Experiment 2 - Docker on Linux/Windows/Mac OS

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>_E2.pdf (Eg. A_PES1UG20CSXXX_Name_E2.pdf)

- **1a.jpg:** Screenshot of running docker hello-world.
- **2a.jpg:** Screenshot of C Program successfully run inside the container.
- **2b.jpg:** Screenshot of the image pushed to Dockerhub.
- **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 python-mongodb application running as a docker-compose application(logs of the application)
- **4b.jpg:** Screenshot of 3 python application writes and reads from MongoDB after scaling the python application.

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.

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

```
nidhi@nidhi-VirtualBox:~$ 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/
```

Task 2: Docker images and docker files

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 <image-name>*
    - Ubuntu 18.04 : *docker pull ubuntu*

```
nidhi@nidhi-VirtualBox:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
677076032cca: Pull complete
Digest: sha256:9a0bdde4188b896a372804be2384015e90e3f84906b750c1a53539b585fbbe7f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

    - Nginx : *docker pull nginx*

```
nidhi@nidhi-VirtualBox:~$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
8740c948ffd4: Pull complete
d2c0556a17c5: Pull complete
c8b9881f2c6a: Pull complete
693c3ffa8f43: Pull complete
8316c5e80e6d: Pull complete
b2fe3577faa4: Pull complete
Digest: sha256:b8f2383a95879e1ae064940d9a200f67a6c79e710ed82ac42263397367e7cc4e
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

    - Python : *docker pull python*

```
nidhi@nidhi-VirtualBox:~$ docker pull python
Using default tag: latest
latest: Pulling from library/python
bbeef03cda1f: Pull complete
f049f75f014e: Pull complete
56261d0e6b05: Pull complete
9bd150679dbd: Pull complete
5b282ee9da04: Pull complete
03f027d5e312: Pull complete
591b0f932310: Pull complete
1047c5f4cc7d: Pull complete
5b5cbe74bf76: Pull complete
Digest: sha256:6b85854518f812d94cf2dfee2386df85b9cb78835a872d4769b4335f584c43ba
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
```

- MongoDB : *docker pull mongo*

```
nidhi@nidhi-VirtualBox:~$ docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
7608715873ec: Pull complete
aa7638a79c68: Pull complete
6edb8e32447e: Pull complete
9b078a16e980: Pull complete
a09083ae9827: Pull complete
1d854f4142ac: Pull complete
b8b27ecbbd3b: Pull complete
af04d8f9bcf8: Pull complete
065e9d0b1cfc: Pull complete
Digest: sha256:3fe527dcddf277d4d5b278f5f03ddea5173cee84c792d12c5ac90c36ba40ba7a
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
```

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

   Use the following C program 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");

}
```

3. 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:

```
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.

```
nidhi@nidhi-VirtualBox:~/task2$ docker build -t task2 .
Sending build context to Docker daemon  3.072kB
Step 1/6 : FROM ubuntu:18.04
 ---> 5d2df19066ac
Step 2/6 : RUN apt-get update
 ---> Using cache
 ---> 362be1e5584d
Step 3/6 : RUN apt-get install gcc -y
 ---> Using cache
 ---> 5b916a9e820e
Step 4/6 : COPY program.c program.c
 ---> 4dc37a101467
Step 5/6 : RUN gcc program.c
 ---> Running in 56f7421576bb
Removing intermediate container 56f7421576bb
 ---> a870a17155a1
Step 6/6 : CMD ["./a.out"]
 ---> Running in c5ffb5492ac2
Removing intermediate container c5ffb5492ac2
 ---> f4999c66f3bf
Successfully built f4999c66f3bf
Successfully tagged task2:latest
```

3. Run the container using *docker run <myimage-name>*

Take a screenshot of the C Program successfully running inside the container (2a.jpg).

```
nidhi@nidhi-VirtualBox:~/task2$ docker run task2
Running this inside a container !
My SRN is <YOUR SRN HERE>
```

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.
WARNING! Your password will be stored unencrypted in /home/nidhi/.docker/config
.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Note: Avoid password with special characters like $ as it will be interpreted by shell and the above login command will not work.

