Deployment Portfolio Task 4

SWE40006 - SOFTWARE DEVELOPMENT AND EVOLUTION

SUMMER – 2024 SUBMITTED ON 30th OF JUNE

Student and Lecturer Details

Name	ID	Lecturer	Class
Vi Luan Dang	103802759	Dr. Thomas Hang Nsam@swin.edu.au	Monday 13:00 PM

Self-Assessment Details

Declaration ò task level attempted (P/C/D/HD)

	Pass	Credit	Distinction	High Distinction
Self-Assessment				✓

	Included & attempted
Task 4.1: Pass	✓
Task 4.2: Credit	✓
Task 4.3: Distinction	✓
Task 4.4: High Distinction	✓

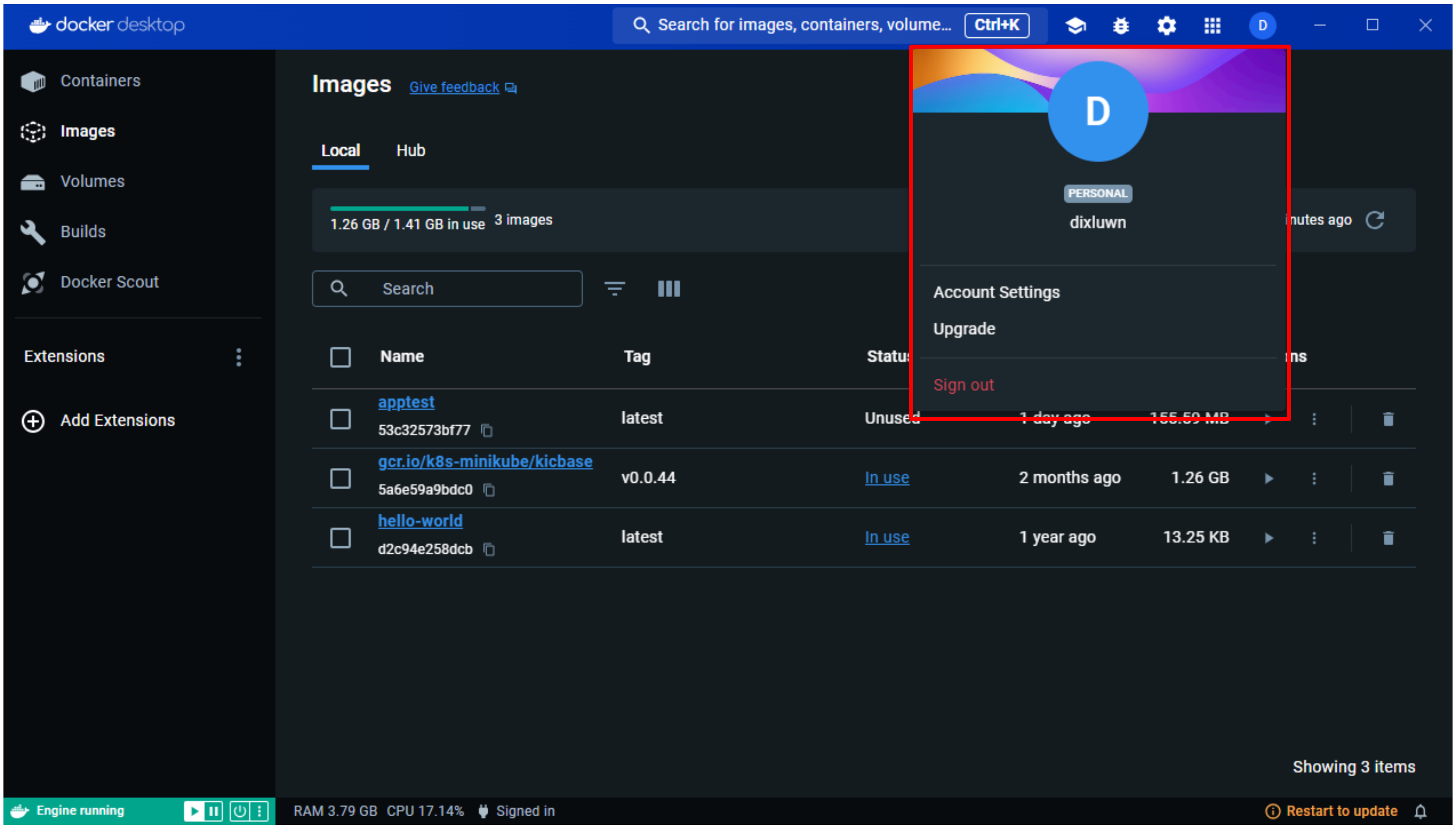
Table of Content

Assignment Report

Task 4.1P:

4.1A and 4.1B. Create Docker account and install Docker Client

I already have been working with Docker lately, so this is already installed on my machine.

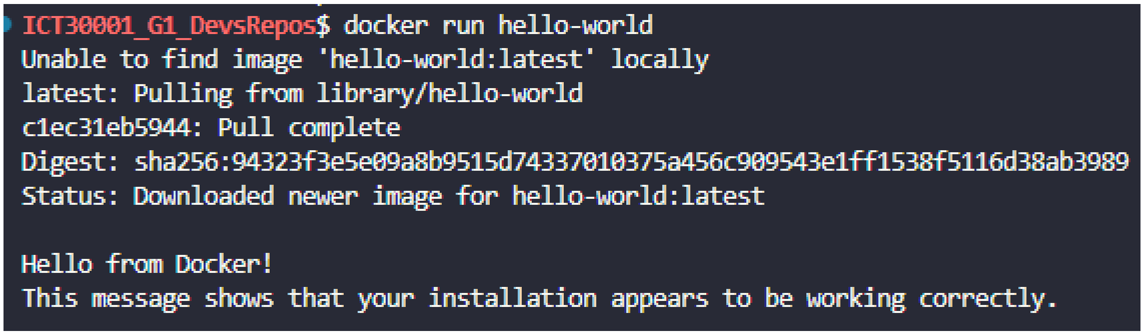


Name	Tag	Status		
apptest 53c32573bf77	latest	Unused	1 day ago	155.59 MB
gcr.io/k8s-minikube/kicbase 5a6e59a9bdc0	v0.0.44	In use	2 months ago	1.26 GB
hello-world d2c94e258dcb	latest	In use	1 year ago	13.25 KB

Figure 1: Docker account on Docker Client

4.2C Deploy Hello-world

Docker's library has a "hello-world" image, so I will just pull that image onto my local engine and run it.



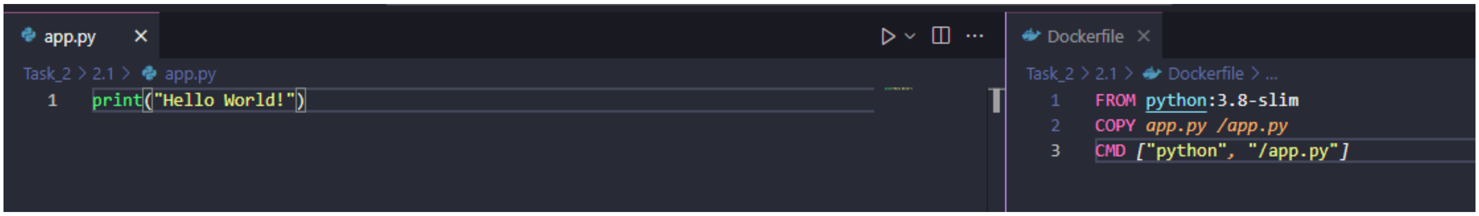
```
ICT30001_G1_DevsRepos$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:94323f3e5e09a8b9515d74337010375a456c909543e1ff1538f5116d38ab3989
Status: Downloaded newer image for hello-world:latest

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

Figure 2: "Hello-world" Docker image.

Task 4.2C:

4.2A Deploy a Python app

For this task, we will create a simple Python app, and dockerize it.

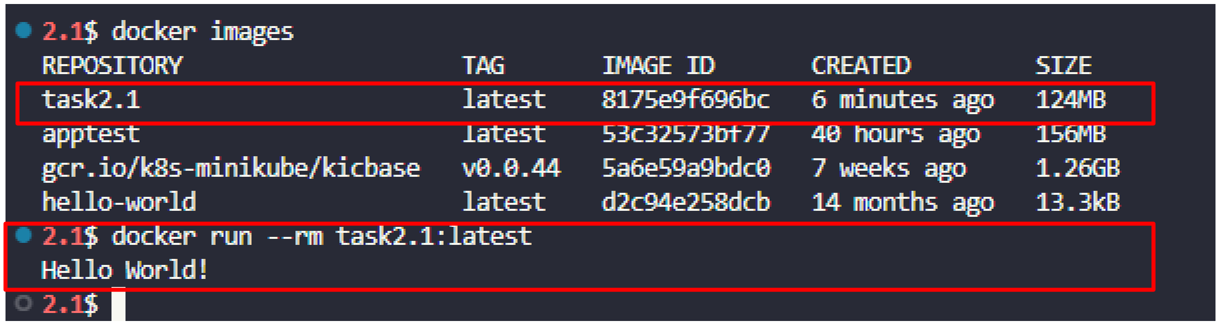


