For any queries please email - tacloudcomputing2024@gmail.com (RR Campus) or ectacloudcomputing2024@gmail.com (EC Campus)

Deliverables
The following screenshots are to be submitted:

Note: The screenshots must be pasted into a Word document and sent in PDF format. The file should be named in this manner
<Section>_<SRN>_<Name>_A2.pdf (Eg. A_PES1UG21CSXXX_Name_A2.pdf)

IMPORTANT:
Every terminal screenshot should include your SRN. The path to your lab folder directory should contain your SRN. Make sure you have a folder named <your_srn> and run the tasks within this folder.

*Eg. "**C:\User\Name\your_srn\lab_folder\task2>**"*

- **1a.jpg:** Screenshot of running docker hello-world.
- **2a.jpg:** Screenshot of python-mongodb application running as a docker-compose application(logs of the application)
- **2b.jpg:** Screenshot of 3 python application writes and reads from MongoDB after scaling the python application.
- **3a.jpg:** Screenshot of docker container running nginx
- **3b.jpg:** Sample.html showing the web page on the browser.
- **3c.jpg:** Screenshot of python application successfully writing and reading from the MongoDB database
- **3d.jpg:** Screenshot showing mongodb being run within the network(docker command has to be clearly highlighted)
- **3e.jpg:** Screenshot showing python file being run within the network and successfully writing and reading from MongoDB(docker command has to be clearly highlighted)
- **4a.jpg:** Screenshot of C Program successfully run inside the container.
- **4b.jpg:** Screenshot of the image pushed to Dockerhub.

Few key points to note:
1. All required Dockerfile(s) have been given.
2. It is very important to go through all reference material given in the pre-reading and installation guide, as this will help you understand and debug the lab tasks.
3. **Ensure docker has been installed** before starting this lab.
4. Apart from the attached resources, you can always refer to the official docker documentation. The docker documentation is well maintained and should help you through all your tasks.https://docs.docker.com/
5. Additional resources have been given at the end of the manual.

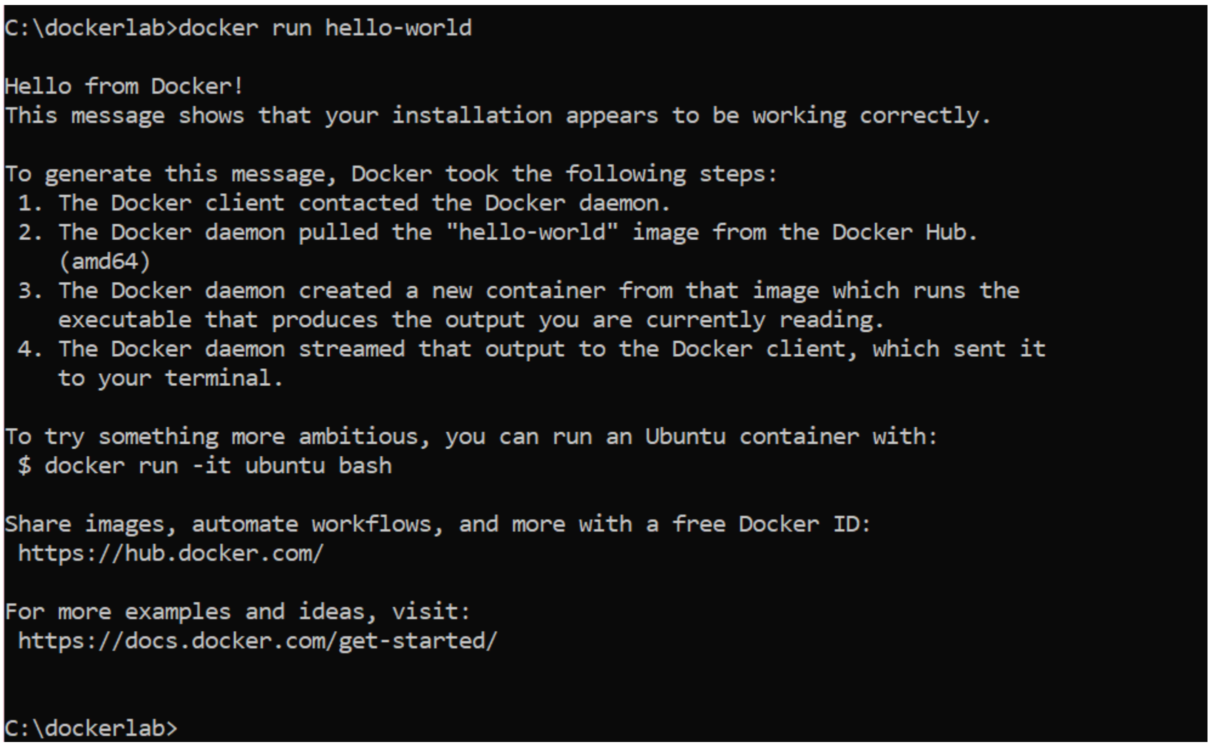
6. When you run docker commands in the foreground, you cannot access the command prompt in which case press [Ctrl+C] and continue with the next step or run docker in the background mode by using -d option.
7. The lab experiment folder contains all the python, html, c and Dockerfiles. Make changes if and when necessary.

Task 1: Installing Docker Engine

Note:
If you are using Play-with-docker, ensure you add a new instance before starting the lab.
Verify that Docker Engine is installed correctly by running:

docker run hello-world

Take a screenshot of running docker hello-world and name it 1a.jpg



```
C:\dockerlab>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/


C:\dockerlab>
```

Task 2 Docker Compose:

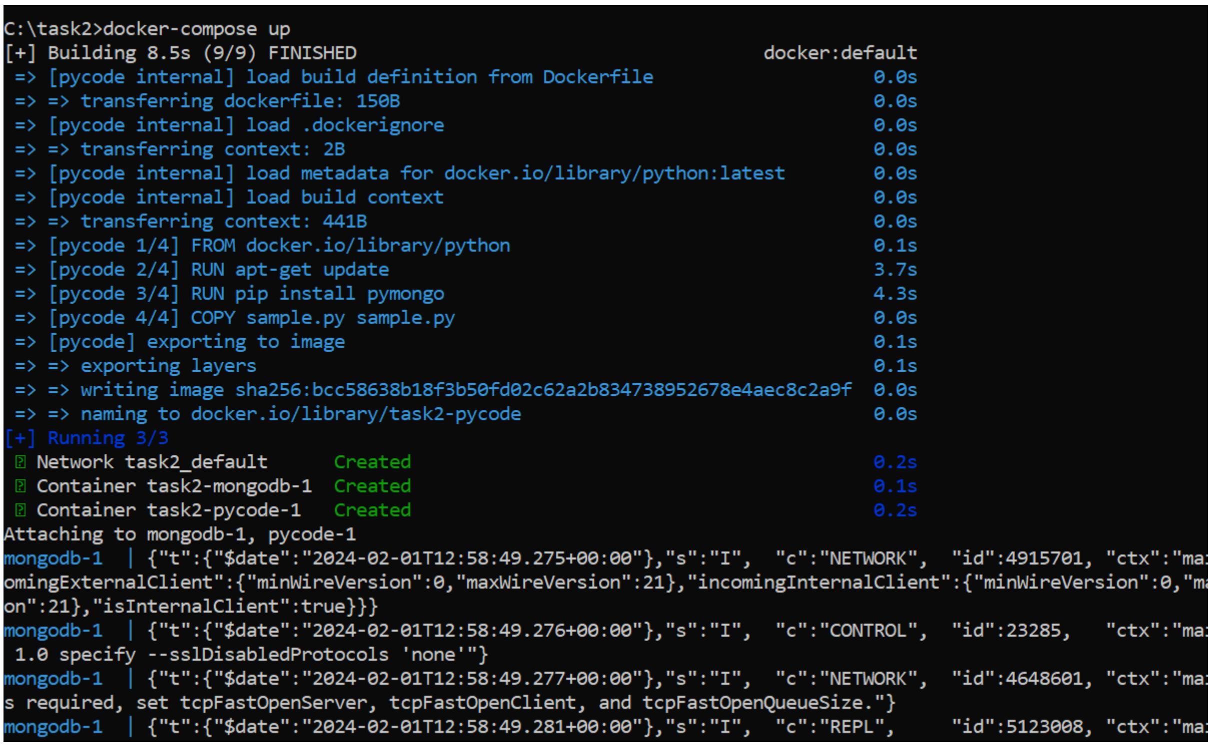
(Dockerfile, sample.py and docker-compose.yml files are present in the lab instructions folder, modify only required details)

1. For the purpose of this lab make sure all the following files (present in the lab instructions folder) are in the same directory:
   a. docker-compose.yml
   b. Dockerfile
   c. sample.py

2. Go to the docker-compose.yml file and try to understand the syntax and what each line does.
3. Within the same directory, run:

   docker-compose up (windows)
   docker compose up (linux)

Take a screenshot of python-mongodb application running as a docker-compose application(logs of the application) (2a.jpg).



