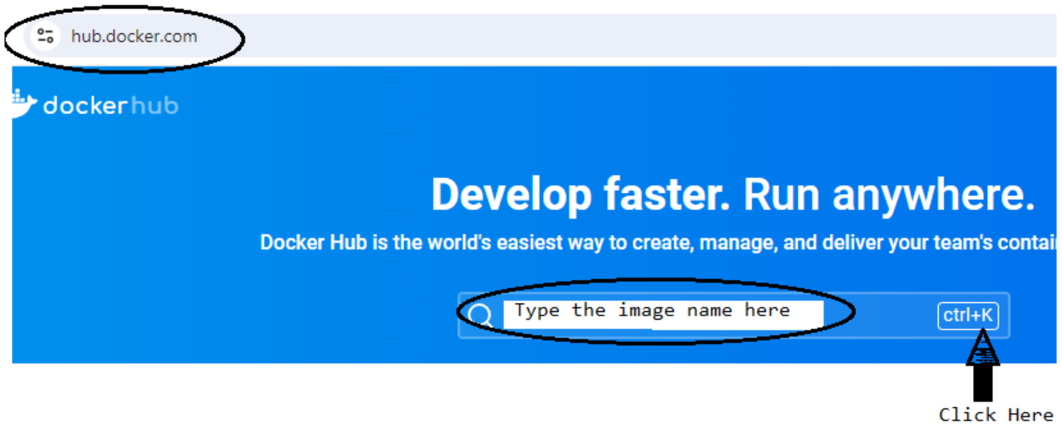


```
[student@ip-172-31-30-188 ~]$ docker ps -a -f "name=heu"
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
547ce3f0bf72   hello-world   "/hello"   11 minutes ago   Exited (0) 11 minutes ago             heuristic_johnson
```

The obvious question, from where are we getting images?

6. Exploring Docker hub

Visit the https://hub.docker.com/ site.


hub.docker.com
dockerhub
Develop faster. Run anywhere.
Docker Hub is the world's easiest way to create, manage, and deliver your team's contai
Type the image name here
ctrl+K
Click Here

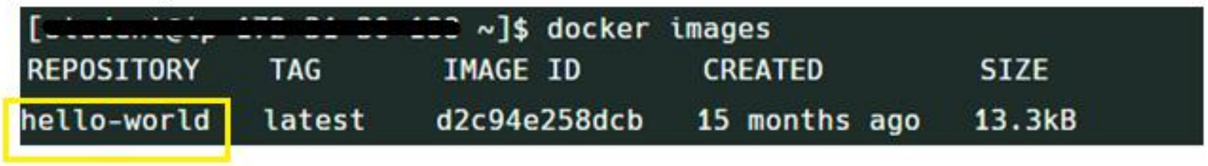
In the Search Docker Hub type the image name you are interested in.

Find the details about following images

1. hello-world
2. busybox
3. nginx

7. Pulling the images from the Docker Hub

Before you pull the images from the docker hub, ensure if the image is locally available. We know that the command *docker images* generates output as shown in image:

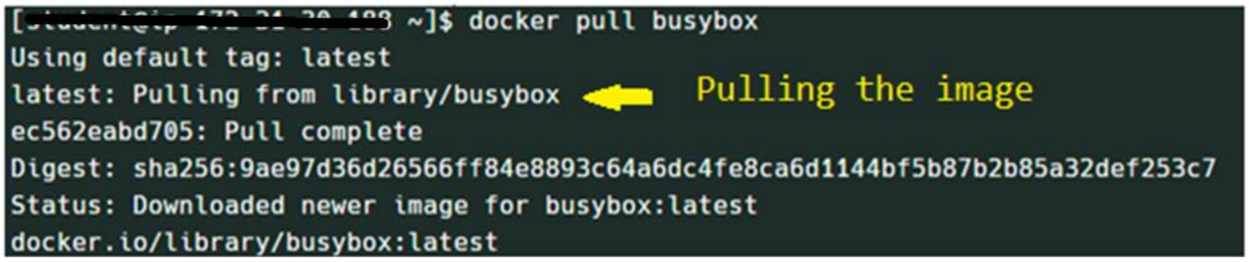


```
[[redacted] ~]$ docker images
REPOSITORY     TAG        IMAGE ID        CREATED          SIZE
hello-world    latest     d2c94e258dcb    15 months ago    13.3kB
```

It is evident that, we have only hello-world image available locally. Now let’s use the pull command to pull the busybox image as:

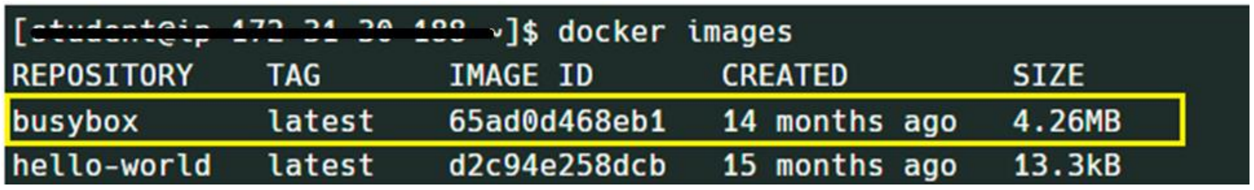
Syntax: docker pull image-name

docker pull busybox



```
[[redacted] ~]$ docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
ec562eabd705: Pull complete
Digest: sha256:9ae97d36d26566ff84e8893c64a6dc4fe8ca6d1144bf5b87b2b85a32def253c7
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
```

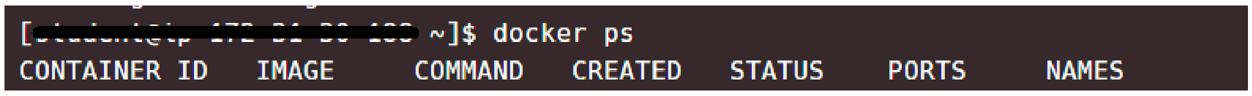
To confirm whether the pull is successful or not, again run the command *docker images*, which should generate following output:



```
[[redacted] ~]$ docker images
REPOSITORY     TAG        IMAGE ID        CREATED          SIZE
busybox        latest     65ad0d468eb1    14 months ago    4.26MB
hello-world    latest     d2c94e258dcb    15 months ago    13.3kB
```

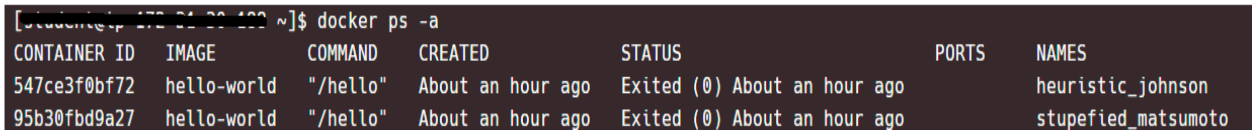
The above output confirms that the image is locally available now. To check, if the container for this image is running use the command:

docker ps



```
[[redacted] ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND    CREATED    STATUS    PORTS     NAMES
```

It might be a possibility that the container has created and stopped as well. Let’s find about all the containers using *docker ps –a* as,



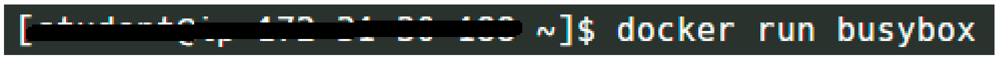
```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED            STATUS                       PORTS     NAMES
547ce3f0bf72   hello-world   "/hello"   About an hour ago  Exited (0) About an hour ago           heuristic_johnson
95b30fbd9a27   hello-world   "/hello"   About an hour ago  Exited (0) About an hour ago           stupefied_matsumoto
```

Note:
Don't worry about how many containers are listed or what's their IDs and NAMEs.