```python
print("Hello World!")
```

```dockerfile
FROM python:3.8-slim
COPY app.py /app.py
CMD ["python", "/app.py"]
```

Figure 3: Python app and its Dockerfile.

The Dockerfile will include a base image, and we will copy our application's source code onto the container, last but not least we will run the application using "CMD".

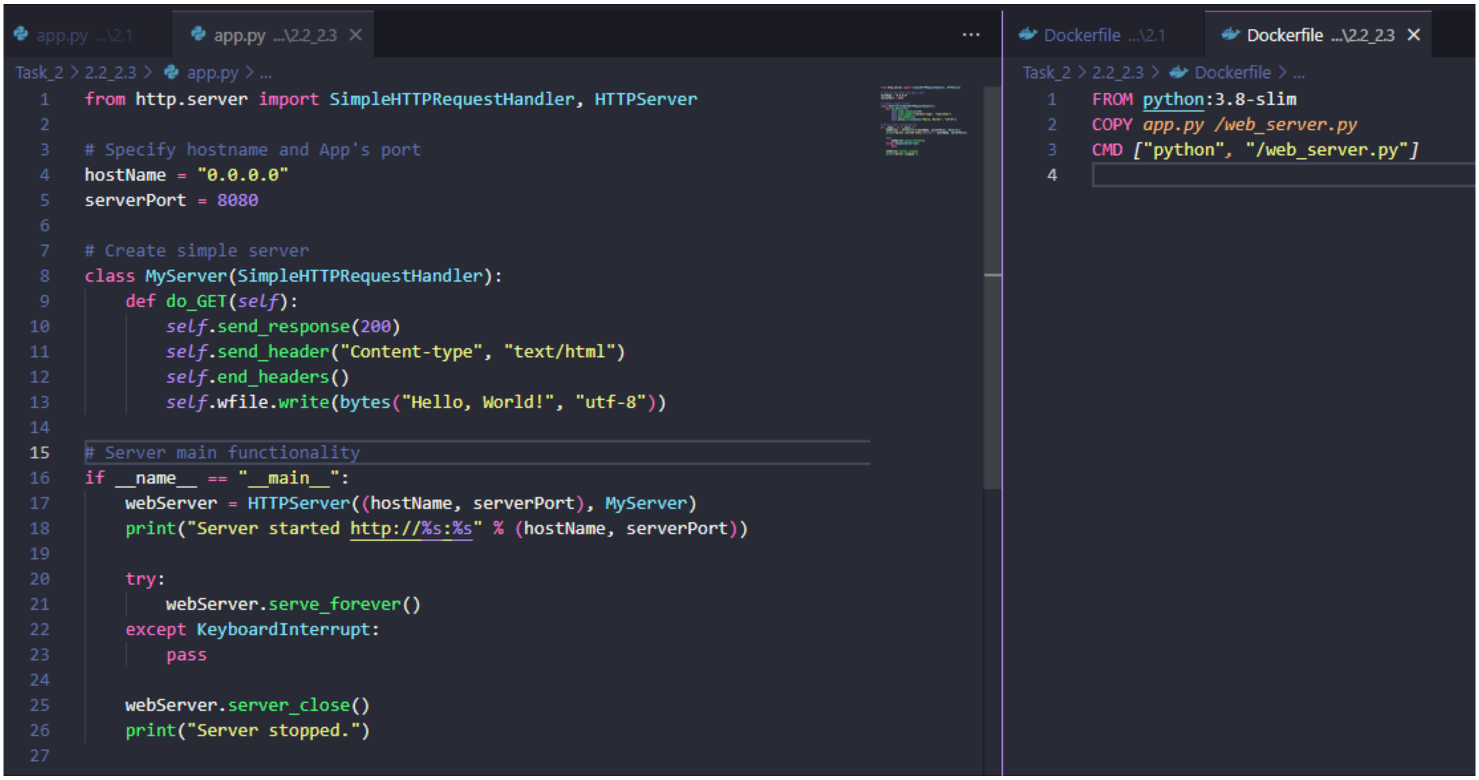


```
2.1$ docker images
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
task2.1                     latest    8175e9f696bc   6 minutes ago    124MB
apptest                     latest    53c32573bf77   40 hours ago     156MB
gcr.io/k8s-minikube/kicbase v0.0.44   5a6e59a9bdc0   7 weeks ago      1.26GB
hello-world                 latest    d2c94e258dcb   14 months ago    13.3kB
2.1$ docker run --rm task2.1:latest
Hello World!
2.1$
```

Figure 4: Python container successfully deployed.

4.2B and 4.2C Add Python Install run-time and Deploy hello-world web server

For this task, we will modify our python application so that it will be a web server and will require Python run-time.

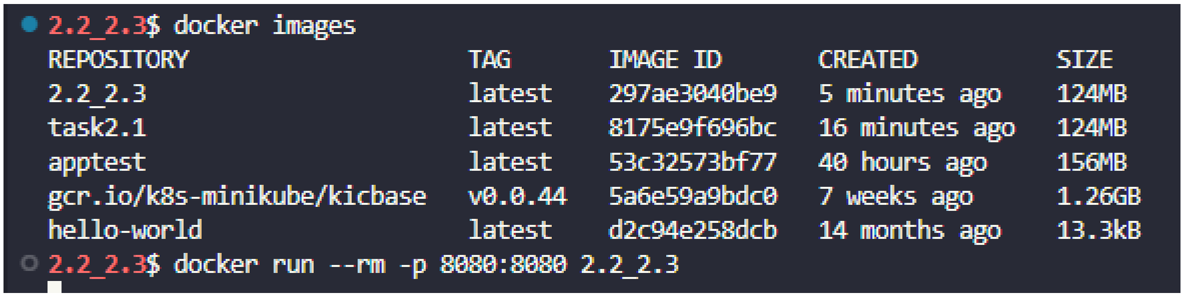


```python
from http.server import SimpleHTTPRequestHandler, HTTPServer

# Specify hostname and App's port
hostName = "0.0.0.0"
serverPort = 8080

# Create simple server
class MyServer(SimpleHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        self.wfile.write(bytes("Hello, World!", "utf-8"))

# Server main functionality
if __name__ == "__main__":
    webServer = HTTPServer((hostName, serverPort), MyServer)
    print("Server started http://%s:%s" % (hostName, serverPort))

    try:
        webServer.serve_forever()
    except KeyboardInterrupt:
        pass

    webServer.server_close()
    print("Server stopped.")
```

```dockerfile
FROM python:3.8-slim
COPY app.py /web_server.py
CMD ["python", "/web_server.py"]
```

Figure 5: Simple Python Web server

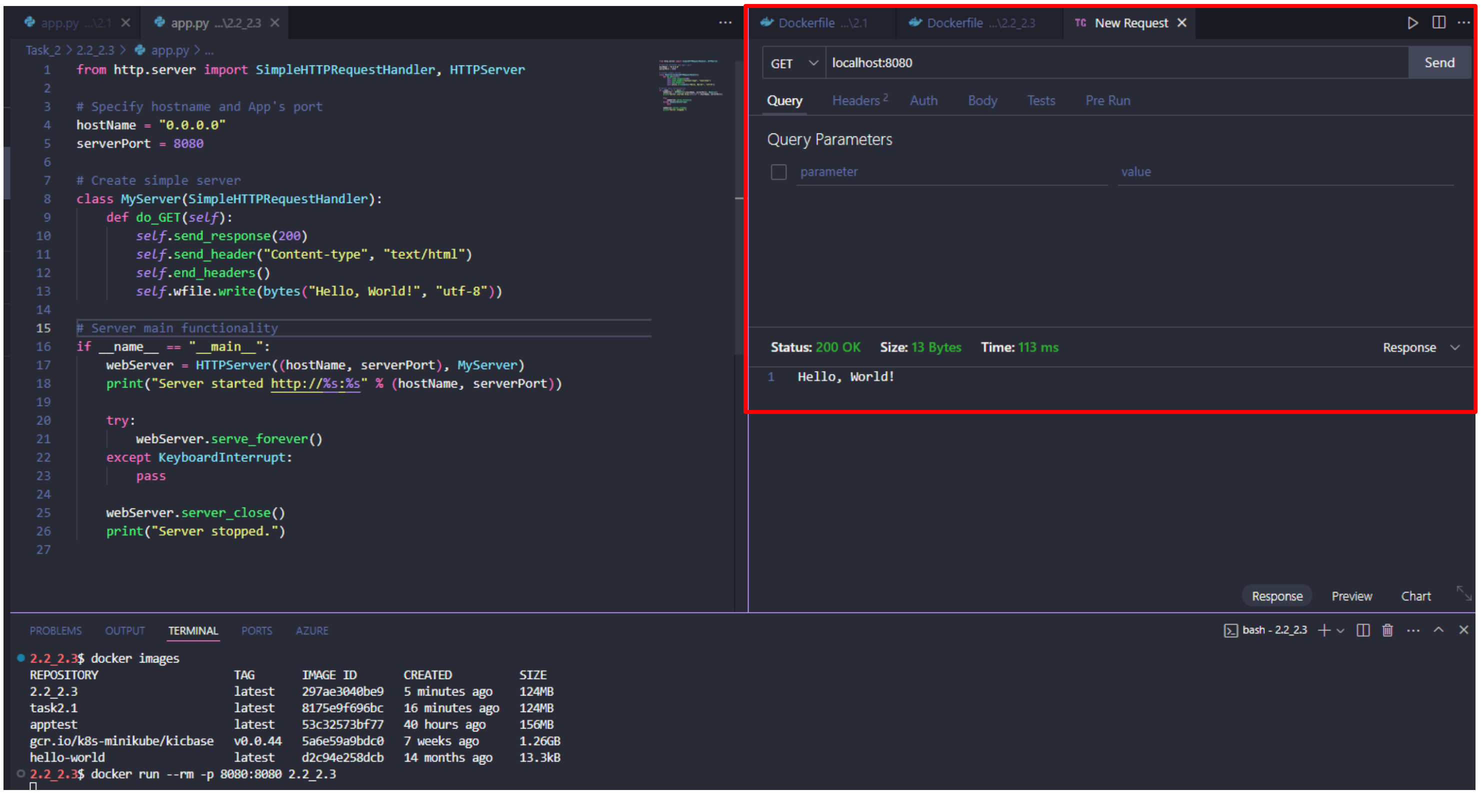
The Python web application will create a simple HTTPServer to handle our request, and it will return the "Hello, World!" upon the request is sent.



