Introduction to Docker

(In this document, you will find explanations of basic terms, commands, and expected outputs. It is strongly recommended that you carefully go through these concepts to understand what each step and output means. This will help you build a strong foundation, as the next lab will cover advanced Docker concepts, which require a clear understanding of the basics covered here.)

In traditional software setup, applications require manual installation of programming languages, libraries, and system dependencies on every computer. This often leads to problems such as version conflicts and **"it works on my machine"** issues.

Docker is a containerization platform that allows applications to be packaged along with all their dependencies into a single unit called a **container**. This container can run consistently on any system where Docker is installed, regardless of the host operating system.

Using Docker, developers can build an application once and run it anywhere, which is a key concept in cloud computing and DevOps.

So to understand in simple words,Think of Docker like a "**Lunch Box"**

- The food is the application
- The box is the container
- Everything needed to eat is packed inside

So wherever you take the box, you can eat the food the same way.
Similarly, wherever you run the Docker container, the application runs the same way.

1. Dockerfile

A Dockerfile is a set of instructions that tells Docker how to create an image. It contains step-by-step commands such as which base system to use, what software to install, and which program to run.

Example:
Think of a cooking recipe. It says: take ingredients, mix them, cook for some time, and serve. In the same way, a Dockerfile tells Docker exactly how to prepare the environment for an application.

2. Docker Images

A Docker image is a ready-to-use package that contains the application along with all the required software and settings needed to run it.

Example:
Imagine buying a packed lunch box from a store. It already contains food, spoon, and napkins.Similarly, a Docker image already contains the app and everything it needs to work.

3. Docker Containers

A Docker container is a running version of an image. When an image is started, it becomes a container that actually performs tasks.

Example:
If an image is like a music file, then a container is when you press play and the song starts playing.The file is stored, but the music is only heard when it is running.

4. Docker Hub

Docker Hub is an online platform where users can store, download, and share Docker images.

Example:
It is like an app store for Docker. Instead of building everything yourself, you can download ready-made applications such as Ubuntu, Python, or MySQL and start using them immediately.

5. Volumes

Volumes are used to store data outside containers so that the data is not lost even if the container is deleted or restarted.

Example:
Think of saving your project on a pen drive. Even if the computer is switched off or changed, your files remain safe on the pen drive.Volumes work in the same way for container data.

6. Docker Networks

Docker networks allow containers to communicate with each other safely and easily.

Example:
Consider different departments in a college connected through internal phones. They can talk directly without using personal mobile numbers.Docker networking creates a private communication system between containers.

Why is Docker Important in Cloud Computing?

In cloud environments:

- Applications are deployed as containers

- Containers are stored in registries like Docker Hub
- Cloud servers pull images and run containers automatically

This makes application deployment faster , more reliable and highly scalable.

In this lab, we will learn how to build, run, and share Docker containers web applications.

IMPORTANT: This lab must be performed only on **Ubuntu (native or WSL)**. Do **not** attempt this lab on **Windows OS**, as the commands and setup may not work correctly.

Step 1: Verifying Docker Installation using hello-world

Procedure:

1. Open the terminal.
2. Create a folder with your SRN

3. Set the SRN as an environment variable:

```
$export SRN=PES1UG23CSXXX
```

3. Display the SRN to confirm:

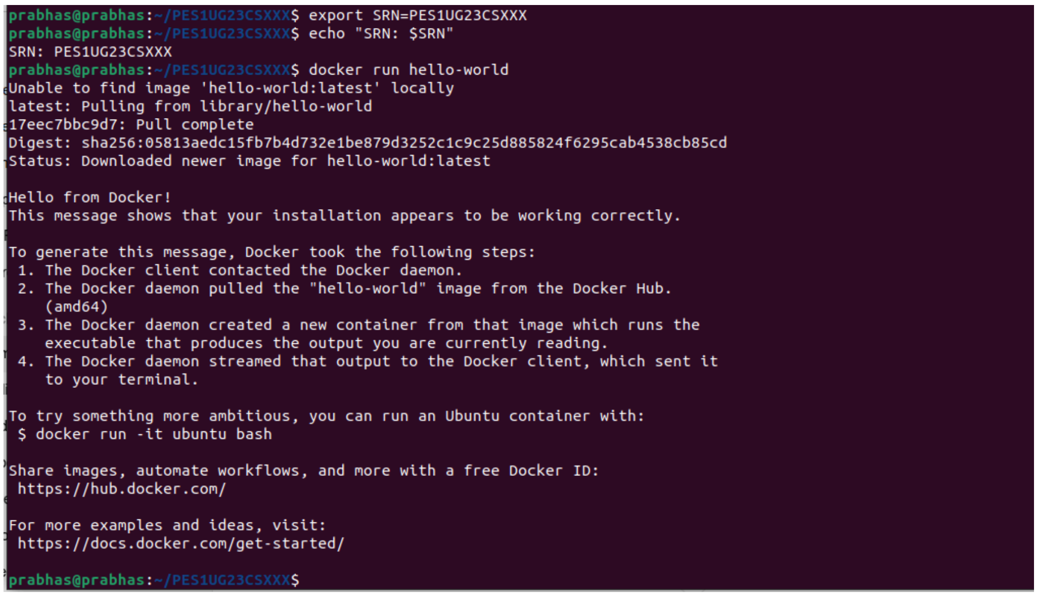
```
$echo "SRN: $SRN"
```

4. Run the Docker test container:

```
$docker run hello-world
```

5. Observe the output message displayed on the terminal and paste it as SS1.

Output SS1:



```
prabhas@prabhas:~/PES1UG23CSXXX$ export SRN=PES1UG23CSXXX
prabhas@prabhas:~/PES1UG23CSXXX$ echo "SRN: $SRN"
SRN: PES1UG23CSXXX
prabhas@prabhas:~/PES1UG23CSXXX$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
Digest: sha256:05813aedc15fb7b4d732e1be879d3252c1c9c25d885824f6295cab4538cb85cd
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/

prabhas@prabhas:~/PES1UG23CSXXX$
```

Output Explanation:

- Since the hello-world image was not present on the system, Docker first downloaded it from Docker Hub.
- After downloading, Docker created a container from the image and executed it.
- The container printed a message confirming that Docker is working correctly.
- Once the message was displayed, the container stopped automatically.

This confirms that:

- Docker daemon is running
- Images can be pulled from Docker Hub
- Containers can be created and executed successfully

Step 2: Using Basic Docker Commands

Procedure:

1. Open the terminal.
2. Run the Below command , this only displays the running containers

$docker ps

(Since the hello-world container has already finished execution, no running containers will be shown).

