



Lab 2 Report

50.046 CCIoT

SUTD 2022 ISTD

Group 3

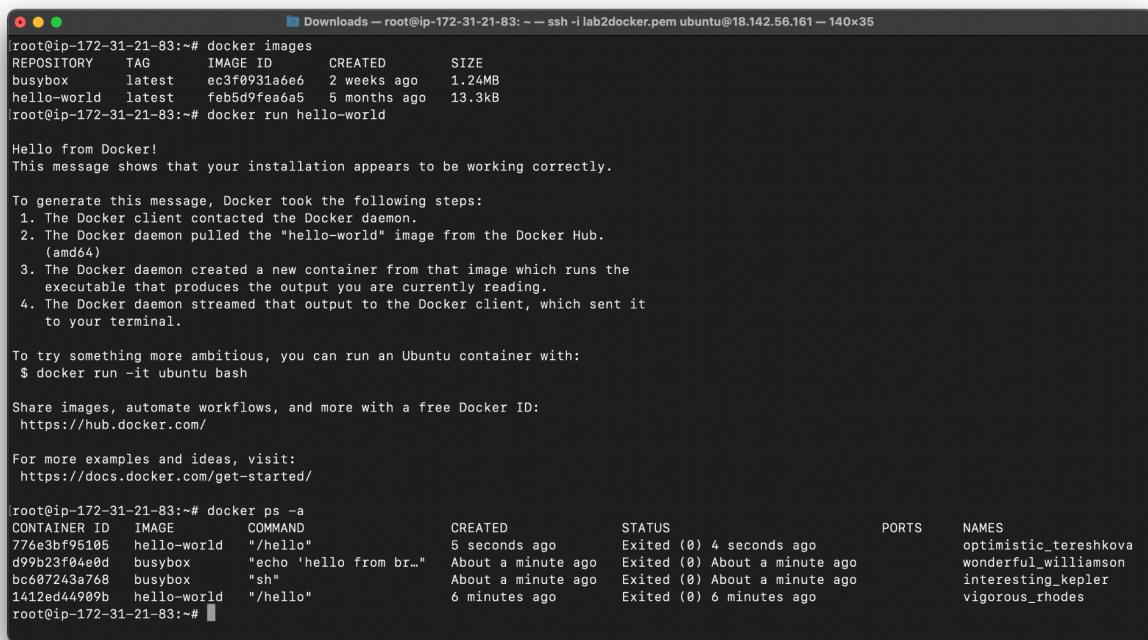
Han Xing Yi	(1004330)
Ang Song Gee	(1004589)
James Raphael Tiovalen	(1004555)
Velusamy Sathiakumar Ragul Balaji	(1004101)

Table of Contents

Task 1: Set up a Docker Environment	3
Task 2: Run a Webapp with Docker	4
Task 3: Make Your First Docker Image	5
Task 4: Multi-Container Environments	9
4.1 Insufficient Memory during Setup	9
Explanation of Microservice Architecture implementation	10