```
2.2_2.3$ docker images
REPOSITORY                   TAG       IMAGE ID       CREATED          SIZE
2.2_2.3                      latest    297ae3040be9   5 minutes ago    124MB
task2.1                      latest    8175e9f696bc   16 minutes ago   124MB
apptest                      latest    53c32573bf77   40 hours ago     156MB
gcr.io/k8s-minikube/kicbase  v0.0.44   5a6e59a9bdc0   7 weeks ago      1.26GB
hello-world                  latest    d2c94e258dcb   14 months ago    13.3kB
2.2_2.3$ docker run --rm -p 8080:8080 2.2_2.3
```

Figure 6: Docker Images is built and run.

Our application is hosted on localhost at port 8080, we can use some API testing application to test if our web is successfully deployed.



```python
from http.server import SimpleHTTPRequestHandler, HTTPServer

# Specify hostname and App's port
hostName = "0.0.0.0"
serverPort = 8080

# Create simple server
class MyServer(SimpleHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        self.wfile.write(bytes("Hello, World!", "utf-8"))

# Server main functionality
if __name__ == "__main__":
    webServer = HTTPServer((hostName, serverPort), MyServer)
    print("Server started http://%s:%s" % (hostName, serverPort))

    try:
        webServer.serve_forever()
    except KeyboardInterrupt:
        pass

    webServer.server_close()
    print("Server stopped.")
```

```
GET  localhost:8080

Status: 200 OK   Size: 13 Bytes   Time: 113 ms

1  Hello, World!
```

```
2.2_2.3$ docker images
REPOSITORY                   TAG       IMAGE ID       CREATED          SIZE
2.2_2.3                      latest    297ae3040be9   5 minutes ago    124MB
task2.1                      latest    8175e9f696bc   16 minutes ago   124MB
apptest                      latest    53c32573bf77   40 hours ago     156MB
gcr.io/k8s-minikube/kicbase  v0.0.44   5a6e59a9bdc0   7 weeks ago      1.26GB
hello-world                  latest    d2c94e258dcb   14 months ago    13.3kB
2.2_2.3$ docker run --rm -p 8080:8080 2.2_2.3
```

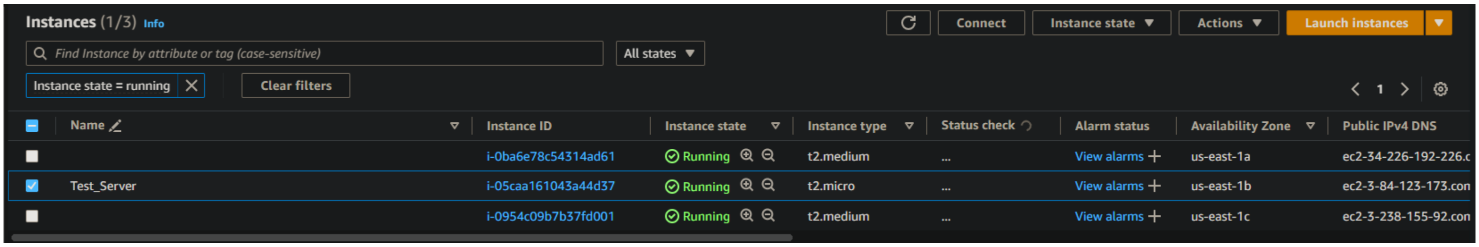
Figure 7: API testing

The API extension return our expected text, we can confirm that our webserver is deployed successfully.

4.2D Pull app to another Docker device

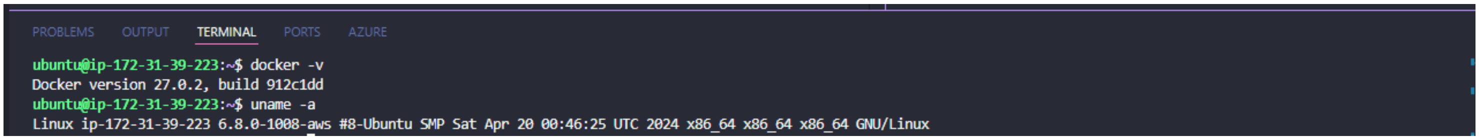
For this task, we will launch an EC2 Instance on AWS, install docker engine on that instance, pull the Docker image, and subsequently run the Docker Image to see if our application can work on other devices.

First, we will create an EC2 instance on AWS. I will use pre-installed user-data to install Docker, we can check if Docker is installed once the instance is launched.



	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
□		i-0ba6e78c54314ad61	Running	t2.medium	...	View alarms +	us-east-1a	ec2-34-226-192-226.c
☑	Test_Server	i-05caa161043a44d37	Running	t2.micro	...	View alarms +	us-east-1b	ec2-3-84-123-173.con
□		i-0954c09b7b37fd001	Running	t2.medium	...	View alarms +	us-east-1c	ec2-3-238-155-92.con

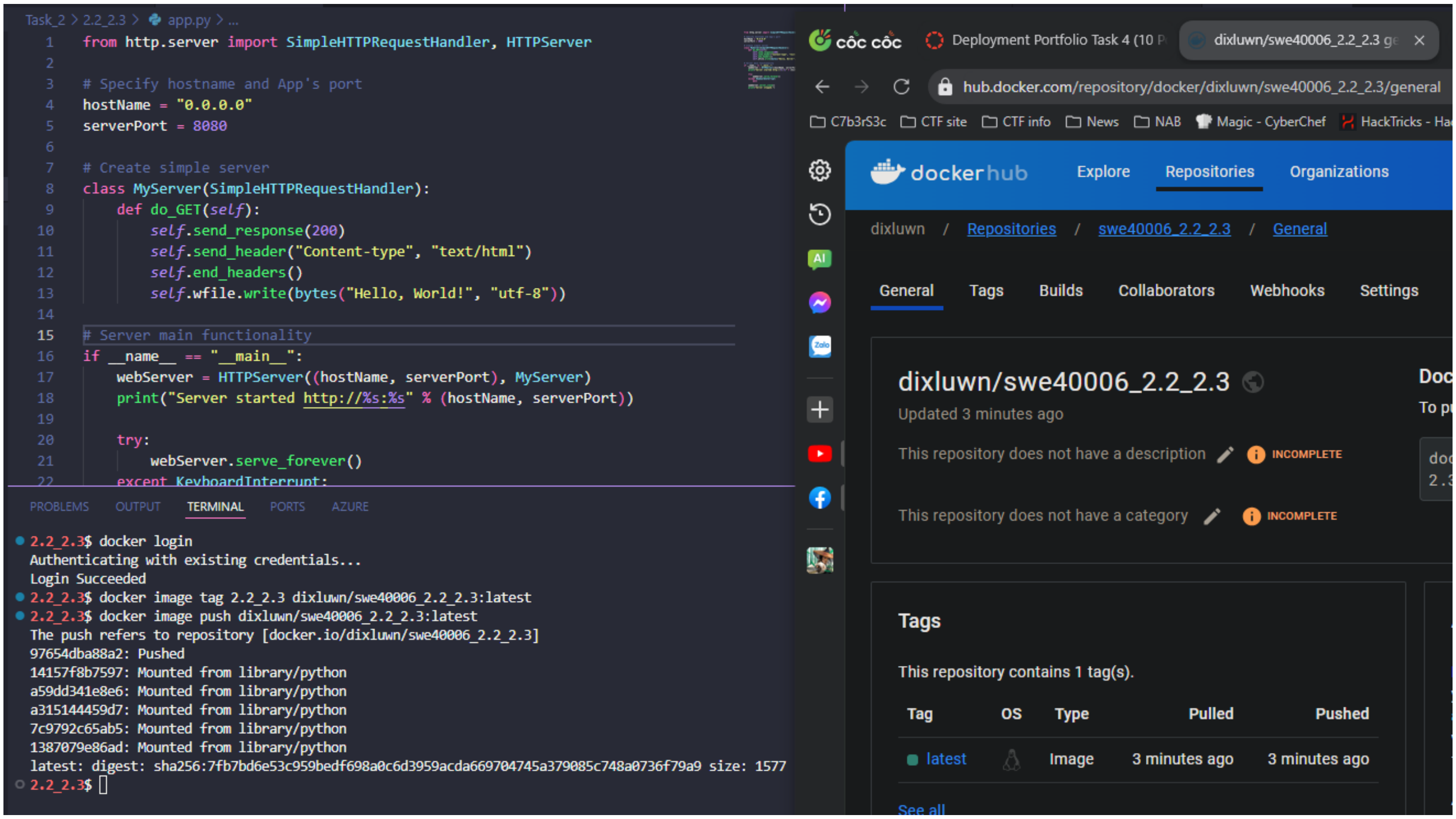
Figure 8: Instance successfully launched.



```
ubuntu@ip-172-31-39-223:~$ docker -v
Docker version 27.0.2, build 912c1dd
ubuntu@ip-172-31-39-223:~$ uname -a
Linux ip-172-31-39-223 6.8.0-1008-aws #8-Ubuntu SMP Sat Apr 20 00:46:25 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
```

Figure 9: Docker successfully installed

Next, we will tag our Docker Image and push it onto Docker Hub.