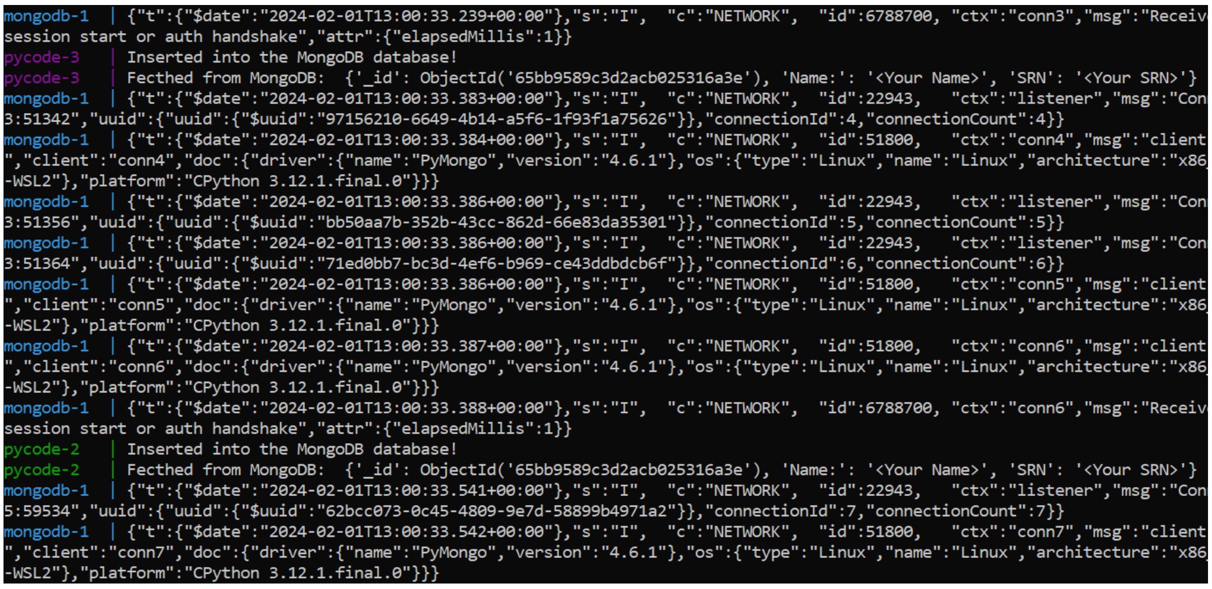
```
C:\task2>docker-compose up
[+] Building 8.5s (9/9) FINISHED                                   docker:default
 => [pycode internal] load build definition from Dockerfile                  0.0s
 => => transferring dockerfile: 150B                                         0.0s
 => [pycode internal] load .dockerignore                                     0.0s
 => => transferring context: 2B                                              0.0s
 => [pycode internal] load metadata for docker.io/library/python:latest      0.0s
 => [pycode internal] load build context                                     0.0s
 => => transferring context: 441B                                            0.0s
 => [pycode 1/4] FROM docker.io/library/python                               0.1s
 => [pycode 2/4] RUN apt-get update                                          3.7s
 => [pycode 3/4] RUN pip install pymongo                                     4.3s
 => [pycode 4/4] COPY sample.py sample.py                                    0.0s
 => [pycode] exporting to image                                              0.1s
 => => exporting layers                                                      0.1s
 => => writing image sha256:bcc58638b18f3b50fd02c62a2b834738952678e4aec8c2a9f  0.0s
 => => naming to docker.io/library/task2-pycode                              0.0s
[+] Running 3/3
 ✔ Network task2_default      Created                                        0.2s
 ✔ Container task2-mongodb-1  Created                                        0.1s
 ✔ Container task2-pycode-1   Created                                        0.2s
Attaching to mongodb-1, pycode-1
mongodb-1  | {"t":{"$date":"2024-02-01T12:58:49.275+00:00"},"s":"I",  "c":"NETWORK",  "id":4915701, "ctx":"ma
omingExternalClient":{"minWireVersion":0,"maxWireVersion":21},"incomingInternalClient":{"minWireVersion":0,"ma
on":21},"isInternalClient":true}}}
mongodb-1  | {"t":{"$date":"2024-02-01T12:58:49.276+00:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"ma
 1.0 specify --sslDisabledProtocols 'none'"}
mongodb-1  | {"t":{"$date":"2024-02-01T12:58:49.277+00:00"},"s":"I",  "c":"NETWORK",  "id":4648601, "ctx":"ma
s required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQueueSize."}
mongodb-1  | {"t":{"$date":"2024-02-01T12:58:49.281+00:00"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"ma
```

What you see is that Docker compose has built your python application, started the MongoDB server, created links internally between the containers (network) and started both the containers together as a unified application. The python container exits since it's done with its utility of writing and reading to the database. (Press *CTRL-C* to exit Docker compose if the shell prompt is not returned)

4. Now we will scale the python application, so that we have 3 containers of the python application, but keep only one container of the mongodb.

docker-compose up --scale pycode=3 (windows)
docker compose up --scale pycode=3 (linux)

Take a screenshot of 3 python application writes and reads from MongoDB after scaling the python application (2b.jpg)



```
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.239+00:00"},"s":"I",  "c":"NETWORK",  "id":6788700, "ctx":"conn3","msg":"Receiv
session start or auth handshake","attr":{"elapsedMillis":1}}
pycode-3   | Inserted into the MongoDB database!
pycode-3   | Fecthed from MongoDB:  {'_id': ObjectId('65bb9589c3d2acb025316a3e'), 'Name:': '<Your Name>', 'SRN': '<Your SRN>'}
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.383+00:00"},"s":"I",  "c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Con
3:51342","uuid":{"uuid":{"$uuid":"97156210-6649-4b14-a5f6-1f93f1a75626"}},"connectionId":4,"connectionCount":4}}
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.384+00:00"},"s":"I",  "c":"NETWORK",  "id":51800,   "ctx":"conn4","msg":"client
","client":"conn4","doc":{"driver":{"name":"PyMongo","version":"4.6.1"},"os":{"type":"Linux","name":"Linux","architecture":"x86
-WSL2"},"platform":"CPython 3.12.1.final.0"}}}
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.386+00:00"},"s":"I",  "c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Con
3:51356","uuid":{"uuid":{"$uuid":"bb50aa7b-352b-43cc-862d-66e83da35301"}},"connectionId":5,"connectionCount":5}}
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.386+00:00"},"s":"I",  "c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Con
3:51364","uuid":{"uuid":{"$uuid":"71ed0bb7-bc3d-4ef6-b969-ce43ddbdcb6f"}},"connectionId":6,"connectionCount":6}}
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.386+00:00"},"s":"I",  "c":"NETWORK",  "id":51800,   "ctx":"conn5","msg":"client
","client":"conn5","doc":{"driver":{"name":"PyMongo","version":"4.6.1"},"os":{"type":"Linux","name":"Linux","architecture":"x86
-WSL2"},"platform":"CPython 3.12.1.final.0"}}}
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.387+00:00"},"s":"I",  "c":"NETWORK",  "id":51800,   "ctx":"conn6","msg":"client
","client":"conn6","doc":{"driver":{"name":"PyMongo","version":"4.6.1"},"os":{"type":"Linux","name":"Linux","architecture":"x86
-WSL2"},"platform":"CPython 3.12.1.final.0"}}}
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.388+00:00"},"s":"I",  "c":"NETWORK",  "id":6788700, "ctx":"conn6","msg":"Receiv
session start or auth handshake","attr":{"elapsedMillis":1}}
pycode-2   | Inserted into the MongoDB database!
pycode-2   | Fecthed from MongoDB:  {'_id': ObjectId('65bb9589c3d2acb025316a3e'), 'Name:': '<Your Name>', 'SRN': '<Your SRN>'}
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.541+00:00"},"s":"I",  "c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Con
5:59534","uuid":{"uuid":{"$uuid":"62bcc073-0c45-4809-9e7d-58899b4971a2"}},"connectionId":7,"connectionCount":7}}
mongodb-1  | {"t":{"$date":"2024-02-01T13:00:33.542+00:00"},"s":"I",  "c":"NETWORK",  "id":51800,   "ctx":"conn7","msg":"client
","client":"conn7","doc":{"driver":{"name":"PyMongo","version":"4.6.1"},"os":{"type":"Linux","name":"Linux","architecture":"x86
-WSL2"},"platform":"CPython 3.12.1.final.0"}}}
```

Task 3 Exposing ports, docker networks :

(Dockerfile and my.html files are present in the lab instructions folder under **task3-nginx.** Run the Docker container in this directory)

1. Create a sample HTML (*my.html*) file as given below, and **modify your SRN.**

(You can use the file present in the lab experiment folder)

```
<html>
  <body>
    <h1>My SRN is your_srn</h1>
    <h2>I am running a nginx container!</h2>
  </body>
</html>
```

2. Create a Dockerfile and then a docker image having an nginx base image, and copying the html file into the default folder in the container.