The output shows, neither the container is running or even exited. Actually, the container is not created. When we execute 'pull' command, it only pulls the image and stores it on the system.

Let's create container from pulled image.

Syntax: docker run name-of-image
docker run busybox

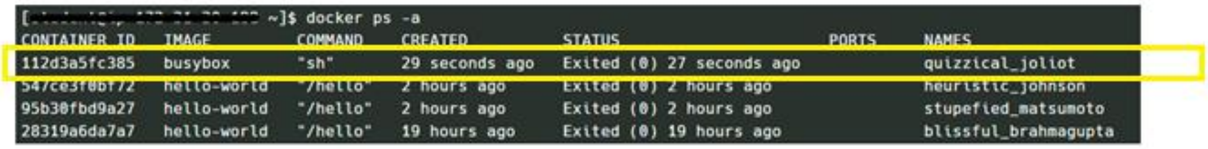


```
[[redacted] ~]$ docker run busybox
```

Surprisingly, nothing would have happened on your system at this point of time.

Did the command execute? Did we get the container from the image? Is it running or not?

Let's cross check using, '*docker ps –a*' command as it will show both running as well as stopped containers,



```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                    PORTS     NAMES
112d3a5fc385   busybox       "sh"       29 seconds ago   Exited (0) 27 seconds ago           quizzical_joliot
547ce3f0bf72   hello-world   "/hello"   2 hours ago      Exited (0) 2 hours ago              heuristic_johnson
95b30fbd9a27   hello-world   "/hello"   2 hours ago      Exited (0) 2 hours ago              stupefied_matsumoto
28319a6da7a7   hello-world   "/hello"   19 hours ago     Exited (0) 19 hours ago             blissful_brahmagupta
```

The above output shows that the container is created with ID as '*112d3a5fc385*'.

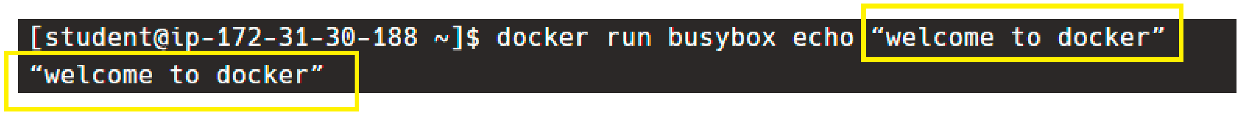
Note:
The ID and NAME may differ on your system. Pay attention to the IMAGE column where it shows "*busybox*"

8. Executing the command in the container

Syntax: docker run image-name command-to-execute

Use following command to run the *busybox* as:

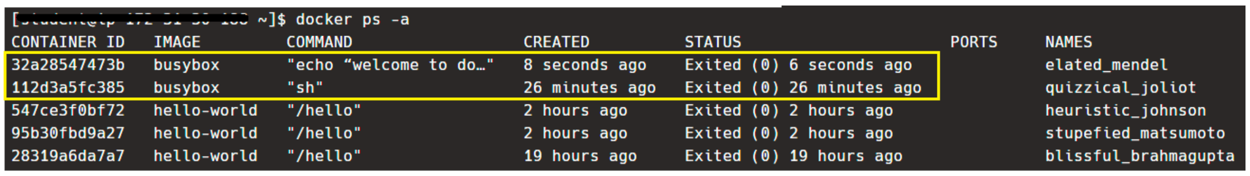
docker run busybox echo "welcome to docker"



```
[student@ip-172-31-30-188 ~]$ docker run busybox echo "welcome to docker"
"welcome to docker"
```

The above output shows, the run command has created a container, and executed echo command in it. Try to execute the same command one more time.

How many containers we created as we ran the *docker run* command twice?



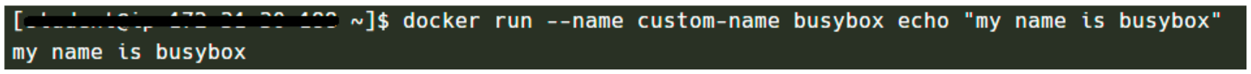
```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                   CREATED          STATUS                       PORTS     NAMES
32a28547473b   busybox       "echo "welcome to do…"    8 seconds ago    Exited (0) 6 seconds ago               elated_mendel
112d3a5fc385   busybox       "sh"                      26 minutes ago   Exited (0) 26 minutes ago              quizzical_joliot
547ce3f0bf72   hello-world   "/hello"                  2 hours ago      Exited (0) 2 hours ago                 heuristic_johnson
95b30fbd9a27   hello-world   "/hello"                  2 hours ago      Exited (0) 2 hours ago                 stupefied_matsumoto
28319a6da7a7   hello-world   "/hello"                  19 hours ago     Exited (0) 19 hours ago                blissful_brahmagupta
```

9. Running the container with names

We can also set the custom name to the container by providing *--name* attribute in the docker run command.

Syntax: docker run --name name-of-container name-of-image command-to-execute

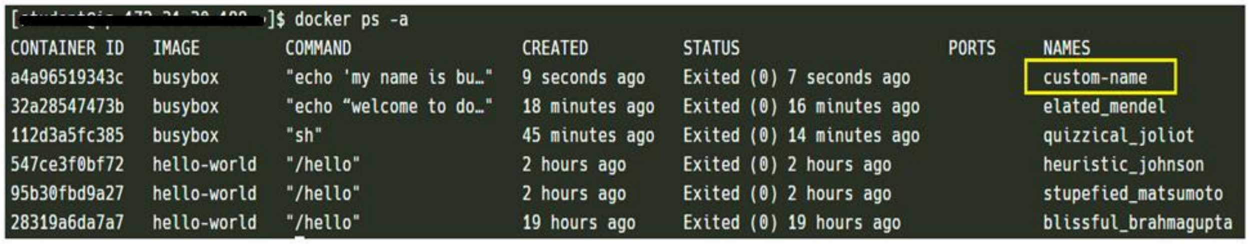
docker run --name custom-name busybox echo "my name is busy box"



```
[[redacted] ~]$ docker run --name custom-name busybox echo "my name is busybox"
my name is busybox
```

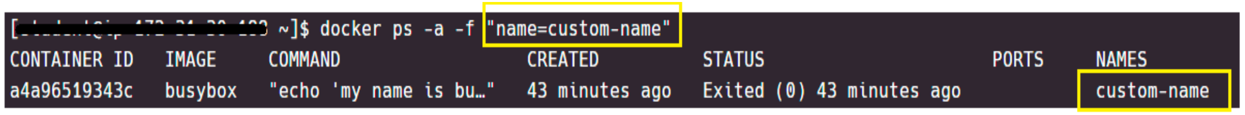
You might have observed that nothing changed as a part of output even if we added the *--name* attribute.