```python
from http.server import SimpleHTTPRequestHandler, HTTPServer

# Specify hostname and App's port
hostName = "0.0.0.0"
serverPort = 8080

# Create simple server
class MyServer(SimpleHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        self.wfile.write(bytes("Hello, World!", "utf-8"))

# Server main functionality
if __name__ == "__main__":
    webServer = HTTPServer((hostName, serverPort), MyServer)
    print("Server started http://%s:%s" % (hostName, serverPort))

    try:
        webServer.serve_forever()
    except KeyboardInterrupt:
```

```
2.2_2.3$ docker login
Authenticating with existing credentials...
Login Succeeded
2.2_2.3$ docker image tag 2.2_2.3 dixluwn/swe40006_2.2_2.3:latest
2.2_2.3$ docker image push dixluwn/swe40006_2.2_2.3:latest
The push refers to repository [docker.io/dixluwn/swe40006_2.2_2.3]
97654dba88a2: Pushed
14157f8b7597: Mounted from library/python
a59dd341e8e6: Mounted from library/python
a315144459d7: Mounted from library/python
7c9792c65ab5: Mounted from library/python
1387079e86ad: Mounted from library/python
latest: digest: sha256:7fb7bd6e53c959bedf698a0c6d3959acda669704745a379085c748a0736f79a9 size: 1577
2.2_2.3$ 
```

dixluwn/swe40006_2.2_2.3

Updated 3 minutes ago

This repository does not have a description INCOMPLETE

This repository does not have a category INCOMPLETE

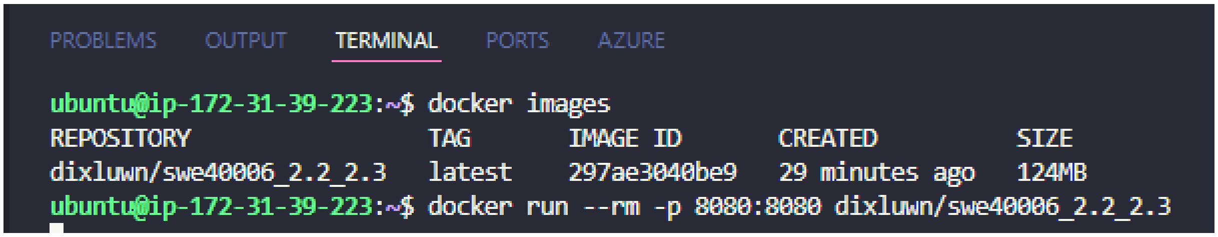
Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	3 minutes ago	3 minutes ago

Figure 10: Image successfully tagged and pushed onto Docker Hub

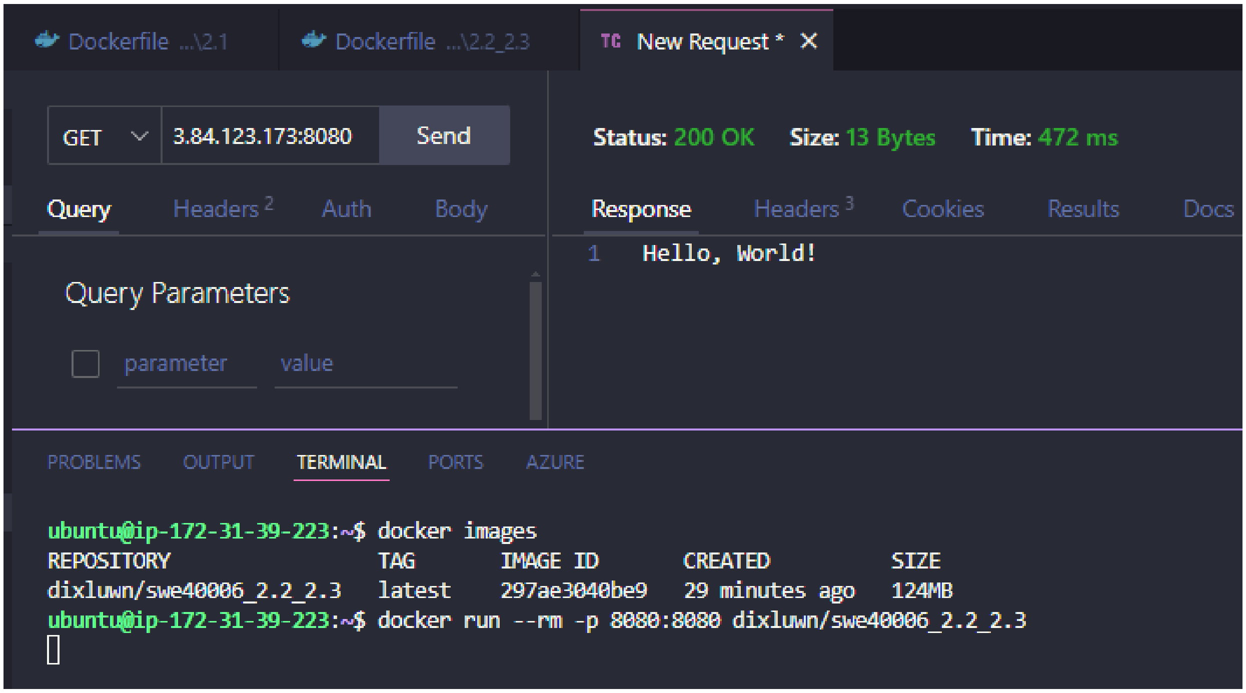
We can see that our image is successfully tagged and pushed onto Docker Hub, we can then pull this image on our EC2 Instance and run the container to see if it works.



```
ubuntu@ip-172-31-39-223:~$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
dixluwn/swe40006_2.2_2.3  latest    297ae3040be9   29 minutes ago   124MB
ubuntu@ip-172-31-39-223:~$ docker run --rm -p 8080:8080 dixluwn/swe40006_2.2_2.3
```

Figure 11: Our application is successfully pulled and run.

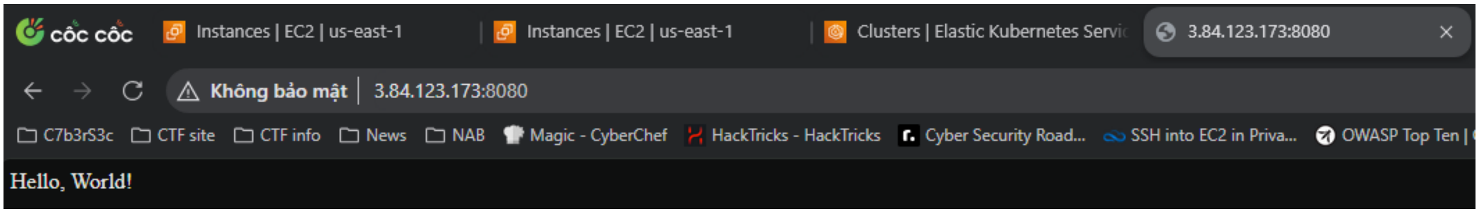
Once the webserver is launched, we can also use API testing extension to see if our application is working as expected.



```
1   Hello, World!
```

```
ubuntu@ip-172-31-39-223:~$ docker images
REPOSITORY                 TAG       IMAGE ID       CREATED          SIZE
dixluwn/swe40006_2.2_2.3   latest    297ae3040be9   29 minutes ago   124MB
ubuntu@ip-172-31-39-223:~$ docker run --rm -p 8080:8080 dixluwn/swe40006_2.2_2.3
```

Figure 12: Application successfully deployed onto EC2 instance.



Hello, World!

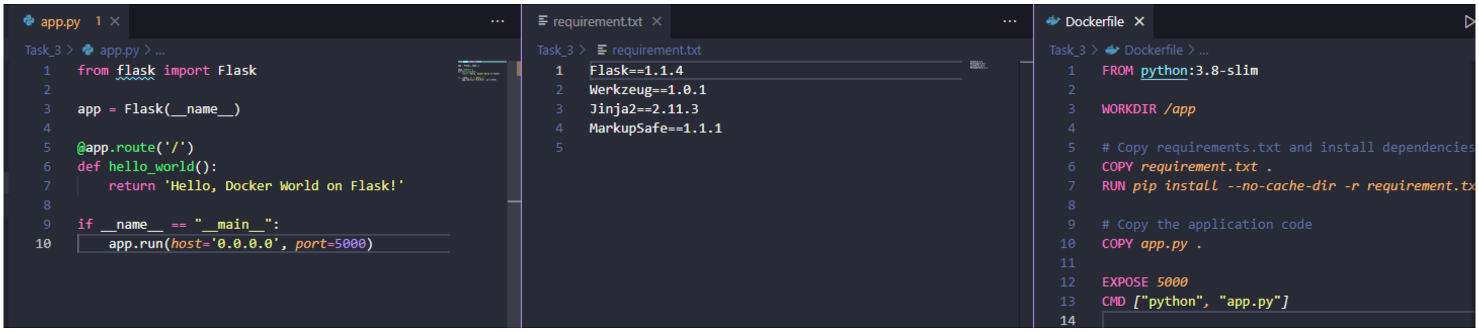
Figure 13: Application on the browser.

This will ensure that our application is successfully deployed and run on the browser, it will also conclude this task.

Task 4.3D:

4.3A Develop a new app

For this task, we will create a Python Webserver, but this time we will use Flask to host our sever.



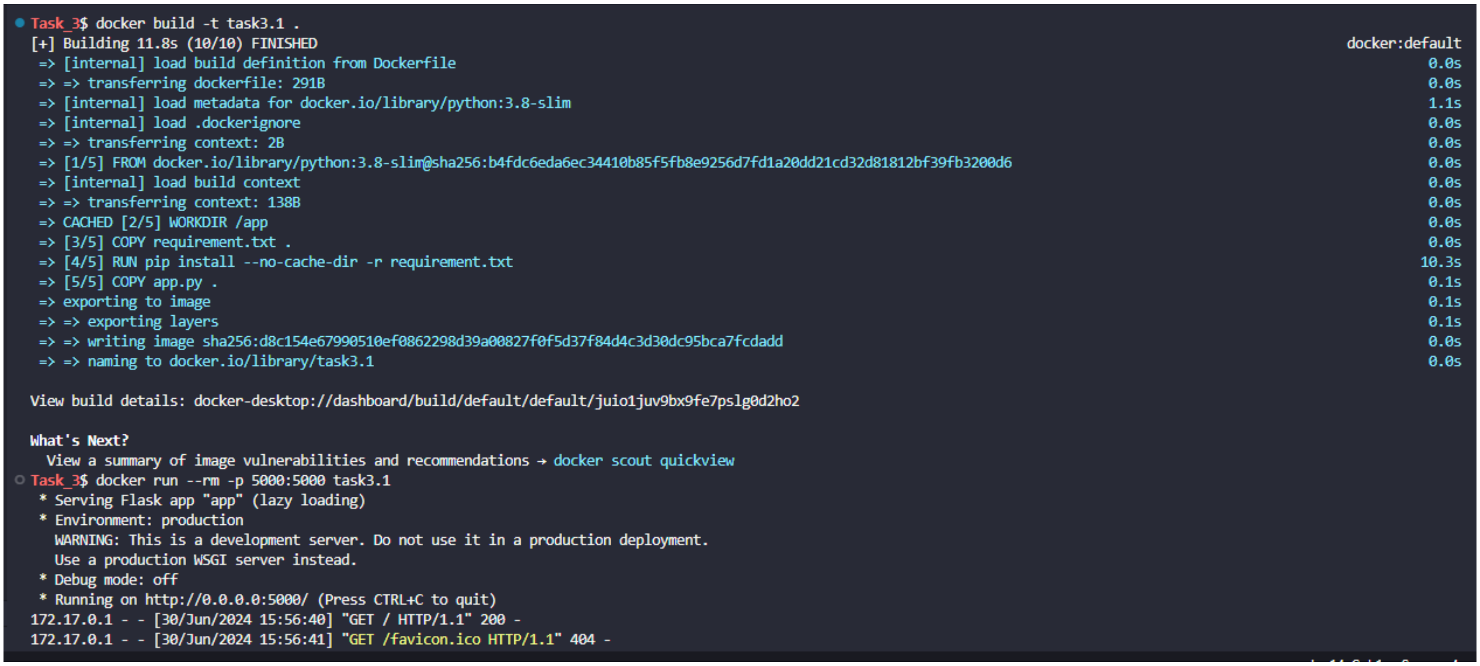
```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, Docker World on Flask!'