   Dockerfile:

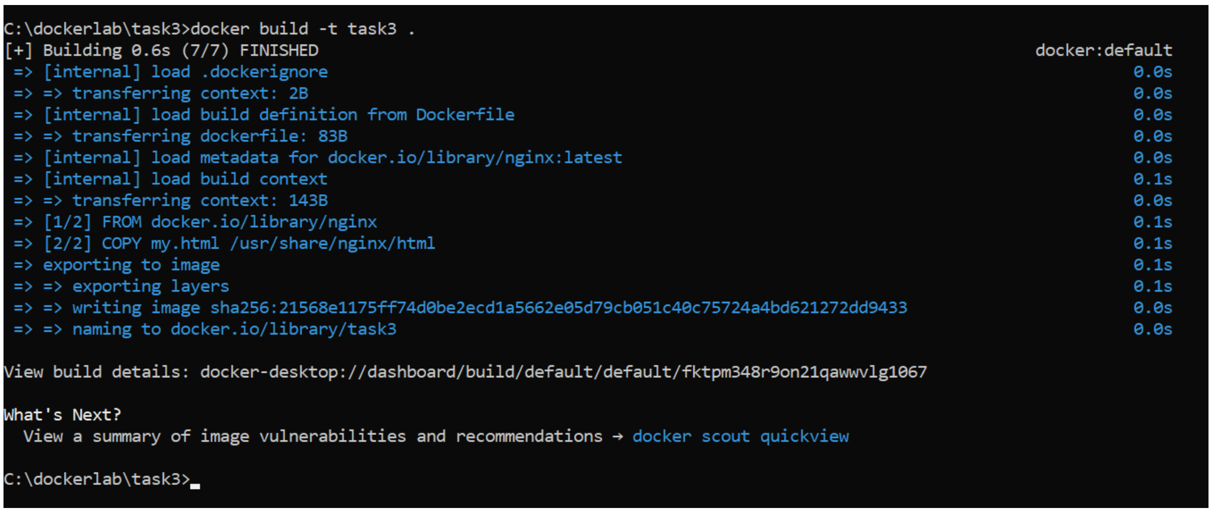
(This file is also present in the lab experiment folder)

```
FROM nginx
COPY my.html /usr/share/nginx/html
```

Save and name the file as Dockerfile.

1. Build the docker image using

docker build -t <myimage-name> .

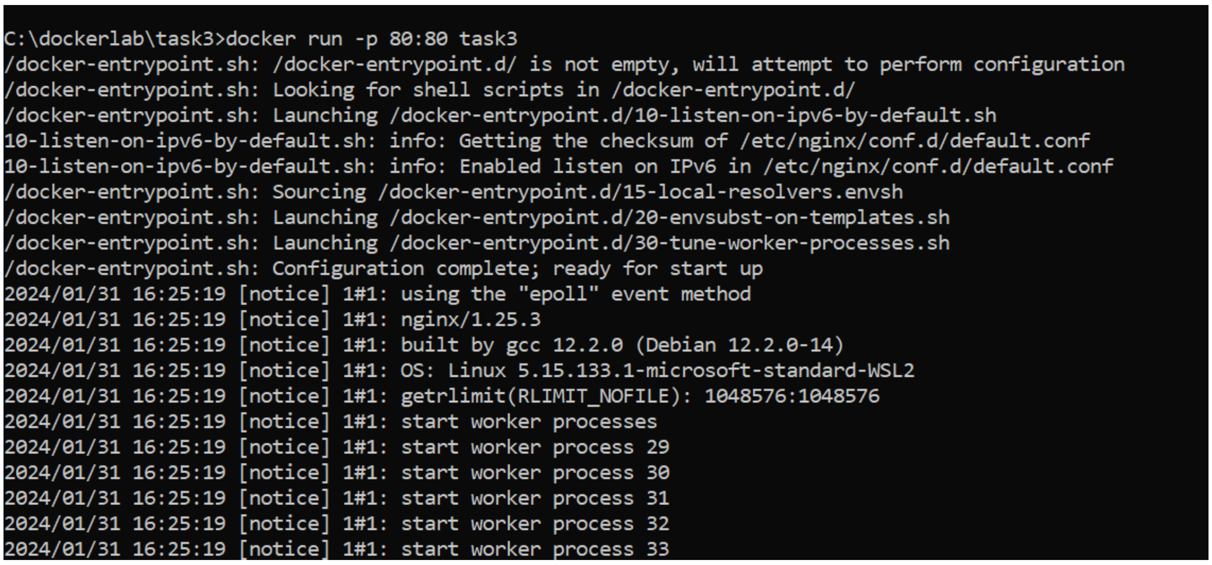


```
C:\dockerlab\task3>docker build -t task3 .
[+] Building 0.6s (7/7) FINISHED                                                          docker:default
 => [internal] load .dockerignore                                                                   0.0s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load build definition from Dockerfile                                                0.0s
 => => transferring dockerfile: 83B                                                                 0.0s
 => [internal] load metadata for docker.io/library/nginx:latest                                     0.0s
 => [internal] load build context                                                                   0.1s
 => => transferring context: 143B                                                                   0.0s
 => [1/2] FROM docker.io/library/nginx                                                              0.1s
 => [2/2] COPY my.html /usr/share/nginx/html                                                        0.1s
 => exporting to image                                                                              0.1s
 => => exporting layers                                                                             0.1s
 => => writing image sha256:21568e1175ff74d0be2ecd1a5662e05d79cb051c40c75724a4bd621272dd9433        0.0s
 => => naming to docker.io/library/task3                                                            0.0s

View build details: docker-desktop://dashboard/build/default/default/fktpm348r9on21qawwvlg1067

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview

C:\dockerlab\task3>
```

2. Run the docker container using the previously created docker image and expose the HTTP port using

   docker run -p 80:80 <myimage-name>

Take a screenshot of docker container running nginx (3a.jpg)

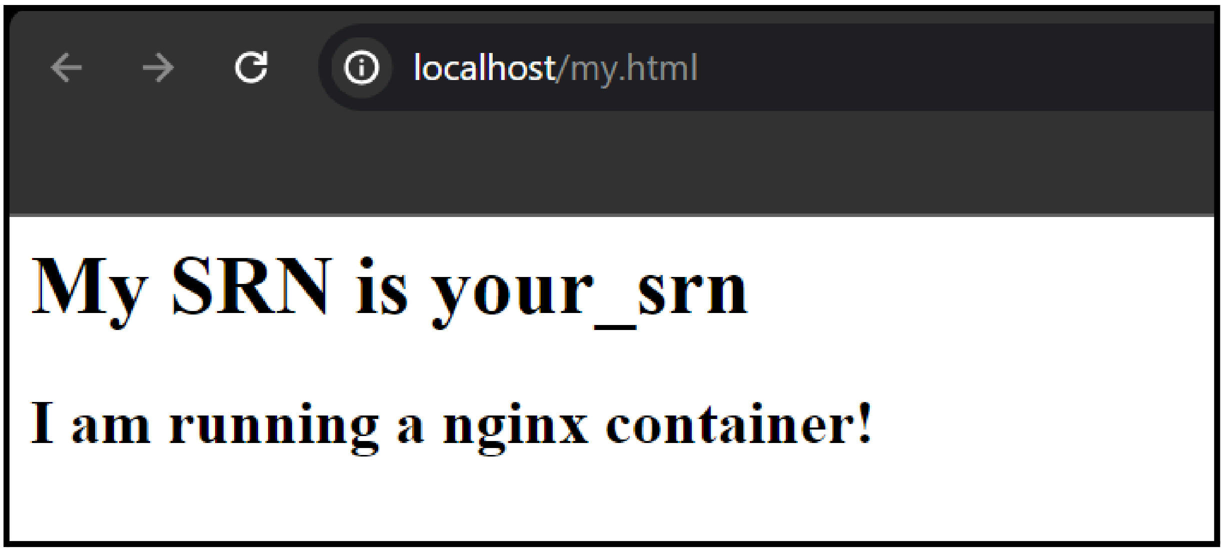


```
C:\dockerlab\task3>docker run -p 80:80 task3
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
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: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/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
2024/01/31 16:25:19 [notice] 1#1: using the "epoll" event method
2024/01/31 16:25:19 [notice] 1#1: nginx/1.25.3
2024/01/31 16:25:19 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/01/31 16:25:19 [notice] 1#1: OS: Linux 5.15.133.1-microsoft-standard-WSL2
2024/01/31 16:25:19 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/01/31 16:25:19 [notice] 1#1: start worker processes
2024/01/31 16:25:19 [notice] 1#1: start worker process 29
2024/01/31 16:25:19 [notice] 1#1: start worker process 30
2024/01/31 16:25:19 [notice] 1#1: start worker process 31
2024/01/31 16:25:19 [notice] 1#1: start worker process 32
2024/01/31 16:25:19 [notice] 1#1: start worker process 33
```