3. To Display all containers including stopped ones execute below:

$docker ps -a

(This shows the previously executed hello-world container with status as *Exited)*.

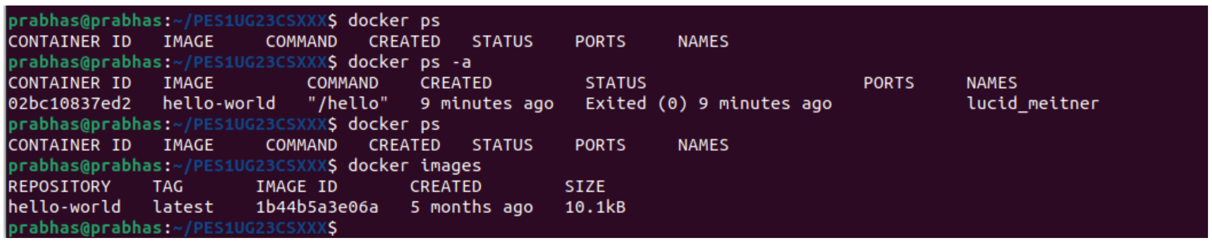
4. To Display all Docker images available on the system execute below command:

$docker images

(This lists the hello-world image that was downloaded in the previous experiment).

5. After Executing Above commands , paste it as SS2 as shown below.

Output SS2:



```
prabhas@prabhas:~/PES1UG23CSXXX$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
prabhas@prabhas:~/PES1UG23CSXXX$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED         STATUS                     PORTS     NAMES
02bc10837ed2   hello-world   "/hello"   9 minutes ago   Exited (0) 9 minutes ago             lucid_meitner
prabhas@prabhas:~/PES1UG23CSXXX$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
prabhas@prabhas:~/PES1UG23CSXXX$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    1b44b5a3e06a   5 months ago    10.1kB
prabhas@prabhas:~/PES1UG23CSXXX$
```

Output Explanation:

- docker ps shows only currently running containers. Since hello-world runs and exits immediately, no container appears here.
- docker ps -a shows all containers including stopped ones, which helps in checking previously executed containers.
- docker images displays all images stored locally, including image name, tag, image ID, and size.

Step 3A: Pull Base Image from Docker Hub

Why Do We Use a Base Image?

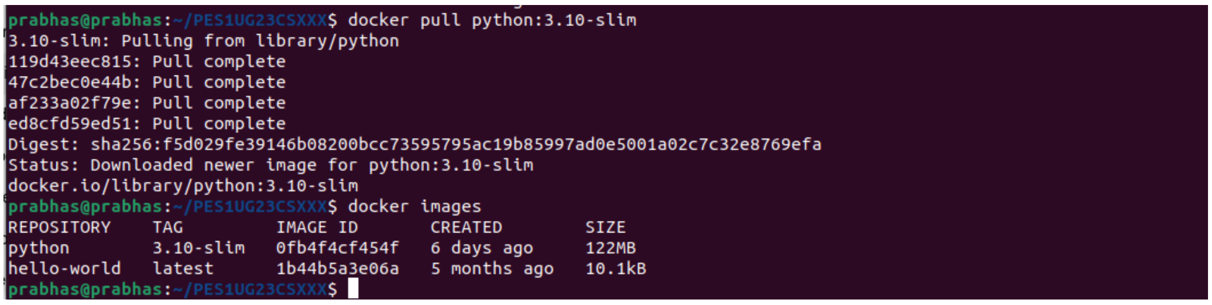
To build our own application container, we first need a **base image** that already contains the required runtime environment.

Since our application which we are going to use in next step is written in Python and uses machine learning libraries, we use an official Python image from Docker Hub.This avoids manual installation of Python and ensures a consistent environment across all systems.We will now pull a lightweight Python image and use it as the foundation for building our application image.

Commands:

1) $ docker pull python:3.10-slim
2) $ docker images

Output SS3:



```
prabhas@prabhas:~/PES1UG23CSXXX$ docker pull python:3.10-slim
3.10-slim: Pulling from library/python
119d43eec815: Pull complete
47c2bec0e44b: Pull complete
af233a02f79e: Pull complete
ed8cfd59ed51: Pull complete
Digest: sha256:f5d029fe39146b08200bcc73595795ac19b85997ad0e5001a02c7c32e8769efa
Status: Downloaded newer image for python:3.10-slim
docker.io/library/python:3.10-slim
prabhas@prabhas:~/PES1UG23CSXXX$ docker images
REPOSITORY    TAG         IMAGE ID       CREATED        SIZE
python        3.10-slim   0fb4f4cf454f   6 days ago     122MB
hello-world   latest      1b44b5a3e06a   5 months ago   10.1kB
prabhas@prabhas:~/PES1UG23CSXXX$
```

Output Explanation:

- Docker images are downloaded from Docker Hub in the form of multiple layers, where each layer represents a part of the software environment such as the base operating system and installed libraries.
 This layered architecture allows Docker to reuse existing layers, which reduces download time and storage usage when the same layers are shared across images.
- The slim variant of the Python image is used to minimize the overall image size while still providing the required Python runtime, which helps in faster image downloads and quicker container startup.

- After executing the pull command, the downloaded Python image can be verified using the docker images command, which shows the locally available images including the newly pulled base image.

Step 3B: Write Docker file and Build image (This is where you will require a zip file given to you).

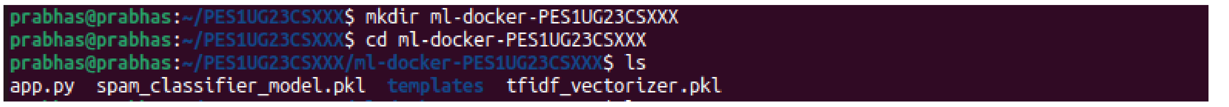
In this step, we will Dockerize a simple machine learning application called "spam-classifier." All the required project files are already provided to you. The main objective of this exercise is not to build the ML model, but to understand the complete process of containerizing an application using Docker—including creating a Dockerfile, building an image, and running the application inside a container.

Procedure:

1. In your present directory which is your SRN , create another folder by using below command and navigate to that dir:

   ```
   $ mkdir ml-docker-PES1UG23CSXXX
   $ cd ml-docker-PES1UG23CSXXX
   ```

2. Now download the Files given to you and put them in this Folder. The project directory should contain files given to you as shown below:

   ```
   prabhas@prabhas:~/PES1UG23CSXXX$ mkdir ml-docker-PES1UG23CSXXX
   prabhas@prabhas:~/PES1UG23CSXXX$ cd ml-docker-PES1UG23CSXXX
   prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ ls
   app.py  spam_classifier_model.pkl  templates  tfidf_vectorizer.pkl
   ```

3. The HTML file for this application is located inside the `templates` folder as `index.html`.You must add basic CSS styling to `templates/index.html` to improve the appearance of the page, without changing the existing structure. Add a heading at the top center of the page with the text **"Your SRN"**. Use simple CSS to make the page look better.

