EX NO: 9a IT22711-DISTRIBUTED AND CLOUD COMPUTING LABORATORY

DATE:

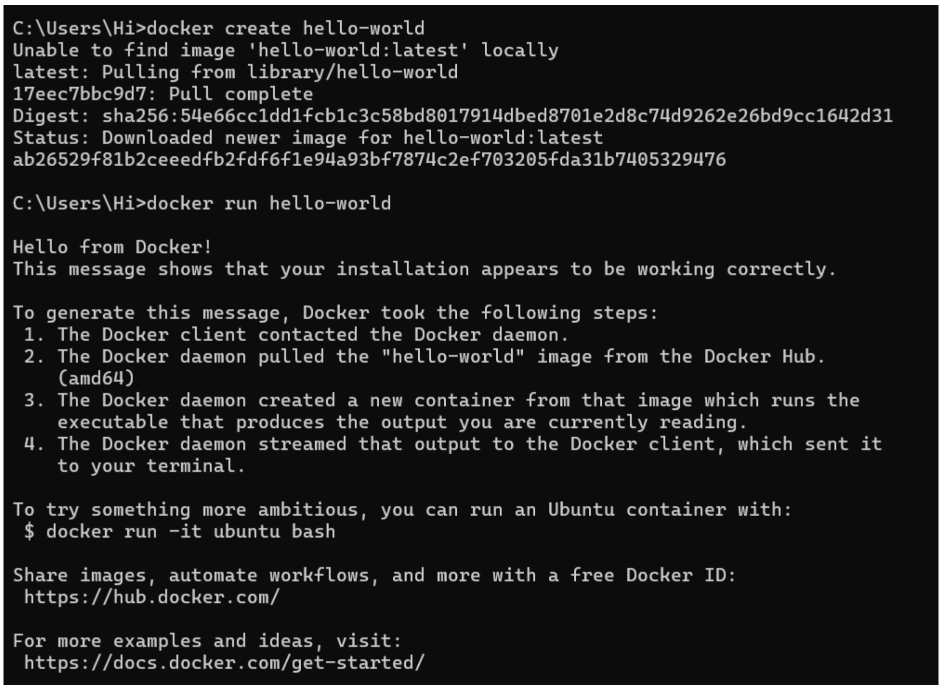
Docker commands

AIM:

To install Docker Desktop and execute the Docker commands.

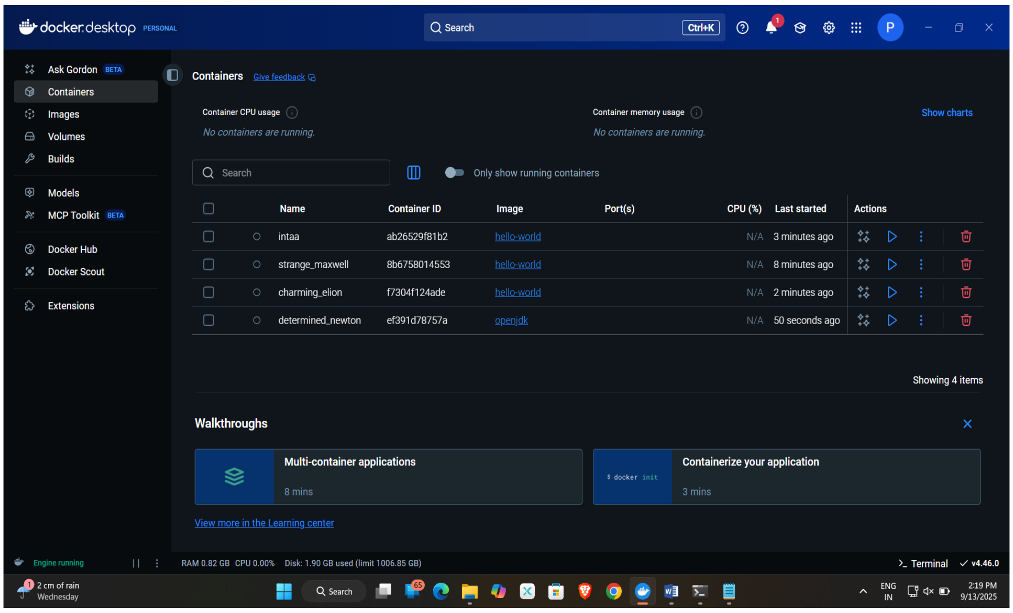
PROCEDURE:

1. Install docker desktop
2. In settings, enable the docker to run once the system is turned on, else you will have error
3. Execute the create and run docker hello-world commands



```
C:\Users\Hi>docker create hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
Digest: sha256:54e66cc1dd1fcb1c3c58bd8017914dbed8701e2d8c74d9262e26bd9cc1642d31
Status: Downloaded newer image for hello-world:latest
ab26529f81b2ceeedfb2fdf6f1e94a93bf7874c2ef703205fda31b7405329476

C:\Users\Hi>docker run hello-world

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

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

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

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

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

4. Check the status of the containers and images in docker desktop



Containers

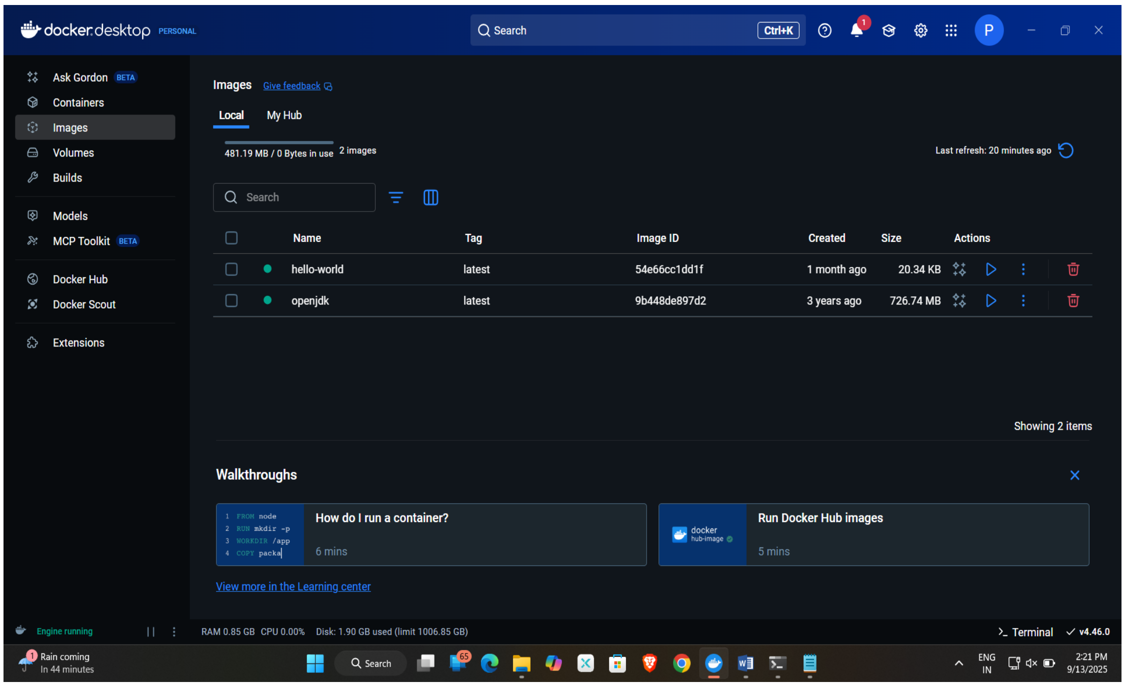
Container CPU usage — No containers are running.

Container memory usage — No containers are running.

Name	Container ID	Image	Port(s)	CPU (%)	Last started
intaa	ab26529f81b2	hello-world		N/A	3 minutes ago
strange_maxwell	8b6758014553	hello-world		N/A	8 minutes ago
charming_elion	f7304f124ade	hello-world		N/A	2 minutes ago
determined_newton	ef391d78757a	openjdk		N/A	50 seconds ago

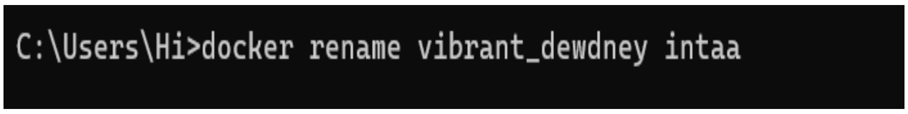
Showing 4 items

Engine running   RAM 0.82 GB CPU 0.00% Disk: 1.90 GB used (limit 1006.85 GB)   v4.46.0

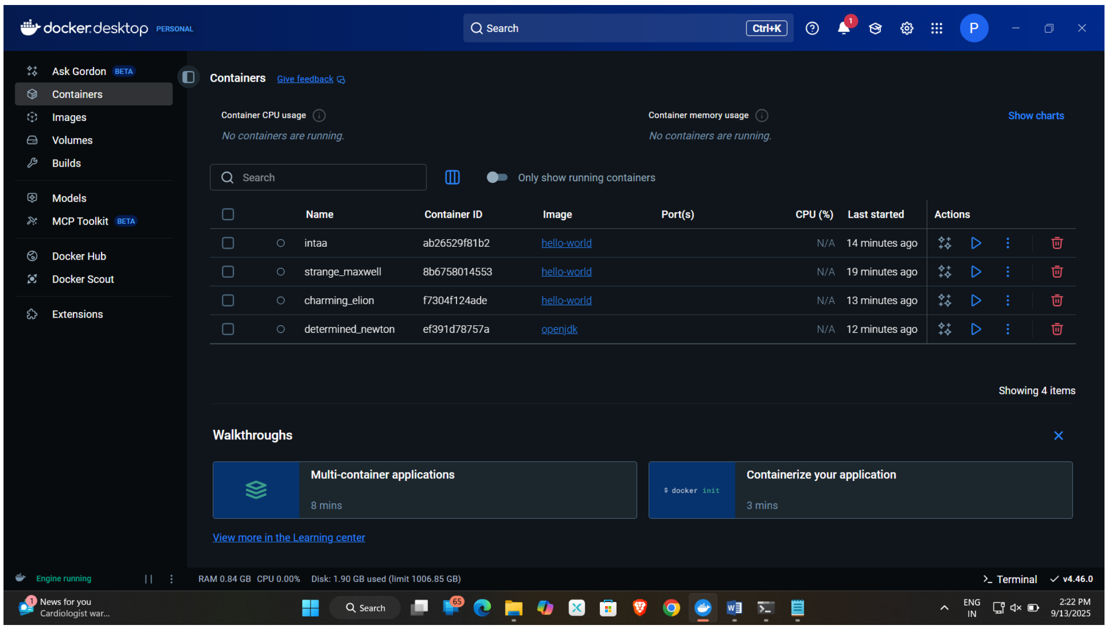


Name	Tag	Image ID	Created	Size	Actions
hello-world	latest	54e66cc1dd1f	1 month ago	20.34 KB	
openjdk	latest	9b448de897d2	3 years ago	726.74 MB	

5. Rename any of the container using commands

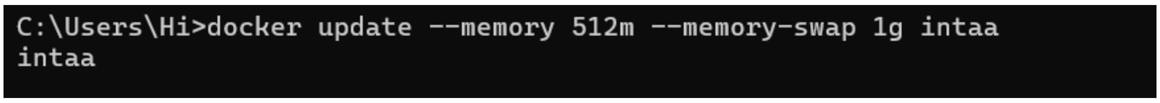