Task 1: Set up a Docker Environment

- Print all local docker images with `docker images`.
- Run the 'hello-world' container in docker with `docker run hello-world`.
- Print all running docker process status with `docker ps -a`.



```

Downloads — root@ip-172-31-21-83:~ — ssh -i lab2docker.pem ubuntu@18.142.56.161 — 140x35
[root@ip-172-31-21-83:~# docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
busybox         latest   ec3f0931a6e6  2 weeks ago   1.24MB
hello-world     latest   feb5d9fea6a5  5 months ago  13.3kB
[root@ip-172-31-21-83:~# 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

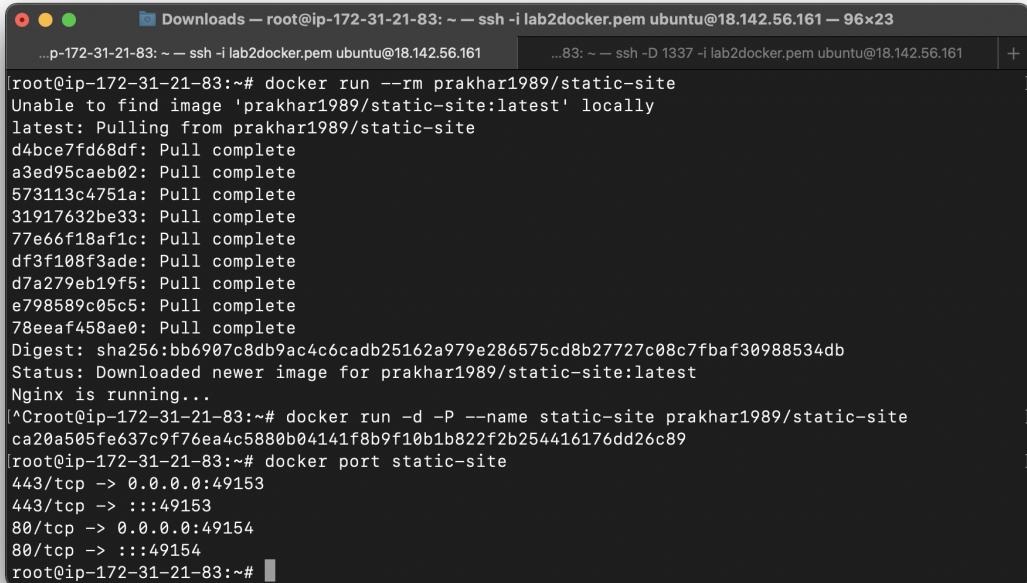
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
[root@ip-172-31-21-83:~# docker ps -a
CONTAINER ID        IMAGE           COMMAND       CREATED          STATUS          PORTS          NAMES
776e3bf95105        hello-world     "/hello"      5 seconds ago   Exited (0) 4 seconds ago   optimistic_tereshkova
d99b23f04e0d        busybox         "echo 'hello from br..."  About a minute ago   Exited (0) About a minute ago   wonderful_williamson
bc607243a768        busybox         "sh"          About a minute ago   Exited (0) About a minute ago   interesting_kepler
1412ed44909b        hello-world     "/hello"      6 minutes ago    Exited (0) 6 minutes ago   vigorous_rhodes
root@ip-172-31-21-83:~# ]

```

Task 2: Run a Webapp with Docker

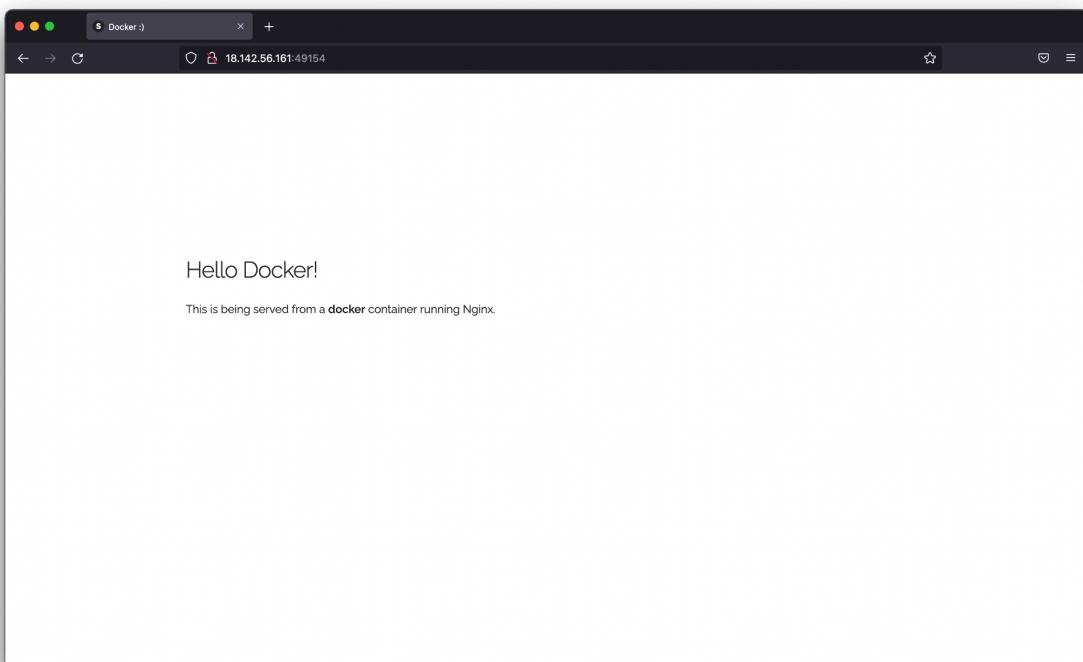
- ✓ Set up static site with `docker run prakhar1989/static-site`.



