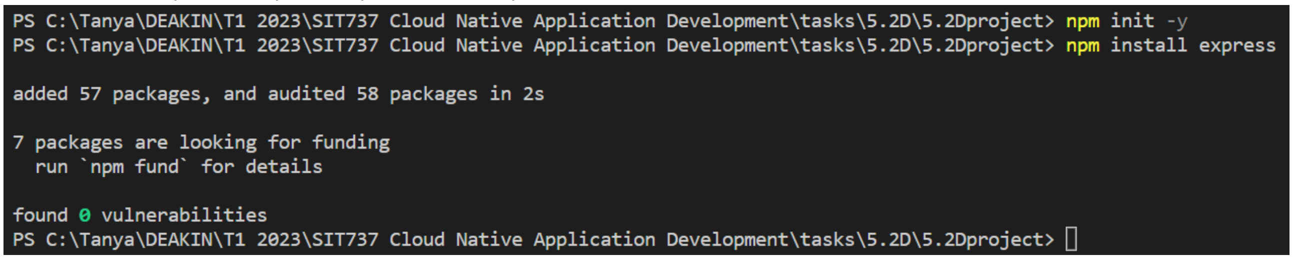
# 5.2D: Configuring networking between containers

1. Started with npm init -y and npm install express



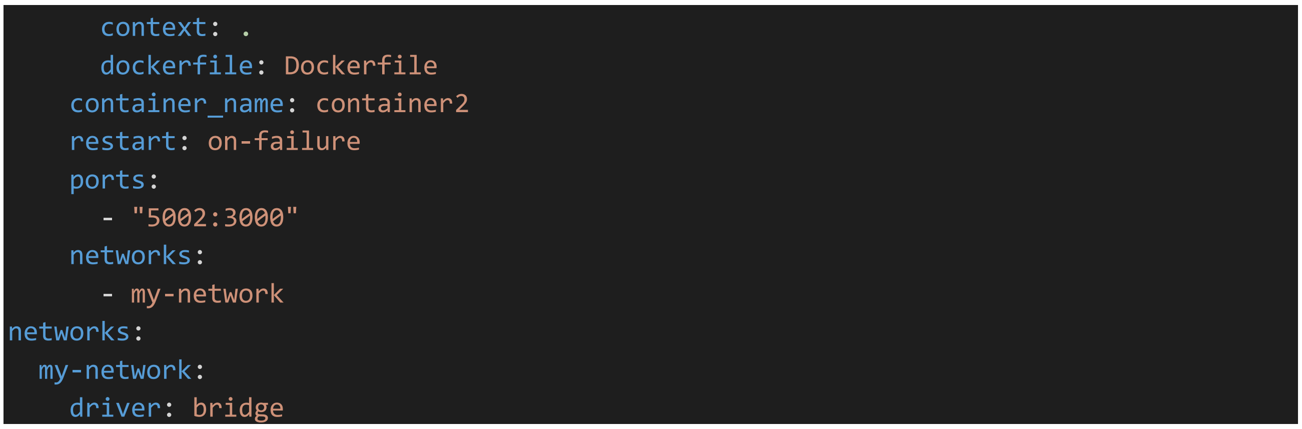
```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.2D\5.2Dproject> npm init -y
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.2D\5.2Dproject> npm install express

added 57 packages, and audited 58 packages in 2s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.2D\5.2Dproject> 
```

2. My Dockerfile looks like –

```
#denotes base image
FROM node:14

#setting working directory
WORKDIR /usr/src/app

COPY package*.json ./

#to install the package listed in package.json file
RUN npm install

COPY index.js index.js

#exposing port outside
EXPOSE 3000
CMD ["node", "index.js"]
```

3. And my docker-compose.yml looks like below. Container1 made from image1 is exposed on port 5001.
Container2 made from image2 is exposed on port 5002
A bridge network called my-network connects the two for communication

```
services:
  app1:
    image: image1
    build:
      context: .
      dockerfile: Dockerfile
    container_name: container1
    restart: on-failure
    ports:
      - "5001:3000"
    networks:
      - my-network
  app2:
    image: image2
    build:
```