   Note: Do not add any advanced or professional CSS styling to the web page. Keep the interface simple, as the primary objective of this lab is to understand Docker concepts and containerized deployment, not front-end design.

4. After completing the previous steps, we now move to the most important part of this experiment — creating a Dockerfile to package the entire machine learning web application into a Docker image.

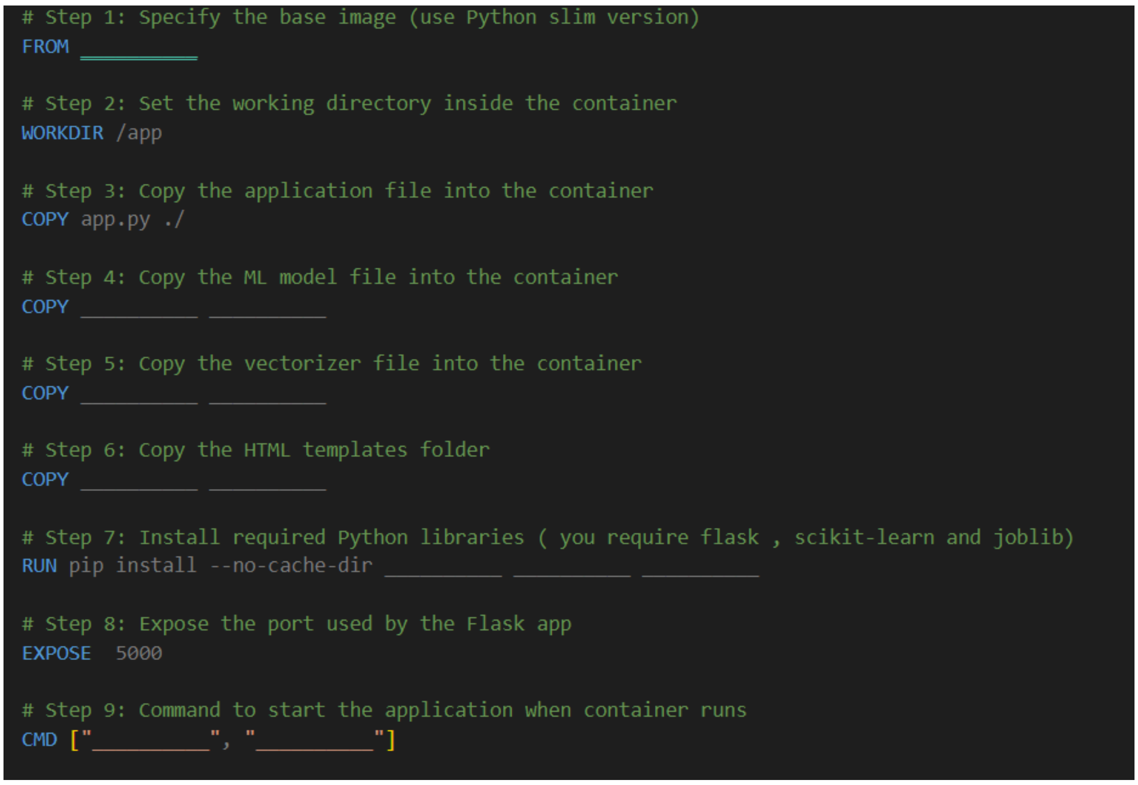
   A Dockerfile is a text file that contains instructions for building a Docker image. It defines the base environment, required software, application files, and the command needed to start the application.

In the same project directory, create a new file named **Dockerfile** (note that no file extension such as .txt should be added).

Complete the Dockerfile by filling in the required instructions in the appropriate blank spaces as shown in the template below, and save the file after making the changes.

While creating the Dockerfile in your virtual machine, type the same template as given below and replace the blanks with suitable values.

Dockerfile Template:



```
# Step 1: Specify the base image (use Python slim version)
FROM __________

# Step 2: Set the working directory inside the container
WORKDIR /app

# Step 3: Copy the application file into the container
COPY app.py ./

# Step 4: Copy the ML model file into the container
COPY __________ __________

# Step 5: Copy the vectorizer file into the container
COPY __________ __________

# Step 6: Copy the HTML templates folder
COPY __________ __________

# Step 7: Install required Python libraries ( you require flask , scikit-learn and joblib)
RUN pip install --no-cache-dir __________ __________ __________

# Step 8: Expose the port used by the Flask app
EXPOSE  5000

# Step 9: Command to start the application when container runs
CMD ["__________", "__________"]
```

5. Now that the Dockerfile has been successfully created, the next step is to build a Docker image using the instructions defined in that Dockerfile.

   Command to build docker image:

   1) $ docker build [OPTIONS] <IMAGE_NAME> <BUILD_CONTEXT>

   [OPTIONS] are Optional flags to modify the build process (for example: -t, --no-cache, etc.).

<BUILD_CONTEXT> is Path to the directory that contains the Dockerfile and all required files.Commonly, . (dot) is used to indicate the current directory.

```
2) $ docker build -t spam-web-app-PES1UG22CSXXX .
```

- **docker build**
  This instructs Docker to start the image build process using the Dockerfile.

- **-t spam-web-app-PES1UG23CSXXX**
  The -t (tag) option assigns a name to the newly created image.
  Here, spam-web-app is the image name that will be used later to run and push the container.

- **. (dot)**
  The dot represents the build context, meaning Docker will use the current directory to locate the Dockerfile and all required project file such as Python application file,ML model files,HTML templates

What Happens During the Build Process?

During the build process, Docker executes each instruction in the Dockerfile step by step, such as selecting the base image, copying application files, installing required libraries, and setting the startup command.
After successful completion, the final output is a ready-to-run Docker image containing the complete ML web application environment.

After the image is built successfully, capture the output displayed in the terminal.