```
C:\Users\Hi>docker rename vibrant_dewdney intaa
```



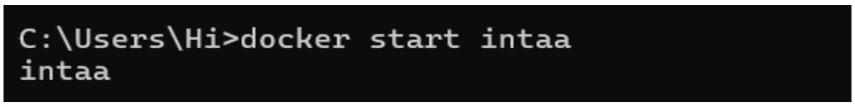
Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
intaa	ab26529f81b2	hello-world		N/A	14 minutes ago	
strange_maxwell	8b6758014553	hello-world		N/A	19 minutes ago	
charming_elion	f7304f124ade	hello-world		N/A	13 minutes ago	
determined_newton	ef391d78757a	openjdk		N/A	12 minutes ago	

6. Now update the memory and memory swap space for the container intaa using the command docker update --memory 512m --memory-swap 1g intaa



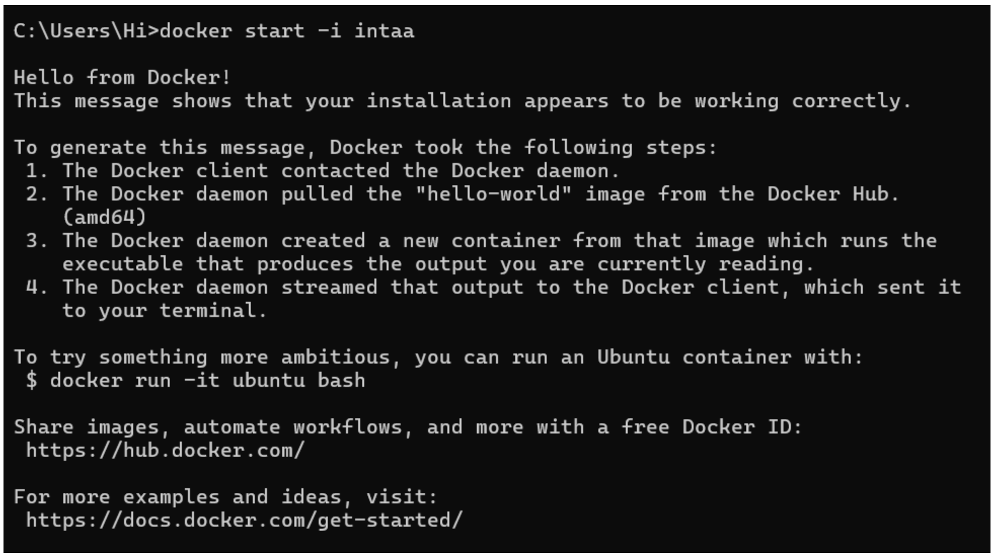
```
C:\Users\Hi>docker update --memory 512m --memory-swap 1g intaa
intaa
```

7. Start the docker using docker start intaa command



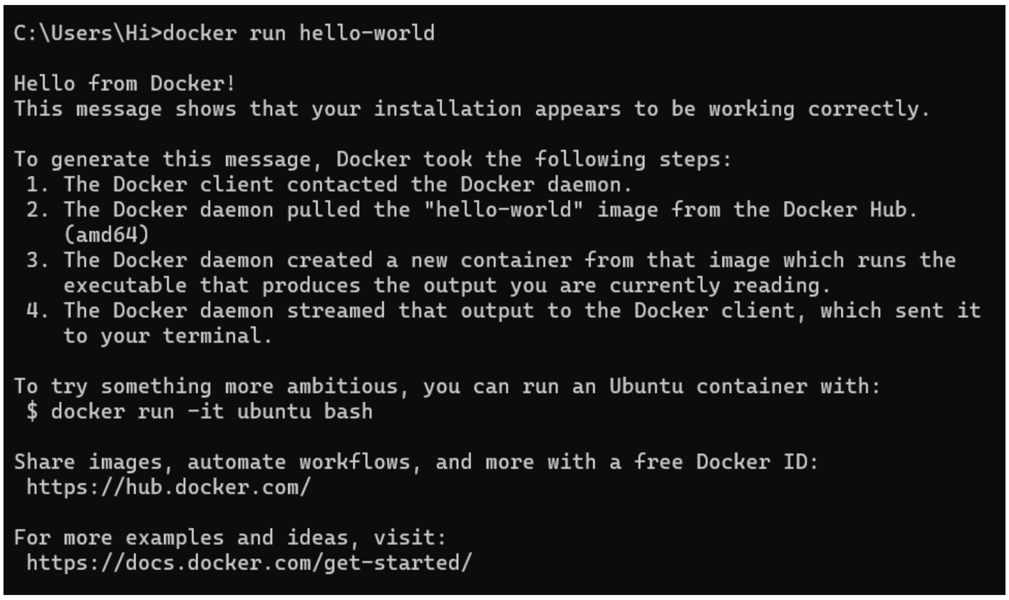
```
C:\Users\Hi>docker start intaa
intaa
```

8. Now execute the command docker start -i intaa to start the docker in interactive mode



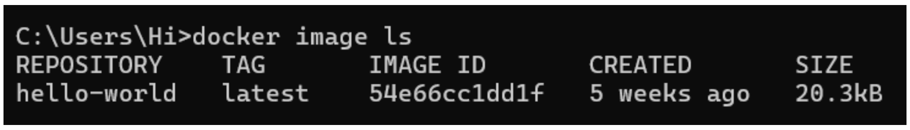
```
C:\Users\Hi>docker start -i intaa

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/
```

9. To check if the docker is installed correctly, execute the command docker run hello-world



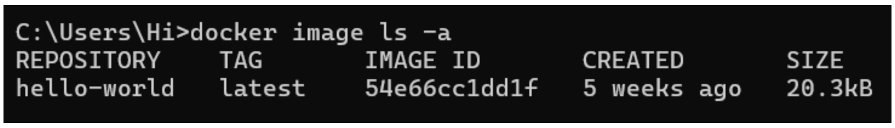
```
C:\Users\Hi>docker run hello-world

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

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

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

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

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

10. Execute the command docker image ls to check the status of docker images



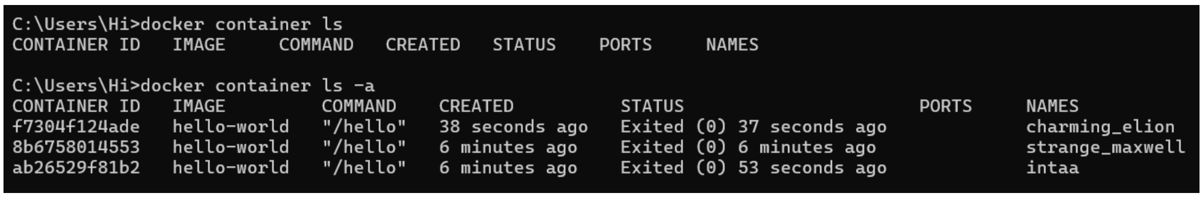
```
C:\Users\Hi>docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
hello-world   latest    54e66cc1dd1f   5 weeks ago   20.3kB
```

11. Execute the command docker image ls -a to check the status of docker images that are active and Inactive



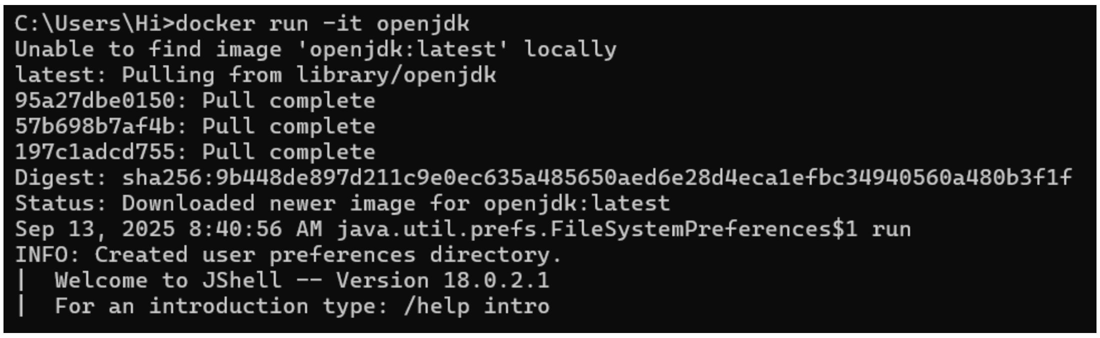
```
C:\Users\Hi>docker image ls -a
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
hello-world   latest    54e66cc1dd1f   5 weeks ago   20.3kB
```