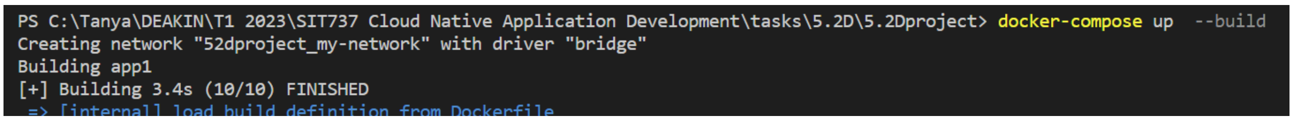


```
      context: .
      dockerfile: Dockerfile
    container_name: container2
    restart: on-failure
    ports:
      - "5002:3000"
    networks:
      - my-network
networks:
  my-network:
    driver: bridge
```

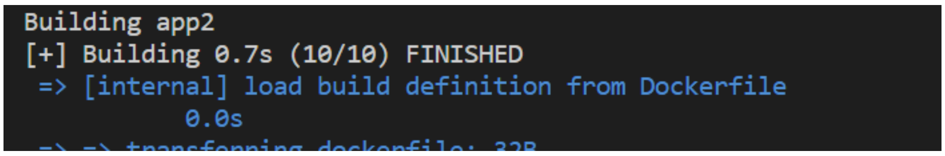
4. Executed docker compose command to make 2 images – image1 and image 2, and 2 containers-container1 and container2.

Command → docker-compose up  --build

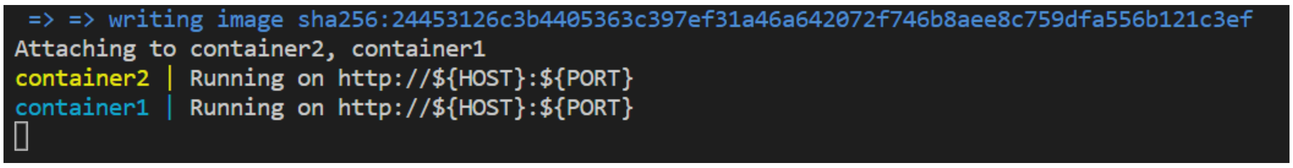
Screenshot –



```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.2D\5.2Dproject> docker-compose up  --build
Creating network "52dproject_my-network" with driver "bridge"
Building app1
[+] Building 3.4s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
```

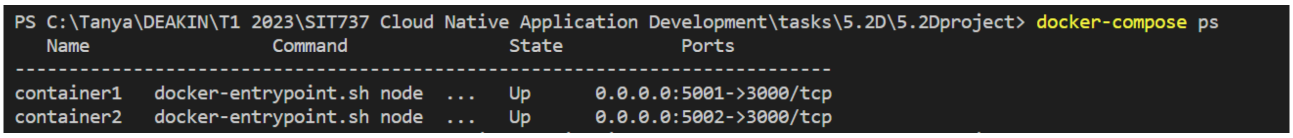


```
Building app2
[+] Building 0.7s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
          0.0s
 => => transferring dockerfile: 32B
```



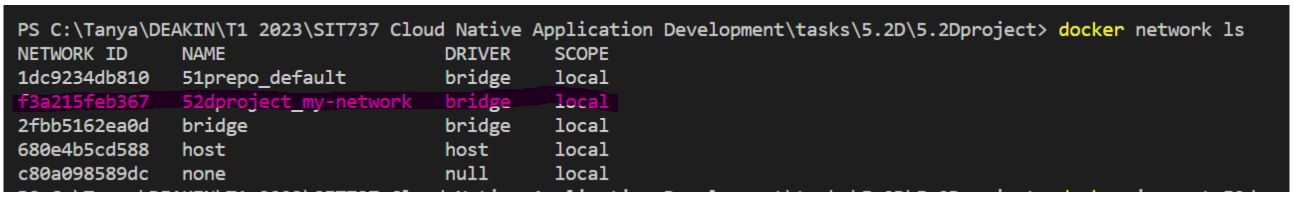
```
 => => writing image sha256:24453126c3b4405363c397ef31a46a642072f746b8aee8c759dfa556b121c3ef
Attaching to container2, container1
container2 | Running on http://${HOST}:${PORT}
container1 | Running on http://${HOST}:${PORT}
```

