**Docker Hands on Exercise**

Contents

[1. Start the Docker 2](#_Toc22212816)

[2. Basic Docker commands 2](#_Toc22212817)

[3. Running interactive Container 5](#_Toc22212818)

[Run a Containers in background 6](#_Toc22212819)

[4. Registering an account in Docker Hub 8](#_Toc22212820)

[5. Packaging and run a custom app using Docker 8](#_Toc22212821)

[Clone the application 8](#_Toc22212822)

[Walk through of Dockerfile 9](#_Toc22212823)

[Building the image 10](#_Toc22212824)

[Running the newly built container image 10](#_Toc22212825)

[Accessing the website 11](#_Toc22212826)

[6. Stop, start and Removing the container 11](#_Toc22212827)

[Stop a container 11](#_Toc22212828)

[Start a container 12](#_Toc22212829)

[Stop and Remove the container 12](#_Toc22212830)

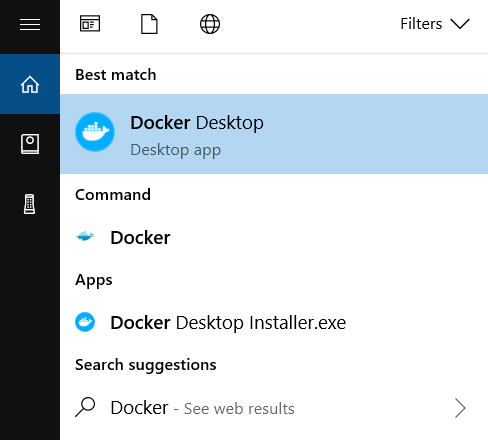
[Remove all containers irrespective of its state 12](#_Toc22212831)

[7. Upload the image to Docker Hub 13](#_Toc22212832)

[8. Play-with Docker 15](#_Toc22212833)

# Start the Docker

Docker Desktop does not start automatically after installation. To start Docker Desktop, search for Docker, and select **Docker Desktop** in the search results. It takes couple of minutes to start the Docker desktop.



When the whale icon in the status bar stays steady, Docker Desktop is up-and-running, and is accessible from any terminal window.

whale on taskbar

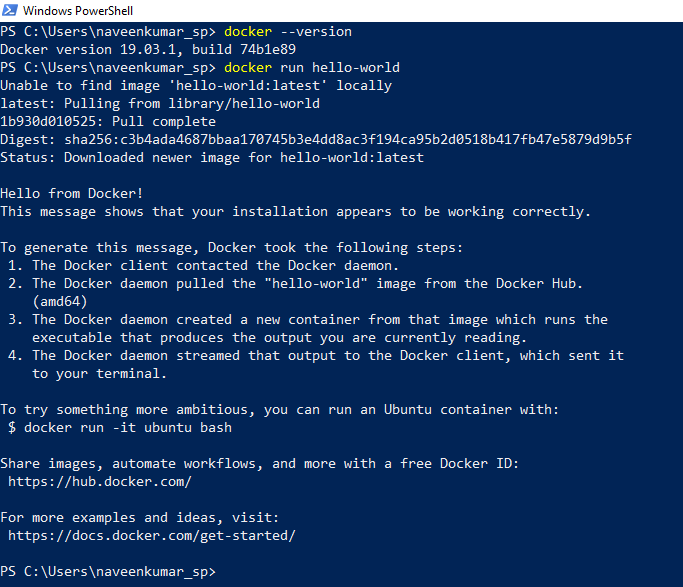
If the whale icon is hidden in the Notifications area, click the up arrow on the taskbar to show it. To learn more, see [Docker Settings](https://docs.docker.com/docker-for-windows/#docker-settings-dialog).

When initialization is complete, click the whale icon in the Notifications area and select **About Docker Desktop** to verify that you have the latest version.

# Basic Docker commands

1. Open a terminal window (Command Prompt or PowerShell, *but not* PowerShell ISE).
2. Run ***docker --version*** to ensure that you have a supported version of Docker:
3. Pull the [hello-world image](https://hub.docker.com/r/library/hello-world/) from Docker Hub and run a container by running the command

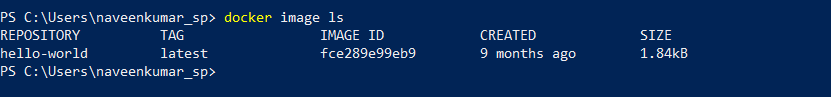
***docker run hello-world***



4. List the all image that was downloaded from Docker Hub using Command

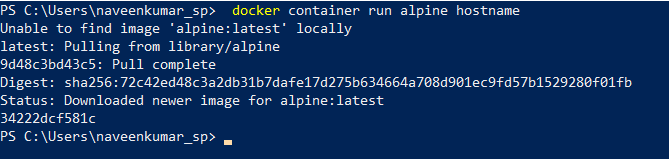
***docker image ls***

Note : Since Docker was installed recently you may not have many images. Re-issue the same command at the end of the exercise to look up all the public images which were downloaded.



5. Run an Alpine Linux container using below command

***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*.

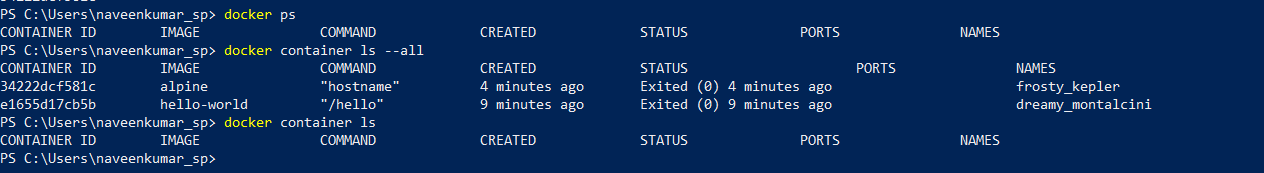
After the image is pulled, the container’s hostname is displayed (34222dcf581c in the example below).