A terminal window titled 'Downloads' showing the execution of a Docker command. The command 'docker run --rm prakhar1989/static-site' is run, which results in an error message: 'Unable to find image 'prakhar1989/static-site:latest' locally'. It then attempts to pull the image from the repository, listing several layers being downloaded. Once the image is pulled, it is run with the command 'docker run -d -P --name static-site prakhar1989/static-site'. The output shows port mappings: 443/tcp to 0.0.0.0:49153 and 80/tcp to 0.0.0.0:49154. Finally, 'docker port static-site' is run to verify the ports are mapped correctly.

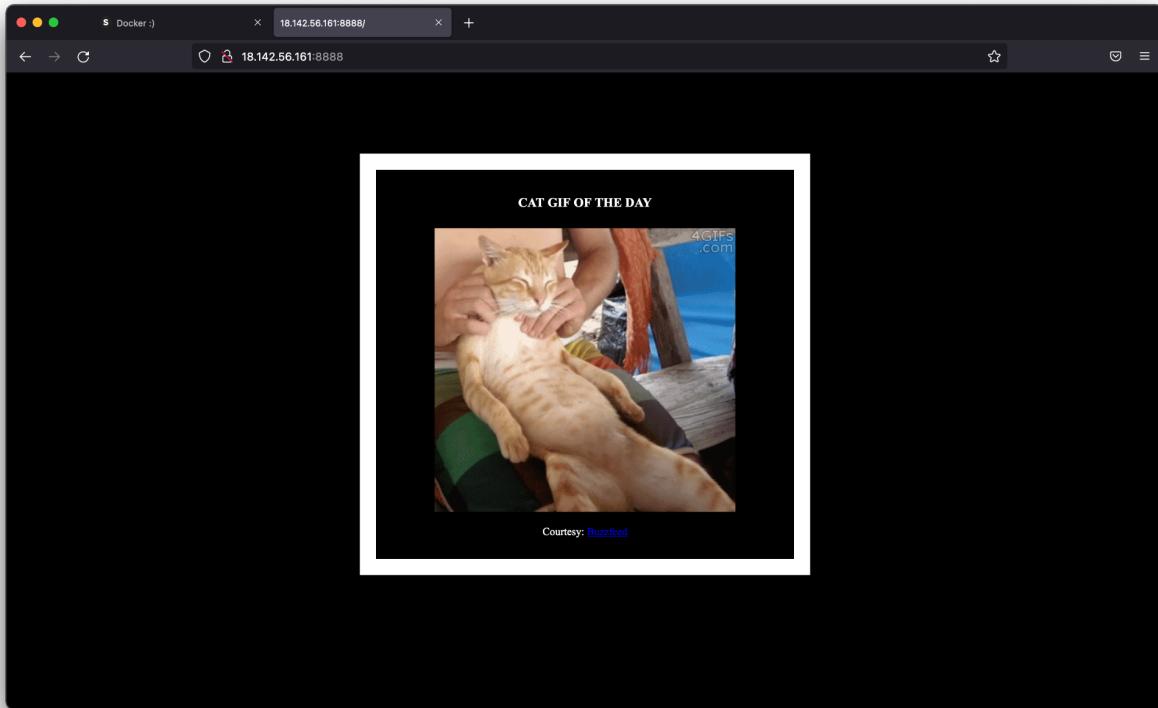
```
...p-172-31-21-83:~ ssh -i lab2docker.pem ubuntu@18.142.56.161 -D 1337 -i lab2docker.pem ubuntu@18.142.56.161 - 96x23
[...]
[root@ip-172-31-21-83:~# docker run --rm prakhar1989/static-site
Unable to find image 'prakhar1989/static-site:latest' locally
latest: Pulling from prakhar1989/static-site
d4bce7fd68df: Pull complete
a3ed95caeb02: Pull complete
573113c4751a: Pull complete
31917632be33: Pull complete
77e66f18af1c: Pull complete
df3f108f3ade: Pull complete
d7a279eb19f5: Pull complete
e798589c05c5: Pull complete
78eeaf458ae0: Pull complete
Digest: sha256:bb6907c8db9ac4c6cadb25162a979e286575cd8b27727c08c7fbaf30988534db
Status: Downloaded newer image for prakhar1989/static-site:latest
Nginx is running...
[^Croot@ip-172-31-21-83:~# docker run -d -P --name static-site prakhar1989/static-site
ca20a505fe637c9f76ea4c5880b04141f8b9f10b1b822f2b254416176dd26c89
[root@ip-172-31-21-83:~# docker port static-site
443/tcp -> 0.0.0.0:49153
443/tcp -> :::49153
80/tcp -> 0.0.0.0:49154
80/tcp -> :::49154
root@ip-172-31-21-83:~# ]
```