Let’s execute the *docker ps –a* command and observe the name of the container.



```
[[redacted]]$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS                      PORTS     NAMES
a4a96519343c   busybox       "echo 'my name is bu…"   9 seconds ago    Exited (0) 7 seconds ago              custom-name
32a28547473b   busybox       "echo “welcome to do…"   18 minutes ago   Exited (0) 16 minutes ago             elated_mendel
112d3a5fc385   busybox       "sh"                     45 minutes ago   Exited (0) 14 minutes ago             quizzical_joliot
547ce3f0bf72   hello-world   "/hello"                 2 hours ago      Exited (0) 2 hours ago                heuristic_johnson
95b30fbd9a27   hello-world   "/hello"                 2 hours ago      Exited (0) 2 hours ago                stupefied_matsumoto
28319a6da7a7   hello-world   "/hello"                 19 hours ago     Exited (0) 19 hours ago               blissful_brahmagupta
```

We can use it to find description and many other commands like:
Syntax: docker ps –a –f “name=container-name”
(Replace container-name with actual NAME of the container assigned)

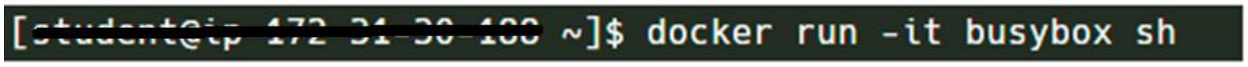


```
[[redacted] ~]$ docker ps -a -f "name=custom-name"
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                       PORTS     NAMES
a4a96519343c   busybox   "echo 'my name is bu…"   43 minutes ago   Exited (0) 43 minutes ago              custom-name
```

10. Interacting with the container

Syntax: docker run –interactive -tty name-of-image
where,
-**interactive (or -i)**: This option keeps the standard input (stdin) of the container open so you can interact with it. It's often used to keep the terminal session alive.
-**tty (or -t)**: This option allocates a pseudo-TTY (teletypewriter) to the container, which provides an interactive terminal session. It's commonly used with -i to provide a fully interactive experience.

docker run -it busybox sh

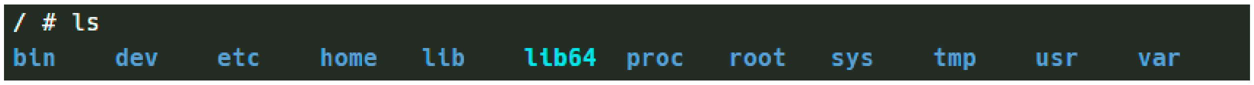


```
[[redacted] ~]$ docker run -it busybox sh
```

This will drop us into the *sh shell* to perform some operations inside a *busybox* system.

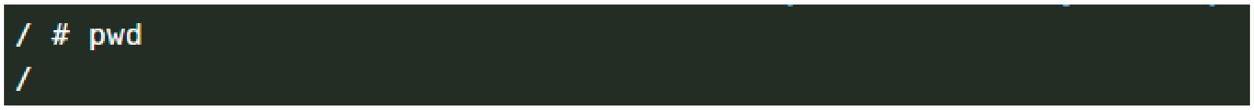
Let us play with it, by some commands such as ls, pwd, cd etc.

Type ls



```
/ # ls
bin     dev     etc     home    lib     lib64   proc    root    sys     tmp     usr     var
```

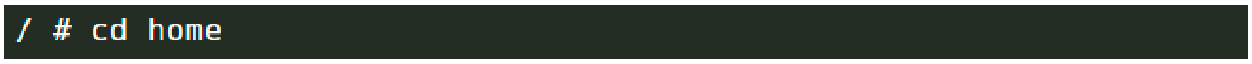
Type pwd



```
/ # pwd
/
```

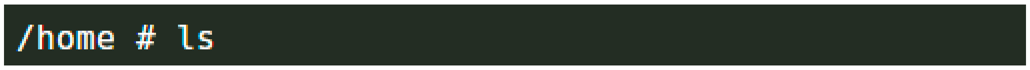
You can change directory using cd as,

Type cd home



```
/ # cd home
```

Try to list directories in home,
Type ls



```
/home # ls
```

There are no directories

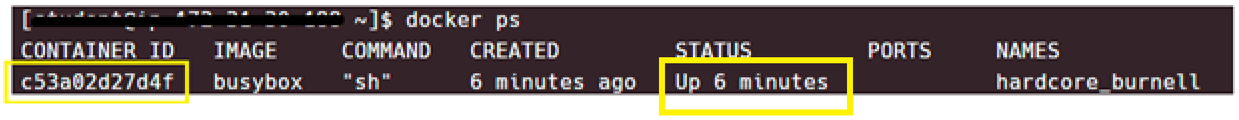
Try few other commands like navigating to root dir, or again at same location where we started '/'.

Once you are done with playing in the container, just open a new shell window and find out the list of running containers using *docker ps* command. We will get list of running containers.

Now you have a running container, and you want to access it after it is started. The run command will start new container, but we want to access the existing one.

Syntax: docker exec –it container-id command

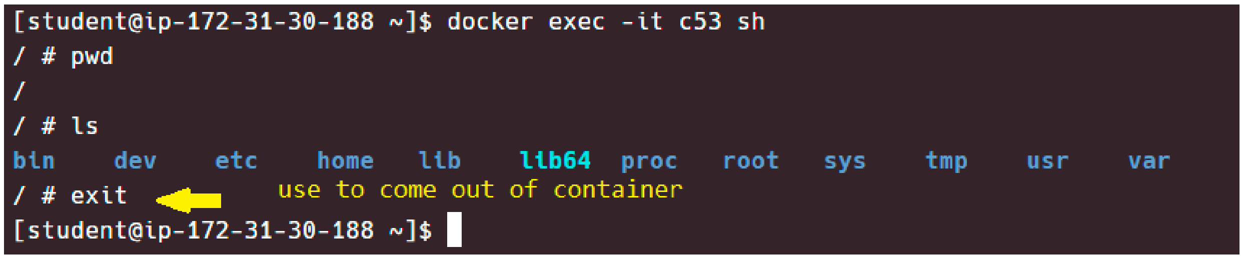
First, find the running container id by using *docker ps*



```
[[redacted] ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
c53a02d27d4f   busybox   "sh"      6 minutes ago   Up 6 minutes             hardcore_burnell
```