Docker keeps a container running as long as the process it started inside the container is still running. All the containers which we have ran so far are exited right after it was started as it has served its purpose. Especially in alpine Linux, we have queried the host of the Linux container which was ran.

6. To list all the containers which are running, we can use below two commands.

***docker ps*** *OR* ***docker container ls.***

Note: to list all containers irrespective of the state, issue the command ***docker container ls –all***



# Running interactive Container

You can run a container based on a different version of Linux than is running on your Docker host. Run a Docker container and access its shell using below command.

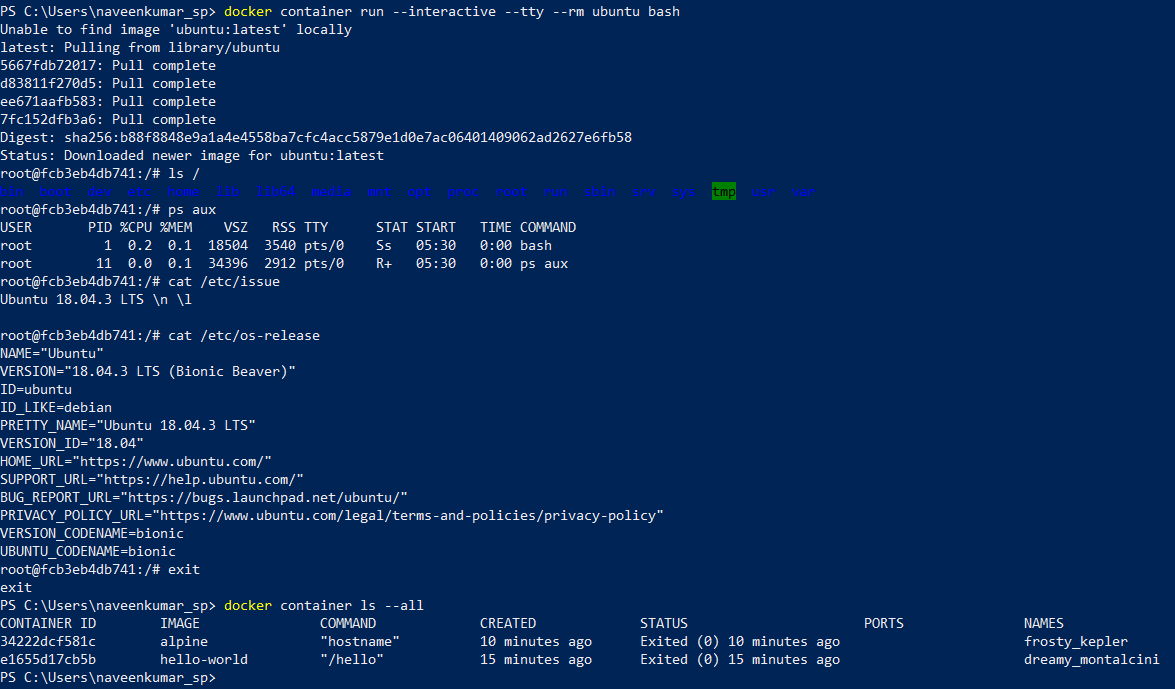
***docker container run --interactive --tty --rm ubuntu bash***

In this example, we’re giving Docker three parameters:

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

The first two parameters allow you to interact with the Docker container.

**ls /** will list the contents of the root director in the container, ps aux will show running processes in the container, ***cat /etc/issue*** and ***cat /etc/os-release*** will show which Linux distro the container is running, in this case Ubuntu 18.04.1 LTS.



Note: As we used the ***--rm*** flag when we started the container, Docker removed the container when it stopped. This means if you run another ***docker container ls --all*** you won’t see the Ubuntu container.

## Run a Containers in background

1. Background containers are how you’ll run most applications. Run a new MySQL container with the following command

***docker container run \***

***--detach \***

***--name mydb \***

***-e MYSQL\_ROOT\_PASSWORD=my-secret-pw \***

***mysql:latest***

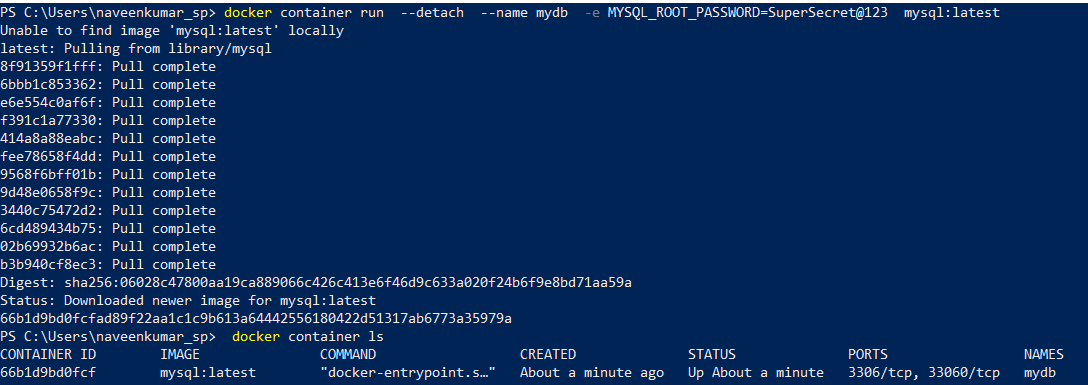
OR Use below one-line command

***docker container run --detach --name mydb -e MYSQL\_ROOT\_PASSWORD=my-secret-pw 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).