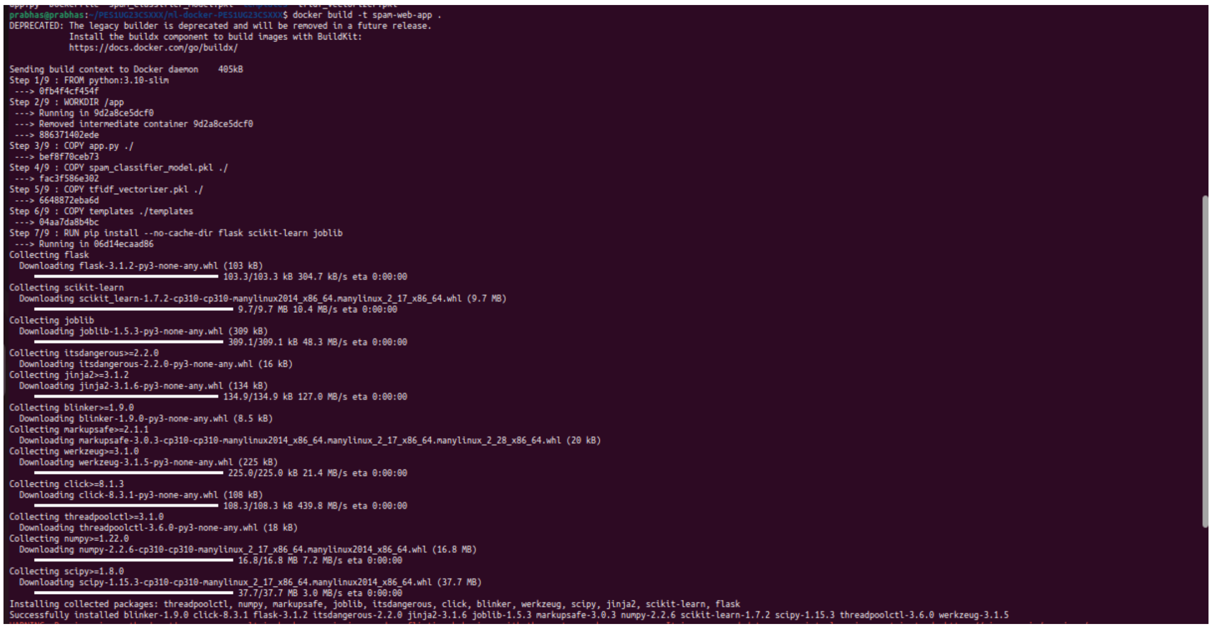
- If the output is long, it may be captured in **two screenshots**.
- Both screenshots should be labeled together as **SS4**.

Make sure the screenshot clearly shows:

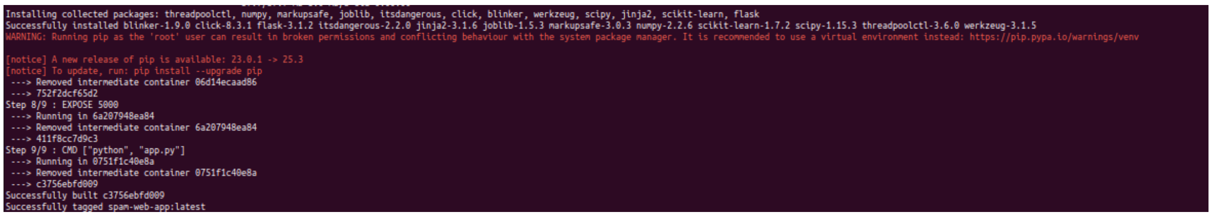
- Successful build message
- Image name (spam-web-app-SRN)
- Your SRN visible in the terminal

(NOTE: Here in the below SS the image name is spam-web-app , But your output should have it as spam-web-app-SRN)

Output SS4:



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker build -t spam-web-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon    405kB
Step 1/9 : FROM python:3.10-slim
 ---> 0fb4f4cf454f
Step 2/9 : WORKDIR /app
 ---> Running in 9d2a8ce5dcf0
 ---> Removed intermediate container 9d2a8ce5dcf0
 ---> 886371402ede
Step 3/9 : COPY app.py ./
 ---> bef8f70ceb73
Step 4/9 : COPY spam_classifier_model.pkl ./
 ---> fac3f586e302
Step 5/9 : COPY tfidf_vectorizer.pkl ./
 ---> 6648872eba6d
Step 6/9 : COPY templates ./templates
 ---> 04aa7da8b4bc
Step 7/9 : RUN pip install --no-cache-dir flask scikit-learn joblib
 ---> Running in 06d14ecaad86
Collecting flask
  Downloading flask-3.1.2-py3-none-any.whl (103 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.3/103.3 kB 304.7 kB/s eta 0:00:00
Collecting scikit-learn
  Downloading scikit_learn-1.7.2-cp310-cp310-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (9.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.7/9.7 MB 10.4 MB/s eta 0:00:00
Collecting joblib
  Downloading joblib-1.5.3-py3-none-any.whl (309 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 309.1/309.1 kB 48.3 MB/s eta 0:00:00
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting jinja2>=3.1.2
  Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.9/134.9 kB 127.0 MB/s eta 0:00:00
Collecting blinker>=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting markupsafe>=2.1.1
  Downloading markupsafe-3.0.3-cp310-cp310-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (20 kB)
Collecting werkzeug>=3.1.0
  Downloading werkzeug-3.1.5-py3-none-any.whl (225 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 225.0/225.0 kB 21.4 MB/s eta 0:00:00
Collecting click>=8.1.3
  Downloading click-8.3.1-py3-none-any.whl (108 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 108.3/108.3 kB 439.8 MB/s eta 0:00:00
Collecting threadpoolctl>=3.1.0
  Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Collecting numpy>=1.22.0
  Downloading numpy-2.2.6-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.8/16.8 MB 7.2 MB/s eta 0:00:00
Collecting scipy>=1.8.0
  Downloading scipy-1.15.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (37.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.7/37.7 MB 3.0 MB/s eta 0:00:00
Installing collected packages: threadpoolctl, numpy, markupsafe, joblib, itsdangerous, click, blinker, werkzeug, scipy, jinja2, scikit-learn, flask
Successfully installed blinker-1.9.0 click-8.3.1 flask-3.1.2 itsdangerous-2.2.0 jinja2-3.1.6 joblib-1.5.3 markupsafe-3.0.3 numpy-2.2.6 scikit-learn-1.7.2 scipy-1.15.3 threadpoolctl-3.6.0 werkzeug-3.1.5
```