- ✓ Server running properly at localhost



Task 3: Make Your First Docker Image

- ✓ Compile Dockerfile and run on localhost



- ✓ Compile Docker image from Dockerfile and upload to my repository

```
[root@ip-172-31-21-83:~# docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
bryxjrsg           latest   315898121af6  45 minutes ago  921MB
xinyiz/bryxjrsg    latest   315898121af6  45 minutes ago  921MB
python              3.9     4819be0df942  7 hours ago   912MB
busybox             latest   ec3f0931a6e6  2 weeks ago   1.24MB
hello-world         latest   feb5d9fea6a5  5 months ago  13.3kB
prakhar1989/foodtrucks-web  latest   38fd22ce00b5  21 months ago  614MB
docker.elastic.co/elasticsearch/elasticsearch  6.3.2    96dd1575de0f  3 years ago   826MB
prakhar1989/static-site    latest   f01030e1dcf3  6 years ago   134MB
root@ip-172-31-21-83:~# ]
```

ISTD 50.046 CCIoT | Lab 2 Report | Group 3

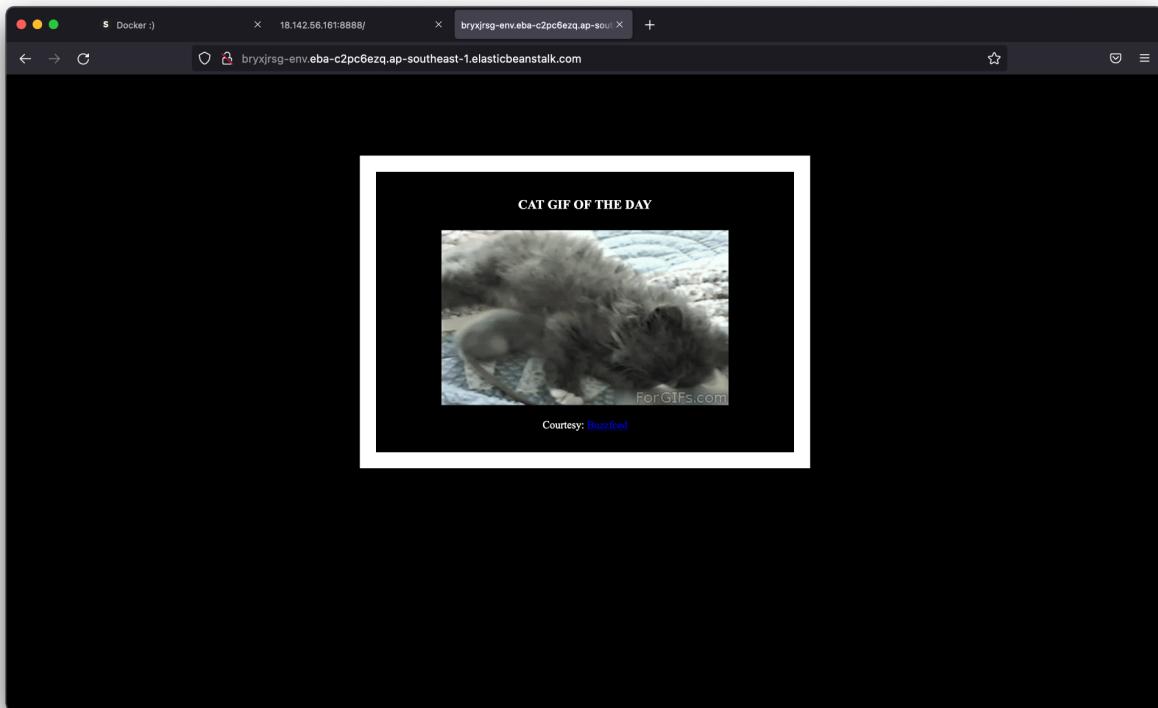
The screenshot shows a Docker Hub repository page for the user 'xingyiz'. The repository name is 'bryxjrsg'. The page includes sections for 'General' (with an 'Advanced Image Management' notice), 'Tags and