12. Similarly execute docker container ls and docker container ls -a to check the status of the containers



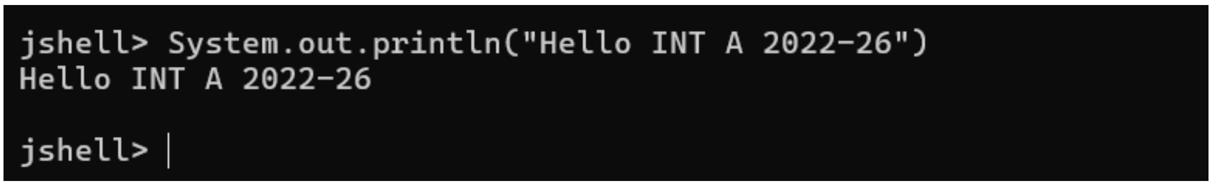
```
C:\Users\Hi>docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\Hi>docker container ls -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
f7304f124ade   hello-world   "/hello"   38 seconds ago   Exited (0) 37 seconds ago             charming_elion
8b6758014553   hello-world   "/hello"   6 minutes ago    Exited (0) 6 minutes ago              strange_maxwell
ab26529f81b2   hello-world   "/hello"   6 minutes ago    Exited (0) 53 seconds ago             intaa
```

13. Running containers interactively allows you to run commands inside the container if it supports it. We can use the openjdk image. This allows us to execute java commands line by line in a Java shell



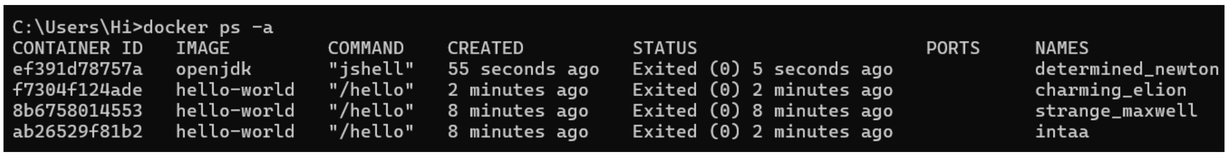
```
C:\Users\Hi>docker run -it openjdk
Unable to find image 'openjdk:latest' locally
latest: Pulling from library/openjdk
95a27dbe0150: Pull complete
57b698b7af4b: Pull complete
197c1adcd755: Pull complete
Digest: sha256:9b448de897d211c9e0ec635a485650aed6e28d4eca1efbc34940560a480b3f1f
Status: Downloaded newer image for openjdk:latest
Sep 13, 2025 8:40:56 AM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
|  Welcome to JShell -- Version 18.0.2.1
|  For an introduction type: /help intro
```

14. Now execute a simple SOP code in jshell. After execution of the code, to terminate the jshell, press Ctrl+D



```
jshell> System.out.println("Hello INT A 2022-26")
Hello INT A 2022-26

jshell> |
```

15. Execute the docker ps -a command to know the information of all the containers and images



```
C:\Users\Hi>docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                     PORTS     NAMES
ef391d78757a   openjdk       "jshell"   55 seconds ago   Exited (0) 5 seconds ago             determined_newton
f7304f124ade   hello-world   "/hello"   2 minutes ago    Exited (0) 2 minutes ago             charming_elion
8b6758014553   hello-world   "/hello"   8 minutes ago    Exited (0) 8 minutes ago             strange_maxwell
ab26529f81b2   hello-world   "/hello"   8 minutes ago    Exited (0) 2 minutes ago             intaa
```

RESULT:

Thus, the installation and execution of Docker commands is done successfully.

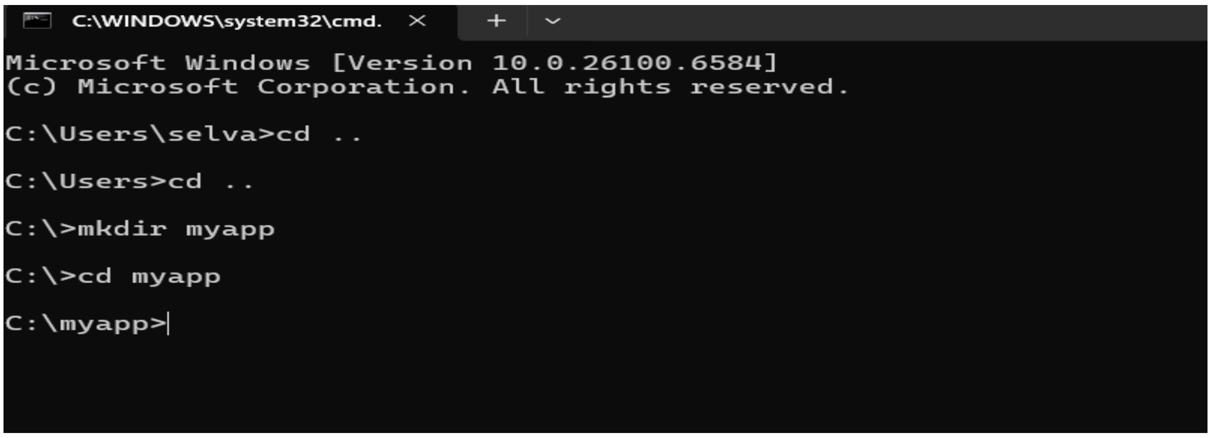
Create a new project and build an image in Docker

AIM:

To create a new project in Docker and build an image in it.

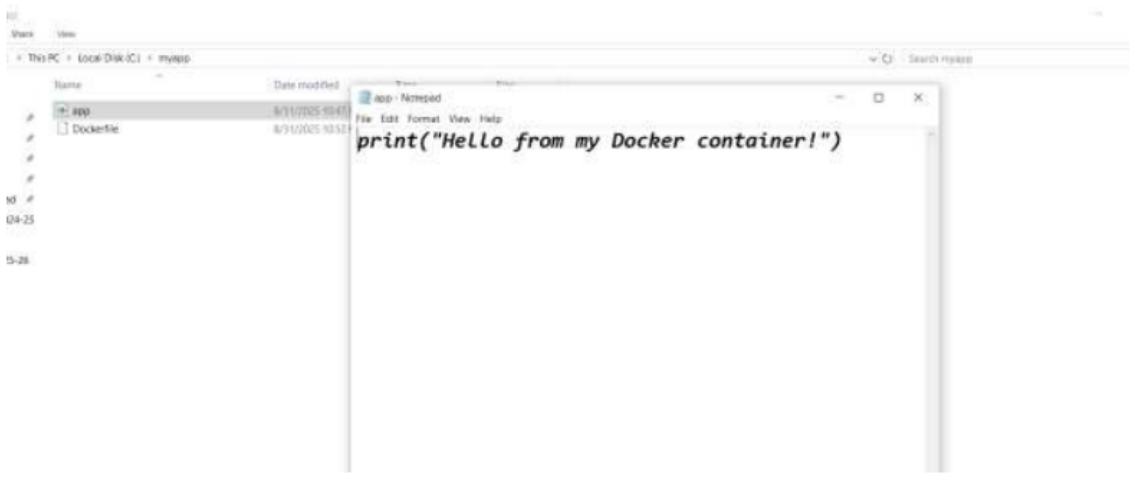
PROCEDURE:

1. Create and access a directory myapp using the commands mkdir and cd



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\selva>cd ..

C:\Users>cd ..

C:\>mkdir myapp

C:\>cd myapp

C:\myapp>
```

2. Create a python file named app.py in notepad with extension all files inside my app folder, write the print statement as “hello, this is docker container”