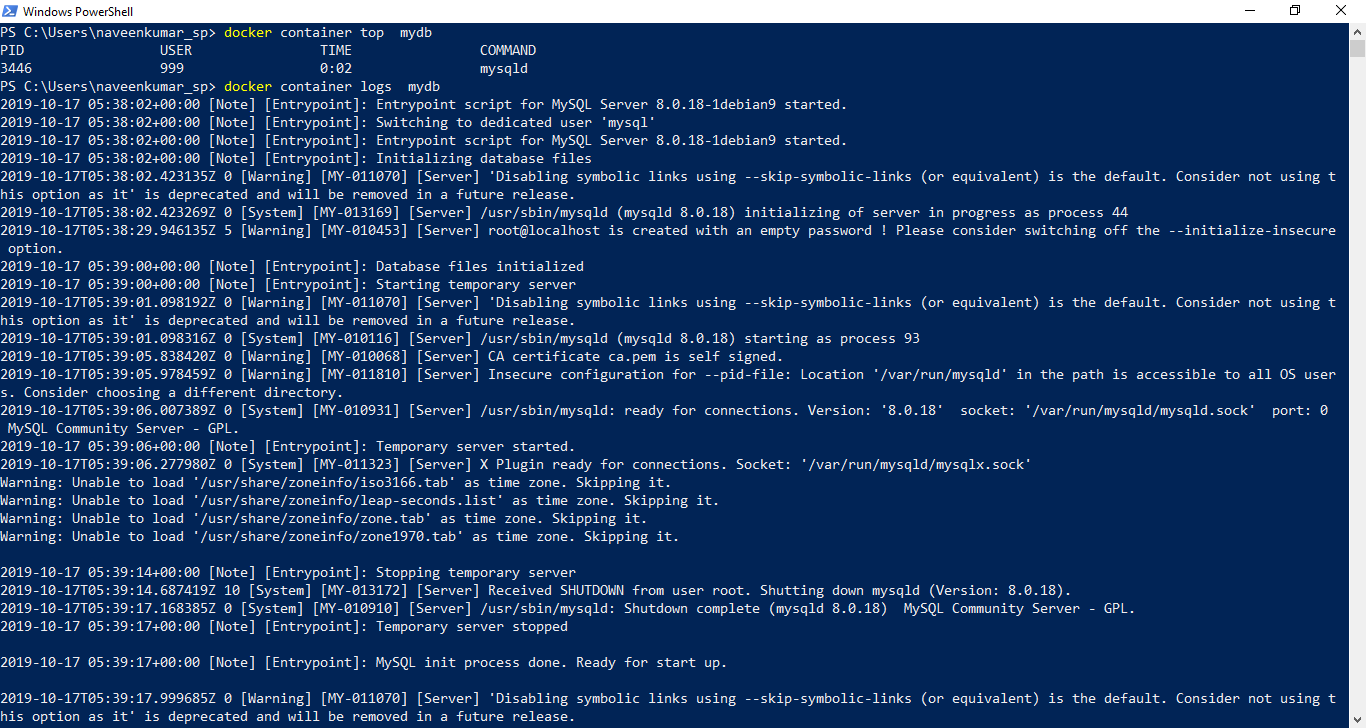
As the MySQL image was not available locally, Docker automatically pulled it from Docker Hub.

2. List the running containers using command ***docker container ls***



3. You can check what’s happening in your containers by using a couple of built-in Docker commands: ***docker container logs <container\_name>***  and ***docker container top*** ***<container\_name>***

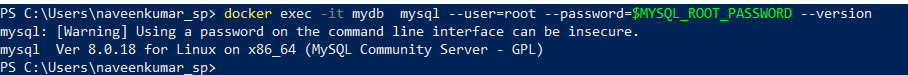
Note: Although MySQL is running, it is isolated within the container because no network ports have been published to the host. Network traffic cannot reach containers from the host unless ports are explicitly published.



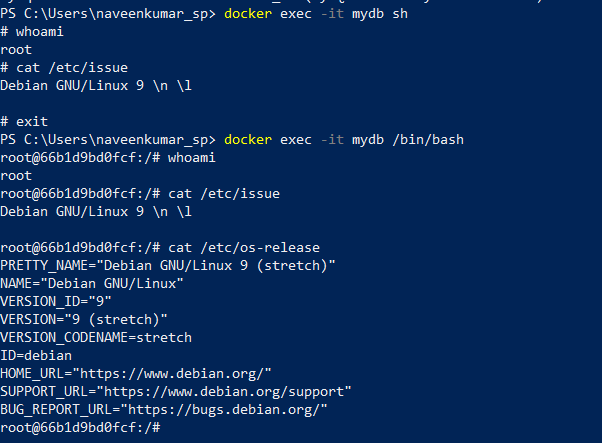
4. List the MySQL version using docker container exec.

***docker container exec*** allows you to run a command inside a container. In this example, we’ll use ***docker container exec*** to run the command-line equivalent of mysql --user=root --password=$MYSQL\_ROOT\_PASSWORD --version inside our MySQL container.