Now as we have ID, let's access it's shell as,

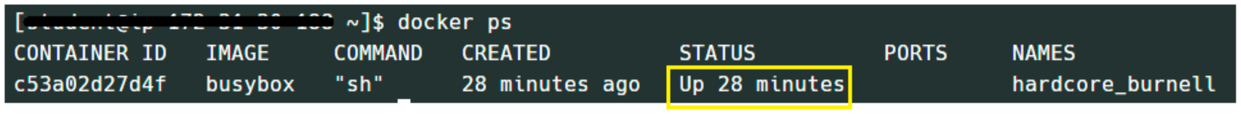
docker exec -it c53 sh



```
[student@ip-172-31-30-188 ~]$ docker exec -it c53 sh
/ # pwd
/
/ # ls
bin    dev    etc    home   lib    lib64  proc   root   sys    tmp    usr    var
/ # exit                use to come out of container
[student@ip-172-31-30-188 ~]$
```

Try to find current location, list of directories etc. commands. To come out of shell type '*exit*'

We come out of container shell. Does '*exit*', stopped the running container? Let's find what happed to our container using *docker ps* command,



```
[[redacted] ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
c53a02d27d4f   busybox   "sh"      28 minutes ago   Up 28 minutes             hardcore_burnell
```

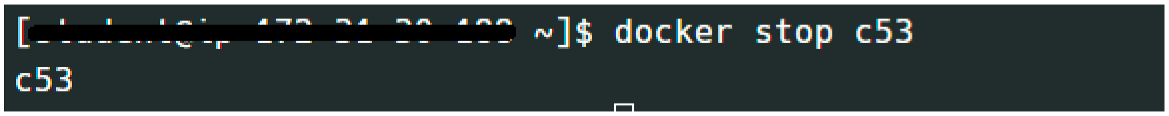
The container is still running. **Great!**

Stopping the container:

We can stop the running container by using the *stop* command:

Syntax: docker stop container-id

docker stop container-id-to-stop



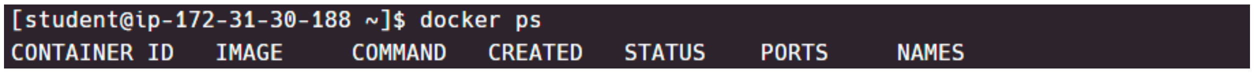
```
[[redacted] ~]$ docker stop c53
c53
```

Start the stopped container:

The way we can stop container we can also start it by using the *start* command:

Syntax: docker start container-id

first let's find the running containers by using the docker ps command



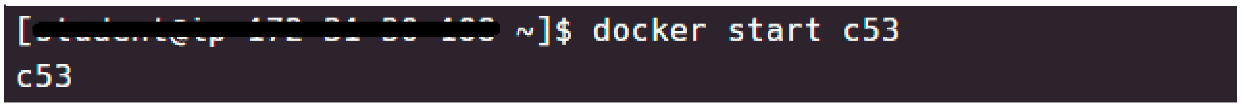
```
[student@ip-172-31-30-188 ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

As you can observe, there are no running containers

We know, we have container with id as c53, which we stopped in earlier command.

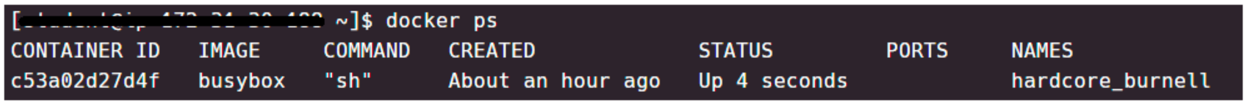
If you have the container id, please first execute the '*docker ps -a*' command.

Let's restart it by using *docker start* command:



```
[[redacted] ~]$ docker start c53
c53
```

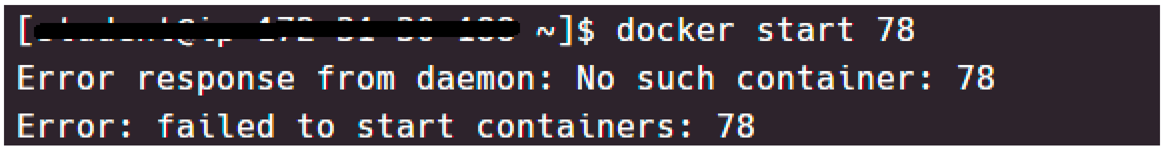
Now confirm is it actually running or not, using *docker ps* command



```
[[redacted] ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED             STATUS         PORTS     NAMES
c53a02d27d4f   busybox   "sh"      About an hour ago   Up 4 seconds             hardcore_burnell
```

Yes! The container is running

What will happen if we give a wrong id or non-existing id? We will get the below error:

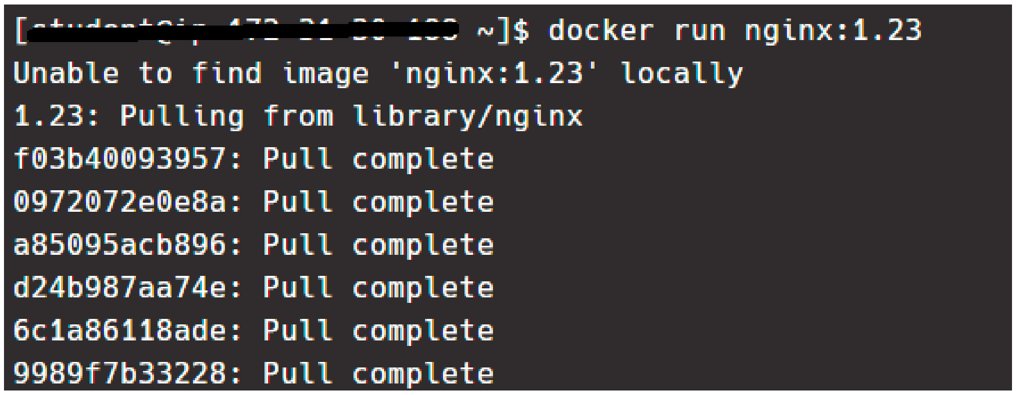


```
[[redacted] ~]$ docker start 78
Error response from daemon: No such container: 78
Error: failed to start containers: 78
```

11. Working with web Servers and Exposing ports of servers

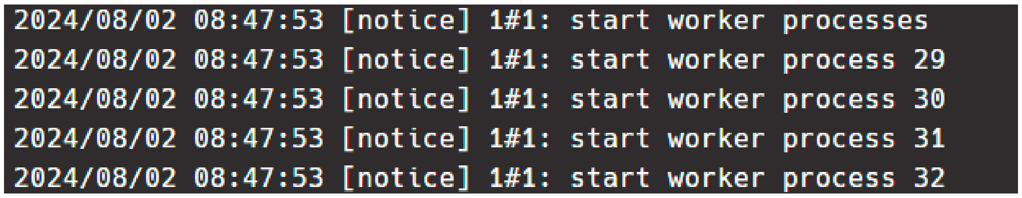
Working with web servers

docker run nginx:1.23



```
[[redacted] ~]$ docker run nginx:1.23
Unable to find image 'nginx:1.23' locally
1.23: Pulling from library/nginx
f03b40093957: Pull complete
0972072e0e8a: Pull complete
a85095acb896: Pull complete
d24b987aa74e: Pull complete
6c1a86118ade: Pull complete
9989f7b33228: Pull complete
```

At the end of the command we will get,