5. Docker compose ps command for the same –



```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.2D\5.2Dproject> docker-compose ps
   Name              Command               State           Ports
-----------------------------------------------------------------------------
container1   docker-entrypoint.sh node   ...   Up      0.0.0.0:5001->3000/tcp
container2   docker-entrypoint.sh node   ...   Up      0.0.0.0:5002->3000/tcp
```

6. The bridge has got created from the docker compose yaml file



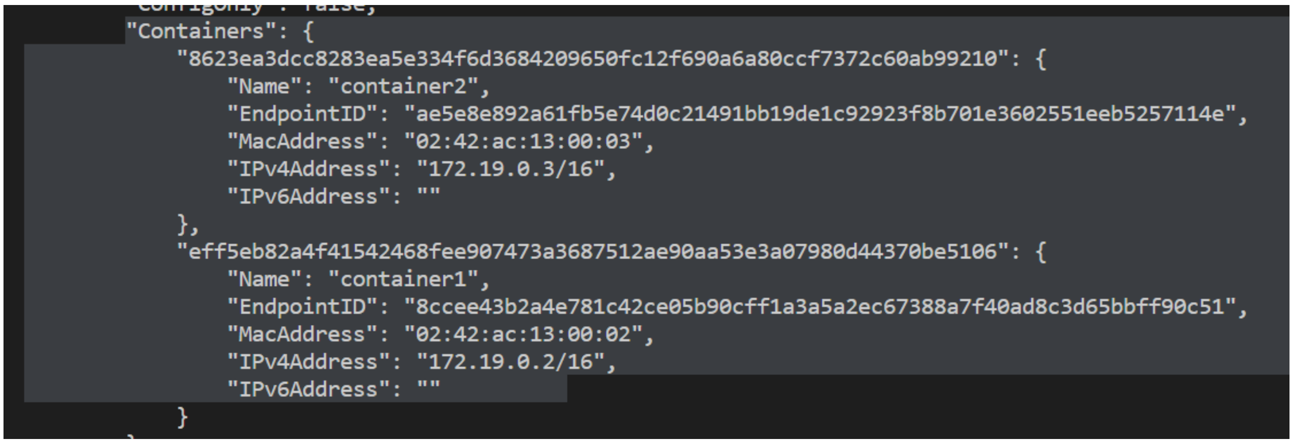
```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.2D\5.2Dproject> docker network ls
NETWORK ID     NAME                    DRIVER    SCOPE
1dc9234db810   51prepo_default         bridge    local
f3a215feb367   52dproject_my-network   bridge    local
2fbb5162ea0d   bridge                  bridge    local
680e4b5cd588   host                    host      local
c80a098589dc   none                    null      local
```

7. Ran docker inspect to check the bridge created.

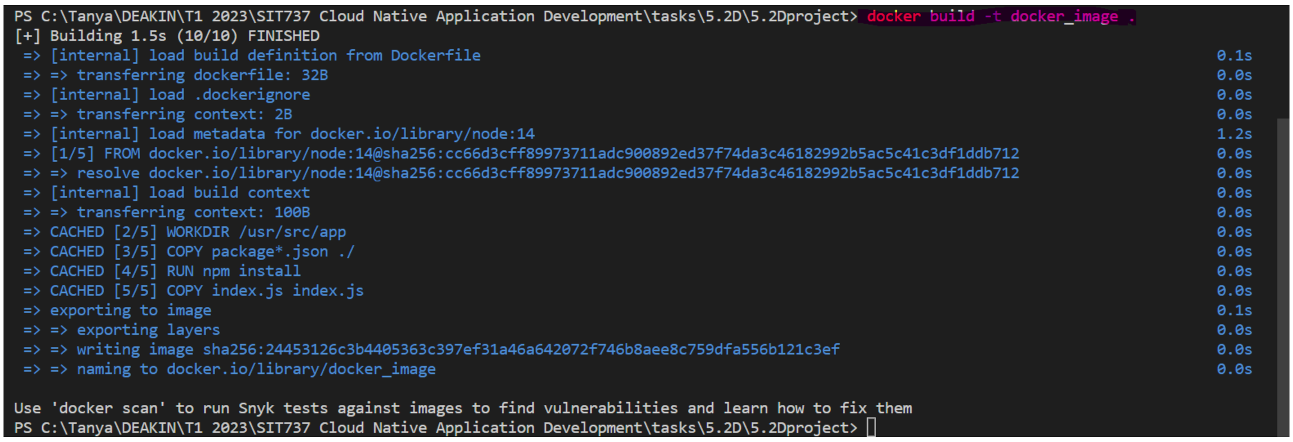
**Command** used→ docker inspect 52dproject_my-network