    b. Create a tag using:

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

```
nidhi@nidhi-VirtualBox:~/task2$ docker tag task2 kubenidhi/task2:1.0
```

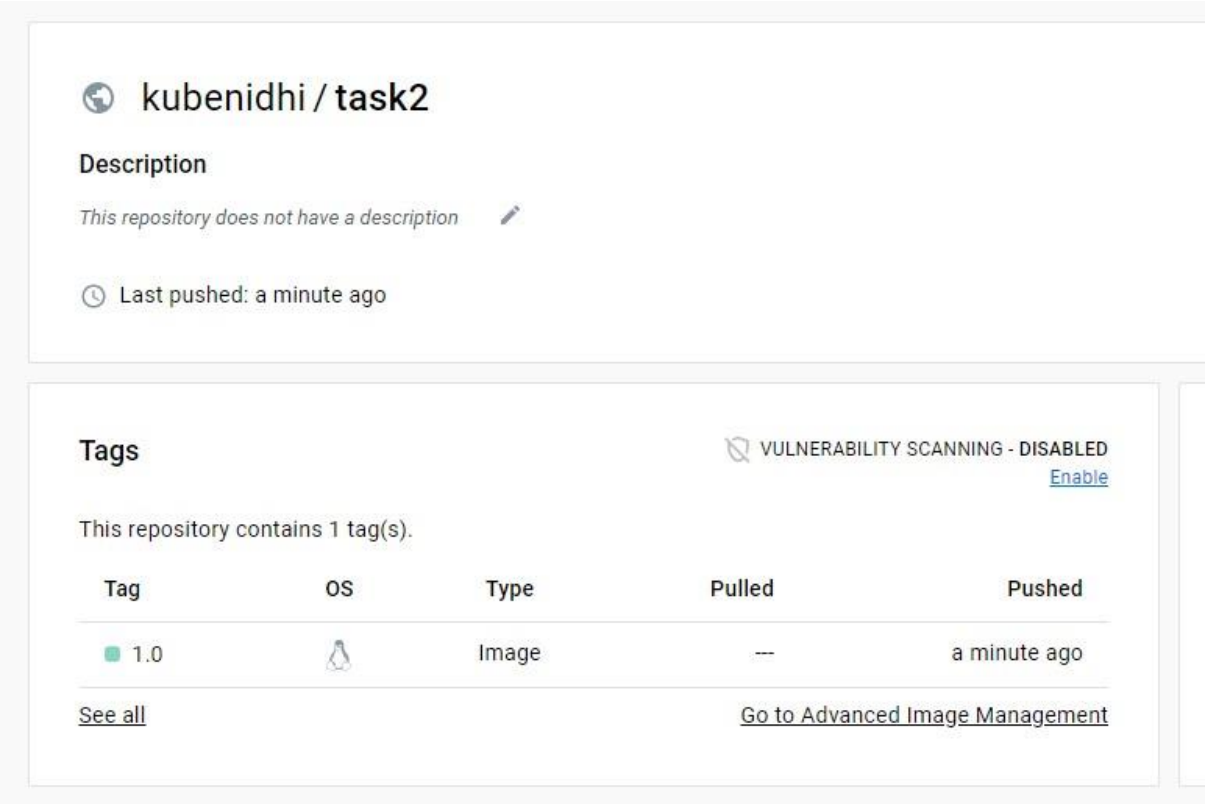
    c. Push the image using

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

```
nidhi@nidhi-VirtualBox:~/task2$ docker push kubenidhi/task2:1.0
The push refers to repository [docker.io/kubenidhi/task2]
8618a1c980b5: Pushed
072b33d29779: Pushed
9ed009b449cc: Pushed
77d3fd00cbbb: Pushed
475a54c2a93d: Mounted from library/ubuntu
1.0: digest: sha256:c377d65be4b8ad94e84560257191620815cf1310d3b9a8a89acae9fb88f
184c8 size: 1368
```

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

Take a screenshot of the image pushed to Dockerhub (2b.jpg).