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

```
Flask==1.1.4
Werkzeug==1.0.1
Jinja2==2.11.3
MarkupSafe==1.1.1
```

```dockerfile
FROM python:3.8-slim

WORKDIR /app

# Copy requirements.txt and install dependencies
COPY requirement.txt .
RUN pip install --no-cache-dir -r requirement.tx

# Copy the application code
COPY app.py .

EXPOSE 5000
CMD ["python", "app.py"]

```

Figure 14: Modified Python Webserver and Dockerfile.

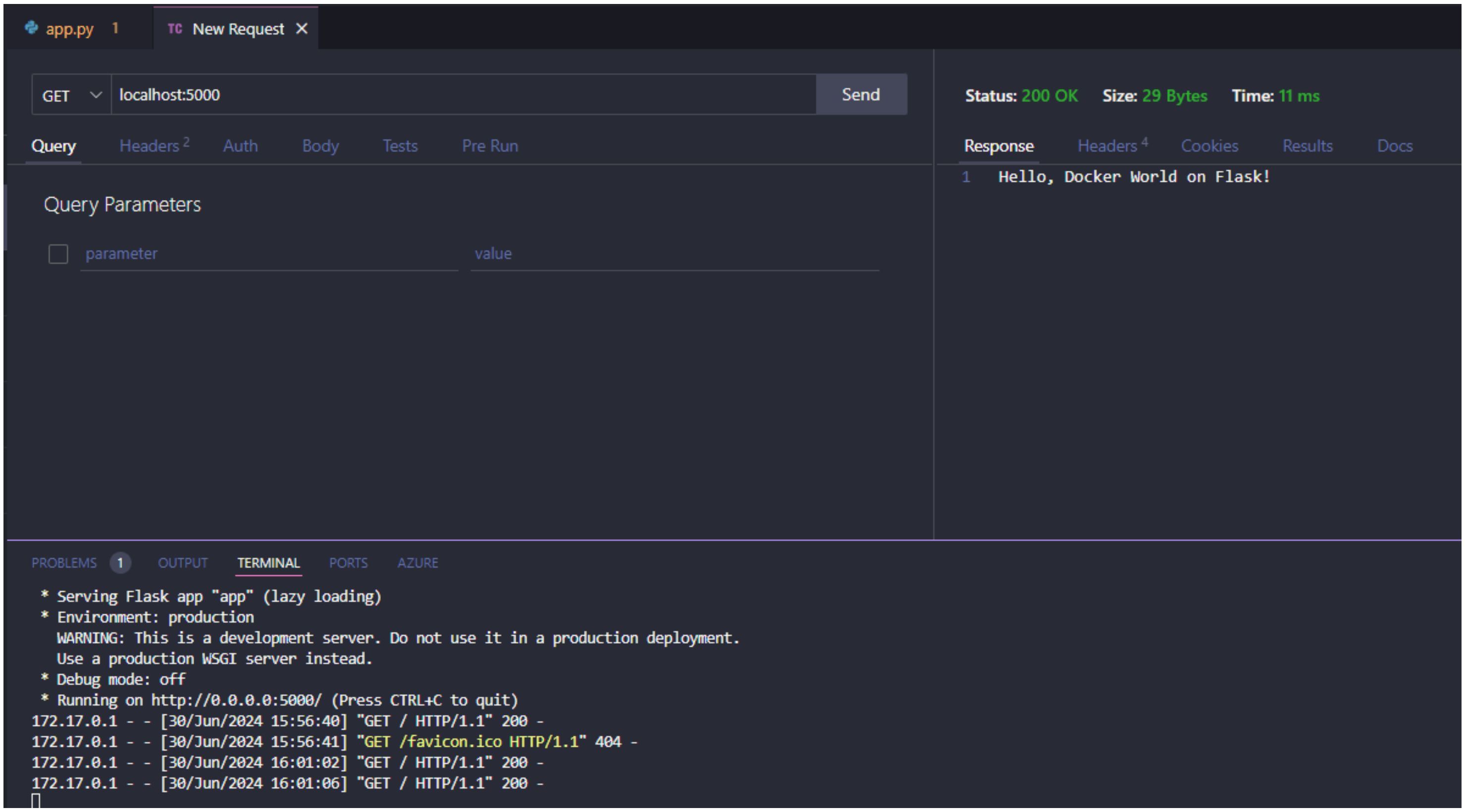
For our application, we will just need to import Flask as our dependency and start our application on the localhost. We will also add a requirement file and include it onto our Docker so that when the container is built, it will also include our required dependencies for the web to start.



```
Task_3$ docker build -t task3.1 .
[+] Building 11.8s (10/10) FINISHED                                                                     docker:default
 => [internal] load build definition from Dockerfile                                                              0.0s
 => => transferring dockerfile: 291B                                                                              0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                                                1.1s
 => [internal] load .dockerignore                                                                                 0.0s
 => => transferring context: 2B                                                                                   0.0s
 => [1/5] FROM docker.io/library/python:3.8-slim@sha256:b4fdc6eda6ec34410b85f5fb8e9256d7fd1a20dd21cd32d81812bf39fb3200d6  0.0s
 => [internal] load build context                                                                                 0.0s
 => => transferring context: 138B                                                                                 0.0s
 => CACHED [2/5] WORKDIR /app                                                                                     0.0s
 => [3/5] COPY requirement.txt .                                                                                  0.0s
 => [4/5] RUN pip install --no-cache-dir -r requirement.txt                                                      10.3s
 => [5/5] COPY app.py .                                                                                           0.1s
 => exporting to image                                                                                            0.1s
 => => exporting layers                                                                                           0.1s
 => => writing image sha256:d8c154e67990510ef0862298d39a00827f0f5d37f84d4c3d30dc95bca7fcdadd                      0.0s
 => => naming to docker.io/library/task3.1                                                                        0.0s

View build details: docker-desktop://dashboard/build/default/default/juio1juv9bx9fe7pslg0d2ho2

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
Task_3$ docker run --rm -p 5000:5000 task3.1
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
172.17.0.1 - - [30/Jun/2024 15:56:40] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [30/Jun/2024 15:56:41] "GET /favicon.ico HTTP/1.1" 404 -
```

Figure 15: Our docker image is successfully built and run

Again, we will use API testing extension to test if our application is successfully built and deployed.



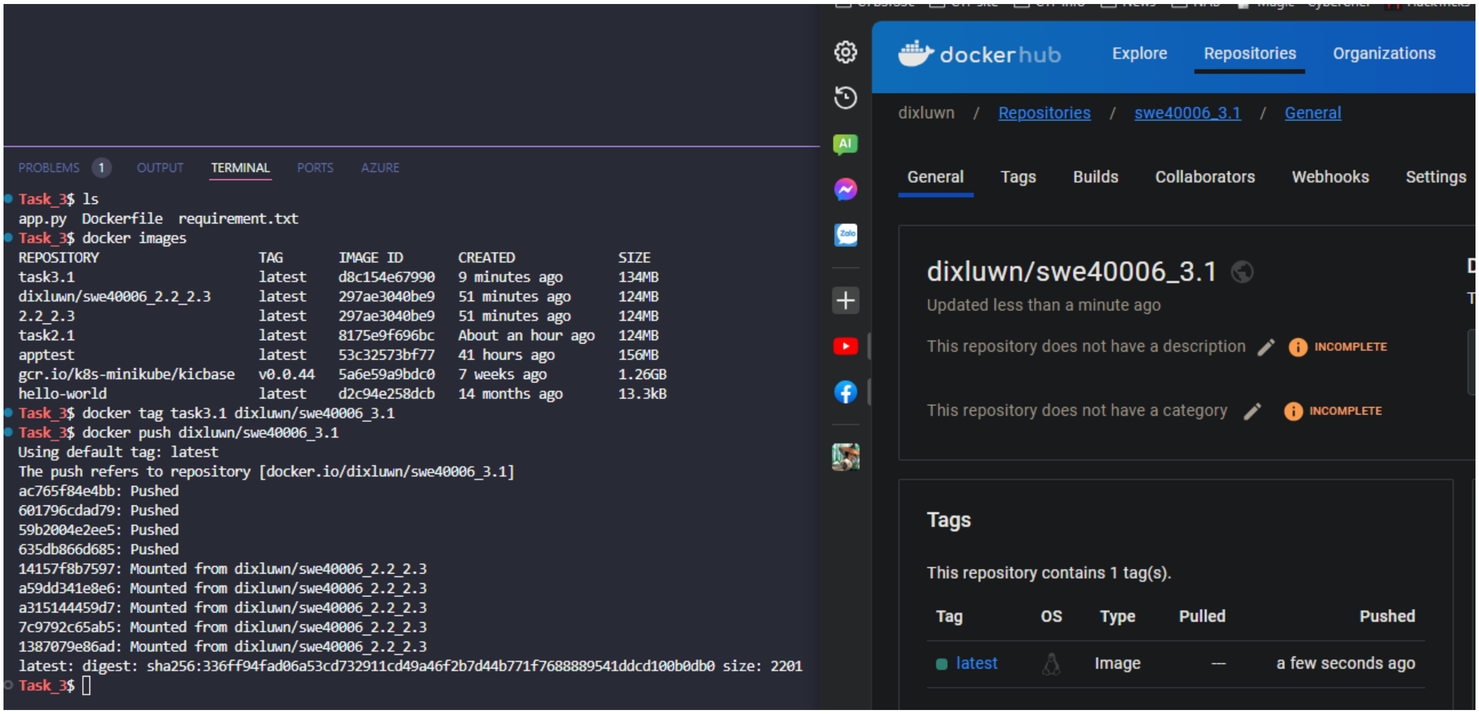
```
1   Hello, Docker World on Flask!
```

```
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
172.17.0.1 - - [30/Jun/2024 15:56:40] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [30/Jun/2024 15:56:41] "GET /favicon.ico HTTP/1.1" 404 -
172.17.0.1 - - [30/Jun/2024 16:01:02] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [30/Jun/2024 16:01:06] "GET / HTTP/1.1" 200 -
```