**Output**: The network shows the containers as expected. IP addresses can be seen.

**Screenshot** –

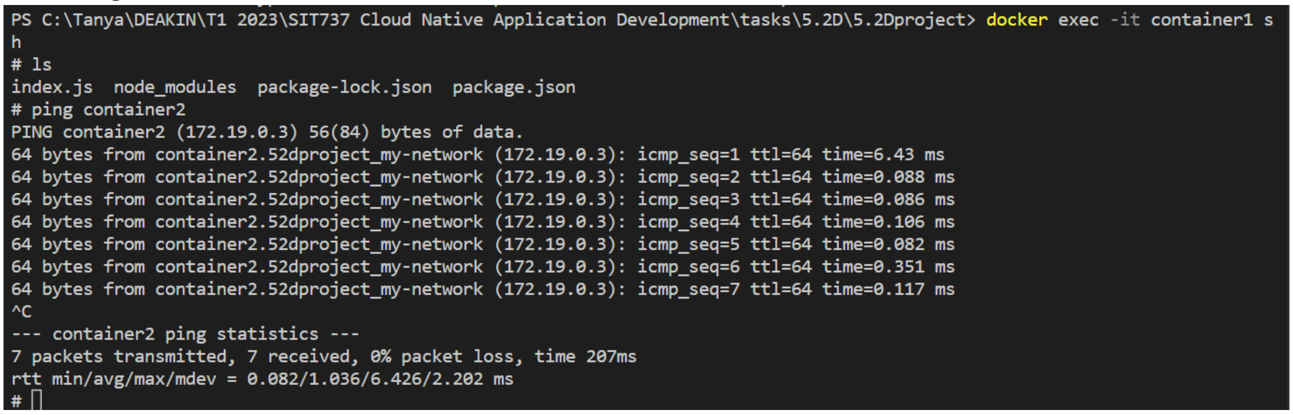


```
        "Containers": {
            "8623ea3dcc8283ea5e334f6d3684209650fc12f690a6a80ccf7372c60ab99210": {
                "Name": "container2",
                "EndpointID": "ae5e8e892a61fb5e74d0c21491bb19de1c92923f8b701e3602551eeb5257114e",
                "MacAddress": "02:42:ac:13:00:03",
                "IPv4Address": "172.19.0.3/16",
                "IPv6Address": ""
            },
            "eff5eb82a4f41542468fee907473a3687512ae90aa53e3a07980d44370be5106": {
                "Name": "container1",
                "EndpointID": "8ccee43b2a4e781c42ce05b90cff1a3a5a2ec67388a7f40ad8c3d65bbff90c51",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            }
```



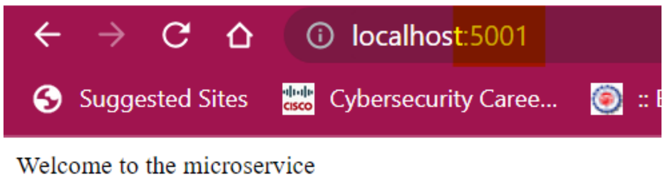
```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.2D\5.2Dproject> docker build -t docker_image .
[+] Building 1.5s (10/10) FINISHED
 => [internal] load build definition from Dockerfile                                                          0.1s
 => => transferring dockerfile: 32B                                                                           0.0s
 => [internal] load .dockerignore                                                                             0.0s
 => => transferring context: 2B                                                                               0.0s
 => [internal] load metadata for docker.io/library/node:14                                                    1.2s
 => [1/5] FROM docker.io/library/node:14@sha256:cc66d3cff89973711adc900892ed37f74da3c46182992b5ac5c41c3df1ddb712  0.0s
 => => resolve docker.io/library/node:14@sha256:cc66d3cff89973711adc900892ed37f74da3c46182992b5ac5c41c3df1ddb712  0.0s
 => [internal] load build context                                                                             0.0s
 => => transferring context: 100B                                                                             0.0s
 => CACHED [2/5] WORKDIR /usr/src/app                                                                         0.0s
 => CACHED [3/5] COPY package*.json ./                                                                        0.0s
 => CACHED [4/5] RUN npm install                                                                              0.0s
 => CACHED [5/5] COPY index.js index.js                                                                       0.0s
 => exporting to image                                                                                        0.1s
 => => exporting layers                                                                                       0.0s
 => => writing image sha256:24453126c3b4405363c397ef31a46a642072f746b8aee8c759dfa556b121c3ef                  0.0s
 => => naming to docker.io/library/docker_image                                                               0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.2D\5.2Dproject>
```