```
print("Hello from my Docker container!")
```

3. Next create a docker file as txt file with the name Dockerfile, inside this file write the following commands

FROM python:3.9-slim //official python runtime as parent image

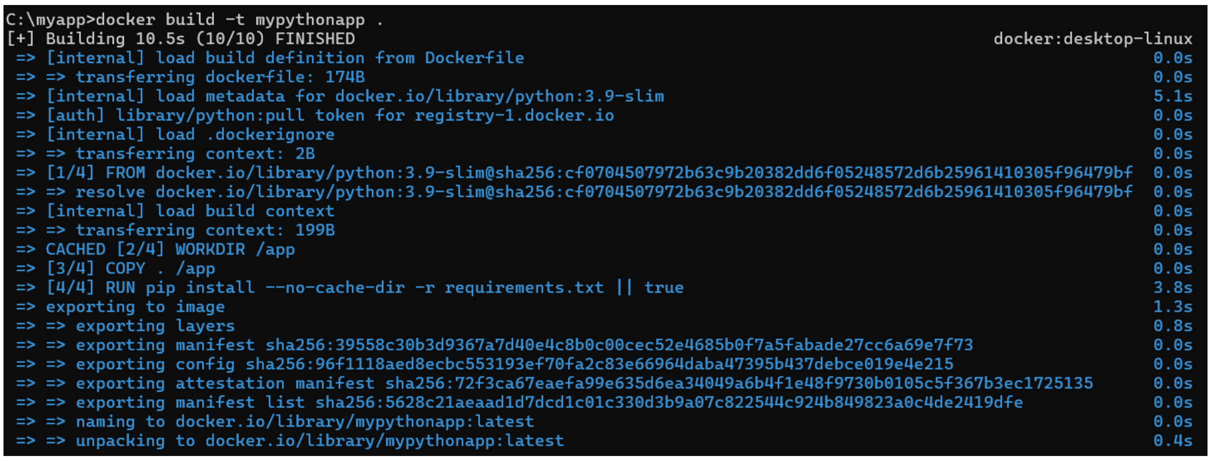
WORKDIR /app //Set the working directory inside the container

COPY . /app //copy the current directory contents into the container

RUN pip install --no-cache-dir -r requirements.txt || true //install dependencies(if you have the requirements text)

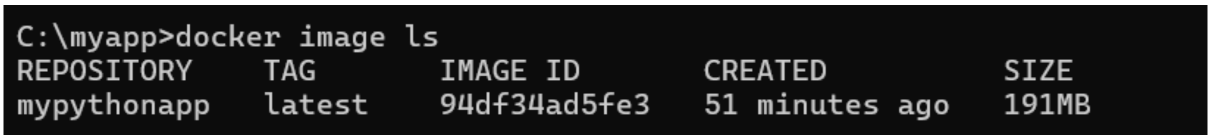
```
CMD ["python", "app.py"] //run your app
```

4. Open the command prompt inside the myapp directory, rename the docker text to docker file using the below command **ren Dockerfile Dockerfile.txt**
5. Build an image from the docker file by executing the command **docker build -t mypythonapp .** where mypythonapp is the image name which has to be created



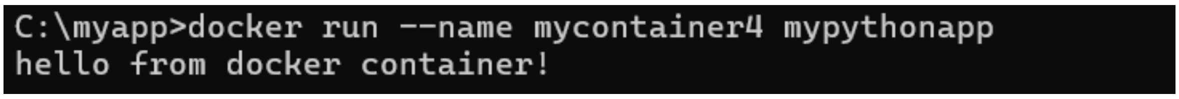
```
C:\myapp>docker build -t mypythonapp .
[+] Building 10.5s (10/10) FINISHED                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                   0.0s
 => => transferring dockerfile: 174B                                                                   0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                     5.1s
 => [auth] library/python:pull token for registry-1.docker.io                                          0.0s
 => [internal] load .dockerignore                                                                      0.0s
 => => transferring context: 2B                                                                        0.0s
 => [1/4] FROM docker.io/library/python:3.9-slim@sha256:cf0704507972b63c9b20382dd6f05248572d6b25961410305f96479bf  0.0s
 => => resolve docker.io/library/python:3.9-slim@sha256:cf0704507972b63c9b20382dd6f05248572d6b25961410305f96479bf  0.0s
 => [internal] load build context                                                                      0.0s
 => => transferring context: 199B                                                                      0.0s
 => CACHED [2/4] WORKDIR /app                                                                          0.0s
 => [3/4] COPY . /app                                                                                  0.0s
 => [4/4] RUN pip install --no-cache-dir -r requirements.txt || true                                   3.8s
 => exporting to image                                                                                 1.3s
 => => exporting layers                                                                                0.8s
 => => exporting manifest sha256:39558c30b3d9367a7d40e4c8b0c00cec52e4685b0f7a5fabade27cc6a69e7f73      0.0s
 => => exporting config sha256:96f1118aed8ecbc553193ef70fa2c83e66964daba47395b437debce019e4e215        0.0s
 => => exporting attestation manifest sha256:72f3ca67eaefa99e635d6ea34049a6b4f1e48f9730b0105c5f367b3ec1725135  0.0s
 => => exporting manifest list sha256:5628c21aeaad1d7dcd1c01c330d3b9a07c822544c924b849823a0c4de2419dfe  0.0s
 => => naming to docker.io/library/mypythonapp:latest                                                  0.0s
 => => unpacking to docker.io/library/mypythonapp:latest                                               0.4s
```

6. After creating an image successfully view the image that has been created with **docker ls** command



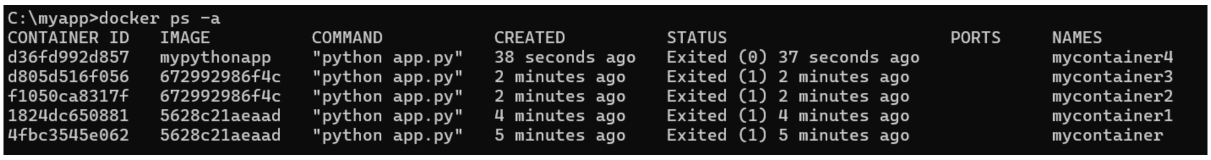
```
C:\myapp>docker image ls
REPOSITORY     TAG       IMAGE ID       CREATED          SIZE
mypythonapp    latest    94df34ad5fe3   51 minutes ago   191MB
```

7. To run the container execute **docker run –name mycontainer mypythonapp**



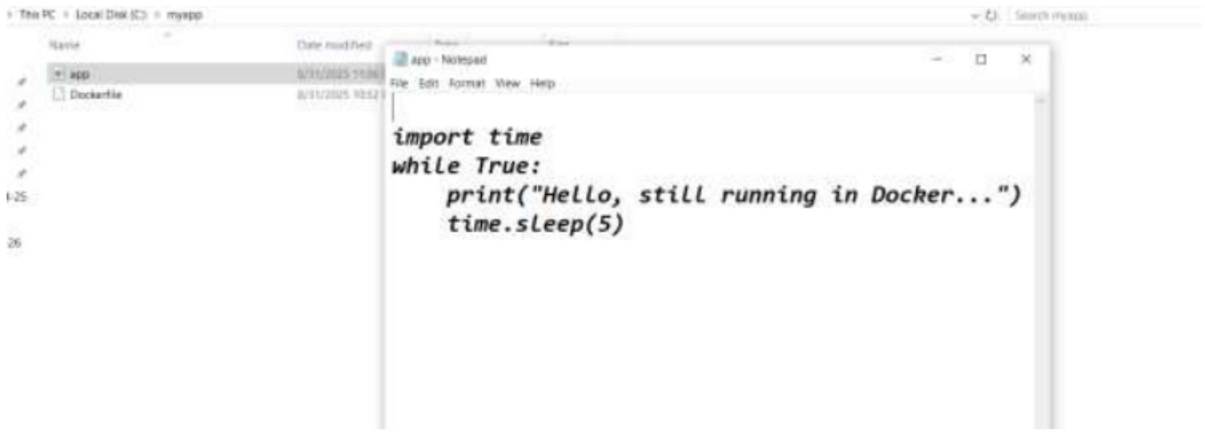
```
C:\myapp>docker run --name mycontainer4 mypythonapp
hello from docker container!
```

8. Execute the **docker ps -a** command to know the information of all the containers and images



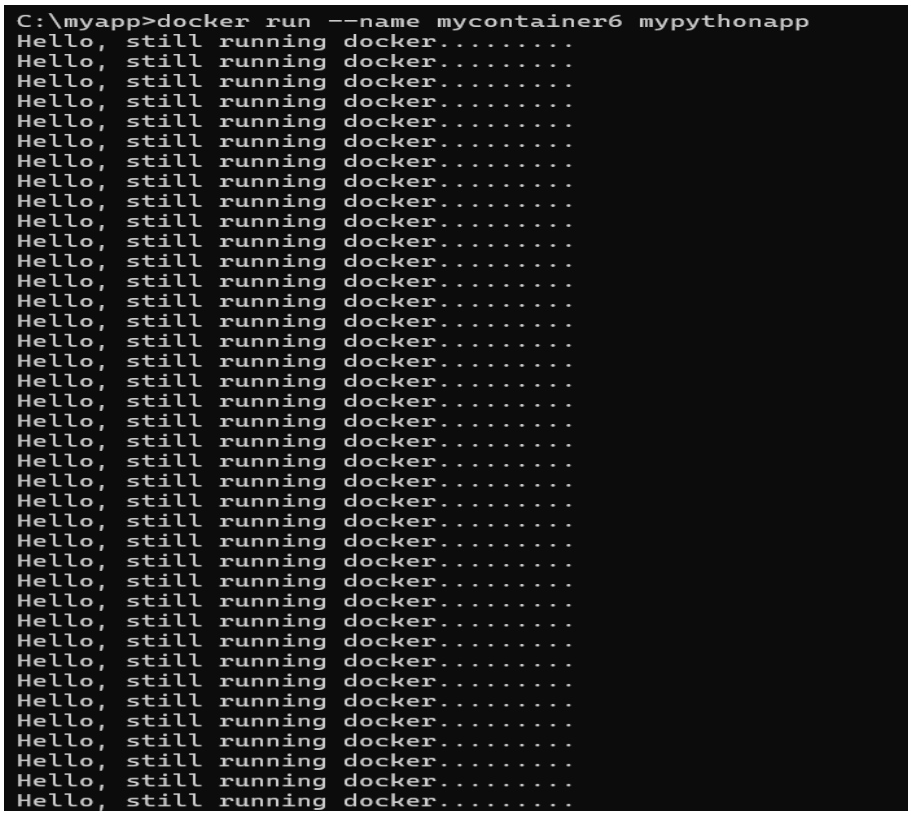
```
C:\myapp>docker ps -a
CONTAINER ID   IMAGE          COMMAND           CREATED          STATUS                       PORTS     NAMES
d36fd992d857   mypythonapp    "python app.py"   38 seconds ago   Exited (0) 37 seconds ago              mycontainer4
d805d516f056   672992986f4c   "python app.py"   2 minutes ago    Exited (1) 2 minutes ago               mycontainer3
f1050ca8317f   672992986f4c   "python app.py"   2 minutes ago    Exited (1) 2 minutes ago               mycontainer2
1824dc650881   5628c21aeaad   "python app.py"   4 minutes ago    Exited (1) 4 minutes ago               mycontainer1
4fbc3545e062   5628c21aeaad   "python app.py"   5 minutes ago    Exited (1) 5 minutes ago               mycontainer
```

9. Now modify the app.py to run the program continuously instead of running once



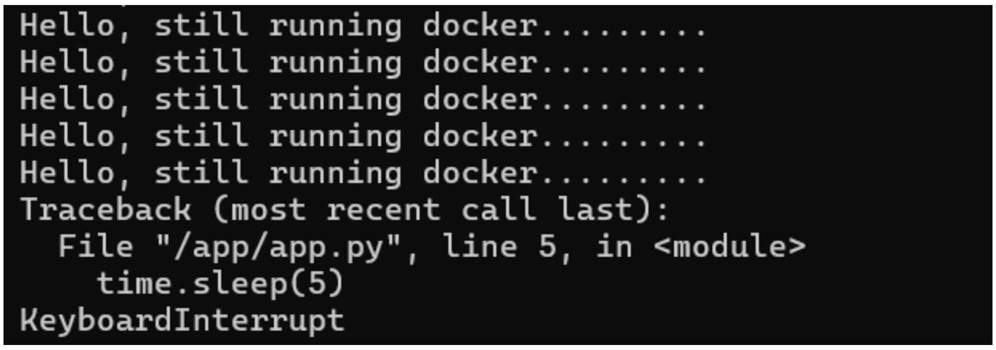
```
import time
while True:
    print("Hello, still running in Docker...")
    time.sleep(5)
```

10. Next rebuild the image using same command as docker build -t mypythonapp, after this execute the **docker run --name mycontainer1**(change the container name to 1 or 2 since container already exists)



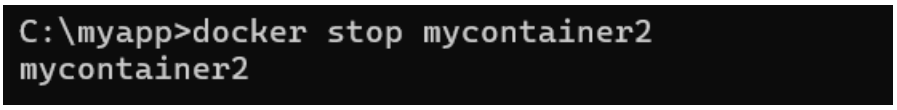
```
C:\myapp>docker run --name mycontainer6 mypythonapp
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
```

11. After execution, give ctrl+c to stop the running