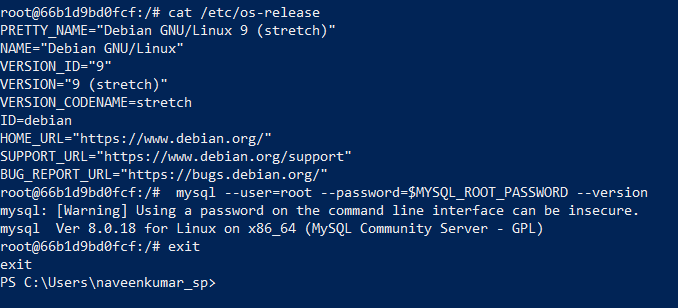
***docker exec -it mydb mysql --user=root --password=$MYSQL\_ROOT\_PASSWORD –version***



5. You can also use ***docker container exec*** to connect to a new shell process inside an already-running container. Executing the command below will give you an interactive shell (sh or bash) inside your MySQL container.



6. Check the version number by running the same command again, only this time from within the new shell session in the container.



# Registering an account in Docker Hub

Access the below URL and signup for an Docker ID. We will be using the Docker Hub account in the next exercise to push the built container image to Docker Hub.

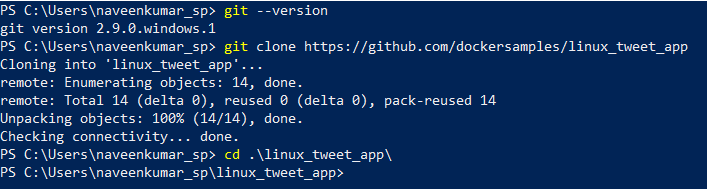
<https://hub.docker.com/signup>

# Packaging and run a custom app using Docker

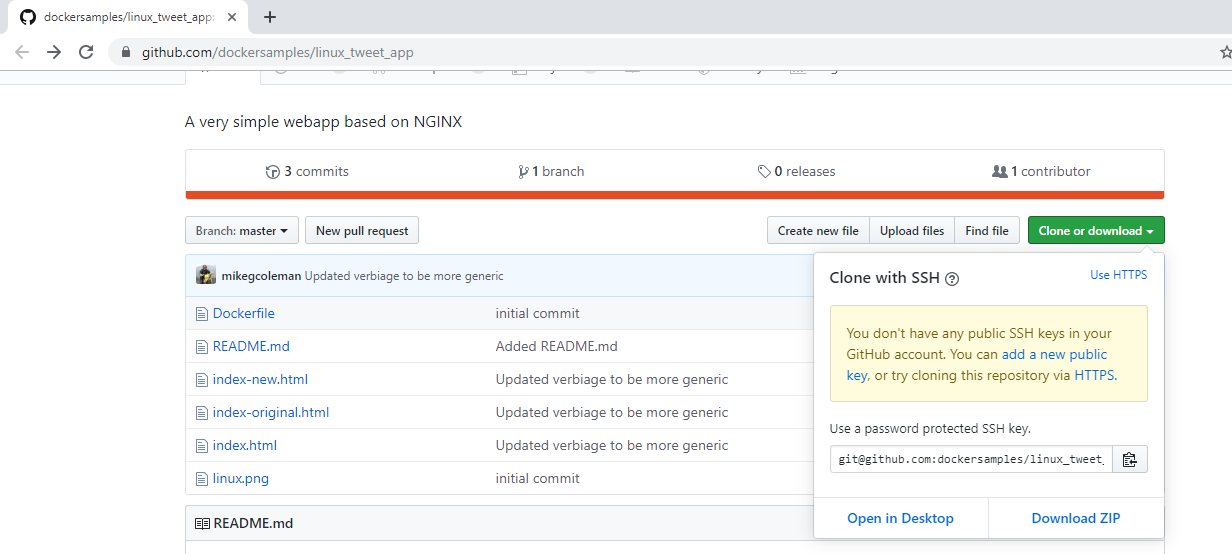
## Clone the application

Clone the below Docker sample application from GitHub using below command and change the command prompt to downloaded repository directory. i.e ***cd .\linux\_tweet\_app***

***git clone*** [***https://github.com/dockersamples/linux\_tweet\_app***](https://github.com/dockersamples/linux_tweet_app)

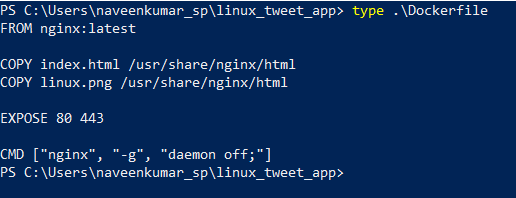


Note: If you don’t have git installed in your machine, Kindly access the URL in web browser and download Zip file from GitHub repository.



## Walk through of Dockerfile

Display the contents of the Dockerfile by typing the command ***type Dockerfile***



Here is the explanation of each step of the Dockerfile does.

* [FROM](https://docs.docker.com/engine/reference/builder/#from) specifies the base image to use as the starting point for this new image you’re creating. For this example we’re starting from ***nginx:latest***.
* [COPY](https://docs.docker.com/engine/reference/builder/#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](https://docs.docker.com/engine/reference/builder/#expose) documents which ports the application uses.
* [CMD](https://docs.docker.com/engine/reference/builder/#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.