3. Access the nginx server displaying the webpage by typing out the following url http://localhost:80/my.html

   (If you are using PWD, access it via the `Open Port` button and access port 80. Append my.html to the end of the url)

Take a screenshot of my.html showing the web page on the browser (3b.jpg)


localhost/my.html

My SRN is your_srn

I am running a nginx container!

4. Press Ctrl+C in the terminal to stop the nginx server

We will explore connectivity without docker networks and see how docker networks make it much easier to connect within containers. To demonstrate this we will create a simple application using a python client and a MongoDB NoSQL server.

Note: You don't need to know how to use mongodb, code to use mongodb has been provided to you.

(Dockerfile and my.html files are present in the lab instructions folder under **task3-pymongo.** Run the Docker container in this directory)

5. Run the mongodb container in a detached mode, exposing the default port (27017) of mongodb using

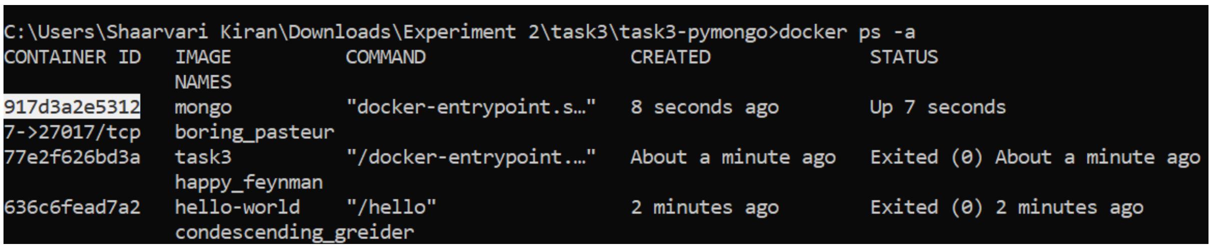
   docker run -dp 27017:27017 mongo

6. Create a sample.py file as given below, and modify your SRN wherever mentioned.
(This file is also present in the lab experiment folder)

```
from pymongo import MongoClient

host = MongoClient("170.17.0.2")
host = MongoClient("mongodb")

db = host["sample_db"]
collection = db["sample_collection"]
```

```
sample_data = {"Name:":"your_name","SRN":"your_srn"}
collection.insert_one(sample_data)
print('Inserted into the MongoDB database!')

rec_data = collection.find_one({"SRN":"your_srn"})
print("Fetched from MongoDB: ",rec_data)
```

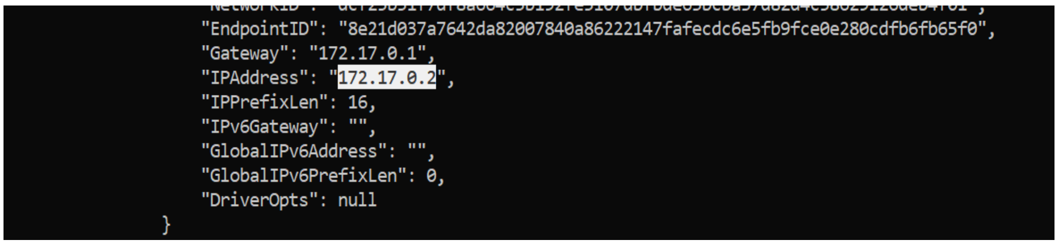
7. The MongoDB container is running, but we need to find out the IP address of the mongodb container.
   a. Find the container ID of the mongodb container using *docker ps -a*



```
C:\Users\Shaarvari Kiran\Downloads\Experiment 2\task3\task3-pymongo>docker ps -a
CONTAINER ID   IMAGE           COMMAND                  CREATED              STATUS
               NAMES
917d3a2e5312   mongo           "docker-entrypoint.s…"   8 seconds ago        Up 7 seconds
7->27017/tcp   boring_pasteur
77e2f626bd3a   task3           "/docker-entrypoint.…"   About a minute ago   Exited (0) About a minute ago
               happy_feynman
636c6fead7a2   hello-world     "/hello"                 2 minutes ago        Exited (0) 2 minutes ago
               condescending_greider
```

   b. Run *docker inspect <container_id>*

   Find this IP Address field and note this down



```
                    "EndpointID": "8e21d037a7642da82007840a86222147fafecdc6e5fb9fce0e280cdfb6fb65f0",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DriverOpts": null
                }
```

Create a Dockerfile using the template given in task 2, which:

a. Uses python base image
b. Updates the apt-repository
c. Installs pymongo using pip (pymongo 3.11.2).
d. Copies the sample.py from the instance to the container.
e. Runs the python command , to run the python file.

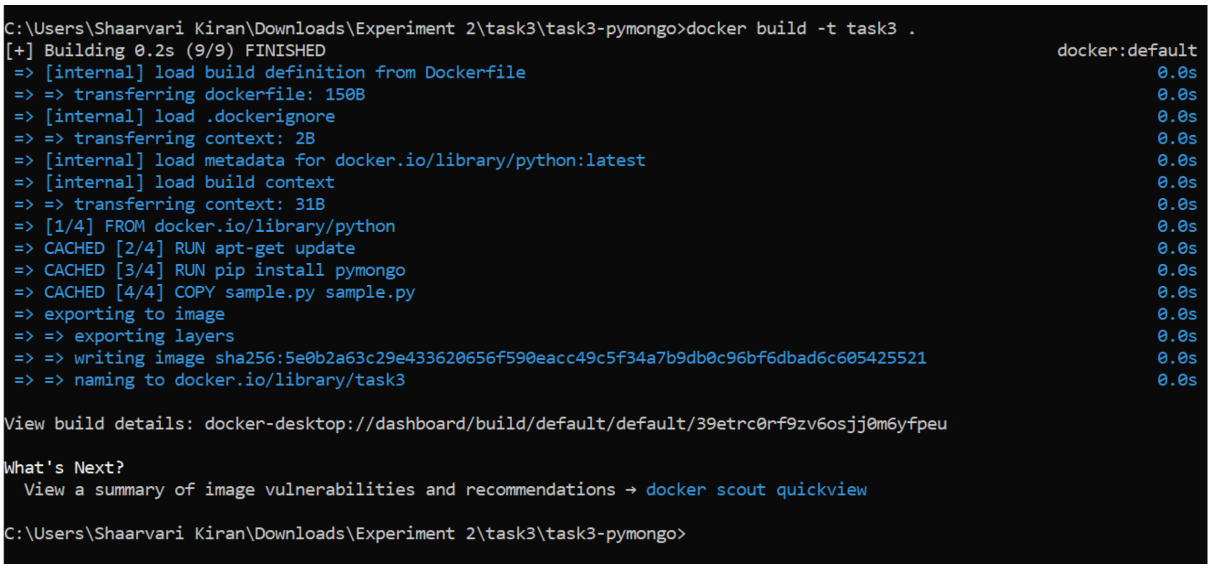
Dockerfile:
(This file is also present in the lab experiment folder)

```
FROM python
RUN apt-get update
RUN pip install pymongo
COPY sample.py sample.py
CMD ["python","sample.py"]
```

Save and name the file as Dockerfile.