```
2024/08/02 08:47:53 [notice] 1#1: start worker processes
2024/08/02 08:47:53 [notice] 1#1: start worker process 29
2024/08/02 08:47:53 [notice] 1#1: start worker process 30
2024/08/02 08:47:53 [notice] 1#1: start worker process 31
2024/08/02 08:47:53 [notice] 1#1: start worker process 32
```

This depicts that the server has started.

Open a new shell and execute *docker ps* command.

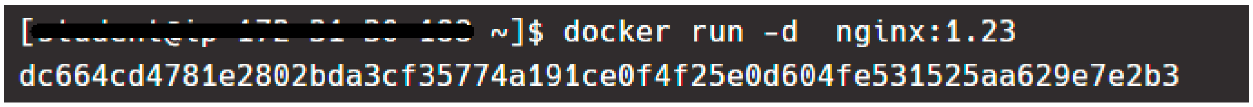
Observe the running container for ngnix.

Now the server is running and we can stop it by typing 'exit'.

We can run the server which will run in background using '-d'.

Syntax: docker run -d name-of-image

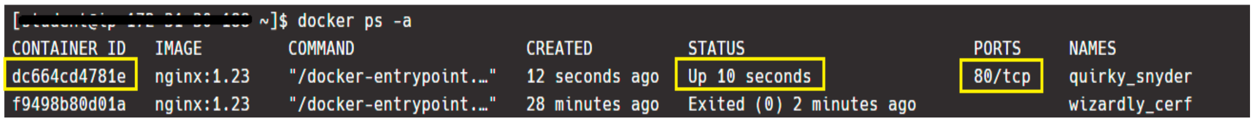
We will execute *docker run -d nginx:1.23*



```
[[redacted] ~]$ docker run -d  nginx:1.23
dc664cd4781e2802bda3cf35774a191ce0f4f25e0d604fe531525aa629e7e2b3
```

If you want to make sure container is running use the command:

docker ps –a



```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS                       PORTS     NAMES
dc664cd4781e   nginx:1.23   "/docker-entrypoint.…"   12 seconds ago   Up 10 seconds                80/tcp    quirky_snyder
f9498b80d01a   nginx:1.23   "/docker-entrypoint.…"   28 minutes ago   Exited (0) 2 minutes ago               wizardly_cerf
```

Try accessing your running server from the browser as http://server-IP:80

We will get 404 error

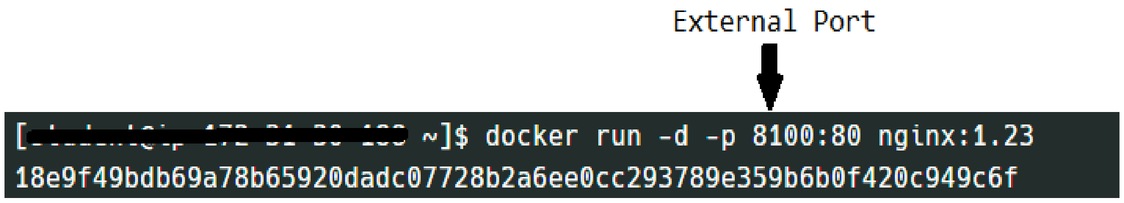
Exposing port

We can expose the port of the running container by adding the *-p* attribute in the *docker run*.

Syntax: docker run -p external-port: server-port

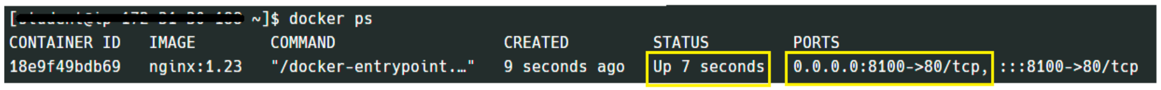
Command:

docker run -d -p 8100:80 nginx:1.23



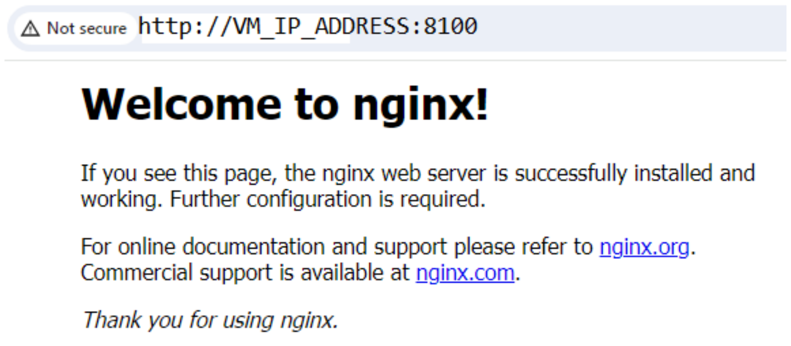
```
[[redacted] ~]$ docker run -d -p 8100:80 nginx:1.23
18e9f49bdb69a78b65920dadc07728b2a6ee0cc293789e359b6b0f420c949c6f
```

Cross check the container is running or not using *docker ps* command:



```
[[redacted] ~]$ docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED         STATUS         PORTS
18e9f49bdb69   nginx:1.23   "/docker-entrypoint.…"   9 seconds ago   Up 7 seconds   0.0.0.0:8100->80/tcp, :::8100->80/tcp
```

Go to browser, and access the URL http://VM_IP_ADDRESS:8100 we will get home page as,



Welcome to nginx!

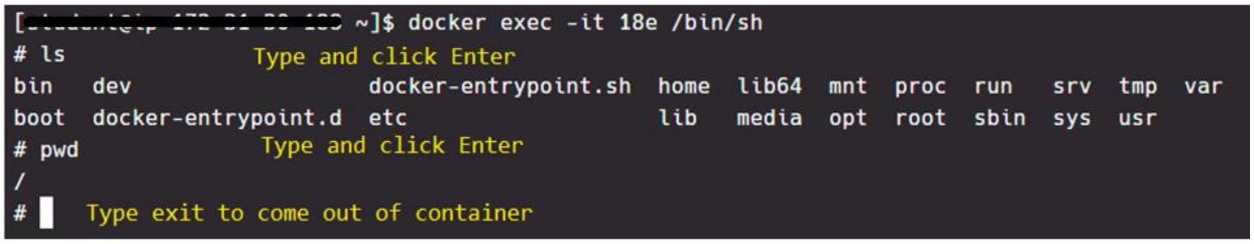
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Note:
Use your own VM_IP_Address to access the page in your VM.

As our container is running, we use ‘exec’ to access it and perform some operations.