8. Messaged container2 from container1



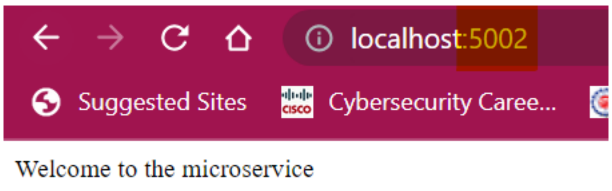
```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.2D\5.2Dproject> docker exec -it container1 s
h
# ls
index.js  node_modules  package-lock.json  package.json
# ping container2
PING container2 (172.19.0.3) 56(84) bytes of data.
64 bytes from container2.52dproject_my-network (172.19.0.3): icmp_seq=1 ttl=64 time=6.43 ms
64 bytes from container2.52dproject_my-network (172.19.0.3): icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from container2.52dproject_my-network (172.19.0.3): icmp_seq=3 ttl=64 time=0.086 ms
64 bytes from container2.52dproject_my-network (172.19.0.3): icmp_seq=4 ttl=64 time=0.106 ms
64 bytes from container2.52dproject_my-network (172.19.0.3): icmp_seq=5 ttl=64 time=0.082 ms
64 bytes from container2.52dproject_my-network (172.19.0.3): icmp_seq=6 ttl=64 time=0.351 ms
64 bytes from container2.52dproject_my-network (172.19.0.3): icmp_seq=7 ttl=64 time=0.117 ms
^C
--- container2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 207ms
rtt min/avg/max/mdev = 0.082/1.036/6.426/2.202 ms
#
```

9. Both web servers are running as below –



Welcome to the microservice

and,



Welcome to the microservice

**PART 2**

**In addition to above steps, what do you suggest for monitoring the network traffic between containers? Is there any solution using Docker dashboard to perform monitoring of your configured network?**

Monitoring is an important step in creating software applications. It helps in optimal performance, security, and compliance of systems. In the dockerized platform it is important to configure and setup the monitoring for the containers because of its ephemeral nature. Prometheus and Grafana are the two popular open source tools that can be used for monitoring and visualizing metrics and creating dashboards. Both the tools have large community support who help in improving the application. Docker images are available for both the application and require various steps to set up.

Prometheus is a pull-based model for collecting metrics applications.It is a widely used application in microservice based architecture. Prometheus is used with tools such as Grafana for visualization. Prometheus also has built-in alerting capabilities to notify when the container use high CPU or when the memory space in the disk is low. It also has a querying language called PromQl to query and aggregate metrics over time.

Below command can be used to pull and create docker container for prometheus

- docker pull bitnami/prometheus

Grafana is used for visualizing metrics and creating dashboards. Various monitoring platforms can be integrated with Grafana to produce Grafana dashboards. It has a web based interface where users can create numerous dashboards according to different use cases. It also has comprehensive data querying and filtering features.

Below command can be used to pull and create docker container for prometheus

- docker pull grafana/grafana

Setting up Prometheus and Grafana in docker platform would have steps like, creating a network interface for grafana-prometheus. Deploying and running both the containers in the same network. Adding data-source configurations in Grafana and linking the configurations to Prometheus. Using the localhost to visualize the dashboards.

Future implementation for the same can be learnt from : https://www.linkedin.com/pulse/running-grafana-prometheus-docker-stephen-townshend/