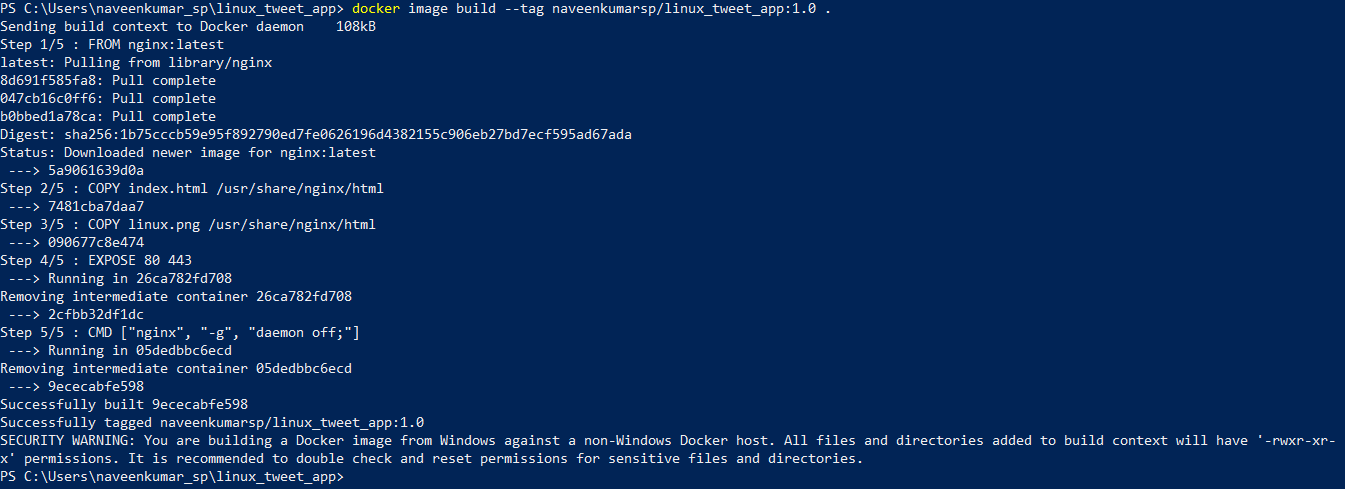
## Building the image

Use the **docker image build** command to create a new Docker image using the instructions in the Dockerfile

***docker image build --tag <dockerHubID>/linux\_tweet\_app:1.0 .***

* ***--tag*** allows us to give the image a custom name. In this case it’s comprised of our DockerID, the application name, and a version. Having the Docker ID attached to the name will allow us to store it on Docker Hub in a later step
* **.** tells Docker to use the current directory as the build context

Be sure to include period (.) at the end of the command.

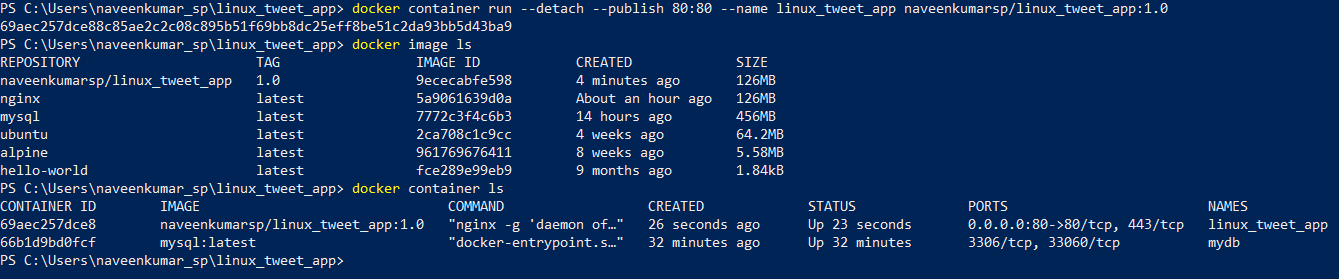


## Running the newly built container image

Use the ***docker container run*** command to start a new container from the image you created using the below command

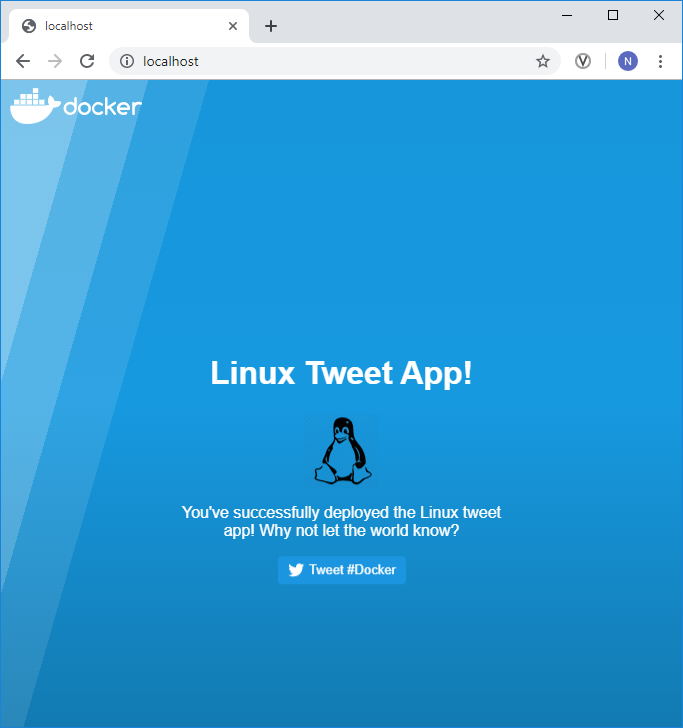
***docker container run --detach --publish 80:80 --name linux\_tweet\_app <dockerHubID>/linux\_tweet\_app:1.0***

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. The format of the ***--publish*** flag is ***host\_port:container\_port***.



## Accessing the website

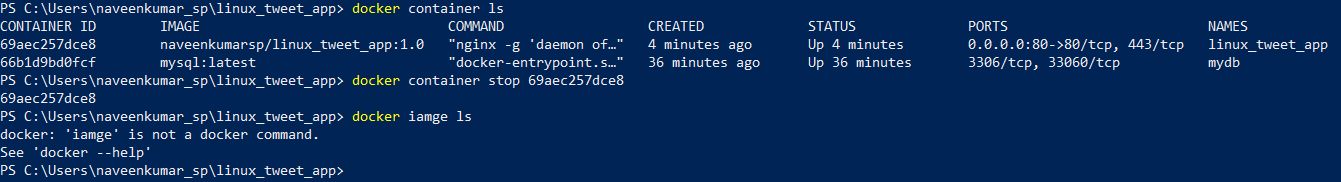
Any external traffic coming into the server on port 80 will now be directed into the container on port 80. Access the url <http://localhost/> to view the webapplication hosted from linux\_tweet\_app container.