Figure 16: Application successfully launched

4.3B Deploy to a Docker installation

For this task, we will repeat similar to 4.2D, we will tag our image then check if our EC2 Instance can use it.



```
Task_3$ ls
app.py  Dockerfile  requirement.txt
Task_3$ docker images
REPOSITORY                   TAG       IMAGE ID       CREATED             SIZE
task3.1                      latest    d8c154e67990   9 minutes ago       134MB
dixluwn/swe40006_2.2_2.3     latest    297ae3040be9   51 minutes ago      124MB
2.2_2.3                      latest    297ae3040be9   51 minutes ago      124MB
task2.1                      latest    8175e9f696bc   About an hour ago   124MB
apptest                      latest    53c32573bf77   41 hours ago        156MB
gcr.io/k8s-minikube/kicbase  v0.0.44   5a6e59a9bdc0   7 weeks ago         1.26GB
hello-world                  latest    d2c94e258dcb   14 months ago       13.3kB
Task_3$ docker tag task3.1 dixluwn/swe40006_3.1
Task_3$ docker push dixluwn/swe40006_3.1
Using default tag: latest
The push refers to repository [docker.io/dixluwn/swe40006_3.1]
ac765f84e4bb: Pushed
601796cdad79: Pushed
59b2004e2ee5: Pushed
635db866d685: Pushed
14157f8b7597: Mounted from dixluwn/swe40006_2.2_2.3
a59dd341e8e6: Mounted from dixluwn/swe40006_2.2_2.3
a315144459d7: Mounted from dixluwn/swe40006_2.2_2.3
7c9792c65ab5: Mounted from dixluwn/swe40006_2.2_2.3
1387079e86ad: Mounted from dixluwn/swe40006_2.2_2.3
latest: digest: sha256:336ff94fad06a53cd732911cd49a46f2b7d44b771f7688889541ddcd100b0db0 size: 2201
Task_3$
```

dixluwn/swe40006_3.1

Updated less than a minute ago

This repository does not have a description INCOMPLETE

This repository does not have a category INCOMPLETE

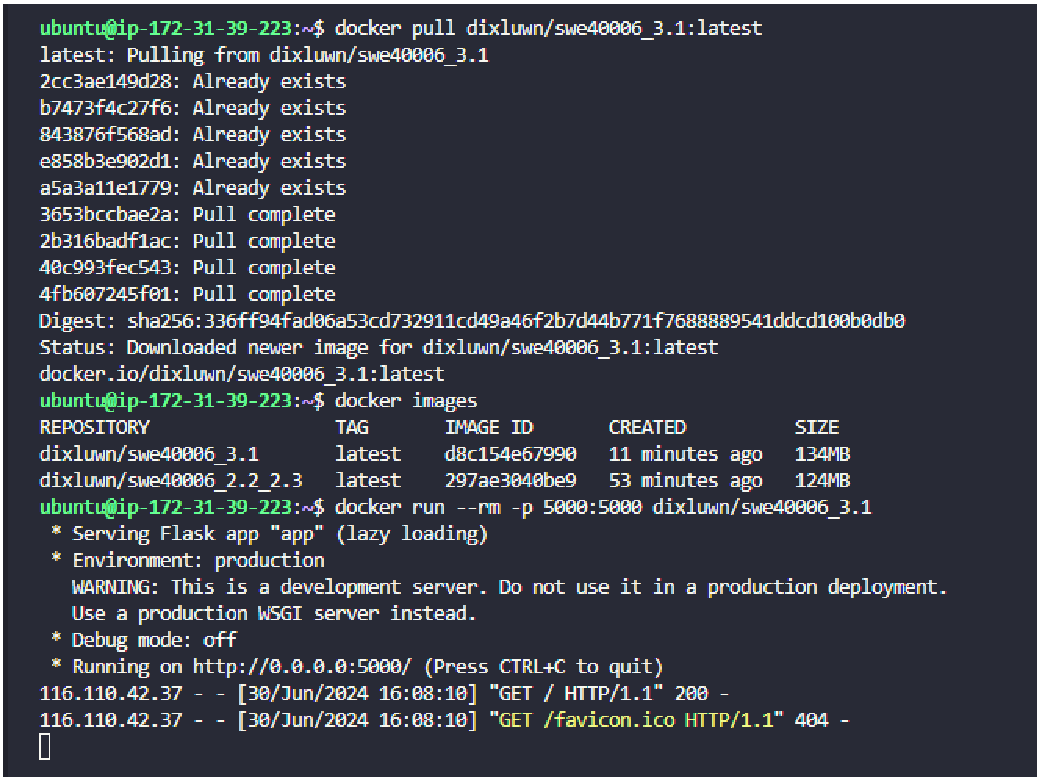
Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	—	a few seconds ago

Figure 17: Docker image successfully tagged and pushed

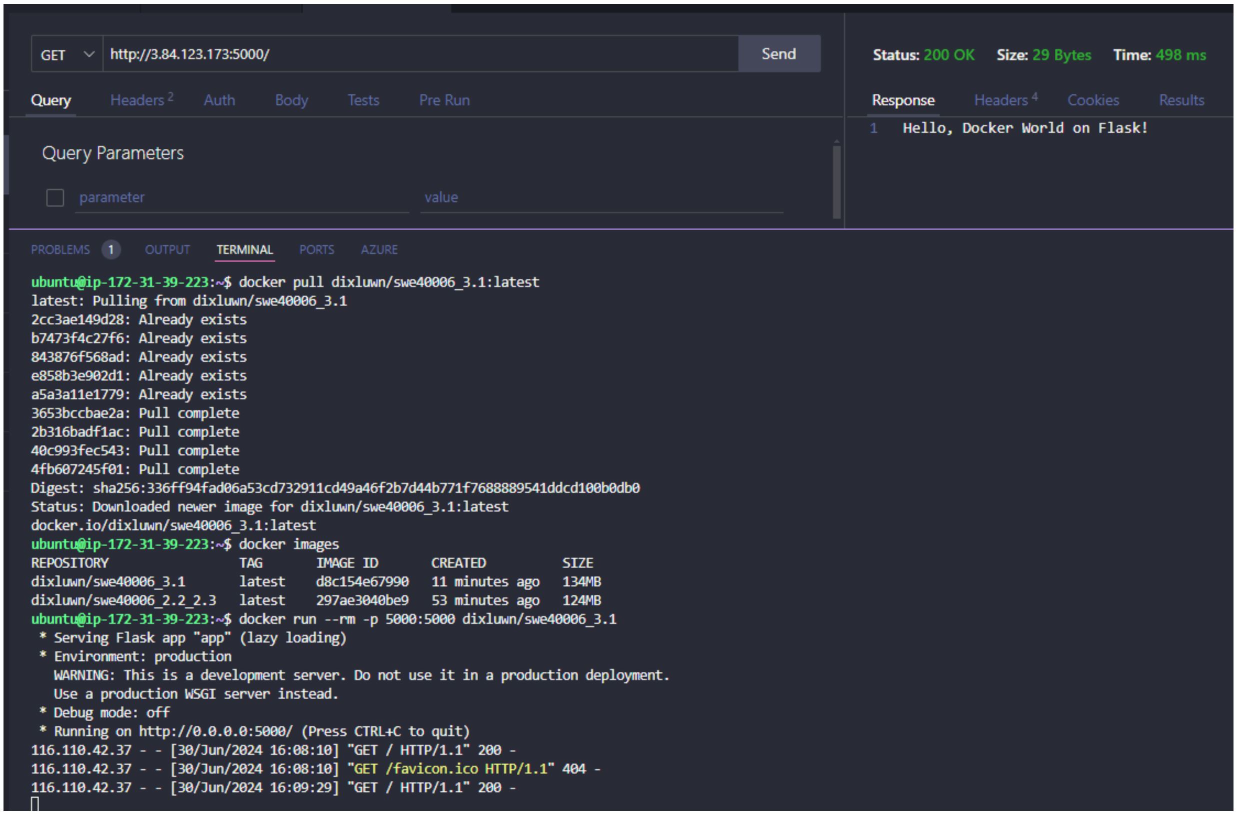
After that we will pull this Docker Image from our EC2 Instance and see if it is working as expected.



```
ubuntu@ip-172-31-39-223:~$ docker pull dixluwn/swe40006_3.1:latest
latest: Pulling from dixluwn/swe40006_3.1
2cc3ae149d28: Already exists
b7473f4c27f6: Already exists
843876f568ad: Already exists
e858b3e902d1: Already exists
a5a3a11e1779: Already exists
3653bccbae2a: Pull complete
2b316badf1ac: Pull complete
40c993fec543: Pull complete
4fb607245f01: Pull complete
Digest: sha256:336ff94fad06a53cd732911cd49a46f2b7d44b771f7688889541ddcd100b0db0
Status: Downloaded newer image for dixluwn/swe40006_3.1:latest
docker.io/dixluwn/swe40006_3.1:latest
ubuntu@ip-172-31-39-223:~$ docker images
REPOSITORY               TAG       IMAGE ID       CREATED          SIZE
dixluwn/swe40006_3.1     latest    d8c154e67990   11 minutes ago   134MB
dixluwn/swe40006_2.2_2.3 latest    297ae3040be9   53 minutes ago   124MB
ubuntu@ip-172-31-39-223:~$ docker run --rm -p 5000:5000 dixluwn/swe40006_3.1
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
116.110.42.37 - - [30/Jun/2024 16:08:10] "GET / HTTP/1.1" 200 -
116.110.42.37 - - [30/Jun/2024 16:08:10] "GET /favicon.ico HTTP/1.1" 404 -
```