Scans' (showing one tag: 'latest'), 'Docker commands' (with a 'Public View' button and a command box containing 'docker push xingyiz/bryxjrsg:tagname'), and 'Automated Builds' (disabled). A 'Readme' section is also present.

✓ Create Elastic Beanstalk website with our json pointing to our uploaded repo.

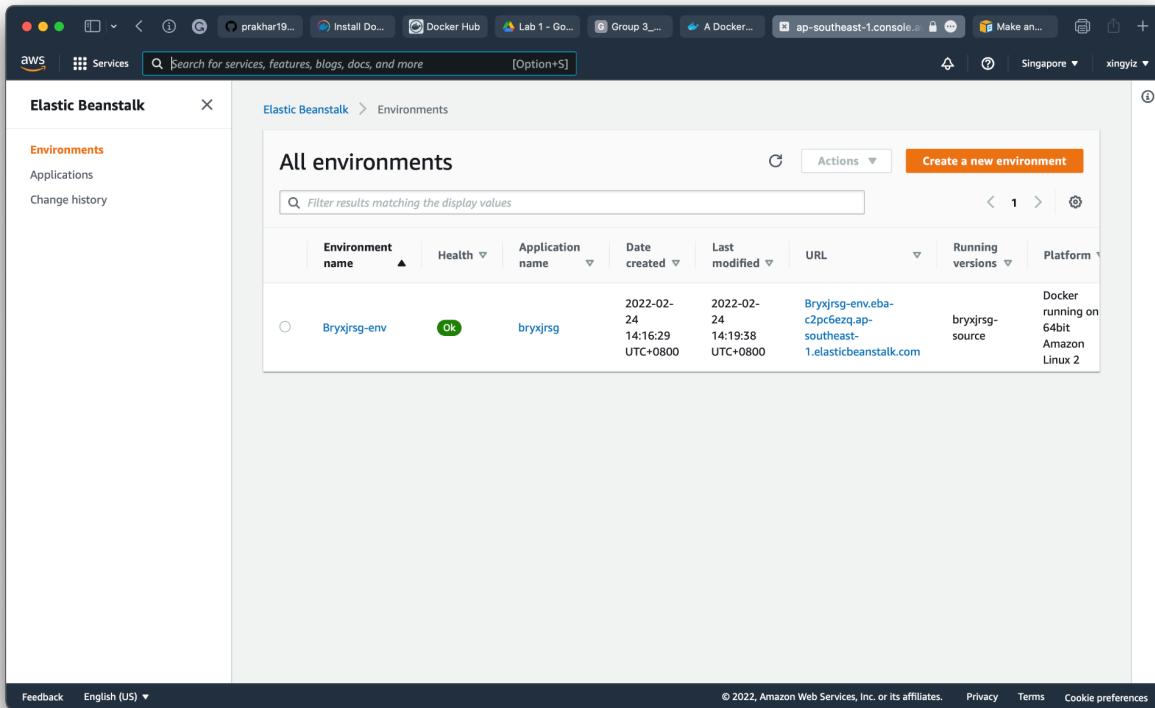
The screenshot shows the AWS Elastic Beanstalk configuration interface. Under the 'Application code' section, the 'Upload your code' option is selected. In the 'Source code origin' section, a local file named 'bryxjrsg-source' is chosen, and a file named 'bryxjrsg.json' has been successfully uploaded. The 'Application code tags' section is shown below. At the bottom, there are 'Cancel', 'Configure more options', and 'Create application' buttons.

```
{} bryxjrsg.json > ...  
1  {  
2      "AWSEBDockerrunVersion": "1",  
3      "Image": {  
4          "Name": "xingyiz/bryxjrsg",  
5          "Update": "true"  
6      },  
7      "Ports": [  
8          {  
9              "ContainerPort": 5000,  
10             "HostPort": 8000  
11         }  
12     ],  
13     "Logging": "/var/log/nginx"  
14 }
```