```
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Hello, still running docker.........
Traceback (most recent call last):
  File "/app/app.py", line 5, in <module>
    time.sleep(5)
KeyboardInterrupt
```

12. Execute **docker stop mycontainer2** to stop the container



```
C:\myapp>docker stop mycontainer2
mycontainer2
```

RESULT:

Thus, the creation of new project and building an image in Docker is done successfully.

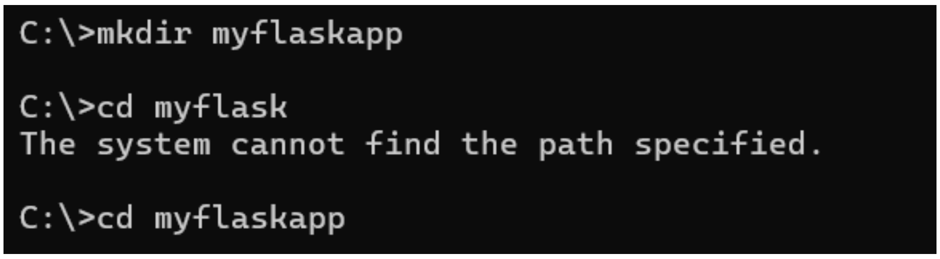
Simple working flask application in Docker

AIM:

To create a simple working flask application in Docker.

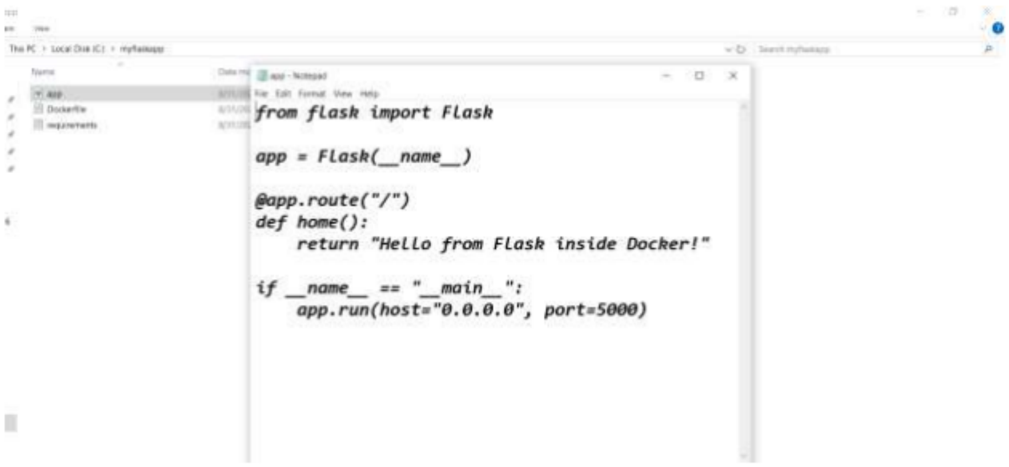
PROCEDURE:

1. Create and access a directory myflaskapp using the commands mkdir and cd



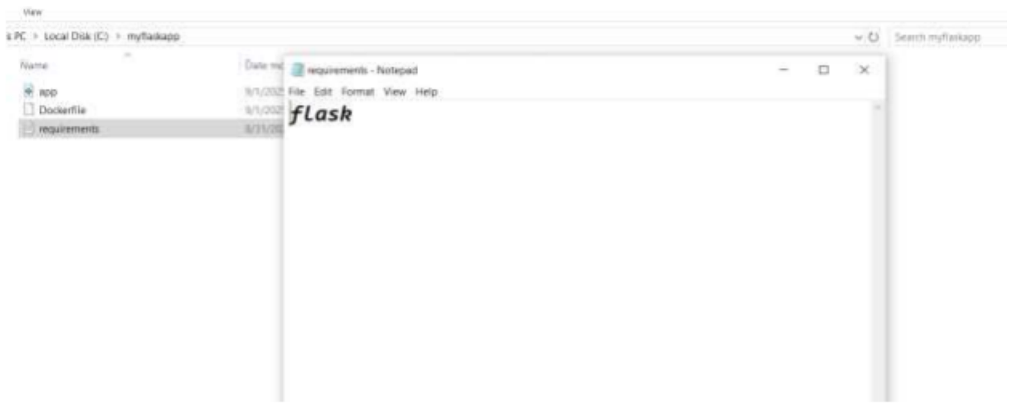
```
C:\>mkdir myflaskapp

C:\>cd myflask
The system cannot find the path specified.

C:\>cd myflaskapp
```

2. Create a python file named app.py in notepad with extension all files inside my app folder, type the below code



```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello from Flask inside Docker!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

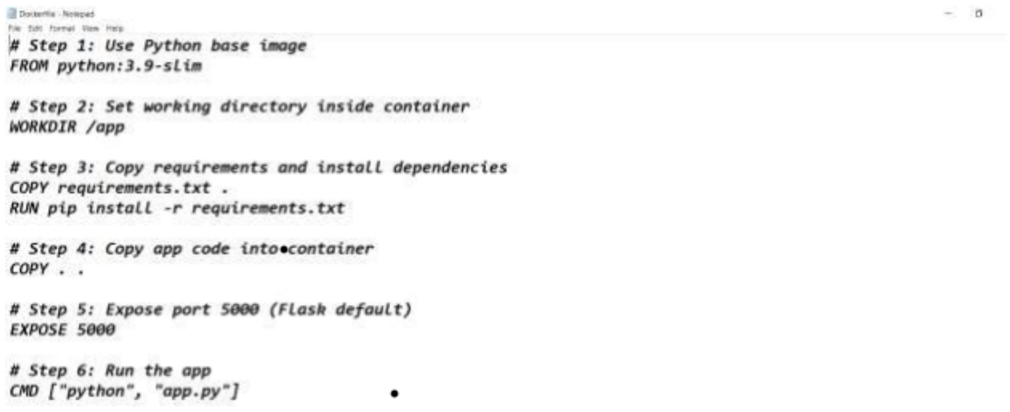
3. Next create a text file named requirements and type **flask** in it



```
flask
```

4. Next create a docker file as txt file with the name Dockerfile, inside this file write the following commands

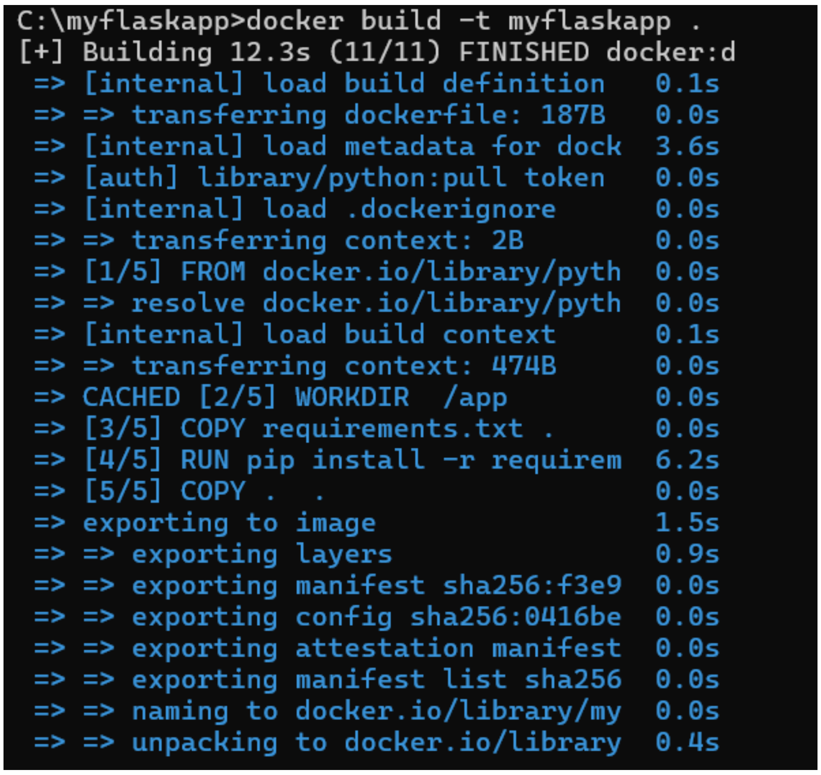
```
FROM python:3.9-slim
WORKDIR  /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY .  .
EXPOSE 5000
CMD ["python",  "app.py"]
```



```
# Step 1: Use Python base image
FROM python:3.9-slim

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

# Step 3: Copy requirements and install dependencies
COPY requirements.txt .
RUN pip install -r requirements.txt

# Step 4: Copy app code into container
COPY . .

# Step 5: Expose port 5000 (Flask default)
EXPOSE 5000

# Step 6: Run the app
CMD ["python", "app.py"]
```