```
Installing collected packages: threadpoolctl, numpy, markupsafe, joblib, itsdangerous, click, blinker, werkzeug, scipy, jinja2, scikit-learn, flask
Successfully installed blinker-1.9.0 click-8.3.1 flask-3.1.2 itsdangerous-2.2.0 jinja2-3.1.6 joblib-1.5.3 markupsafe-3.0.3 numpy-2.2.6 scikit-learn-1.7.2 scipy-1.15.3 threadpoolctl-3.6.0 werkzeug-3.1.5
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.3
[notice] To update, run: pip install --upgrade pip
 ---> Removed intermediate container 06d14ecaad86
 ---> 752f2dcf65d2
Step 8/9 : EXPOSE 5000
 ---> Running in 6a207948ea84
 ---> Removed intermediate container 6a207948ea84
 ---> 411f8cc7d9c3
Step 9/9 : CMD ["python", "app.py"]
 ---> Running in 0751f1c40e8a
 ---> Removed intermediate container 0751f1c40e8a
 ---> c3756ebfd009
Successfully built c3756ebfd009
Successfully tagged spam-web-app:latest
```

(NOTE: In Docker containers, applications usually run as root hence the warning you see in SS is harmless and does not affect execution)

Output Explanation:

The successful completion of the docker build command indicates that Docker has created a new image by following all the instructions specified in the Dockerfile.
 During this process, Docker packages the entire ML application — including the Python runtime, required libraries, pretrained model files, and web interface — into a single portable image.

This image represents a complete software environment that can run identically on any system where Docker is available, independent of the host operating system or installed software.
 Instead of manually installing dependencies on every machine, the environment is defined once in the Dockerfile and automatically reproduced by Docker during the image build process.

As a result, the generated Docker image acts as a self-contained deployment unit that can be shared, deployed on servers, or pushed to cloud registries such as Docker Hub for reuse.

6. Now that the Docker image has been successfully created, the next step is to run a container using this image and start the ML web application.

   Execute the following command to run the container:

   ```
   $ docker run [OPTIONS] <IMAGE_NAME>

   $ docker run -d -p 5000:5000 spam-web-app-PES1UG23CSXXX
   ```

This command creates and starts a container from the previously built Docker image and runs the application.

- docker run
  This tells Docker to create a new container from the specified image and start executing it.
- -d (detached mode)
  This option runs the container in the background, allowing the terminal to be used for other commands while the application continues to run.
- -p 5000:5000 (port mapping)
  This maps port **5000 of the host machine** to port **5000 inside the container**.
  Since the Flask web application inside the container listens on port 5000, this mapping allows users to access the application using a web browser at http://localhost:5000.
- spam-web-app
  This is the name of the Docker image from which the container is created.

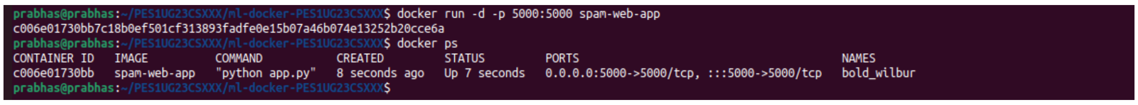
  After executing this command, the ML web service runs inside the container, and users can interact with it through the browser, while the application remains isolated from the host operating system.

7. Now once you create container , check if the container is running using below command:

   ```
   $ docker ps
   ```

   Your SS5 should be both step-7 and step-8.

Output SS5:



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker run -d -p 5000:5000 spam-web-app
c006e01730bb7c18b0ef501cf313893fadfe0e15b07a46b074e13252b20cce6a
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker ps
CONTAINER ID   IMAGE          COMMAND           CREATED         STATUS         PORTS                                       NAMES
c006e01730bb   spam-web-app   "python app.py"   8 seconds ago   Up 7 seconds   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   bold_wilbur
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$
```

8. You are running your container now , open http://localhost:5000/ (here in my case i used port 5000 , if u get any errors try with some other port or check if something else is running on that port). Once you open that in your browser , you will be able to see the page there which you can use.paste that page output as **SS6** (you ,might have already added some CSS and SRN at the top-center of the page , you should be able to see that when go to the link).

Output SS6: (Add the webpage SS)

Step 3C: Push Created Image to Docker Hub

1. Login to the Docker Hub using the below command :

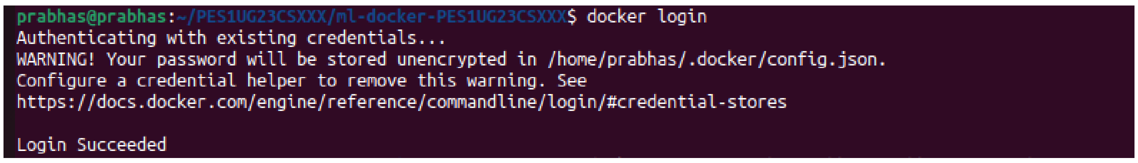
   $ docker login

   Enter your Docker Hub **username** and **password** when prompted.

   If the login is successful, the terminal will display as "Login succeeded"

   This will be your **SS7**

Output SS7:



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/prabhas/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
```

2. Tag the Image for Docker Hub

   Docker Hub requires images to follow the naming format as :

   username/repository:tag

Therefore, the locally built image must be tagged with the Docker Hub username.

```
$docker tag <SOURCE_IMAGE> <USERNAME>/<REPOSITORY>:<TAG>

$docker tag spam-web-app-SRN <dockerhub-username>/spam-web-app-SRN:v1
```

Replace <dockerhub-username> with your actual Docker Hub username.

This command assigns a new name to the same image so that it can be uploaded to your Docker Hub account.

To verify the tagging, execute:

```
$ docker images
```

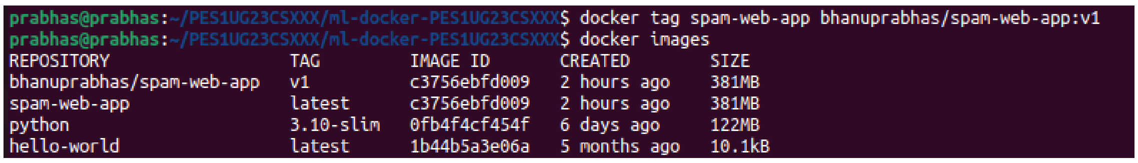
You should now see two entries pointing to the same image ID:

- spam-web-app-SRN
- <dockerhub-username>/spam-web-app-SRN:v1

This is your output **SS8**.

(Image name spam-web-app in the below SS , for you it should be spam-web-app-SRN)

Output SS8:



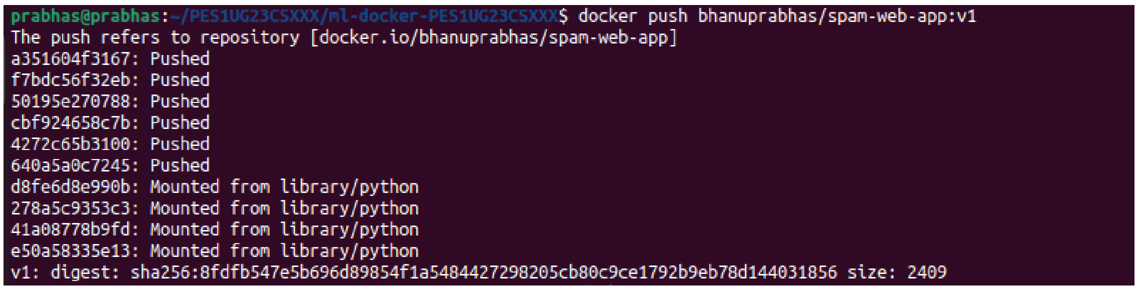
```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker tag spam-web-app bhanuprabhas/spam-web-app:v1
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker images
REPOSITORY                  TAG         IMAGE ID       CREATED         SIZE
bhanuprabhas/spam-web-app   v1          c3756ebfd009   2 hours ago     381MB
spam-web-app                latest      c3756ebfd009   2 hours ago     381MB
python                      3.10-slim   0fb4f4cf454f   6 days ago      122MB
hello-world                 latest      1b44b5a3e06a   5 months ago    10.1kB
```

3. Now push the tagged image to Docker Hub using the following command:

```
$ docker push <USERNAME>/<REPOSITORY>:<TAG>
$ docker push <dockerhub-username>/spam-web-app-SRN:v1
```

Docker will upload the image layers to Docker Hub.
Once the upload is complete, the terminal will display confirmation that all layers have been pushed successfully.
This is your output **SS9.**

Output SS9:



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker push bhanuprabhas/spam-web-app:v1
The push refers to repository [docker.io/bhanuprabhas/spam-web-app]
a351604f3167: Pushed
f7bdc56f32eb: Pushed
50195e270788: Pushed
cbf924658c7b: Pushed
4272c65b3100: Pushed
640a5a0c7245: Pushed
d8fe6d8e990b: Mounted from library/python
278a5c9353c3: Mounted from library/python
41a08778b9fd: Mounted from library/python
e50a58335e13: Mounted from library/python
v1: digest: sha256:8fdfb547e5b696d89854f1a5484427298205cb80c9ce1792b9eb78d144031856 size: 2409
```

4. Verify Image on Docker Hub website

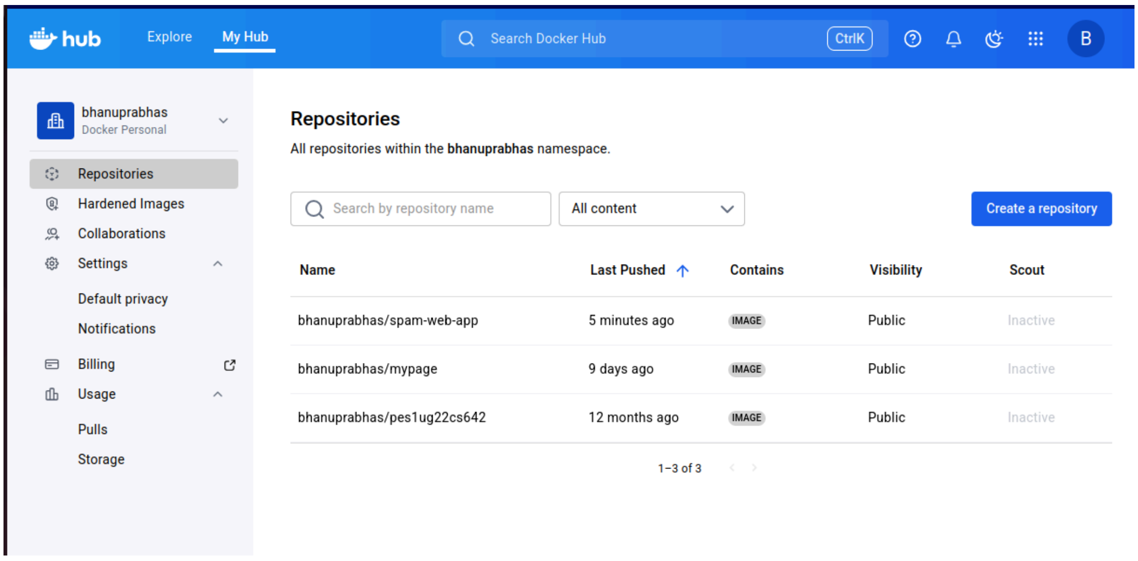
   Open a web browser and go to: https://hub.docker.com/

Login using your Docker Hub account and navigate to Repositories.
You should see the repository named spam-web-app-SRN with tag v1.

This confirms that the Docker image is now stored in the cloud and can be accessed from any machine.

This will be your output **SS10.**

Output SS10:



Repositories

All repositories within the **bhanuprabhas** namespace.

Name	Last Pushed	Contains	Visibility	Scout
bhanuprabhas/spam-web-app	5 minutes ago	IMAGE	Public	Inactive
bhanuprabhas/mypage	9 days ago	IMAGE	Public	Inactive
bhanuprabhas/pes1ug22cs642	12 months ago	IMAGE	Public	Inactive

1–3 of 3

Step 4: Stop Containers

1. First, list all currently running containers using the following command:

$docker ps

2. Stop the running container using below command:

   $docker stop <container_id>

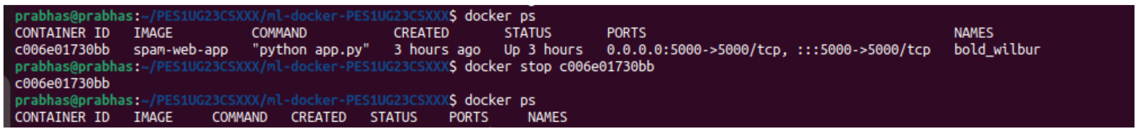
   (Replace <container_id> with the container ID obtained

   from the docker ps output.)

3. Now again verify using the same docker ps command.

   This will be your output **SS11**.

Output SS11:



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker ps
CONTAINER ID   IMAGE          COMMAND           CREATED       STATUS       PORTS                                         NAMES
c006e01730bb   spam-web-app   "python app.py"   3 hours ago   Up 3 hours   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   bold_wilbur
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker stop c006e01730bb
c006e01730bb
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Step 5: Delete Containers

1. Execute the following command to list all containers(this command displays both running containers and stopped containers as well):

   $docker ps -a

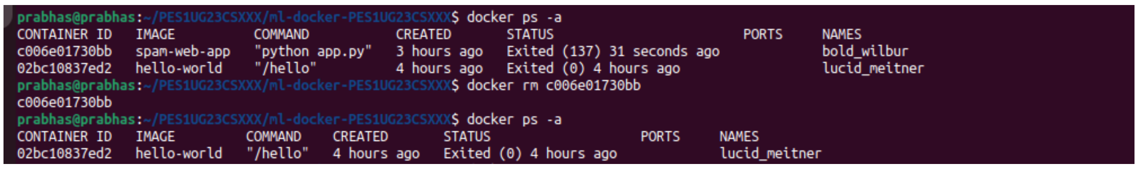
2. Remove the stopped container using the command below:

   $docker rm <container_id>

3. Now again verify using the same command docker ps -a

   This will be your output **SS12**.