✓ Elastic Beanstalk loads Docker image and we can access the URL provided



ISTD 50.046 CCIoT | Lab 2 Report | Group 3

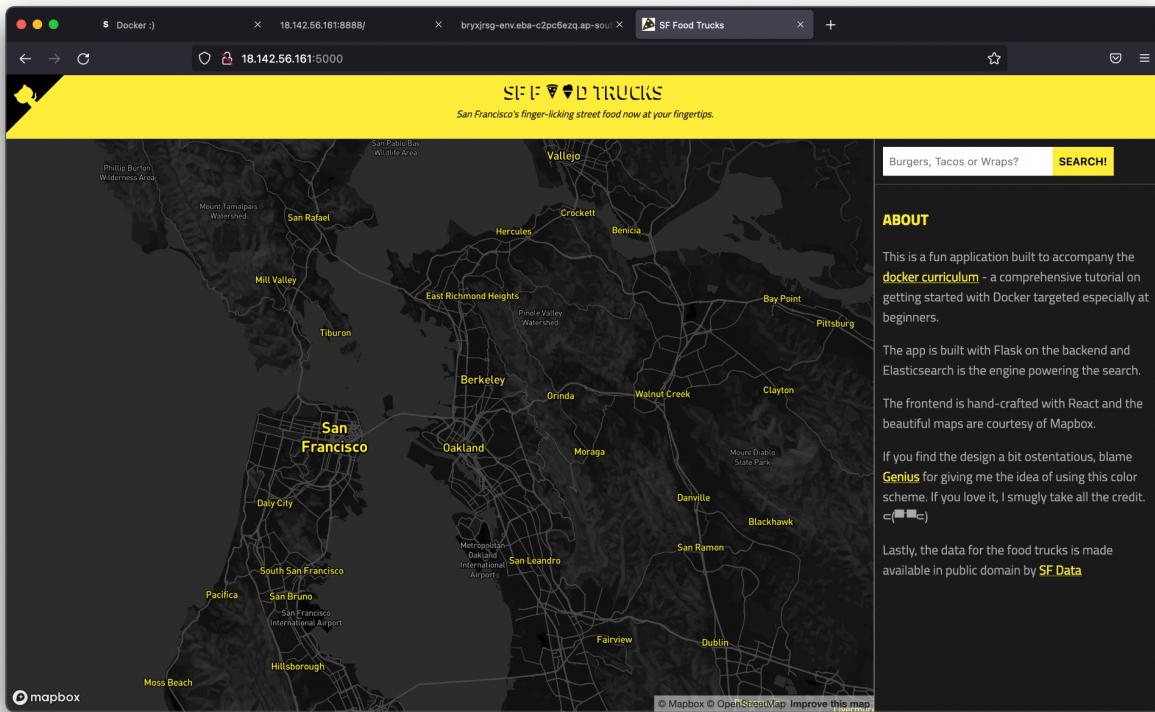


The screenshot shows the AWS Elastic Beanstalk console. The left sidebar has 'Environments' selected. The main area is titled 'All environments' and contains a table with one row. The table columns are: Environment name, Health, Application name, Date created, Last modified, URL, Running versions, and Platform. The single environment listed is 'Bryxjrsg-env' with a green 'Ok' status indicator. The application name is 'bryxjrsg'. The date created was '2022-02-24 14:16:29 UTC+0800' and last modified was '2022-02-24 14:19:38 UTC+0800'. The URL is 'Bryxjrsg-env.eba-c2pc6ezq.ap-southeast-1.elasticbeanstalk.com'. The running version is 'bryxjrsg-source' and the platform is 'Docker running on 64bit Amazon Linux 2'.

Environment name	Health	Application name	Date created	Last modified	URL	Running versions	Platform
Bryxjrsg-env	Ok	bryxjrsg	2022-02-24 14:16:29 UTC+0800	2022-02-24 14:19:38 UTC+0800	Bryxjrsg-env.eba-c2pc6ezq.ap-southeast-1.elasticbeanstalk.com	bryxjrsg-source	Docker running on 64bit Amazon Linux 2

Task 4: Multi-Container Environments

- ✓ Able to run the FoodTrucks example



4.1 Insufficient Memory during Setup

The ElasticSearch container has a massive heap size of > 1GB and thus it could not be started because there was insufficient memory allocated.

For our case, we added a swap section by following this article to add a swap space of 4GB: [How To Add Swap Space on Ubuntu 20.04 | DigitalOcean](#).

If the ElasticSearch container is killed with error code 137 (out of memory), additional RAM/swap space can be added by attaching additional Amazon EBS volumes to the EC2 instance and mounting it accordingly.

Explain how a microservice architecture is implemented using the example of Docker Network and Docker Compose.