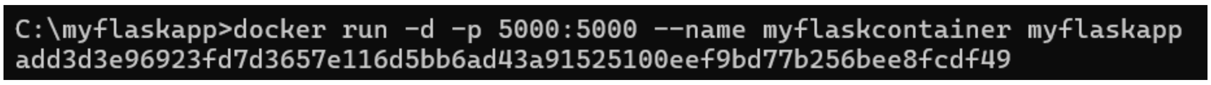
5. Open the command prompt inside the myapp directory, rename the docker text to docker file using the below command **ren Dockerfile.txt Dockerfile**

6. Build an image from the docker file by executing the command **docker build -t mypythonapp .** where mypythonapp is the image name which has to be created

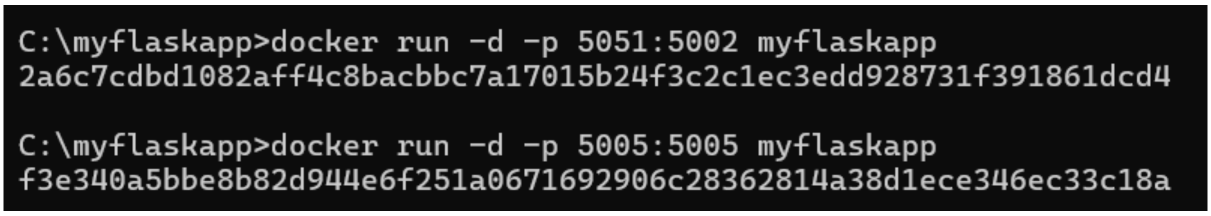


```
C:\myflaskapp>docker build -t myflaskapp .
[+] Building 12.3s (11/11) FINISHED docker:d
 => [internal] load build definition   0.1s
 => => transferring dockerfile: 187B   0.0s
 => [internal] load metadata for dock  3.6s
 => [auth] library/python:pull token   0.0s
 => [internal] load .dockerignore      0.0s
 => => transferring context: 2B        0.0s
 => [1/5] FROM docker.io/library/pyth  0.0s
 => => resolve docker.io/library/pyth  0.0s
 => [internal] load build context      0.1s
 => => transferring context: 474B      0.0s
 => CACHED [2/5] WORKDIR  /app         0.0s
 => [3/5] COPY requirements.txt .      0.0s
 => [4/5] RUN pip install -r requirem  6.2s
 => [5/5] COPY .  .                    0.0s
 => exporting to image                 1.5s
 => => exporting layers                0.9s
 => => exporting manifest sha256:f3e9  0.0s
 => => exporting config sha256:0416be  0.0s
 => => exporting attestation manifest  0.0s
 => => exporting manifest list sha256  0.0s
 => => naming to docker.io/library/my  0.0s
 => => unpacking to docker.io/library  0.4s
```

7. Now, run the container in detached mode. A container is a running instance of the image. -p 5000:5000 maps host port 5000 → container port 5000 (format: HOST:CONTAINER).

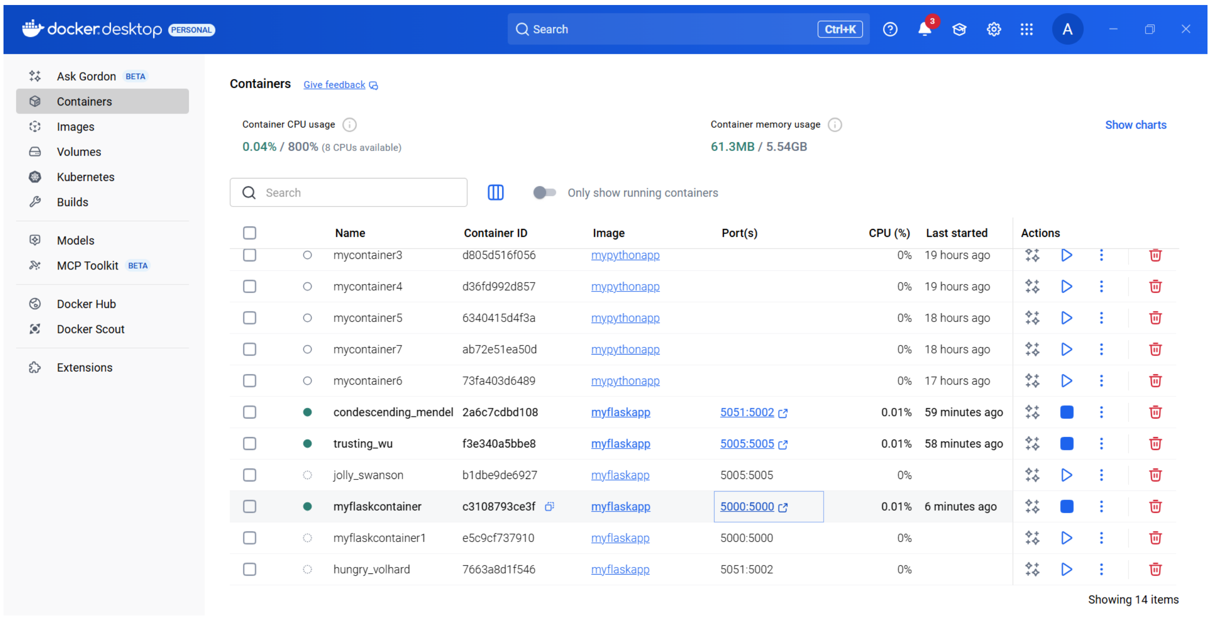


```
C:\myflaskapp>docker run -d -p 5000:5000 --name myflaskcontainer myflaskapp
add3d3e96923fd7d3657e116d5bb6ad43a91525100eef9bd77b256bee8fcdf49
```



```
C:\myflaskapp>docker run -d -p 5051:5002 myflaskapp
2a6c7cdbd1082aff4c8bacbbc7a17015b24f3c2c1ec3edd928731f391861dcd4

C:\myflaskapp>docker run -d -p 5005:5005 myflaskapp
f3e340a5bbe8b82d944e6f251a0671692906c28362814a38d1ece346ec33c18a
```

8. To run the app, go to docker check for the myflaskcontainer in this list and give run to see the output in the server localhost



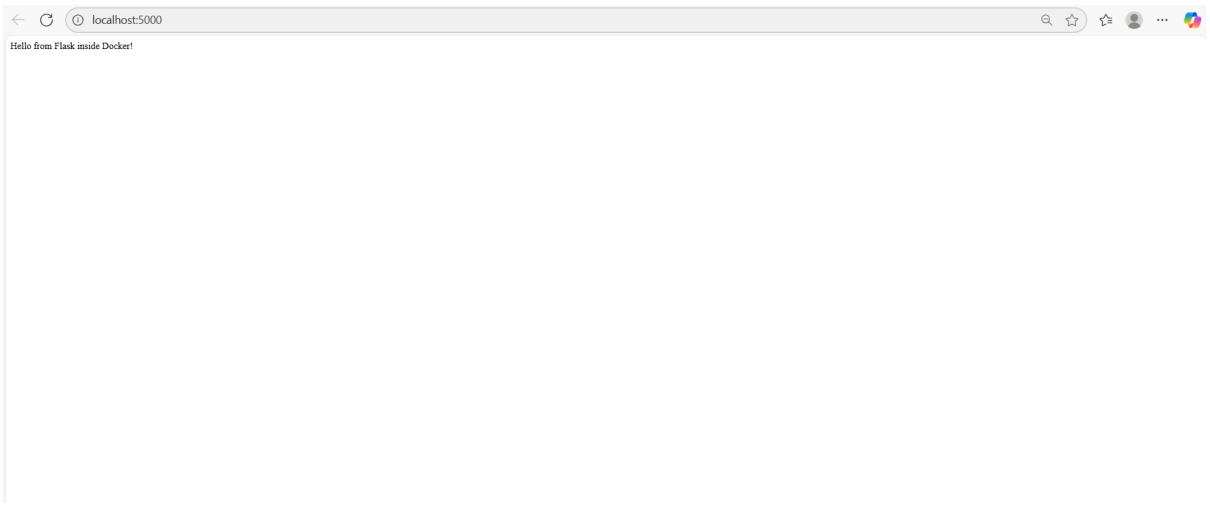
Containers

Container CPU usage
0.04% / 800% (8 CPUs available)