Task 3: Exposing ports,docker networks

1. Download the sample HTML file from the Lab 2 drive folder, and modify your SRN.
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:

```
FROM nginx

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

Save and name the file as Dockerfile.

3. Build the docker image using *docker build -t <myimage-name> .*

```
nidhi@nidhi-VirtualBox:~/task3$ docker build -t task3 .
Sending build context to Docker daemon  3.072kB
Step 1/2 : FROM nginx
 ---> a99a39d070bf
Step 2/2 : COPY my.html /usr/share/nginx/html
 ---> 88d891d7ecb2
Successfully built 88d891d7ecb2
Successfully tagged task3:latest
```

4. 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).

```
nidhi@nidhi-VirtualBox:~/task3$ docker run -p 80:80 task3
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perf
orm configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-defa
ult.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/con
f.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.
sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.
sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/02/02 12:05:32 [notice] 1#1: using the "epoll" event method
2023/02/02 12:05:32 [notice] 1#1: nginx/1.23.3
2023/02/02 12:05:32 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6
)
2023/02/02 12:05:32 [notice] 1#1: OS: Linux 5.15.0-58-generic
2023/02/02 12:05:32 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/02/02 12:05:32 [notice] 1#1: start worker processes
2023/02/02 12:05:32 [notice] 1#1: start worker process 28
2023/02/02 12:05:32 [notice] 1#1: start worker process 29
2023/02/02 12:05:32 [notice] 1#1: start worker process 30
2023/02/02 12:05:32 [notice] 1#1: start worker process 31
```

5. 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).

6. 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.

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

docker run -dp 27017:27017 mongo

8. Download sample.py from the drive folder. Modify the SRN wherever mentioned.
9. 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*

```
nidhi@nidhi-VirtualBox:~/task3$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED              STA
TUS                      PORTS                                          NAMES
b54bf9c3985f   mongo          "docker-entrypoint.s…"   About a minute ago   Up
About a minute           0.0.0.0:27017->27017/tcp, :::27017->27017/tcp  friend
ly_wu
d581a0be6681   task3          "/docker-entrypoint.…"   3 minutes ago        Exi
ted (0) 2 minutes ago                                                     jolly_
ptolemy
4d75f20fbc7a   task2          "./a.out"                16 minutes ago       Exi
ted (0) 16 minutes ago                                                    sweet_
galois
8ab0f93baf48   a94141c89665   "/bin/sh -c 'gcc pro…"   18 minutes ago       Exi
ted (1) 18 minutes ago                                                    friend
ly_mahavira
7dc2a4980a97   hello-world    "/hello"                 32 minutes ago       Exi
ted (0) 32 minutes ago                                                    clever
_thompson
6903d9e79340   hello-world    "/hello"                 28 hours ago         Exi
ted (0) 28 hours ago                                                      blissf
ul_mahavira
b59af8aabcf0   hello-world    "/hello"                 29 hours ago         Exi
ted (0) 29 hours ago                                                      xenodo
chial_jepsen
```

   b. Run *docker inspect <container-id>*. Find this IPAddress field and note this down. Modify this IP address in the sample.py file.

```
                    "EndpointID": "ce43db0117a65be53717d9db97b6ee1db4d5358c
4306941b9bc8726af16e",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "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:

```
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.

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

docker build -t <myimage-name> .

```
nidhi@nidhi-VirtualBox:~/task3$ docker build -t task3 .
Sending build context to Docker daemon  4.096kB
Step 1/5 : FROM python
 ---> 63490c269128
Step 2/5 : RUN apt-get update
 ---> Using cache
 ---> 39d40ca8a31d
Step 3/5 : RUN pip install pymongo
 ---> Using cache
 ---> a192957627ed
Step 4/5 : COPY sample.py sample.py
 ---> Using cache
 ---> a0e95bae5348
Step 5/5 : CMD ["python", "sample.py"]
 ---> Using cache
 ---> ad3f916ce169
Successfully built ad3f916ce169
Successfully tagged task3:latest
```

docker run <myimage-name>

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

```
nidhi@nidhi-VirtualBox:~/task3$ docker run task3
Inserted into the MongoDB database!
Fecthed from MongoDB:  {'_id': ObjectId('63dba98904d7888ae8be5f93'), 'Name:': '
<Your Name>', 'SRN': '<YOUR_SRN>'}
nidhi@nidhi-VirtualBox:~/task3$
```

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.