Figure 18: Docker Image successfully pulled and run on EC2 Instance

We can check this application on our API extension or using the public IP.



http://3.84.123.173:5000/

Status: 200 OK Size: 29 Bytes Time: 498 ms

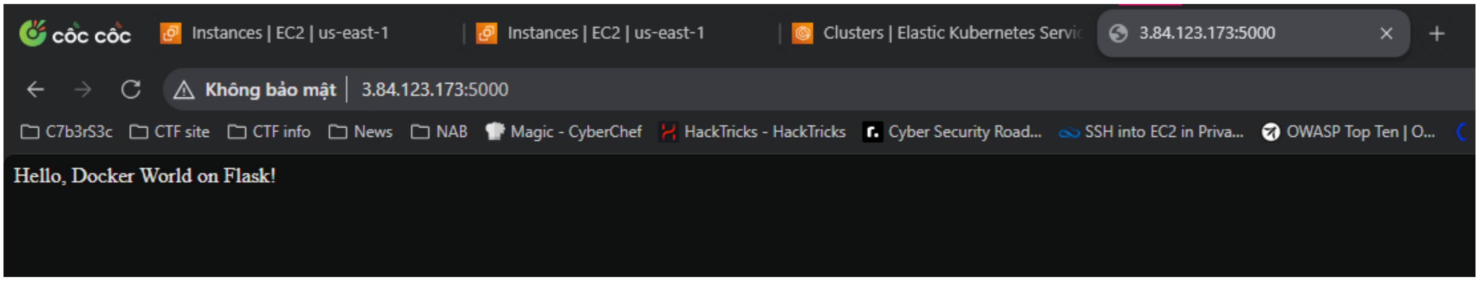
Query Parameters

□ parameter value

```
1   Hello, Docker World on Flask!
```

```
ubuntu@ip-172-31-39-223:~$ docker pull dixluwn/swe40006_3.1:latest
latest: Pulling from dixluwn/swe40006_3.1
2cc3ae149d28: Already exists
b7473f4c27f6: Already exists
843876f568ad: Already exists
e858b3e902d1: Already exists
a5a3a11e1779: Already exists
3653bccbae2a: Pull complete
2b316badf1ac: Pull complete
40c993fec543: Pull complete
4fb607245f01: Pull complete
Digest: sha256:336ff94fad06a53cd732911cd49a46f2b7d44b771f7688889541ddcd100b0db0
Status: Downloaded newer image for dixluwn/swe40006_3.1:latest
docker.io/dixluwn/swe40006_3.1:latest
ubuntu@ip-172-31-39-223:~$ docker images
REPOSITORY               TAG       IMAGE ID       CREATED          SIZE
dixluwn/swe40006_3.1     latest    d8c154e67990   11 minutes ago   134MB
dixluwn/swe40006_2.2_2.3 latest    297ae3040be9   53 minutes ago   124MB
ubuntu@ip-172-31-39-223:~$ docker run --rm -p 5000:5000 dixluwn/swe40006_3.1
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
116.110.42.37 - - [30/Jun/2024 16:08:10] "GET / HTTP/1.1" 200 -
116.110.42.37 - - [30/Jun/2024 16:08:10] "GET /favicon.ico HTTP/1.1" 404 -
116.110.42.37 - - [30/Jun/2024 16:09:29] "GET / HTTP/1.1" 200 -
```

Figure 19: Application is working as expected.



Không bảo mật | 3.84.123.173:5000

```
Hello, Docker World on Flask!
```

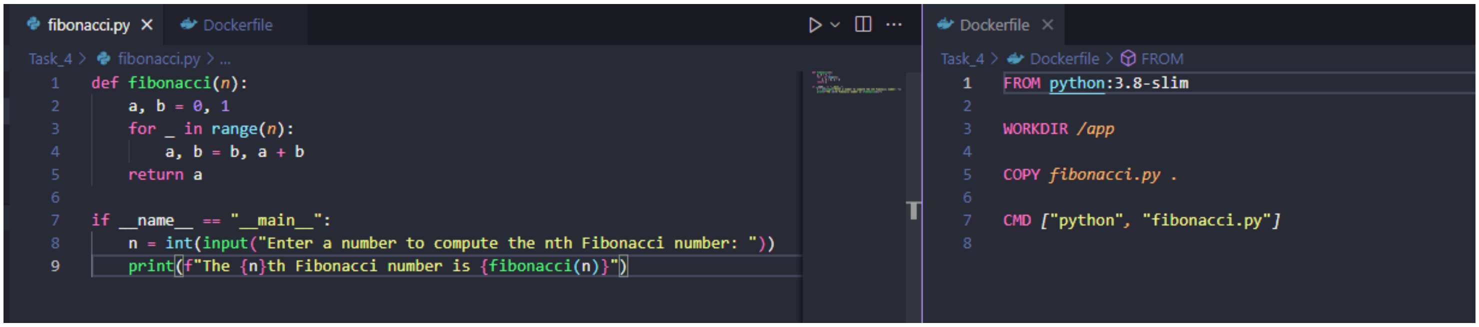
Figure 20: Application is working as expected on the public IP

We can confirm that our application is working as expected. This will also conclude this task, we will move on to our final task.

Task 4.4HD:

4.4A Develop a non-web-based app

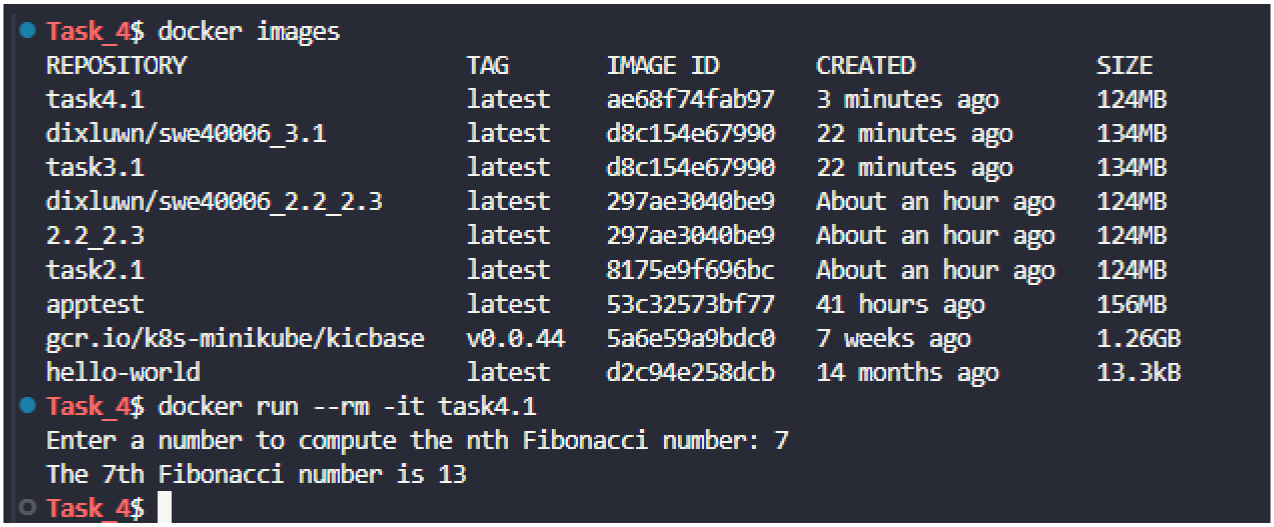
For this task, we will write a simple Python program to count the “n” Fibonacci number based on user input. After that we will dockerize that application to see if it works as expected.



```python
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

if __name__ == "__main__":
    n = int(input("Enter a number to compute the nth Fibonacci number: "))
    print(f"The {n}th Fibonacci number is {fibonacci(n)}")
```

```dockerfile
FROM python:3.8-slim

WORKDIR /app

COPY fibonacci.py .

CMD ["python", "fibonacci.py"]
```