Output SS12:



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker ps -a
CONTAINER ID   IMAGE          COMMAND           CREATED       STATUS                       PORTS     NAMES
c006e01730bb   spam-web-app   "python app.py"   3 hours ago   Exited (137) 31 seconds ago            bold_wilbur
02bc10837ed2   hello-world    "/hello"          4 hours ago   Exited (0) 4 hours ago                 lucid_meitner
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker rm c006e01730bb
c006e01730bb
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED       STATUS                   PORTS     NAMES
02bc10837ed2   hello-world   "/hello"   4 hours ago   Exited (0) 4 hours ago             lucid_meitner
```

Step 5: Delete Images locally

1. List all the images using the below command:

   $ docker images

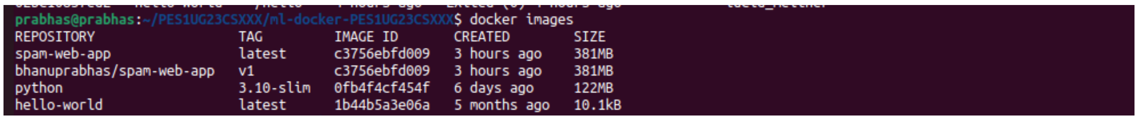
2. Delete the image you created using the command below:

$ docker rmi <image_name>

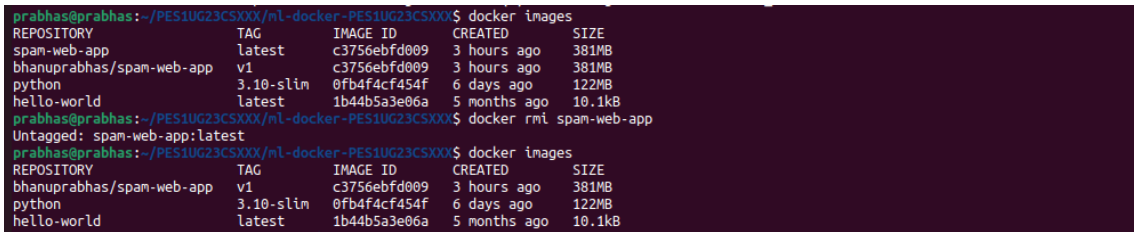
3. Now again verify using docker images command

   This will be your output **SS13**.

Output SS13:



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker images
REPOSITORY                   TAG         IMAGE ID       CREATED         SIZE
spam-web-app                 latest      c3756ebfd009   3 hours ago     381MB
bhanuprabhas/spam-web-app    v1          c3756ebfd009   3 hours ago     381MB
python                       3.10-slim   0fb4f4cf454f   6 days ago      122MB
hello-world                  latest      1b44b5a3e06a   5 months ago    10.1kB
```



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker images
REPOSITORY                   TAG         IMAGE ID       CREATED         SIZE
spam-web-app                 latest      c3756ebfd009   3 hours ago     381MB
bhanuprabhas/spam-web-app    v1          c3756ebfd009   3 hours ago     381MB
python                       3.10-slim   0fb4f4cf454f   6 days ago      122MB
hello-world                  latest      1b44b5a3e06a   5 months ago    10.1kB
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker rmi spam-web-app
Untagged: spam-web-app:latest
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker images
REPOSITORY                   TAG         IMAGE ID       CREATED         SIZE
bhanuprabhas/spam-web-app    v1          c3756ebfd009   3 hours ago     381MB
python                       3.10-slim   0fb4f4cf454f   6 days ago      122MB
hello-world                  latest      1b44b5a3e06a   5 months ago    10.1kB
```

Observe the output carefully , did the docker delete all the images with the same image id?? NO.

- Docker did NOT delete the image
- It only removed the **tag (name)** spam-web-app:latest

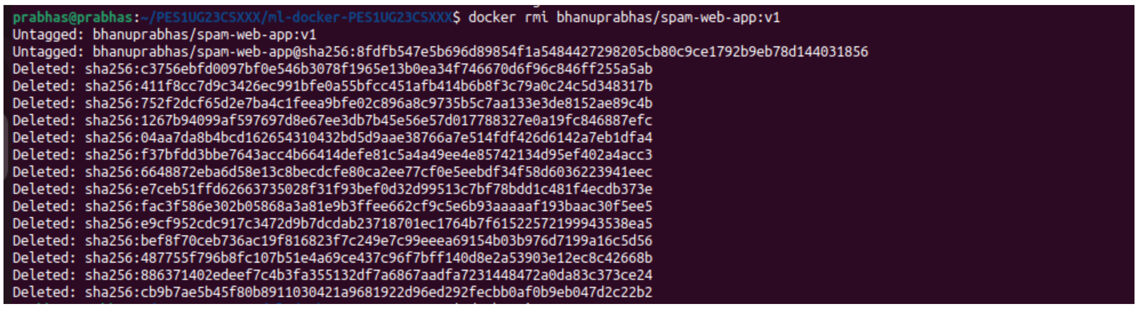
Because the image was still referenced by: bhanuprabhas/spam-web-app:v1

A single Docker image can have multiple tags associated with the same image ID. Removing one tag only deletes that reference to the image, but the actual image will not be removed from the system as long as at least one other tag still exists.

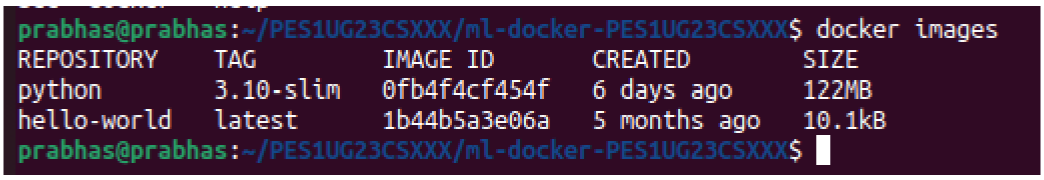
Now delete the other image as well using the same command and then verify using docker images

Together these both will be your output **SS14.**

Output SS14:



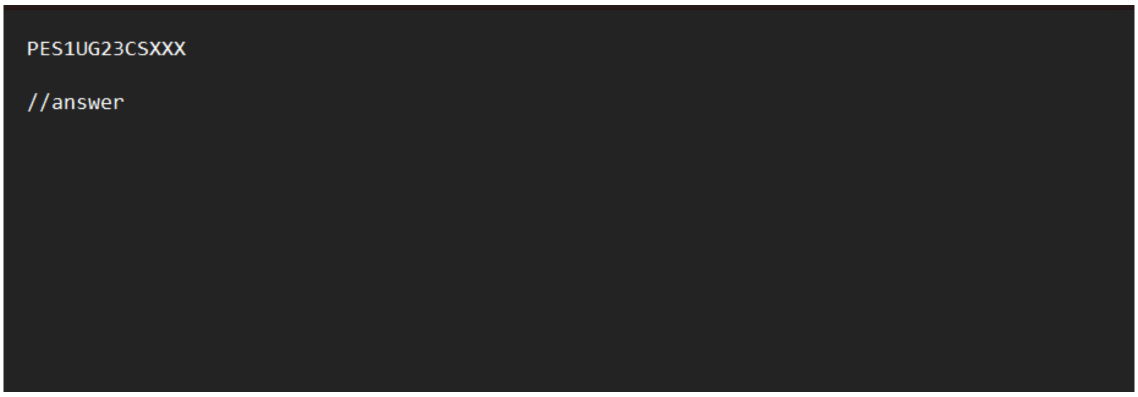
```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker rmi bhanuprabhas/spam-web-app:v1
Untagged: bhanuprabhas/spam-web-app:v1
Untagged: bhanuprabhas/spam-web-app@sha256:8fdfb547e5b696d89854f1a5484427298205cb80c9ce1792b9eb78d144031856
Deleted: sha256:c3756ebfd0097bf0e546b3078f1965e13b0ea34f746670d6f96c846ff255a5ab
Deleted: sha256:411f8cc7d9c3426ec991bfe0a55bfcc451afb414b6b8f3c79a0c24c5d348317b
Deleted: sha256:752f2dcf65d2e7ba4c1feea9bfe02c896a8c9735b5c7aa133e3de8152ae89c4b
Deleted: sha256:1267b94099af597697d8e67ee3db7b45e56e57d017788327e0a19fc846887efc
Deleted: sha256:04aa7da8b4bcd162654310432bd5d9aae38766a7e514fdf426d6142a7eb1dfa4
Deleted: sha256:f37bfdd3bbe7643acc4b66414defe81c5a4a49ee4e85742134d95ef402a4acc3
Deleted: sha256:6648872eba6d58e13c8becdcfe80ca2ee77cf0e5eebdf34f58d6036223941eec
Deleted: sha256:e7ceb51ffd62663735028f31f93bef0d32d99513c7bf78bdd1c481f4ecdb373e
Deleted: sha256:fac3f586e302b05868a3a81e9b3ffee662cf9c5e6b93aaaaaf193baac30f5ee5
Deleted: sha256:e9cf952cdc917c3472d9b7dcdab23718701ec1764b7f61522572199943538ea5
Deleted: sha256:bef8f70ceb736ac19f816823f7c249e7c99eeea69154b03b976d7199a16c5d56
Deleted: sha256:487755f796b8fc107b51e4a69ce437c96f7bff140d8e2a53903e12ec8c42668b
Deleted: sha256:886371402edeef7c4b3fa355132df7a6867aadfa7231448472a0da83c373ce24
Deleted: sha256:cb9b7ae5b45f80b8911030421a9681922d96ed292fecbb0af0b9eb047d2c22b2
```



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker images
REPOSITORY    TAG         IMAGE ID       CREATED         SIZE
python        3.10-slim   0fb4f4cf454f   6 days ago      122MB
hello-world   latest      1b44b5a3e06a   5 months ago    10.1kB
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$
```