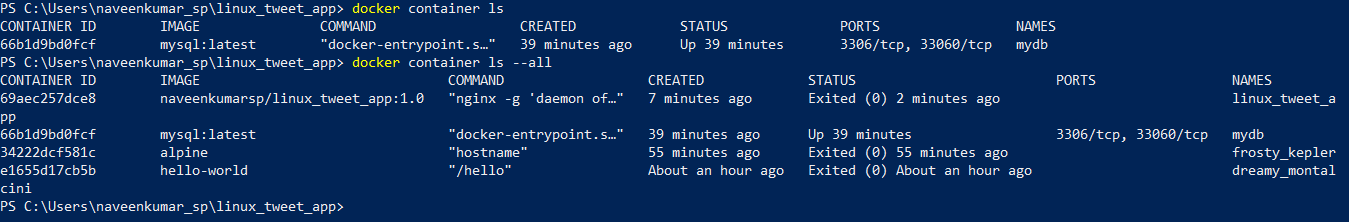
# Stop, start and Removing the container

### Stop a container

To stop the container, you can use the command **docker *stop <container\_name/Container\_ID>***



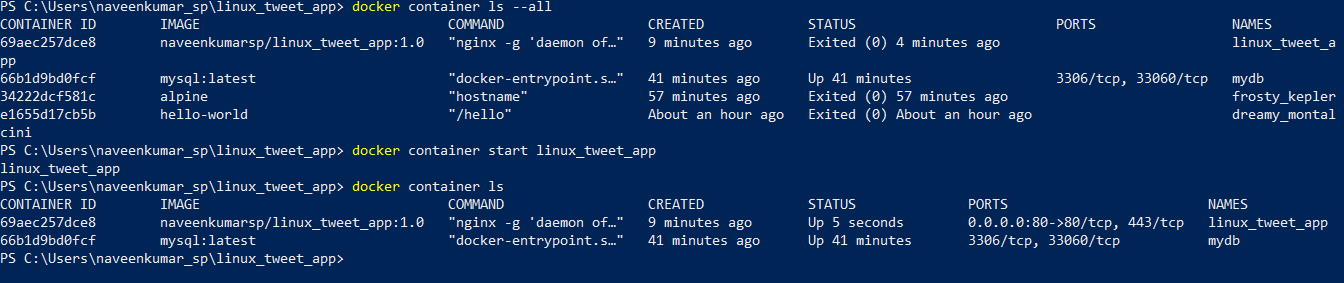
Note : Here, we have used the container ID instead of container name to stop the container.

As you may see below, only one container is running i.e mydb. Rest are exited. 

### Start a container

You may start the container which is exited once again by running the command

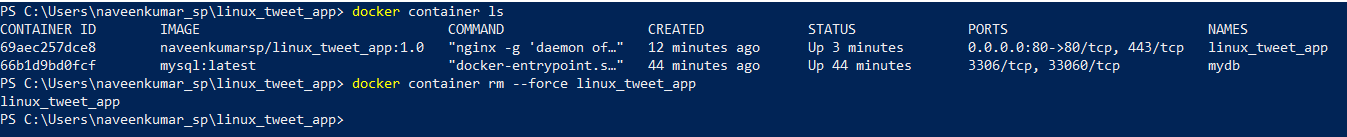
***docker container start <container\_Name/Container\_ID>***



### Stop and Remove the container

To stop and remove the container forcefully we can use the below command

***docker container rm --force linux\_tweet\_app***

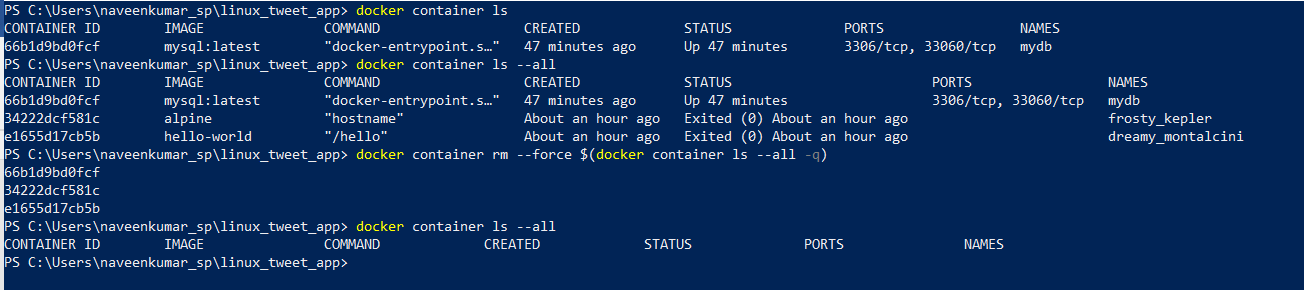


Note: We used the ***--force*** parameter to remove the running container without shutting it down. This will ungracefully shutdown the container and permanently remove it from the Docker host.

### Remove all containers irrespective of its state

Issue the below command where container ID will be passed to docker container rm command to remove all containers.

