

## Task 5.1P

### 1. Docker Installation

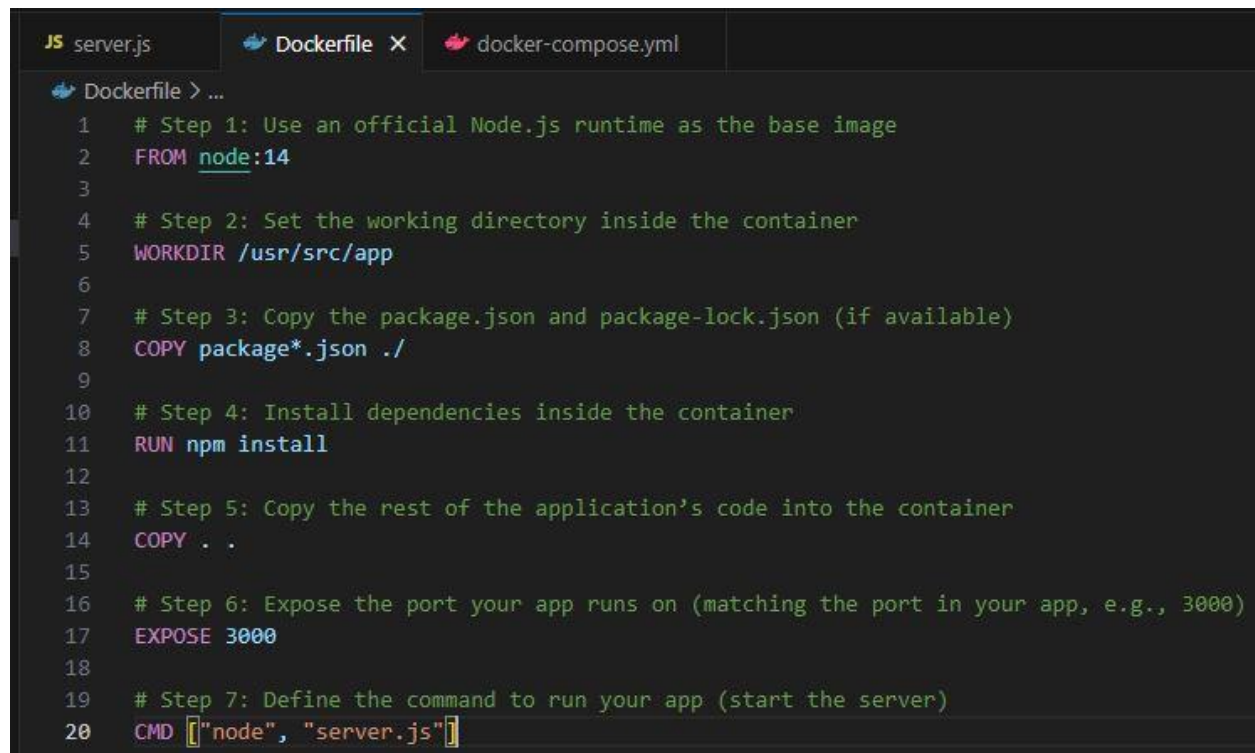
- Ensure and check if Docker is installed:
- `docker --version`

### 2. Sample Web Application

- I chose the application that I was working on during week 4, the calculator application
- <https://github.com/mustafaT96/sit737-2025-prac4p.git>

### 3. Create a Dockerfile

- A **Dockerfile** defines the environment and steps needed to run your application inside a container.
- I created the **Dockerfile** in the root directory of the cloned project as shown below:



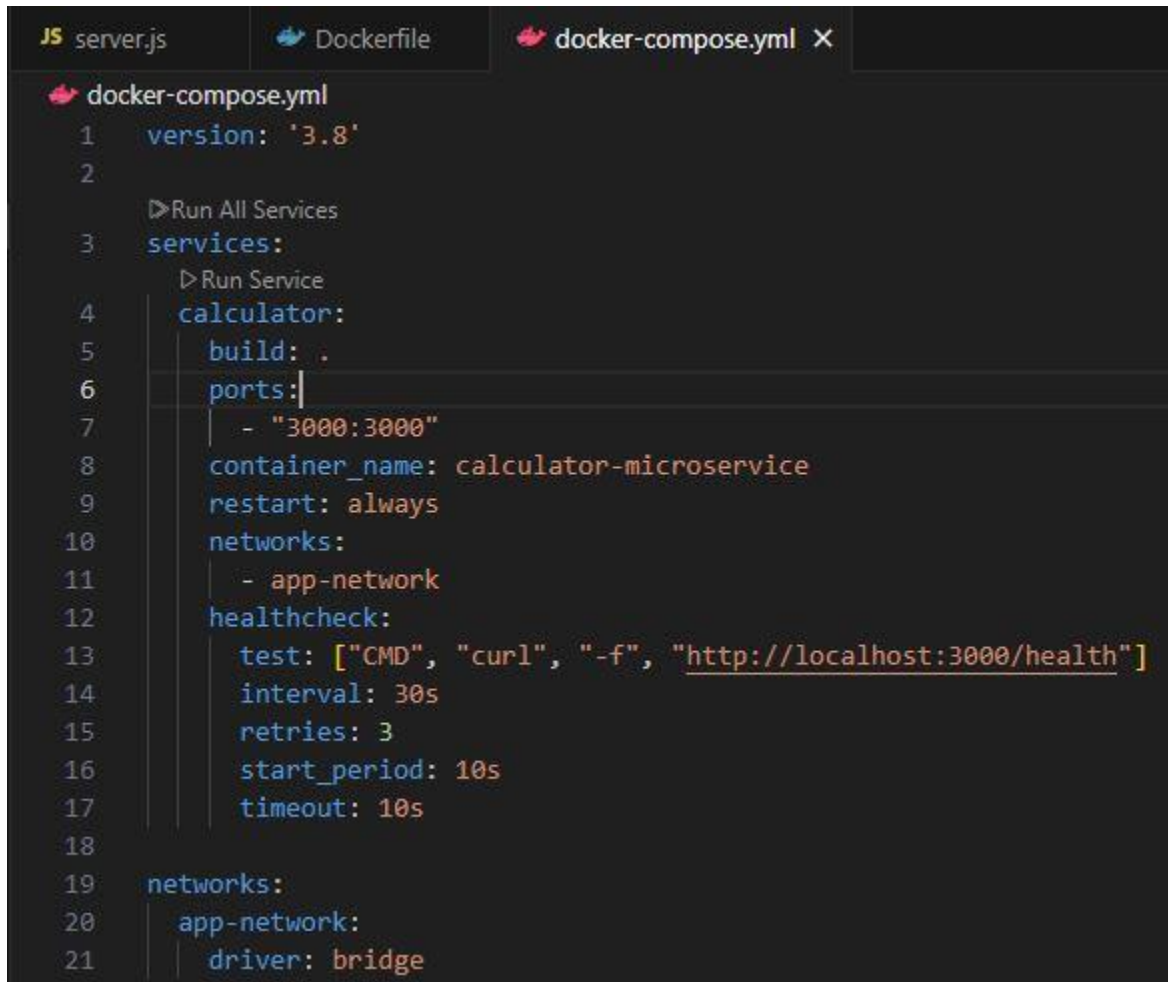
```
JS server.js Dockerfile X docker-compose.yml
Dockerfile > ...
1  # Step 1: Use an official Node.js runtime as the base image
2  FROM node:14
3
4  # Step 2: Set the working directory inside the container
5  WORKDIR /usr/src/app
6
7  # Step 3: Copy the package.json and package-lock.json (if available)
8  COPY package*.json ./
9
10 # Step 4: Install dependencies inside the container
11 RUN npm install
12
13 # Step 5: Copy the rest of the application's code into the container
14 COPY . .
15
16 # Step 6: Expose the port your app runs on (matching the port in your app, e.g., 3000)
17 EXPOSE 3000
18
19 # Step 7: Define the command to run your app (start the server)
20 CMD ["node", "server.js"]
```

### 4. Build the Docker Image

- Created the Docker image using the *Dockerfile*:
- `docker build -t calculator-microservice .`
- This command will create a Docker image tagged as *calculator-microservice* from the current directory (.).