8. Build the docker image using the above Dockerfile and run the container.

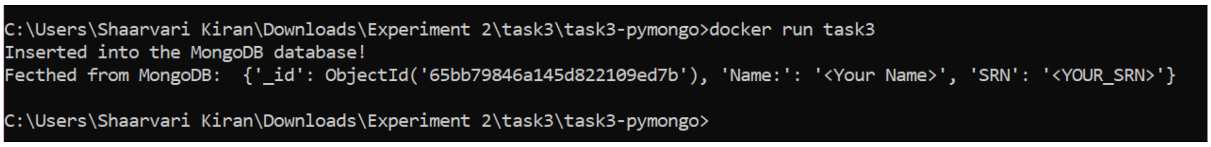
docker build -t <myimage-name> .



```
C:\Users\Shaarvari Kiran\Downloads\Experiment 2\task3\task3-pymongo>docker build -t task3 .
[+] Building 0.2s (9/9) FINISHED                                                        docker:default
 => [internal] load build definition from Dockerfile                                              0.0s
 => => transferring dockerfile: 150B                                                              0.0s
 => [internal] load .dockerignore                                                                 0.0s
 => => transferring context: 2B                                                                   0.0s
 => [internal] load metadata for docker.io/library/python:latest                                  0.0s
 => [internal] load build context                                                                 0.0s
 => => transferring context: 31B                                                                  0.0s
 => [1/4] FROM docker.io/library/python                                                           0.0s
 => CACHED [2/4] RUN apt-get update                                                               0.0s
 => CACHED [3/4] RUN pip install pymongo                                                          0.0s
 => CACHED [4/4] COPY sample.py sample.py                                                         0.0s
 => exporting to image                                                                            0.0s
 => => exporting layers                                                                           0.0s
 => => writing image sha256:5e0b2a63c29e433620656f590eacc49c5f34a7b9db0c96bf6dbad6c605425521      0.0s
 => => naming to docker.io/library/task3                                                          0.0s

View build details: docker-desktop://dashboard/build/default/default/39etrc0rf9zv6osjj0m6yfpeu

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview

C:\Users\Shaarvari Kiran\Downloads\Experiment 2\task3\task3-pymongo>
```

docker run <myimage-name>

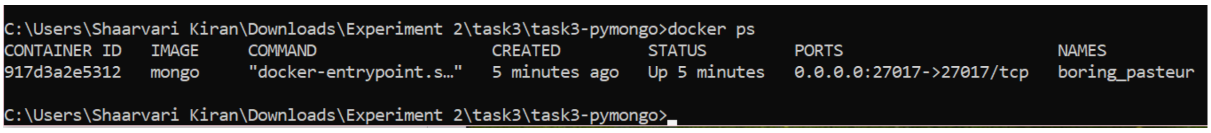
Take a screenshot of python application successfully writing and reading from the MongoDB database (3c.jpg).



```
C:\Users\Shaarvari Kiran\Downloads\Experiment 2\task3\task3-pymongo>docker run task3
Inserted into the MongoDB database!
Fecthed from MongoDB:  {'_id': ObjectId('65bb79846a145d822109ed7b'), 'Name:': '<Your Name>', 'SRN': '<YOUR_SRN>'}

C:\Users\Shaarvari Kiran\Downloads\Experiment 2\task3\task3-pymongo>
```

You should see that the data was correctly inserted and fetched from the database container.

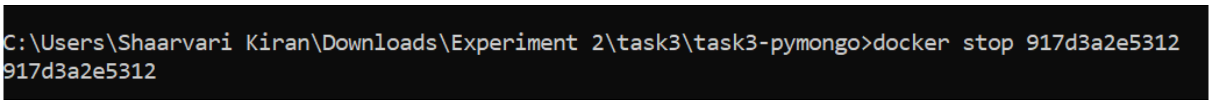
11. Use *docker ps* to get the container id of the mongo image.



```
C:\Users\Shaarvari Kiran\Downloads\Experiment 2\task3\task3-pymongo>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                      NAMES
917d3a2e5312   mongo     "docker-entrypoint.s…"   5 minutes ago   Up 5 minutes   0.0.0.0:27017->27017/tcp   boring_pasteur

C:\Users\Shaarvari Kiran\Downloads\Experiment 2\task3\task3-pymongo>
```

12. Note the container id and stop running the container using

docker stop <container-id>

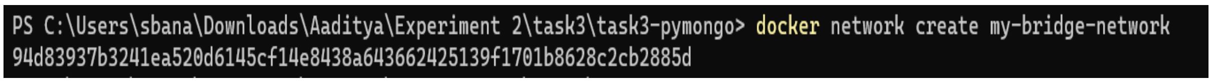


```
C:\Users\Shaarvari Kiran\Downloads\Experiment 2\task3\task3-pymongo>docker stop 917d3a2e5312
917d3a2e5312
```

The above tasks showed the connectivity between 2 containers without a network. This can cause problems as every time a container is created it could have a different IP address. This is the issue docker networks tries to solve.

1. Create a docker bridge network called my-bridge-network.

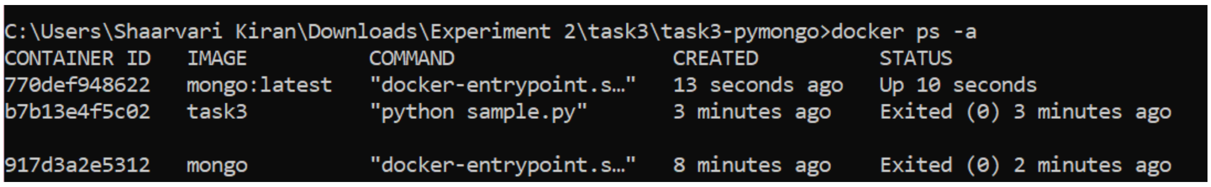
   docker network create my-bridge-network



```
PS C:\Users\sbana\Downloads\Aaditya\Experiment 2\task3\task3-pymongo> docker network create my-bridge-network
94d83937b3241ea520d6145cf14e8438a643662425139f1701b8628c2cb2885d
```

2. Run a mongodb container again, but now with the following parameters;
   a. network : **my-bridge-network**
   b. name: **mongodb**
   c. Exposed ports: **27017**
   d. Image: **mongo:latest**

docker run -dp 27017:27017 --network=my-bridge-network --name=mongodb mongo:latest
Take a screenshot showing mongodb being run within the network(docker command has to be clearly highlighted) (3d.jpg).

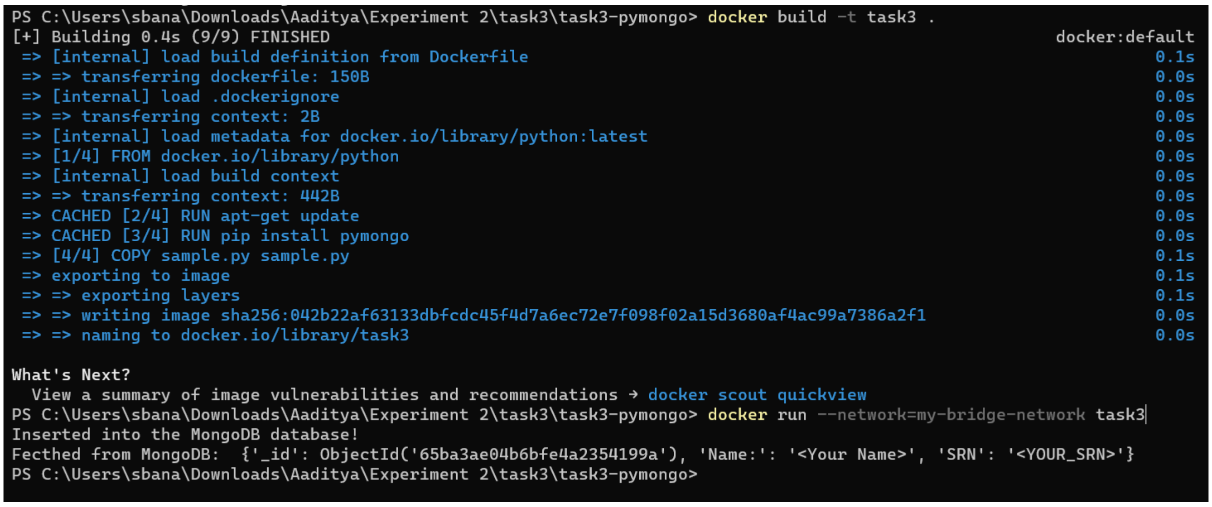


```
C:\Users\Shaarvari Kiran\Downloads\Experiment 2\task3\task3-pymongo>docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
770def948622   mongo:latest   "docker-entrypoint.s…"   13 seconds ago   Up 10 seconds
b7b13e4f5c02   task3          "python sample.py"       3 minutes ago    Exited (0) 3 minutes ago

917d3a2e5312   mongo          "docker-entrypoint.s…"   8 minutes ago    Exited (0) 2 minutes ago
```

3. Go back to sample.py, comment line 3 and uncomment line 4. We are now going to use the name of the database containers as the host name, leaving the ip address resolution to docker.
4. Build the python app docker image again as you did previously and run the container using the built image. You should see that insertion and retrieval have been done successfully.

   docker build -t <myimage-name> .
   docker run --network=my-bridge-network <myimage-name>

Take a screenshot showing python file being run within the network and successfully writing and reading from MongoDB(docker command has to be clearly highlighted) (3e.jpg).