TO THINK : Is there any other command to delete images completely in one go? Explore the above question , type the answer in the notepad and take a screenshot of it. This will be your **SS14**.

Output SS14: This will be the SS of answer to the above question , make sure before you answer in the first line add your SRN and then continue in the next line with the answer.



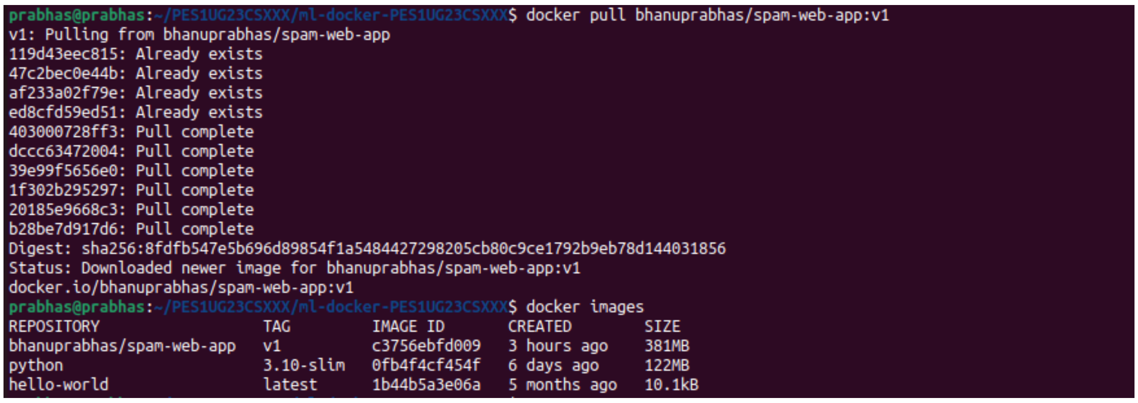
```
PES1UG23CSXXX

//answer
```

Step 6: Pull Previously pushed image from Docker Hub

1. Here in this step , you need to pull the image from DockerHub which you pushed earlier using command below:
   $docker pull <image_name>

This will be your output **SS15**.

Output SS15:



```
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker pull bhanuprabhas/spam-web-app:v1
v1: Pulling from bhanuprabhas/spam-web-app
119d43eec815: Already exists
47c2bec0e44b: Already exists
af233a02f79e: Already exists
ed8cfd59ed51: Already exists
403000728ff3: Pull complete
dccc63472004: Pull complete
39e99f5656e0: Pull complete
1f302b295297: Pull complete
20185e9668c3: Pull complete
b28be7d917d6: Pull complete
Digest: sha256:8fdfb547e5b696d89854f1a5484427298205cb80c9ce1792b9eb78d144031856
Status: Downloaded newer image for bhanuprabhas/spam-web-app:v1
docker.io/bhanuprabhas/spam-web-app:v1
prabhas@prabhas:~/PES1UG23CSXXX/ml-docker-PES1UG23CSXXX$ docker images
REPOSITORY                  TAG         IMAGE ID       CREATED        SIZE
bhanuprabhas/spam-web-app   v1          c3756ebfd009   3 hours ago    381MB
python                      3.10-slim   0fb4f4cf454f   6 days ago     122MB
hello-world                 latest      1b44b5a3e06a   5 months ago   10.1kB
```