```
[[redacted] ~]$ docker exec -it 18e /bin/sh
# ls                  Type and click Enter
bin   dev                    docker-entrypoint.sh  home  lib64  mnt  proc  run   srv  tmp  var
boot  docker-entrypoint.d    etc                   lib   media  opt  root  sbin  sys  usr
# pwd                  Type and click Enter
/
#    Type exit to come out of container
```

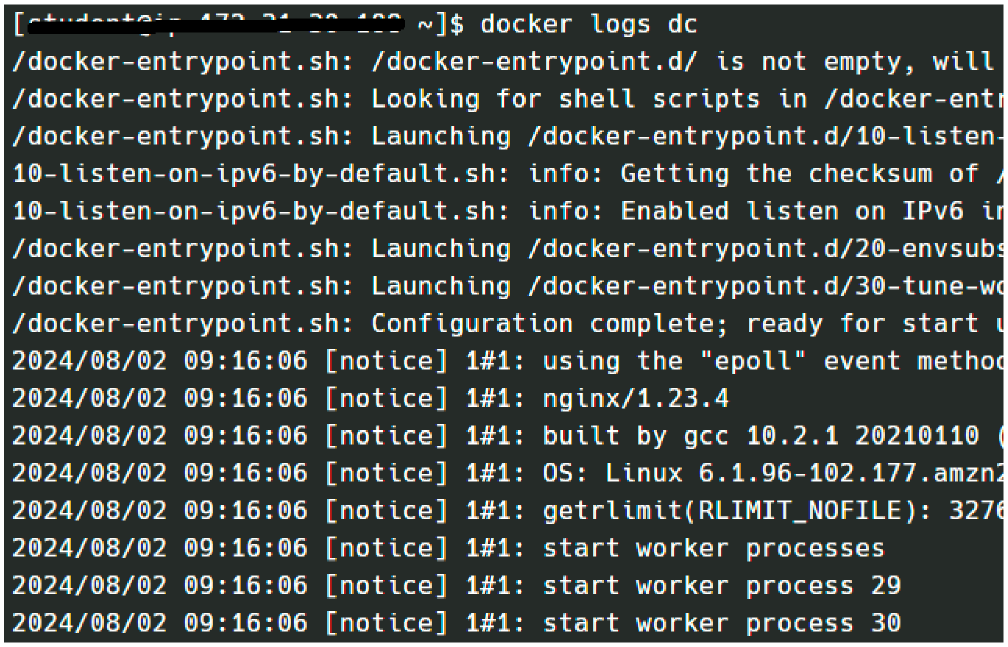
12. Inspecting the container

Reading the Logs

Way 1:

Syntax: docker logs container-id

As the container id of our *nginx* starts from *dc*, we can use the command, *docker log dc*:



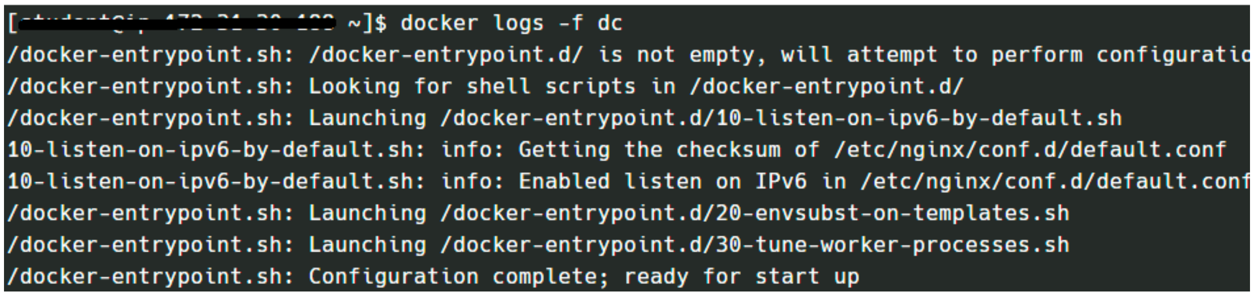
```
[[redacted] ~]$ docker logs dc
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will
/docker-entrypoint.sh: Looking for shell scripts in /docker-entr
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubs
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-wo
/docker-entrypoint.sh: Configuration complete; ready for start u
2024/08/02 09:16:06 [notice] 1#1: using the "epoll" event method
2024/08/02 09:16:06 [notice] 1#1: nginx/1.23.4
2024/08/02 09:16:06 [notice] 1#1: built by gcc 10.2.1 20210110 (
2024/08/02 09:16:06 [notice] 1#1: OS: Linux 6.1.96-102.177.amzn2
2024/08/02 09:16:06 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 3276
2024/08/02 09:16:06 [notice] 1#1: start worker processes
2024/08/02 09:16:06 [notice] 1#1: start worker process 29
2024/08/02 09:16:06 [notice] 1#1: start worker process 30
```

Way 2:

Getting real time logs using -f as,

Syntax: docker logs -f id-of-container

In our case we use the command, *docker logs -f dc*



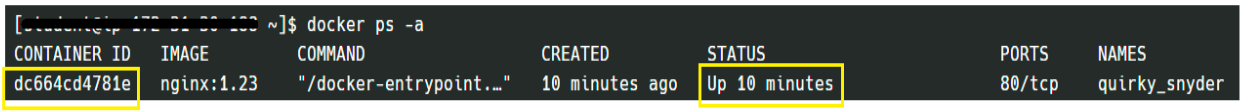
```
[[redacted] ~]$ docker logs -f dc
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuratio
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

The logs keep on displaying as time progress.

We can type *ctrl+c* to stop logs displaying on the shell. We didn’t stop the server it is still running.

Finding the container status

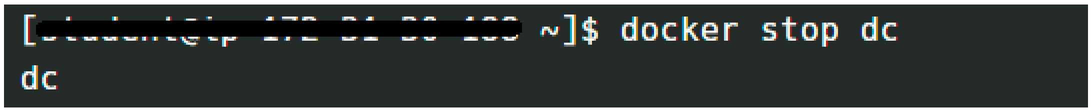
We typed *ctrl+c* to stop logs displaying on the shell. We didn’t stop the server. It is still running and we track it by using *docker ps* or *docker ps –a*



```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS          PORTS     NAMES
dc664cd4781e   nginx:1.23   "/docker-entrypoint.…"   10 minutes ago   Up 10 minutes   80/tcp    quirky_snyder
```

You can stop it using stop command as,

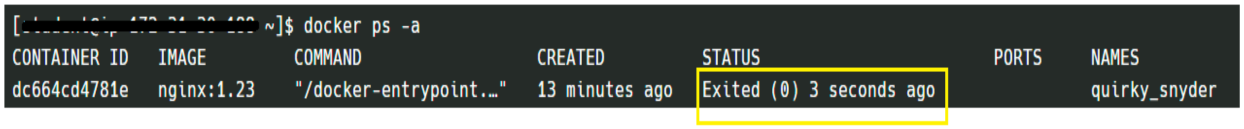
docker stop dc



```
[[redacted] ~]$ docker stop dc
dc
```