Container memory usage
61.3MB / 5.54GB

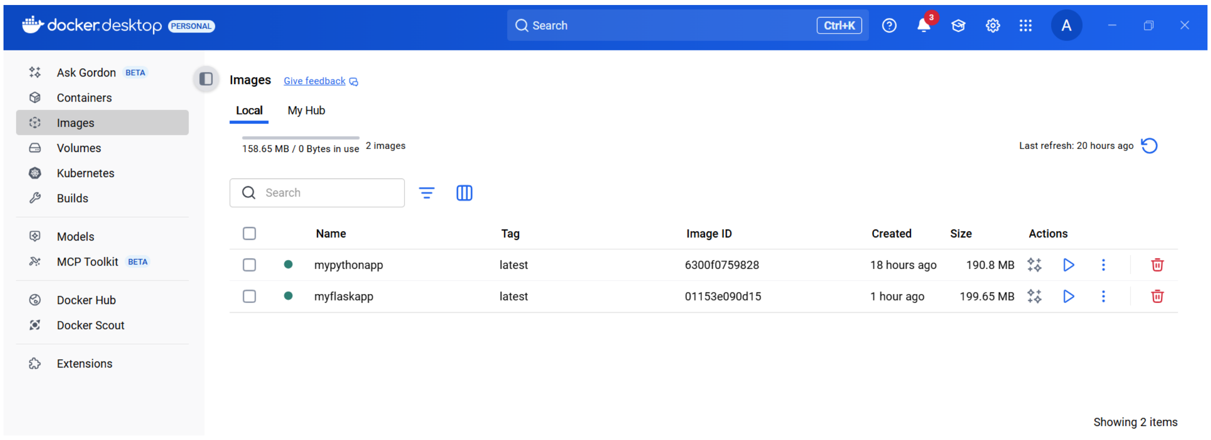
	Name	Container ID	Image	Port(s)	CPU (%)	Last started
○	mycontainer3	d805d516f056	mypythonapp		0%	19 hours ago
○	mycontainer4	d36fd992d857	mypythonapp		0%	19 hours ago
○	mycontainer5	6340415d4f3a	mypythonapp		0%	18 hours ago
○	mycontainer7	ab72e51ea50d	mypythonapp		0%	18 hours ago
○	mycontainer6	73fa403d6489	mypythonapp		0%	17 hours ago
●	condescending_mendel	2a6c7cdbd108	myflaskapp	5051:5002	0.01%	59 minutes ago
●	trusting_wu	f3e340a5bbe8	myflaskapp	5005:5005	0.01%	58 minutes ago
○	jolly_swanson	b1dbe9de6927	myflaskapp	5005:5005	0%	
●	myflaskcontainer	c3108793ce3f	myflaskapp	5000:5000	0.01%	6 minutes ago
○	myflaskcontainer1	e5c9cf737910	myflaskapp	5000:5000	0%	
○	hungry_volhard	7663a8d1f546	myflaskapp	5051:5002	0%	

Showing 14 items



localhost:5000

Hello from Flask inside Docker!



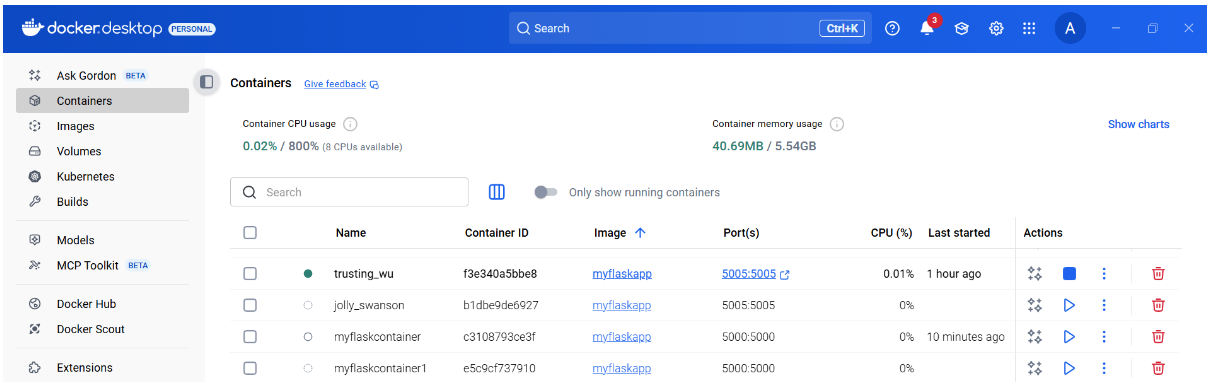
158.65 MB / 0 Bytes in use 2 images

Last refresh: 20 hours ago

Name	Tag	Image ID	Created	Size
mypythonapp	latest	6300f0759828	18 hours ago	190.8 MB
myflaskapp	latest	01153e090d15	1 hour ago	199.65 MB

Showing 2 items

9. Execute **docker stop myflaskcontainer** to stop the container

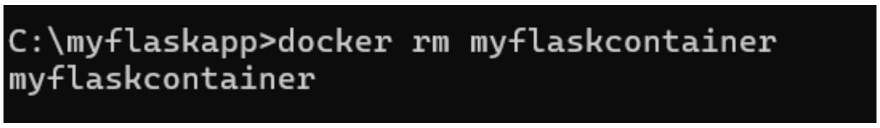


Container CPU usage 0.02% / 800% (8 CPUs available)

Container memory usage 40.69MB / 5.54GB

Name	Container ID	Image	Port(s)	CPU (%)	Last started
trusting_wu	f3e340a5bbe8	myflaskapp	5005:5005	0.01%	1 hour ago
jolly_swanson	b1dbe9de6927	myflaskapp	5005:5005	0%	
myflaskcontainer	c3108793ce3f	myflaskapp	5000:5000	0%	10 minutes ago
myflaskcontainer1	e5c9cf737910	myflaskapp	5000:5000	0%	

10. To remove the container execute **docker rm myflaskcontainer**



```
C:\myflaskapp>docker rm myflaskcontainer
myflaskcontainer
```

RESULT:

Thus, the creation of simple working flask application in Docker is done successfully.

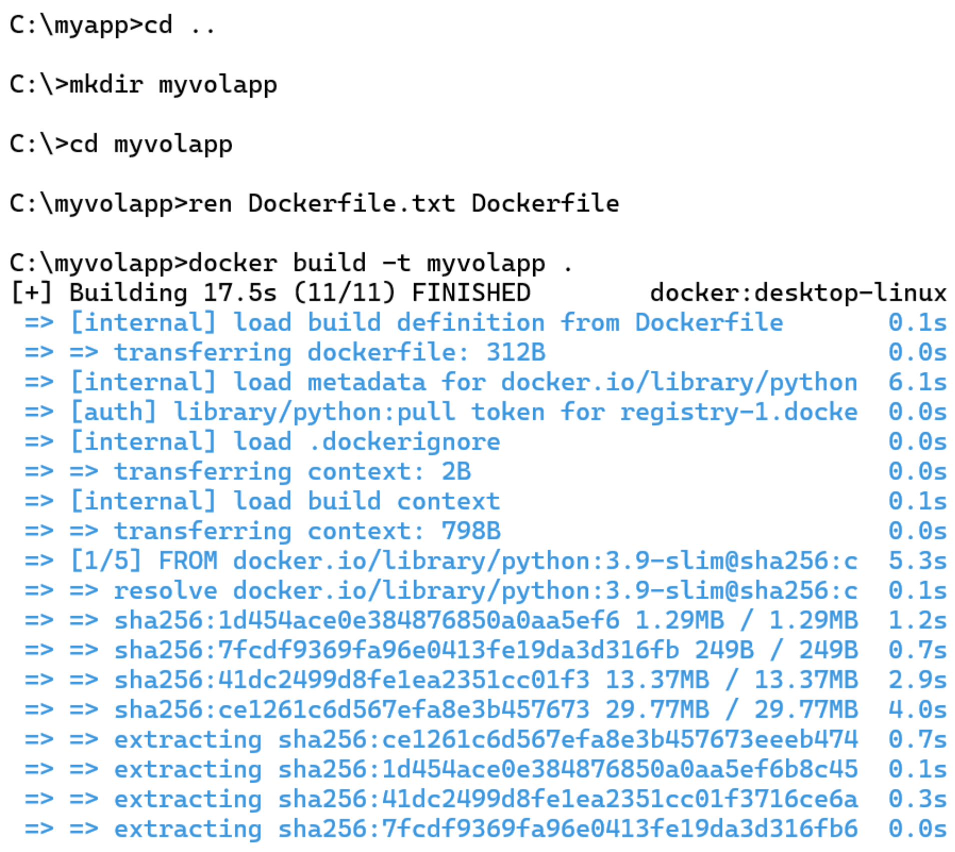
Managing Docker containers and images

AIM:

To manage containers and images in Docker.