***docker container rm --force $(docker container ls --all -q)***



# Upload the image to Docker Hub

1. Build a new image and tag it as 2.0 with below command

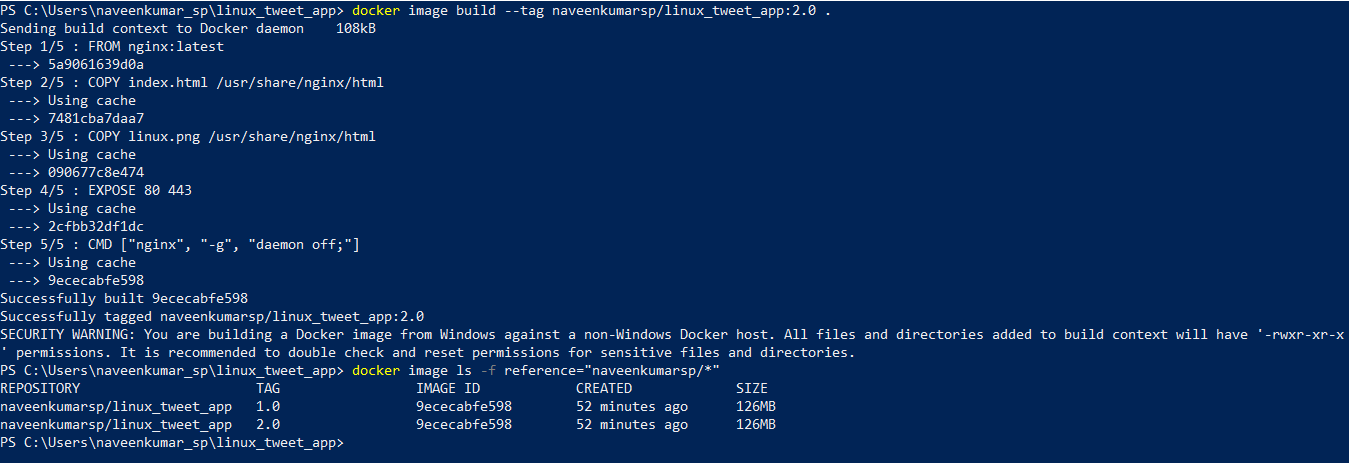
***docker image build --tag <DOCKERID>/linux\_tweet\_app:2.0***

Notice how fast that built! This is because Docker only modified the portion of the image that changed vs. rebuilding the whole image.

2. List the images on your Docker host.

***docker image ls -f reference="<dockerID>/\*"***

You will see that you now have two ***linux\_tweet\_app*** images - one tagged as ***1.0*** and the other as ***2.0***.



3. Before you can push your images, you will need to log into Docker Hub using below command

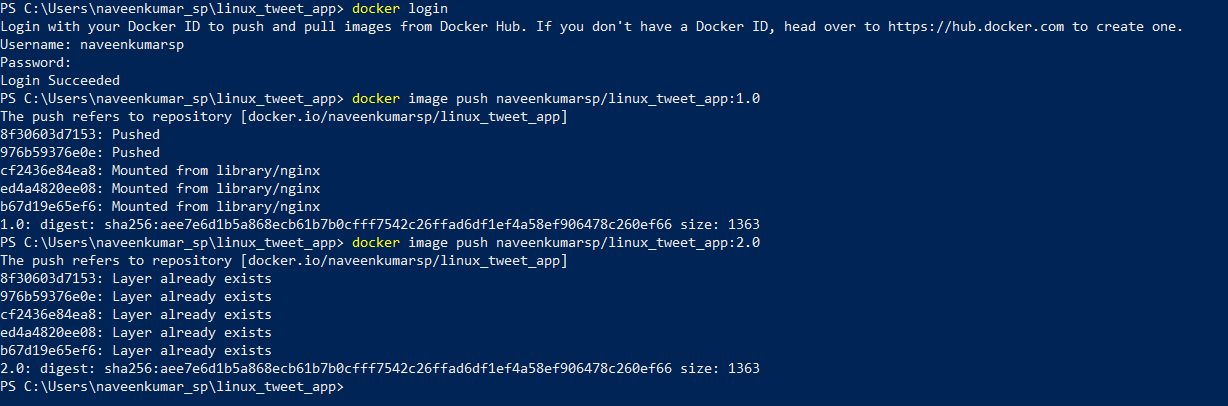
***docker login***

4. Push version 1.0 of your web app using docker image push command as listed below.

***docker image push <dockerID>/linux\_tweet\_app:1.0***

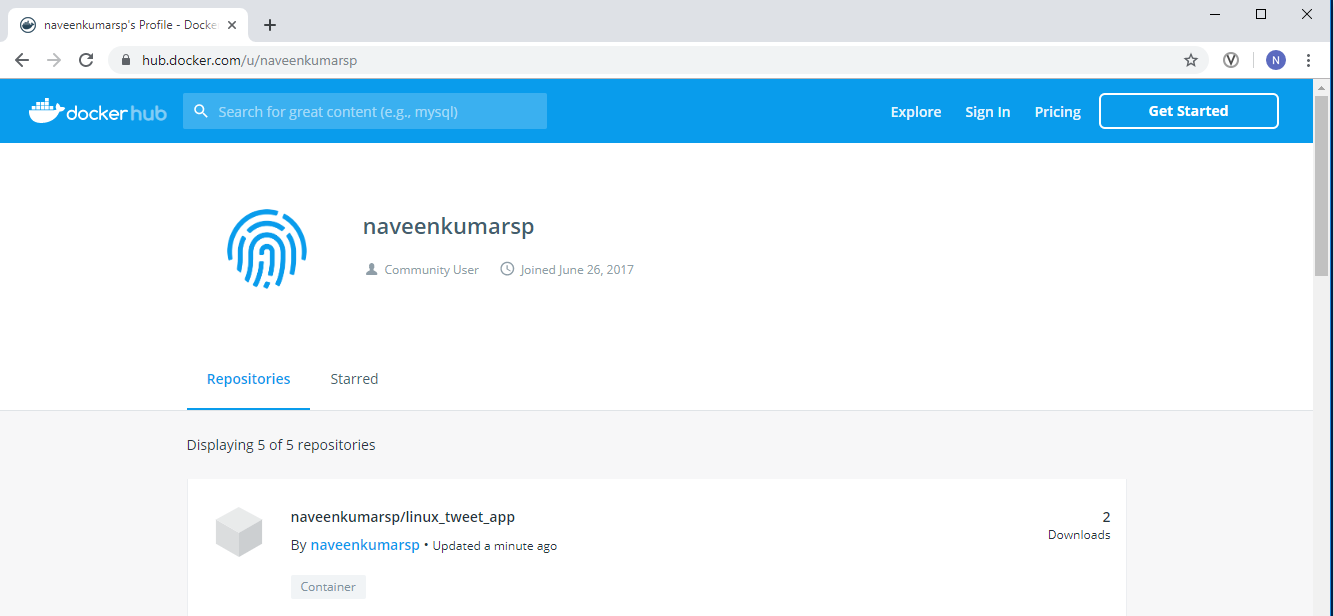
5. Push version 2.0 as well.