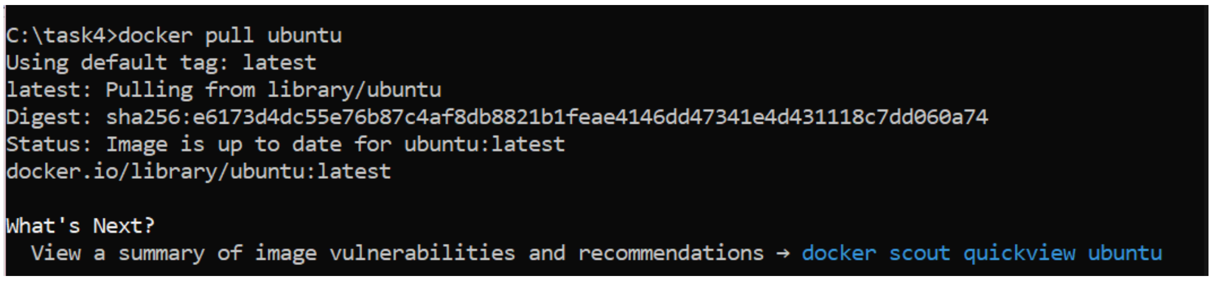
```
PS C:\Users\sbana\Downloads\Aaditya\Experiment 2\task3\task3-pymongo> docker build -t task3 .
[+] Building 0.4s (9/9) FINISHED                                                        docker:default
 => [internal] load build definition from Dockerfile                                              0.1s
 => => transferring dockerfile: 150B                                                              0.0s
 => [internal] load .dockerignore                                                                 0.0s
 => => transferring context: 2B                                                                   0.0s
 => [internal] load metadata for docker.io/library/python:latest                                  0.0s
 => [1/4] FROM docker.io/library/python                                                           0.0s
 => [internal] load build context                                                                 0.0s
 => => transferring context: 442B                                                                 0.0s
 => CACHED [2/4] RUN apt-get update                                                               0.0s
 => CACHED [3/4] RUN pip install pymongo                                                          0.0s
 => [4/4] COPY sample.py sample.py                                                                0.1s
 => exporting to image                                                                            0.1s
 => => exporting layers                                                                           0.1s
 => => writing image sha256:042b22af63133dbfcdc45f4d7a6ec72e7f098f02a15d3680af4ac99a7386a2f1      0.0s
 => => naming to docker.io/library/task3                                                          0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\sbana\Downloads\Aaditya\Experiment 2\task3\task3-pymongo> docker run --network=my-bridge-network task3
Inserted into the MongoDB database!
Fecthed from MongoDB:  {'_id': ObjectId('65ba3ae04b6bfe4a2354199a'), 'Name:': '<Your Name>', 'SRN': '<YOUR_SRN>'}
PS C:\Users\sbana\Downloads\Aaditya\Experiment 2\task3\task3-pymongo>
```

Task 4: Docker images and docker files

Make sure you have completed the sub tasks (docker pull tasks) as a part of the pre-installations

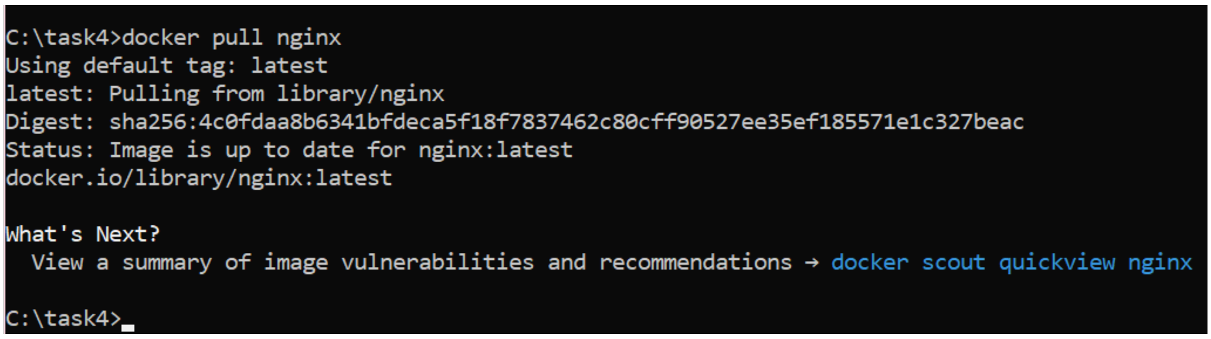
Sub tasks (Ensure you have created an account on Docker Hub before starting this task):
1. Pull the following images onto your docker instance using docker pull

- Ubuntu 18.04 : *docker pull ubuntu*



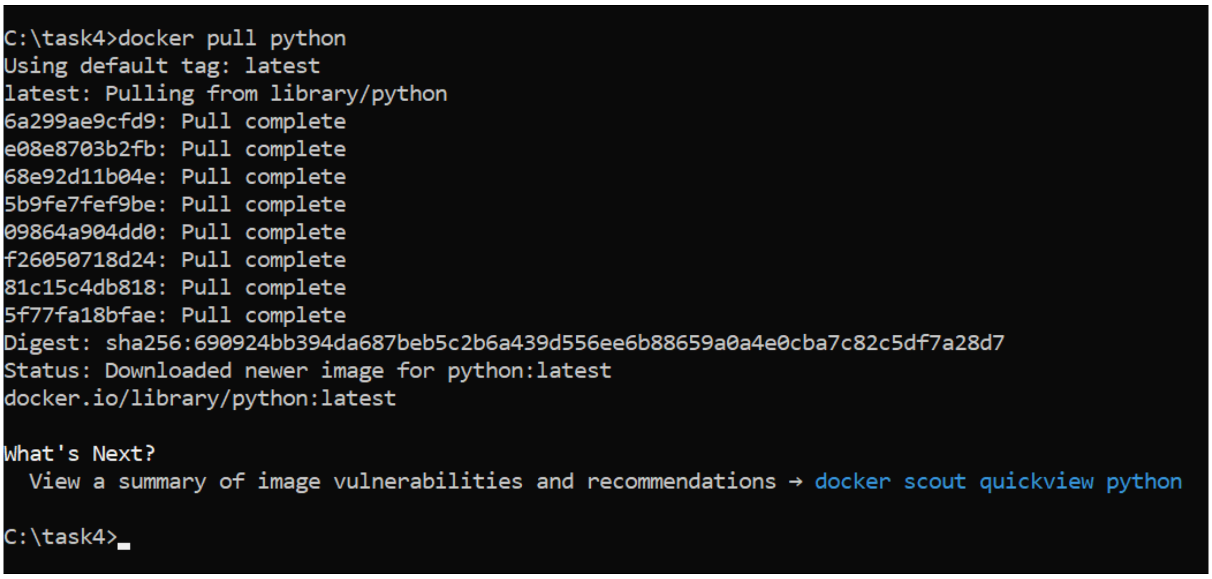
```
C:\task4>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
Digest: sha256:e6173d4dc55e76b87c4af8db8821b1feae4146dd47341e4d431118c7dd060a74
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview ubuntu
```