```
nidhi@nidhi-VirtualBox:~/task3$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
  PORTS                                          NAMES
b54bf9c3985f   mongo     "docker-entrypoint.s…"   9 minutes ago    Up 9 minutes
  0.0.0.0:27017->27017/tcp, :::27017->27017/tcp  friendly_wu
```

12. Note the container id andstop running the container using

docker stop <container-id>

```
nidhi@nidhi-VirtualBox:~/task3$ docker stop b54bf9c3985f
b54bf9c3985f
```

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 possibly* 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

```
nidhi@nidhi-VirtualBox:~/task3$ docker network create my-bridge-network
784e155d973a09283171b4b05a2e3b64f7facf426ffbb46d6b53bc36d0283acb
nidhi@nidhi-VirtualBox:~/task3$
```

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).

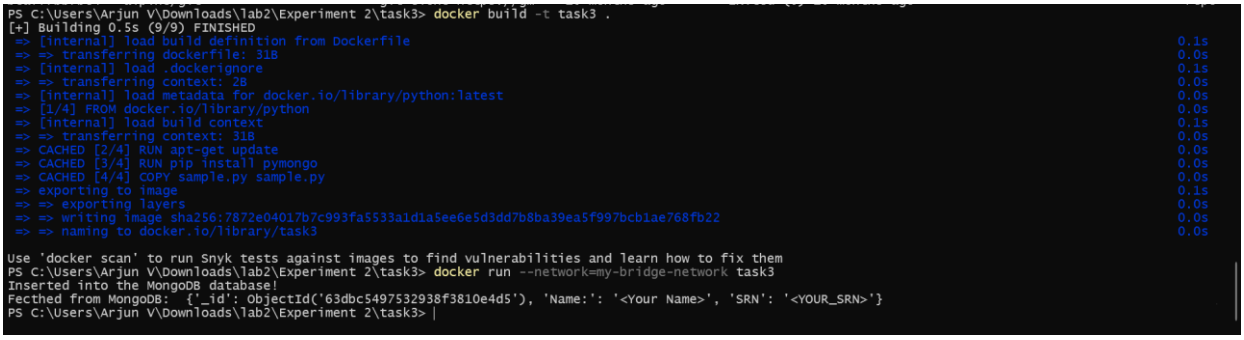
```
PS C:\Users\Arjun V\Downloads\lab2\Experiment 2\task3> docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED              STATUS                          PORTS                     NAMES
9c507218f9bb   mongo:latest         "docker-entrypoint.s…"   About a minute ago   Up About a minute               0.0.0.0:27017->27017/tcp  mongodb
76085aa82e63   task41-pycode        "python sample.py"       9 minutes ago        Exited (0) 9 minutes ago                                  task41-pycode-1
c6ee445d0500   task41-pycode        "python sample.py"       9 minutes ago        Exited (0) 9 minutes ago                                  task41-pycode-3
0b37a6c834de   task41-pycode        "python sample.py"       9 minutes ago        Exited (0) 9 minutes ago                                  task41-pycode-2
a98690ac4ac9   mongo                "docker-entrypoint.s…"   17 minutes ago       Exited (0) 4 minutes ago                                  task41-mongodb-1
cbb052fd9888   task3                "python sample.py"       18 minutes ago       Exited (0) 18 minutes ago                                 sad_kowalevski
```

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).



Task 4: Docker compose

1. For the purposes of this lab make sure all the following files are in the same directory
    a. docker-compose.yml (present on the drive)
    b. Dockerfile (same file used for task 3)
    c. sample.py (same file used for task 3)
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) (4a.jpg).

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 (4b.jpg)

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 CLI & Dockerfile Cheat Sheet
5. Docker Build: A Beginner's Guide to Building Docker Images
6. Docker Container Tutorial #8 Exposing container ports
7. Overview of Docker Compose