Check the status by using the command,

docker ps –a



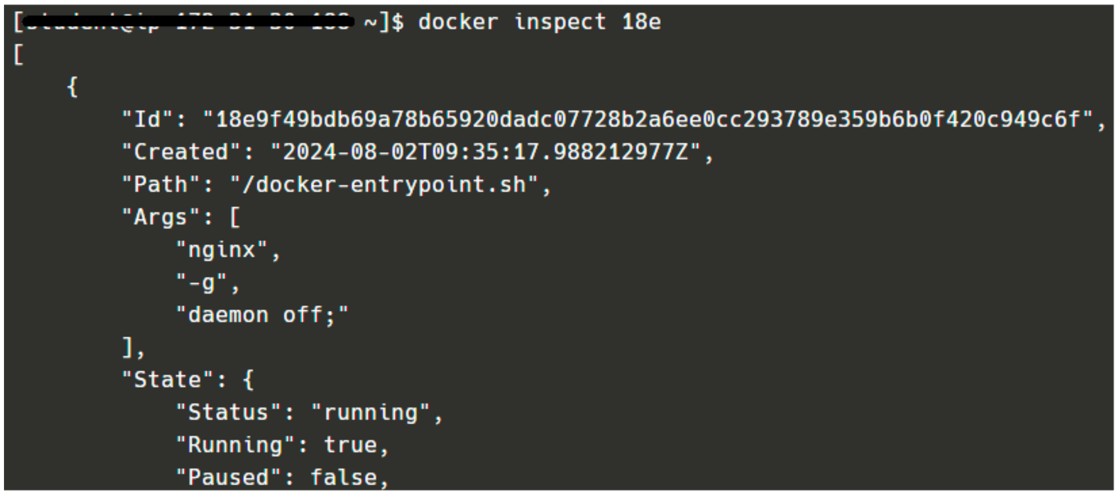
```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS                    PORTS     NAMES
dc664cd4781e   nginx:1.23   "/docker-entrypoint.…"   13 minutes ago   Exited (0) 3 seconds ago            quirky_snyder
```

The container is stopped.

Finding details using inspect command

To inspect the container, we can use docker inspect command

Syntax: docker inspect container-id



```
[[redacted] ~]$ docker inspect 18e
[
    {
        "Id": "18e9f49bdb69a78b65920dadc07728b2a6ee0cc293789e359b6b0f420c949c6f",
        "Created": "2024-08-02T09:35:17.988212977Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
```

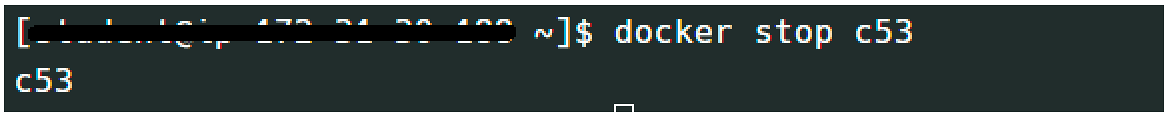
13. Stopping containers

Stopping individual containers

We can stop the running container by using the *stop* command:

Syntax: docker stop container-id

docker stop container-id-to-stop

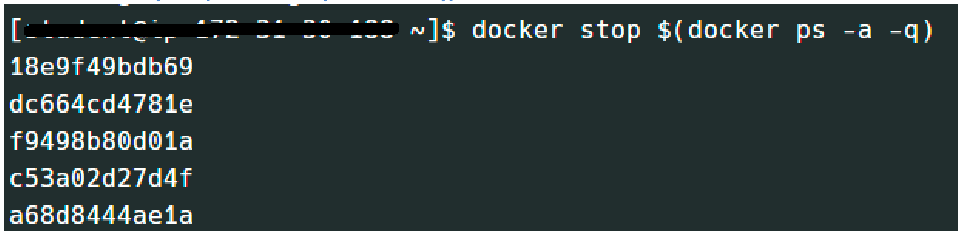


```
[[redacted] ~]$ docker stop c53
c53
```

Stopping all the running containers

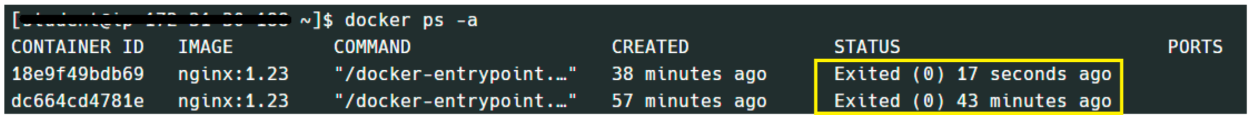
To stop all the running containers, use the command:

docker stop $(docker ps –a –q)



```
[[redacted] ~]$ docker stop $(docker ps -a -q)
18e9f49bdb69
dc664cd4781e
f9498b80d01a
c53a02d27d4f
a68d8444ae1a
```

Now, let's confirm whether the containers are stopped or not. To do this, use the command *docker ps -a* as shown below:



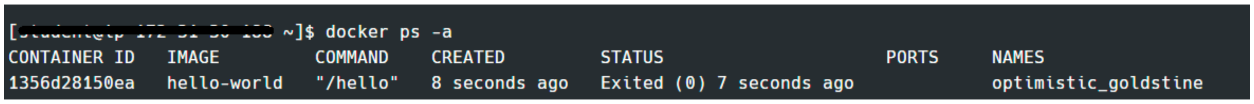
```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS                      PORTS
18e9f49bdb69   nginx:1.23   "/docker-entrypoint.…"   38 minutes ago   Exited (0) 17 seconds ago
dc664cd4781e   nginx:1.23   "/docker-entrypoint.…"   57 minutes ago   Exited (0) 43 minutes ago
```

Observe the STATUS, all running containers have stopped.

14. Removing containers

Removing individual container

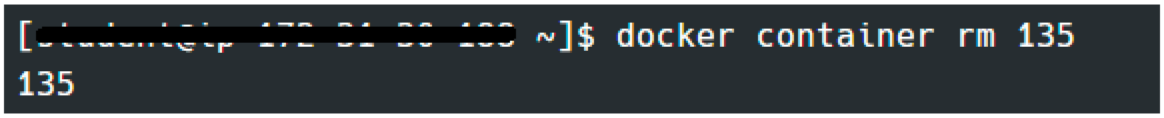
To find all containers first use *docker ps -a* command as shown below. This is important to know the ID of the container.



```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED         STATUS                     PORTS     NAMES
1356d28150ea   hello-world   "/hello"   8 seconds ago   Exited (0) 7 seconds ago             optimistic_goldstine
```

Now we have container ID (In my case, it is 135 ….). Let's remove it by giving the command docker container rm,

Syntax: docker container rm container-id
Use the command: *docker container rm 135*

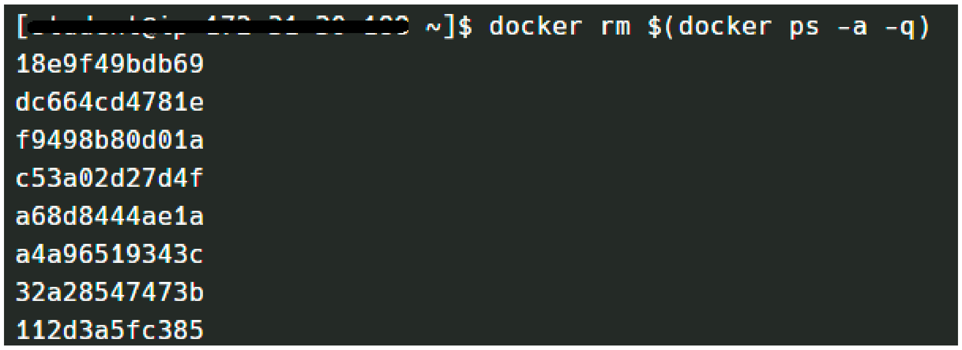


```
[[redacted] ~]$ docker container rm 135
135
```

You can verify, if the container has been removed or not using the command *docker ps -a* command.