- Nginx : *docker pull nginx*



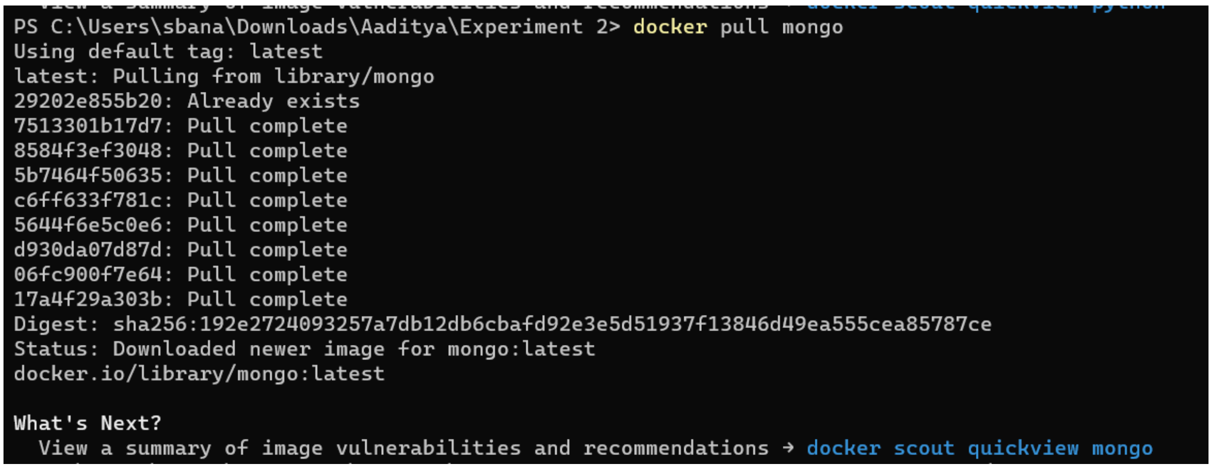
```
C:\task4>docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:4c0fdaa8b6341bfdeca5f18f7837462c80cff90527ee35ef185571e1c327beac
Status: Image is up to date for nginx:latest
docker.io/library/nginx:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview nginx

C:\task4>
```

- Python : *docker pull python*



```
C:\task4>docker pull python
Using default tag: latest
latest: Pulling from library/python
6a299ae9cfd9: Pull complete
e08e8703b2fb: Pull complete
68e92d11b04e: Pull complete
5b9fe7fef9be: Pull complete
09864a904dd0: Pull complete
f26050718d24: Pull complete
81c15c4db818: Pull complete
5f77fa18bfae: Pull complete
Digest: sha256:690924bb394da687beb5c2b6a439d556ee6b88659a0a4e0cba7c82c5df7a28d7
Status: Downloaded newer image for python:latest
docker.io/library/python:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview python

C:\task4>_
```

- MongoDB : *docker pull mongo*



```
PS C:\Users\sbana\Downloads\Aaditya\Experiment 2> docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
29202e855b20: Already exists
7513301b17d7: Pull complete
8584f3ef3048: Pull complete
5b7464f50635: Pull complete
c6ff633f781c: Pull complete
5644f6e5c0e6: Pull complete
d930da07d87d: Pull complete
06fc900f7e64: Pull complete
17a4f29a303b: Pull complete
Digest: sha256:192e2724093257a7db12db6cbafd92e3e5d51937f13846d49ea555cea85787ce
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview mongo
```

(Dockerfile and program.c is present in the lab instructions folder, modify only required details)

1a. Open a text editor and create a C program and name it program.c

Use the following C program given in the lab manual folder as a base, only modify your SRN:

```c
#include <stdio.h>

int main()
{
   printf("Running this inside a container !\n");

   printf("My SRN is <YOUR SRN HERE>\n");
}
```

1b. Create your own Docker image by writing a Dockerfile (template shown below). The image you will create will run a simple C program, after installing the GCC compiler.
The Dockerfile must:

a. Specify the base image as ubuntu:18.04
b. Update the "apt" repository and install the GCC compiler.
c. Copy the program.c file from your instance to the docker image.
d. Compile the C program.
e. Run the ./a.out command.

Dockerfile:
(This file is also present in the lab experiment folder)

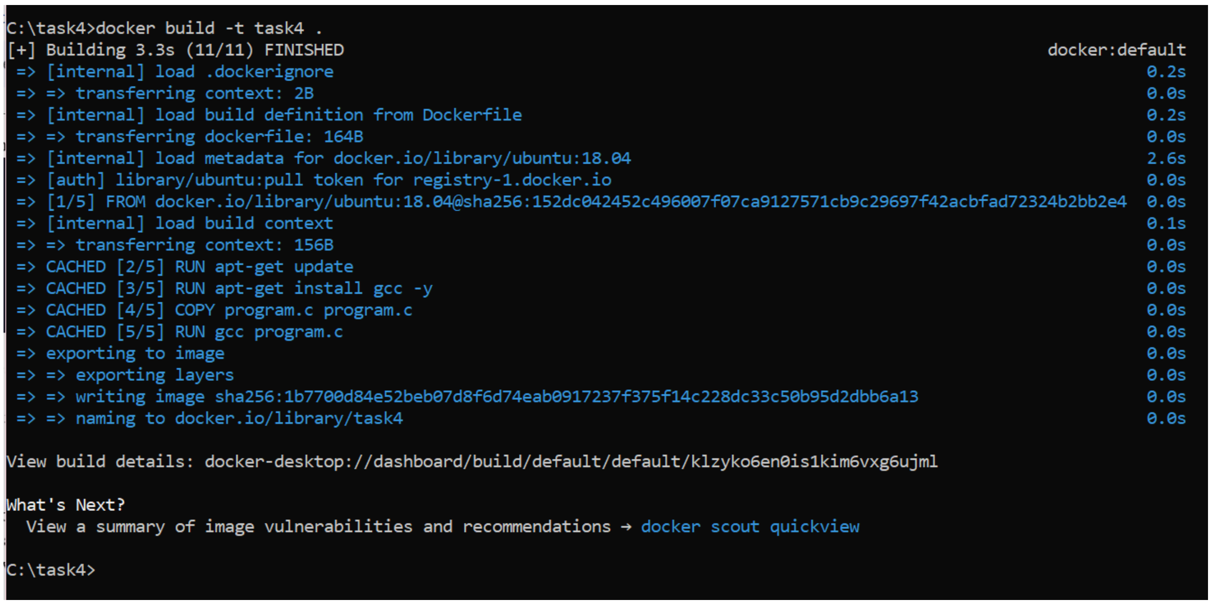
```
FROM ubuntu:18.04
RUN apt-get update
RUN apt-get install gcc -y
COPY program.c program.c
RUN gcc program.c
CMD ["./a.out"]
```

Save and name the file as **Dockerfile**.

2. After you have your Dockerfile, build the image using

```
docker build -t <myimage-name> .
```

'myimage-name' is the name you will give for your newly created docker image. Do not forget to include the period after the image name. The period indicates the use of Dockerfile in the local repository. You can replace this with the path to your Dockerfile

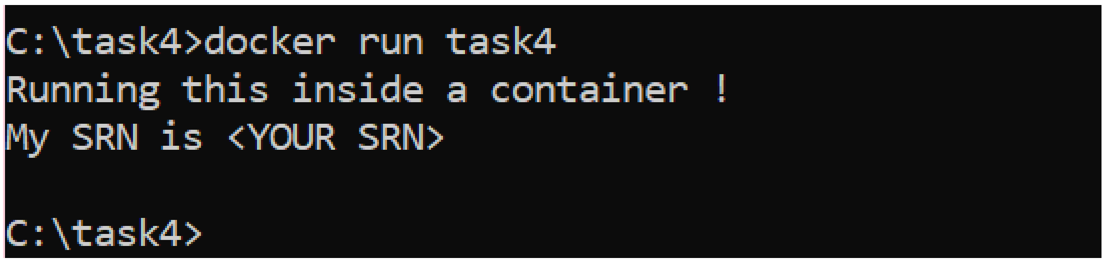


```
C:\task4>docker build -t task4 .
[+] Building 3.3s (11/11) FINISHED                                                              docker:default
 => [internal] load .dockerignore                                                                          0.2s
 => => transferring context: 2B                                                                            0.0s
 => [internal] load build definition from Dockerfile                                                       0.2s
 => => transferring dockerfile: 164B                                                                       0.0s
 => [internal] load metadata for docker.io/library/ubuntu:18.04                                            2.6s
 => [auth] library/ubuntu:pull token for registry-1.docker.io                                              0.0s
 => [1/5] FROM docker.io/library/ubuntu:18.04@sha256:152dc042452c496007f07ca9127571cb9c29697f42acbfad72324b2bb2e4  0.0s
 => [internal] load build context                                                                          0.1s
 => => transferring context: 156B                                                                          0.0s
 => CACHED [2/5] RUN apt-get update                                                                        0.0s
 => CACHED [3/5] RUN apt-get install gcc -y                                                                0.0s
 => CACHED [4/5] COPY program.c program.c                                                                  0.0s
 => CACHED [5/5] RUN gcc program.c                                                                         0.0s
 => exporting to image                                                                                     0.0s
 => => exporting layers                                                                                    0.0s
 => => writing image sha256:1b7700d84e52beb07d8f6d74eab0917237f375f14c228dc33c50b95d2dbb6a13               0.0s
 => => naming to docker.io/library/task4                                                                   0.0s

View build details: docker-desktop://dashboard/build/default/default/klzyko6en0is1kim6vxg6ujml

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview

C:\task4>
```