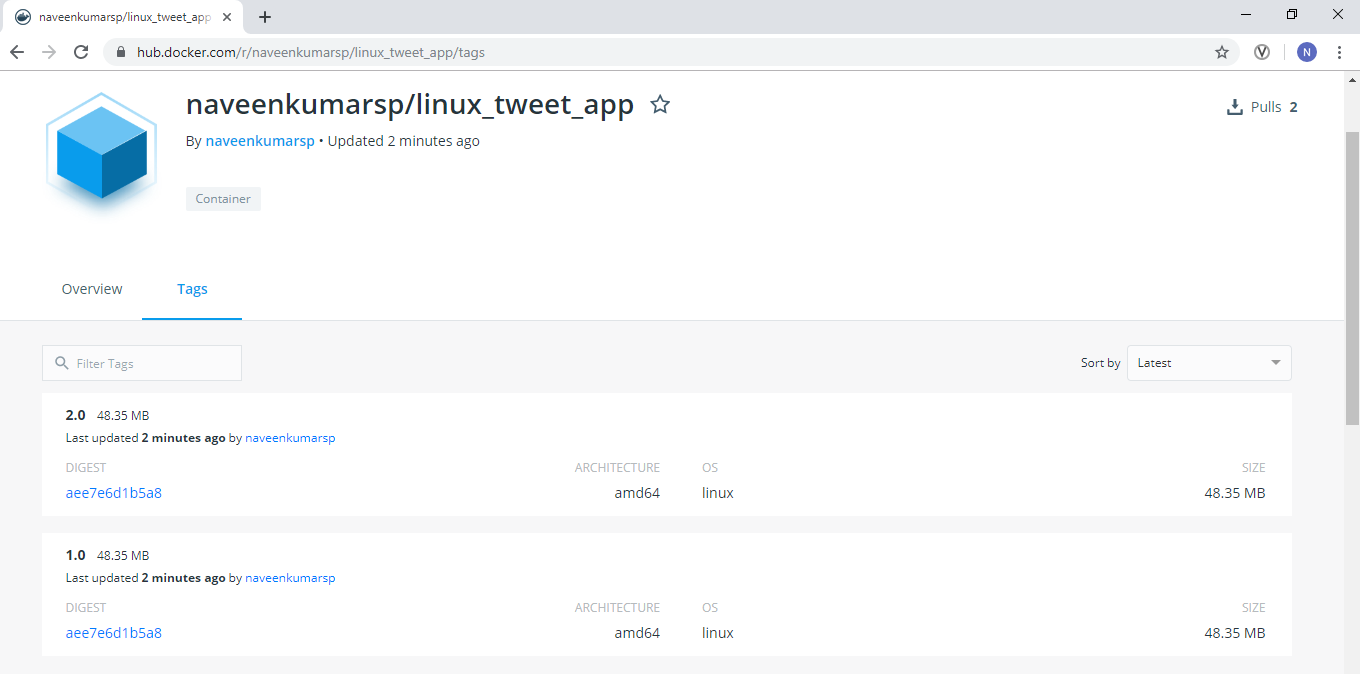
***docker image push <dockerID>/linux\_tweet\_app:2.0***



Notice that several lines of the output say ***Layer already exists***. This is because Docker will leverage read-only layers that are the same as any previously uploaded image layers*.*

6. You can browse to**https://hub.docker.com/r/<docker id>/**and see your newly-pushed Docker images. These are public repositories, so anyone can pull the image





**Note:** you don’t even need a Docker ID to pull public images. Docker Hub also supports private repositories. Also it is possible to deploy private registry in your organization to push and pull containers to deploy applications.

E.g : registry.eqslearning.com:5000 is the private registry of our hands on lab and Visualization can be done on <http://docker.eqslearning.com/> . You may notice many container images listed which were part of the hands on exercises and Demo CI/CD Pipelines.

# Play-with Docker

If you are unable to install Docker desktop in your workstation, you may access the below URL and use public instance to practice the lab exercise. (In order to use the play-with-docker, kindly Login/register an account for Docker ID on <https://hub.docker.com/signup> which will be used in exercise as well.

<https://labs.play-with-docker.com/>

Note: The Play with Docker is an Alpine Linux platform on internet. When you expose the port, you may see the port number on the Top will be created. Click on the port number to access the web application.

Note: In this session I have used version/tag 2.0 which was pulled from my Docker hub.