In task 4 of this lab, we used a *docker-compose.yml* file to define two microservices, *es* and *web*. Docker Compose reads from this *docker-compose.yml* to spawn 2 containers, each acting as one microservice. This allows **isolation between different microservices** and allows us to **spin up multiple microservices using one file** using the '*docker-compose up*' command, instead of having to manually set up each service one by one. This makes the whole application **more portable and maintainable**.

Furthermore, as different services have different resource needs and grow at different rates, another advantage of Docker Compose is that separating each service into individual containers allows for **easier scalability and management** of different container resource needs.

By default, all containers are connected to the default '*bridge*' network, which allows them to communicate with each other. Docker Network also allows us to **create user-defined networks**, which allows **isolation of the network to particular containers**, since the '*bridge*' network is shared by every container by default.

When we use the default '*bridge*' network, we were not able to connect the *es* service to the *web* service. This is because, inside the Flask web app's code, there is this line: `es = Elasticsearch(host='es')`. To do this properly, the web app would need to resolve and access each other by name or alias. However, containers on the default '*bridge*' network can only access each other by IP addresses because, by default, Docker runs all the containers on '`0.0.0.0`'. On the contrary, user-defined networks provide automatic DNS resolution between containers. Hence, this allows the web app container to connect to the Elasticsearch container via the '`es`' alias. An advantage of this would be to **provide network isolation between the containers**, which would **enhance security**.