PROCEDURE:

1. Install Docker Desktop on your system
2. Create a new folder named myvolapp using mkdir myvolapp
3. Move into the new folder with cd myvolapp
4. Rename Dockerfile.txt to Dockerfile using ren
5. Build the docker image with docker build -t myvolapp .



```
C:\myapp>cd ..

C:\>mkdir myvolapp

C:\>cd myvolapp

C:\myvolapp>ren Dockerfile.txt Dockerfile

C:\myvolapp>docker build -t myvolapp .
[+] Building 17.5s (11/11) FINISHED        docker:desktop-linux
 => [internal] load build definition from Dockerfile       0.1s
 => => transferring dockerfile: 312B                        0.0s
 => [internal] load metadata for docker.io/library/python  6.1s
 => [auth] library/python:pull token for registry-1.docke  0.0s
 => [internal] load .dockerignore                          0.0s
 => => transferring context: 2B                             0.0s
 => [internal] load build context                          0.1s
 => => transferring context: 798B                           0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:c  5.3s
 => => resolve docker.io/library/python:3.9-slim@sha256:c  0.1s
 => => sha256:1d454ace0e384876850a0aa5ef6 1.29MB / 1.29MB  1.2s
 => => sha256:7fcdf9369fa96e0413fe19da3d316fb 249B / 249B  0.7s
 => => sha256:41dc2499d8fe1ea2351cc01f3 13.37MB / 13.37MB  2.9s
 => => sha256:ce1261c6d567efa8e3b457673 29.77MB / 29.77MB  4.0s
 => => extracting sha256:ce1261c6d567efa8e3b457673eeeb474  0.7s
 => => extracting sha256:1d454ace0e384876850a0aa5ef6b8c45  0.1s
 => => extracting sha256:41dc2499d8fe1ea2351cc01f3716ce6a  0.3s
 => => extracting sha256:7fcdf9369fa96e0413fe19da3d316fb6  0.0s
```

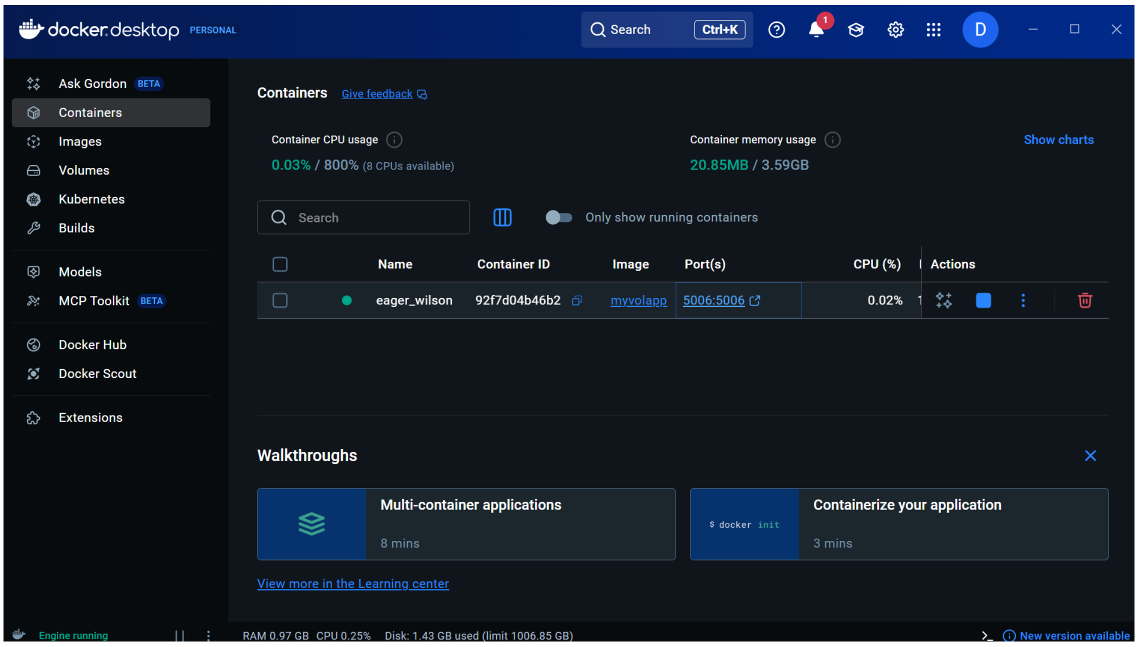
6. Run a new container from the image myvolapp with port mapping and volume mount with docker run -d -p 5006:5006 -v mydata:/data myvolapp
7. List the available docker volumes to check if mydata volume has been created with docker volume ls
8. Use the alpine container temporarily to access the mounted volume and read a file log.txt inside it with docker run --rm -it -v mydata:/data alpine cat /data/log.txt
9. List running containers to see active ones using docker ps

```
C:\myvolapp>docker run -d -p 5006:5006 -v mydata:/data myvolapp
d91cf5dac36adc35158a5c252a0f89cac4ca9a534f0ab0a2b97a3303c3916b04

C:\myvolapp>docker volume ls
DRIVER    VOLUME NAME
local     mydata

C:\myvolapp>docker run --rm -it -v mydata:/data alpine cat /data
/log.txt
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
9824c27679d3: Pull complete
Digest: sha256:4bcff63911fcb4448bd4fdacec207030997caf25e9bea4045
fa6c8c44de311d1
Status: Downloaded newer image for alpine:latest
Visited at 2025-09-17 08:17:31

C:\myvolapp>docker ps
CONTAINER ID   IMAGE      COMMAND            CREATED           STA
TUS           PORTS                                        NAMES
d91cf5dac36a   myvolapp   "python app.py"    18 minutes ago    Up
18 minutes    0.0.0.0:5006->5006/tcp, [::]:5006->5006/tcp   adori
```

10. Stop the container using its container id like docker stop d91cf5dac36a
11. Remove the stopped container using its name like docker rm adoring_volhard
12. Run a new container in detached mode with port mapping and volume mount with docker run -d -p 5006:5006 -v mydata:/data myvolapp

```
C:\myvolapp>docker stop d91cf5dac36adc35158a5c252a0f89cac4ca9a53
4f0ab0a2b97a3303c3916b04
d91cf5dac36adc35158a5c252a0f89cac4ca9a534f0ab0a2b97a3303c3916b04

C:\myvolapp>docker rm adoring_volhard
adoring_volhard

C:\myvolapp>run -d -p 5006:5006 -v mydata:/data myvolapp
'run' is not recognized as an internal or external command,
operable program or batch file.

C:\myvolapp>docker run -d -p 5006:5006 -v mydata:/data myvolapp
92f7d04b46b2477f89f1c4ad57afc6e41d2b1e07b83a9d9bf1cd8b01363a39f9
```

13. Open Docker Desktop → Images tab to verify that the built images (myvolapp and alpine) are available



Containers

Container CPU usage 0.03% / 800% (8 CPUs available)

Container memory usage 20.85MB / 3.59GB

	Name	Container ID	Image	Port(s)	CPU (%)	Actions
☐ ●	eager_wilson	92f7d04b46b2	myvolapp	5006:5006	0.02%	

Walkthroughs

Multi-container applications 8 mins

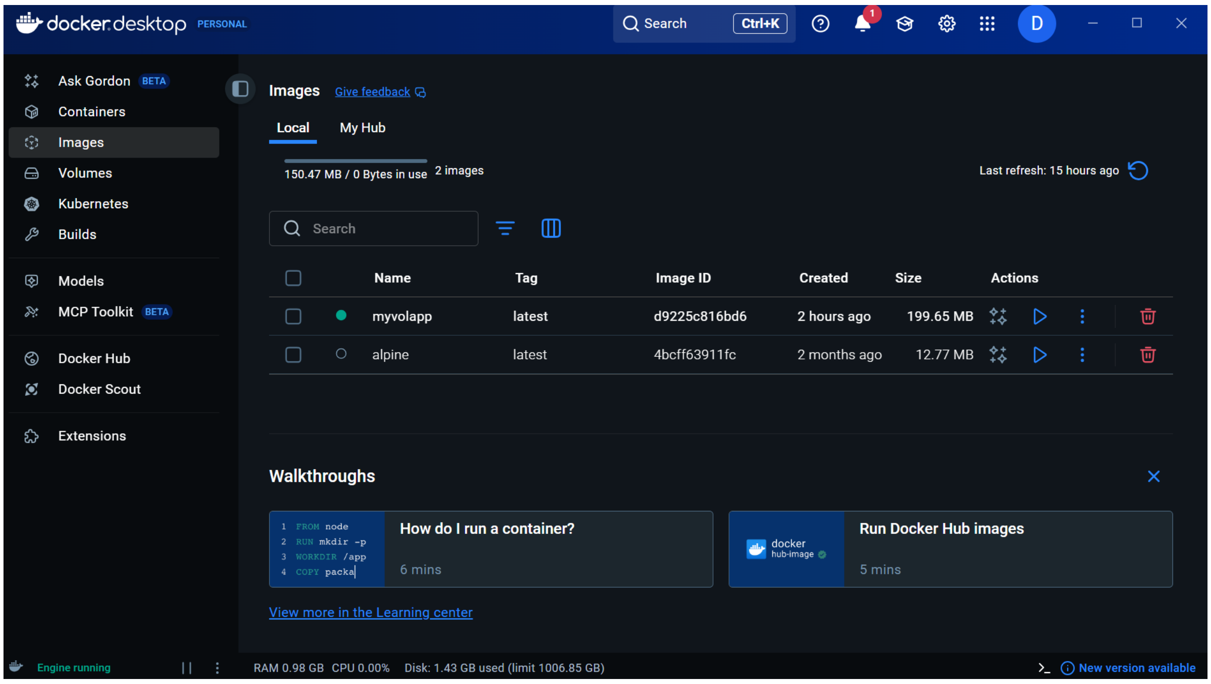
$ docker init — Containerize your application 3 mins

View more in the Learning center

Engine running RAM 0.97 GB CPU 0.25% Disk: 1.43 GB used (limit 1006.85 GB) New version available

14. Open Docker Desktop → Containers tab to check that the container created from myvolapp is running with the mapped port 5006:5006

15. Use Docker Desktop interface to start, stop, or remove containers/images



150.47 MB / 0 Bytes in use 2 images

Last refresh: 15 hours ago

	Name	Tag	Image ID	Created	Size	Actions
●	myvolapp	latest	d9225c816bd6	2 hours ago	199.65 MB	
○	alpine	latest	4bcff63911fc	2 months ago	12.77 MB	

Walkthroughs

How do I run a container? 6 mins

Run Docker Hub images 5 mins

View more in the Learning center

Engine running RAM 0.98 GB CPU 0.00% Disk: 1.43 GB used (limit 1006.85 GB) New version available

RESULT:

Thus, the managing of containers and images in Docker is done successfully.