Figure 21: Fibonacci application and its Dockerfile



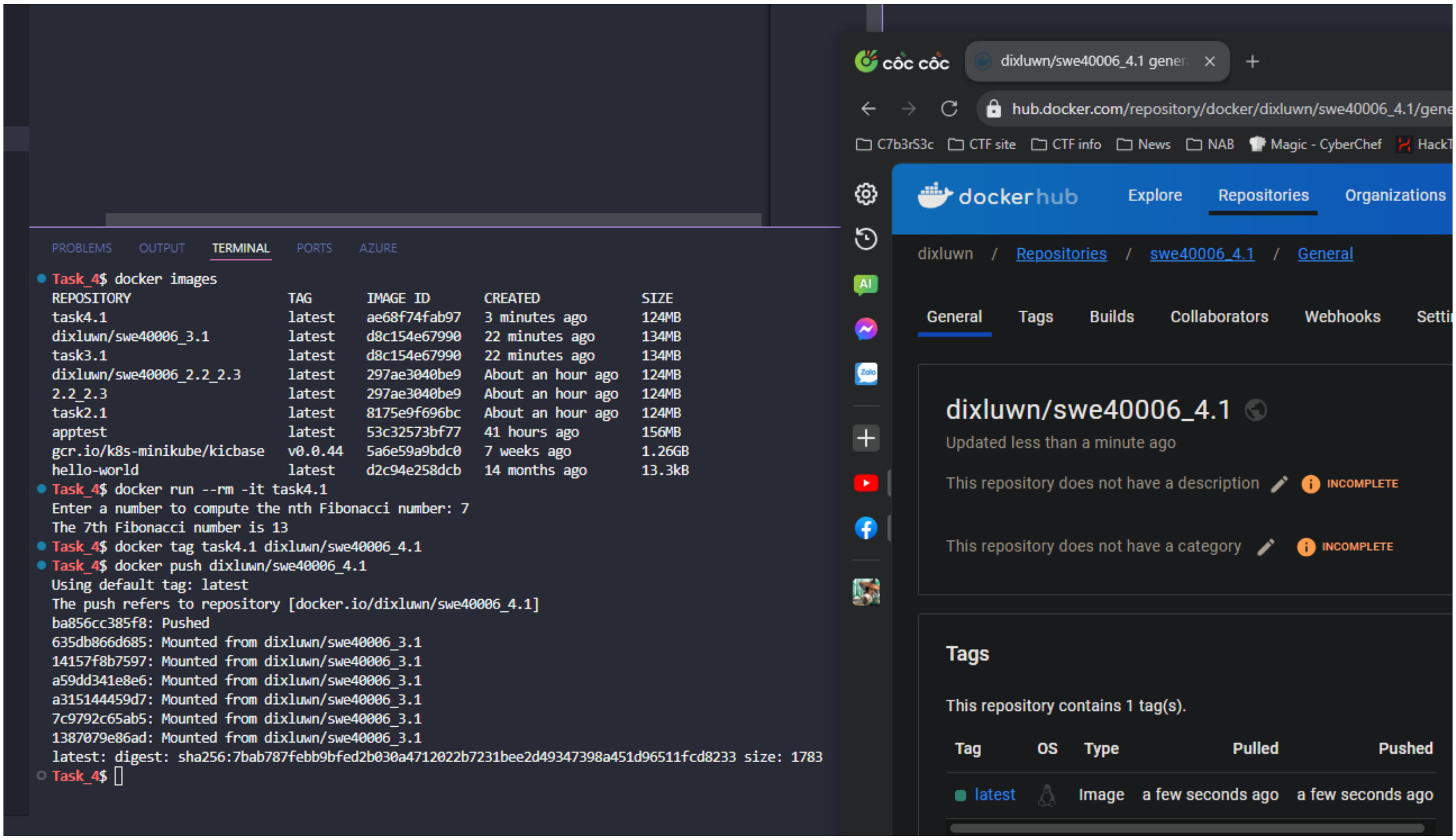
```
Task_4$ docker images
REPOSITORY                      TAG       IMAGE ID       CREATED            SIZE
task4.1                         latest    ae68f74fab97   3 minutes ago      124MB
dixluwn/swe40006_3.1            latest    d8c154e67990   22 minutes ago     134MB
task3.1                         latest    d8c154e67990   22 minutes ago     134MB
dixluwn/swe40006_2.2_2.3        latest    297ae3040be9   About an hour ago  124MB
2.2_2.3                         latest    297ae3040be9   About an hour ago  124MB
task2.1                         latest    8175e9f696bc   About an hour ago  124MB
apptest                         latest    53c32573bf77   41 hours ago       156MB
gcr.io/k8s-minikube/kicbase     v0.0.44   5a6e59a9bdc0   7 weeks ago        1.26GB
hello-world                     latest    d2c94e258dcb   14 months ago      13.3kB
Task_4$ docker run --rm -it task4.1
Enter a number to compute the nth Fibonacci number: 7
The 7th Fibonacci number is 13
Task_4$
```

Figure 22: Application successfully built and run

We can see that our application is successfully dockerized and it is running as expected. We can run this docker image using “-it” tag to make it interactive. We will move to our final task.

4.4B Deploy to a Docker

Similar to task 4.2D and 4.3B, we will tag this image and push it onto Docker Hub, finally we will run this image on our EC2 instance to confirm that our image can work cross-platform.



```
● Task_4$ docker images
  REPOSITORY                   TAG       IMAGE ID       CREATED            SIZE
  task4.1                      latest    ae68f74fab97   3 minutes ago      124MB
  dixluwn/swe40006_3.1         latest    d8c154e67990   22 minutes ago     134MB
  task3.1                      latest    d8c154e67990   22 minutes ago     134MB
  dixluwn/swe40006_2.2_2.3     latest    297ae3040be9   About an hour ago  124MB
  2.2_2.3                      latest    297ae3040be9   About an hour ago  124MB
  task2.1                      latest    8175e9f696bc   About an hour ago  124MB
  apptest                      latest    53c32573bf77   41 hours ago       156MB
  gcr.io/k8s-minikube/kicbase  v0.0.44   5a6e59a9bdc0   7 weeks ago        1.26GB
  hello-world                  latest    d2c94e258dcb   14 months ago      13.3kB
● Task_4$ docker run --rm -it task4.1
  Enter a number to compute the nth Fibonacci number: 7
  The 7th Fibonacci number is 13
● Task_4$ docker tag task4.1 dixluwn/swe40006_4.1
● Task_4$ docker push dixluwn/swe40006_4.1
  Using default tag: latest
  The push refers to repository [docker.io/dixluwn/swe40006_4.1]
  ba856cc385f8: Pushed
  635db866d685: Mounted from dixluwn/swe40006_3.1
  14157f8b7597: Mounted from dixluwn/swe40006_3.1
  a59dd341e8e6: Mounted from dixluwn/swe40006_3.1
  a315144459d7: Mounted from dixluwn/swe40006_3.1
  7c9792c65ab5: Mounted from dixluwn/swe40006_3.1
  1387079e86ad: Mounted from dixluwn/swe40006_3.1
  latest: digest: sha256:7bab787febb9bfed2b030a4712022b7231bee2d49347398a451d96511fcd8233 size: 1783
○ Task_4$ ▯
```

dixluwn/swe40006_4.1

Updated less than a minute ago

This repository does not have a description INCOMPLETE

This repository does not have a category INCOMPLETE

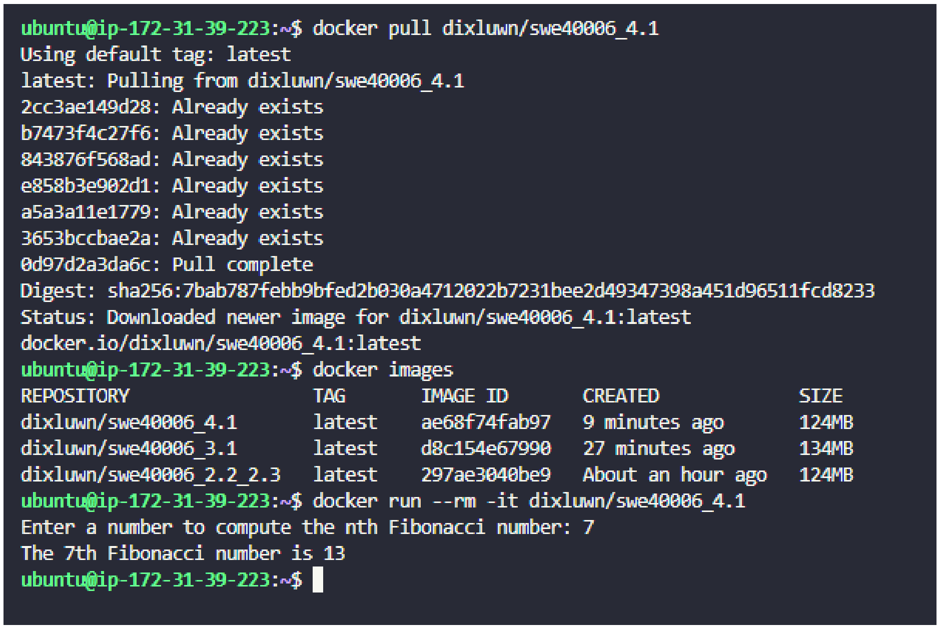
Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	a few seconds ago	a few seconds ago

Figure 23: Docker image successfully tagged and pushed onto Docker Hub.

With that done, our final task is to pull this image onto our EC2 instance and check if it is working properly.



```
ubuntu@ip-172-31-39-223:~$ docker pull dixluwn/swe40006_4.1
Using default tag: latest
latest: Pulling from dixluwn/swe40006_4.1
2cc3ae149d28: Already exists
b7473f4c27f6: Already exists
843876f568ad: Already exists
e858b3e902d1: Already exists
a5a3a11e1779: Already exists
3653bccbae2a: Already exists
0d97d2a3da6c: Pull complete
Digest: sha256:7bab787febb9bfed2b030a4712022b7231bee2d49347398a451d96511fcd8233
Status: Downloaded newer image for dixluwn/swe40006_4.1:latest
docker.io/dixluwn/swe40006_4.1:latest
ubuntu@ip-172-31-39-223:~$ docker images
REPOSITORY                 TAG       IMAGE ID       CREATED            SIZE
dixluwn/swe40006_4.1       latest    ae68f74fab97   9 minutes ago      124MB
dixluwn/swe40006_3.1       latest    d8c154e67990   27 minutes ago     134MB
dixluwn/swe40006_2.2_2.3   latest    297ae3040be9   About an hour ago  124MB
ubuntu@ip-172-31-39-223:~$ docker run --rm -it dixluwn/swe40006_4.1
Enter a number to compute the nth Fibonacci number: 7
The 7th Fibonacci number is 13
ubuntu@ip-172-31-39-223:~$ ▌
```

Figure 24: Non-web-based application work as expected.

We can now confirm that our non-web-based application is working as expected across different platforms. This will also conclude this assignment.

Resources:

The code and configuration files of this assignment can be found here via this link