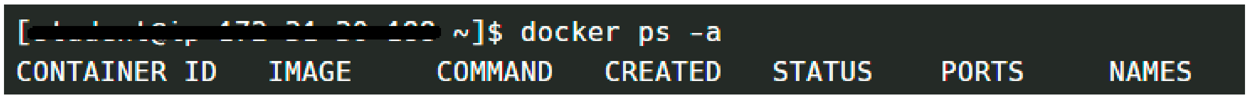
Remove all containers

To remove all the stopped containers, use the command:
docker rm $(docker ps -a -q)



```
[[redacted] ~]$ docker rm $(docker ps -a -q)
18e9f49bdb69
dc664cd4781e
f9498b80d01a
c53a02d27d4f
a68d8444ae1a
a4a96519343c
32a28547473b
112d3a5fc385
```

If we give *docker ps -a*, we will get all the available containers as shown below:



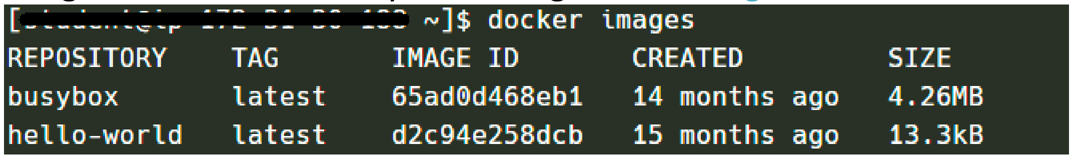
```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

The list is empty, which means now all the created containers have been removed.

15. Removing images

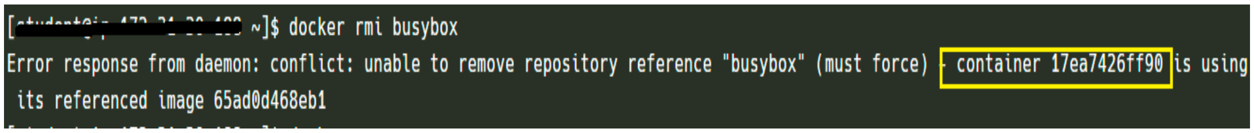
Remove individual image

To remove individual image, we can use image NAME. Let us first, find out images available on our system using docker images.



```
[[redacted] ~]$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
busybox       latest    65ad0d468eb1   14 months ago   4.26MB
hello-world   latest    d2c94e258dcb   15 months ago   13.3kB
```

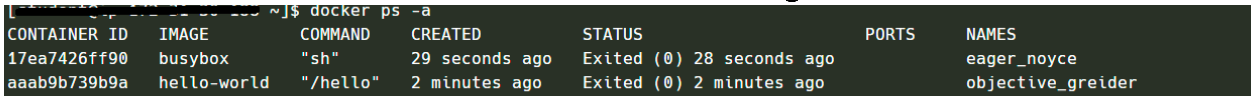
Now, let us remove busybox image using docker rmi image-name as a command.



```
[[redacted] ~]$ docker rmi busybox
Error response from daemon: conflict: unable to remove repository reference "busybox" (must force) - container 17ea7426ff90 is using
 its referenced image 65ad0d468eb1
```

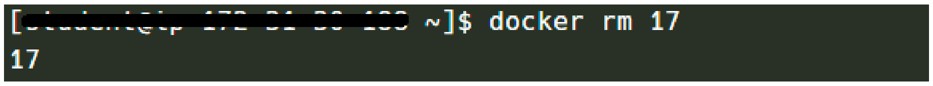
Though, we have image we can't remove it and that's what the output is showing. We cannot remove the image as, there are one or many containers associated with it. It means we first need to remove the containers and then we can remove the image. Let us do it:

a. First find out containers associated with our image



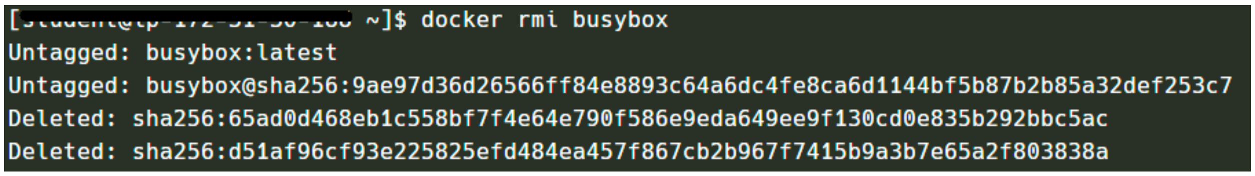
```
[[redacted] ~]$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
17ea7426ff90   busybox       "sh"       29 seconds ago   Exited (0) 28 seconds ago             eager_noyce
aaab9b739b9a   hello-world   "/hello"   2 minutes ago    Exited (0) 2 minutes ago              objective_greider
```

b. Container with ID 17, is associated with busybox. We need to stop that first



```
[[redacted] ~]$ docker rm 17
17
```

c. Now let us remove the image using docker rmi



```
[[redacted] ~]$ docker rmi busybox
Untagged: busybox:latest
Untagged: busybox@sha256:9ae97d36d26566ff84e8893c64a6dc4fe8ca6d1144bf5b87b2b85a32def253c7
Deleted: sha256:65ad0d468eb1c558bf7f4e64e790f586e9eda649ee9f130cd0e835b292bbc5ac
Deleted: sha256:d51af96cf93e225825efd484ea457f867cb2b967f7415b9a3b7e65a2f803838a
```

Remove dangling images

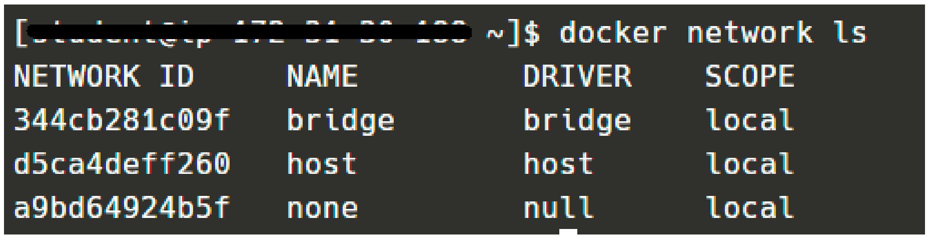
To remove all the dangling images in the system, we can use the following command:

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

16. Working with docker network

Finding the available networks:

Syntax: docker network ls



```
[[redacted] ~]$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
344cb281c09f   bridge    bridge    local
d5ca4deff260   host      host      local
a9bd64924b5f   none      null      local
```

Creating the network

Syntax: docker network create name-of-network

Execute the command docker network create mynetwork