3. Run the container using

docker run <myimage-name>

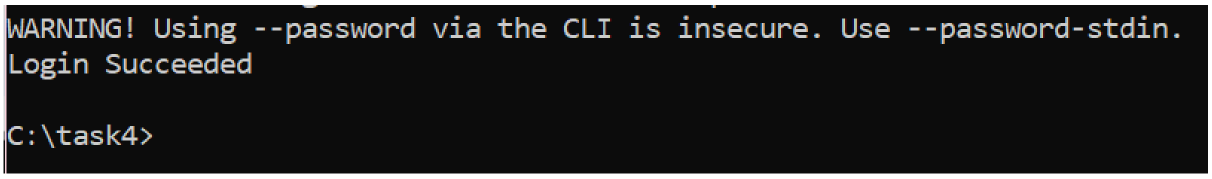
Take a screenshot of the C Program successfully running inside the container (4a.jpg).



```
C:\task4>docker run task4
Running this inside a container !
My SRN is <YOUR SRN>

C:\task4>
```

4. To push the image to Docker hub:
   a. Login from the terminal using

docker login -u "myusername" -p "mypassword" docker.io



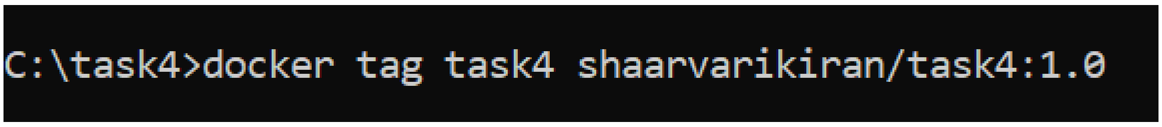
```
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded

C:\task4>
```

Note:

- *Avoid passwords with special characters like $ as it will be interpreted by shell and the above login command will not work.*
- *Write your username and password within double quotes*
  Eg. docker login -u "abc" -p "abc12" docker.io

b. Create a tag using:

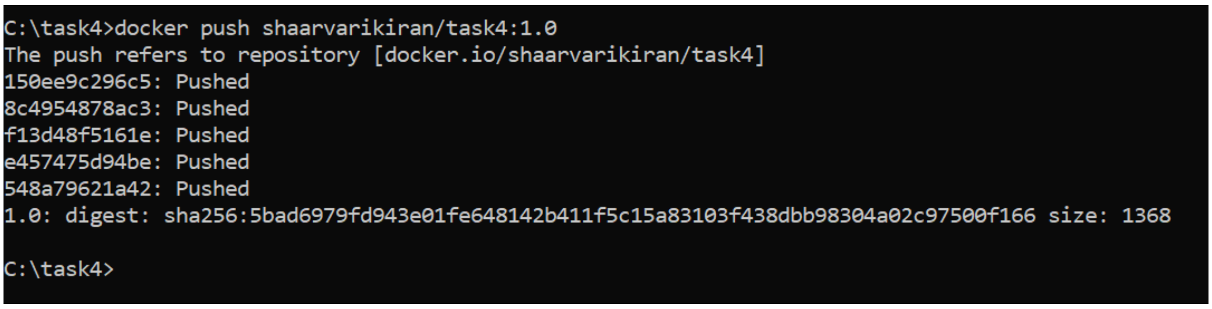
docker tag <myimage-name> <myusername>/<myimage-name>:<version-number>



```
C:\task4>docker tag task4 shaarvarikiran/task4:1.0
```

c. Push the image using

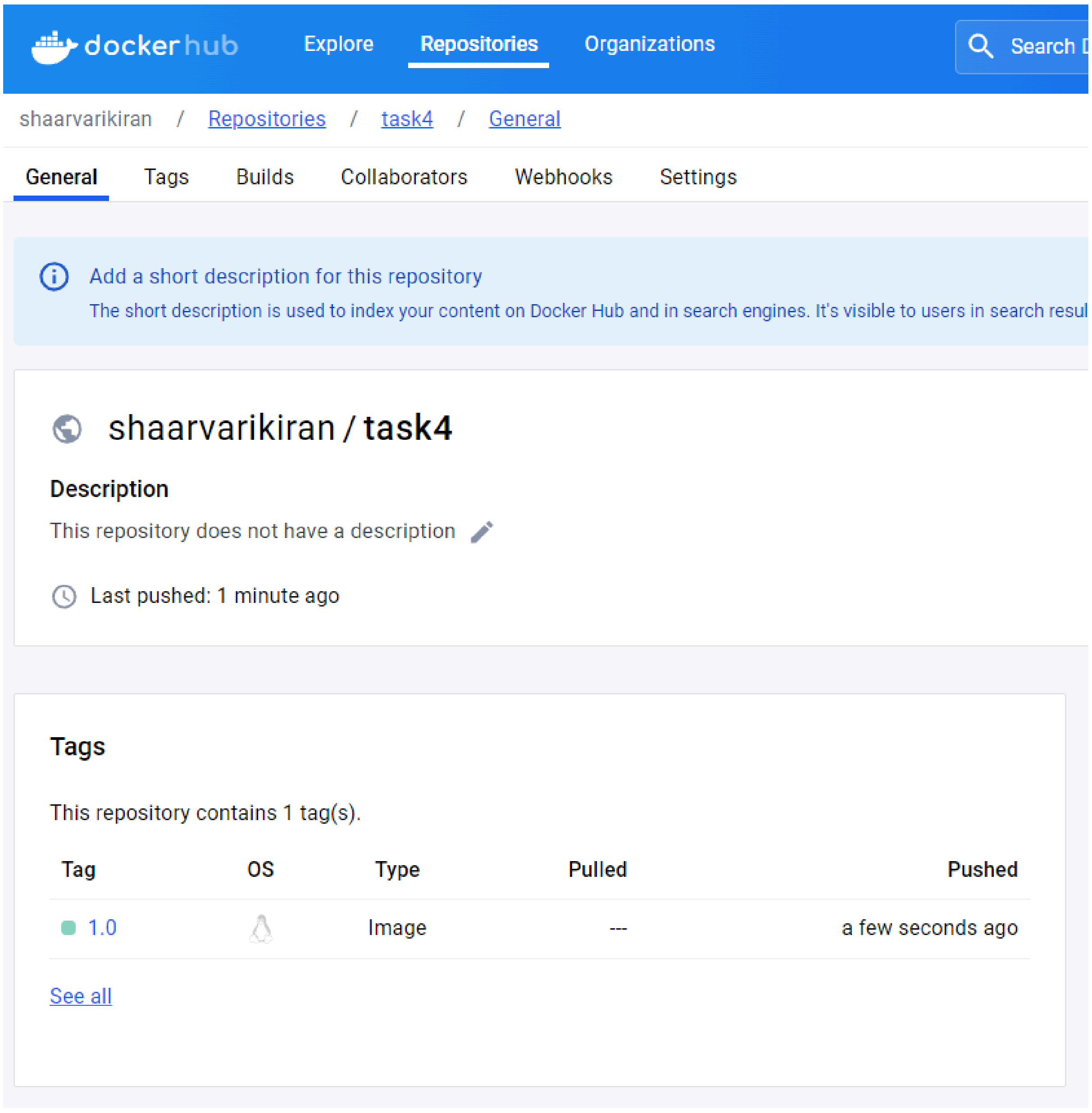
docker push <myusername>/<myimage-name>:<version-number>



```
C:\task4>docker push shaarvarikiran/task4:1.0
The push refers to repository [docker.io/shaarvarikiran/task4]
150ee9c296c5: Pushed
8c4954878ac3: Pushed
f13d48f5161e: Pushed
e457475d94be: Pushed
548a79621a42: Pushed
1.0: digest: sha256:5bad6979fd943e01fe648142b411f5c15a83103f438dbb98304a02c97500f166 size: 1368

C:\task4>
```

Login to docker hub and the image you pushed will appear in your repository.

Take a screenshot of the image pushed to Dockerhub (4b.jpg).



shaarvarikiran / Repositories / task4 / General

General Tags Builds Collaborators Webhooks Settings

Add a short description for this repository
The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search resul

shaarvarikiran / task4

Description
This repository does not have a description

Last pushed: 1 minute ago

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
🟩 1.0		Image	---	a few seconds ago

See all

Additional Resources/ Common bugs you might encounter:

1. How to debug and fix common docker issues.
2. Not able to build/run docker containers due to insufficient space: How To Remove Docker Images, Containers, and Volumes
3. Docker - Container is not running
4. Docker Build: A Beginner's Guide to Building Docker Images
5. Docker Container Tutorial #8 Exposing container ports
6. Overview of Docker Compose