## 5. Create a Docker Compose File

- The *docker-compose.yml* file defines the services (containers) required for the application.



```
JS server.js Dockerfile docker-compose.yml X
docker-compose.yml
1  version: '3.8'
2
3  > Run All Services
4  services:
5    > Run Service
6    calculator:
7      build: .
8      ports:
9        - "3000:3000"
10     container_name: calculator-microservice
11     restart: always
12     networks:
13       - app-network
14     healthcheck:
15       test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
16       interval: 30s
17       retries: 3
18       start_period: 10s
19       timeout: 10s
20
21     networks:
22       app-network:
23         driver: bridge
```

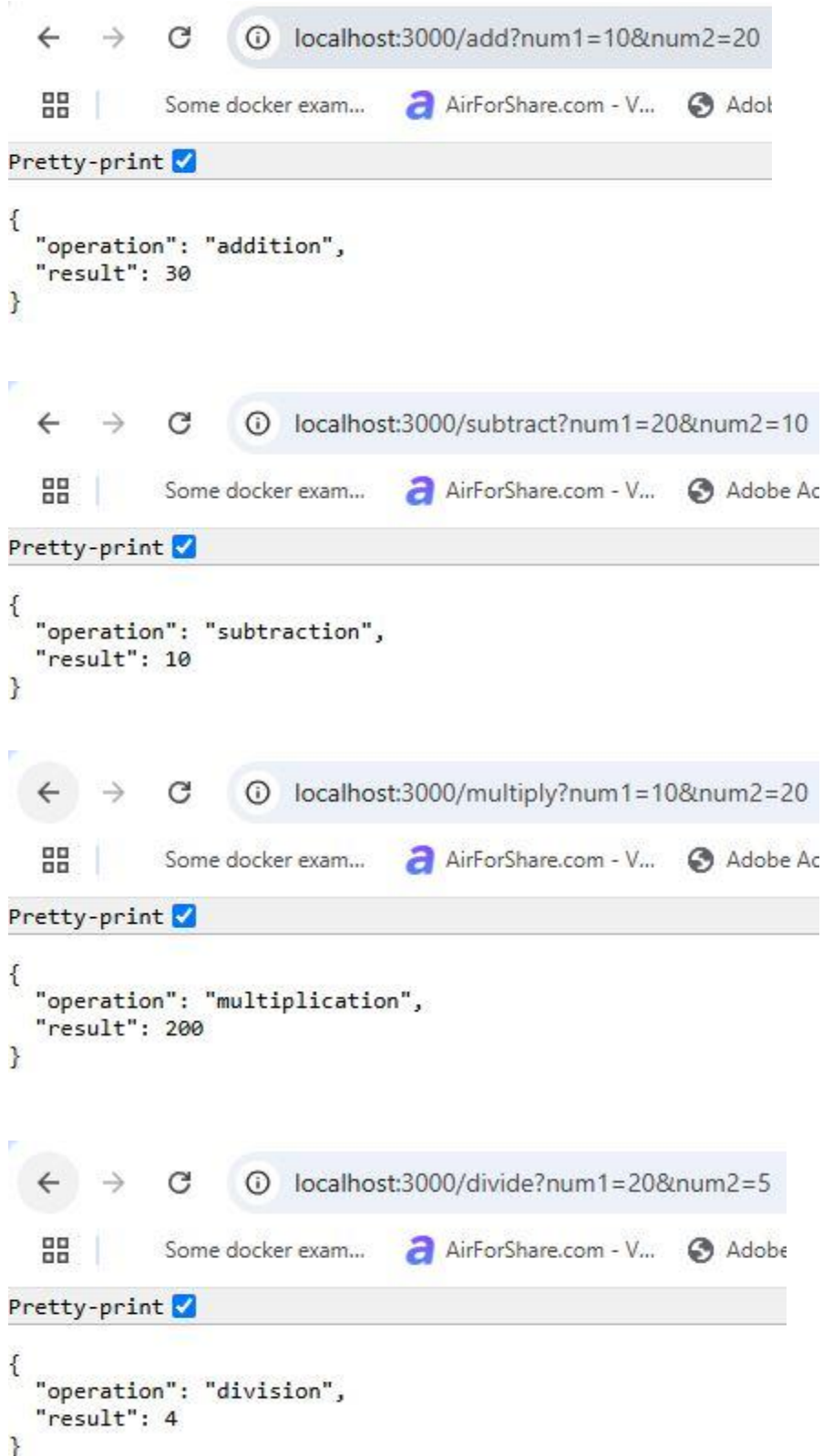
- This defines a single service (*calculator*) and includes the *healthcheck* block.

## 6. Start the Docker Compose Environment

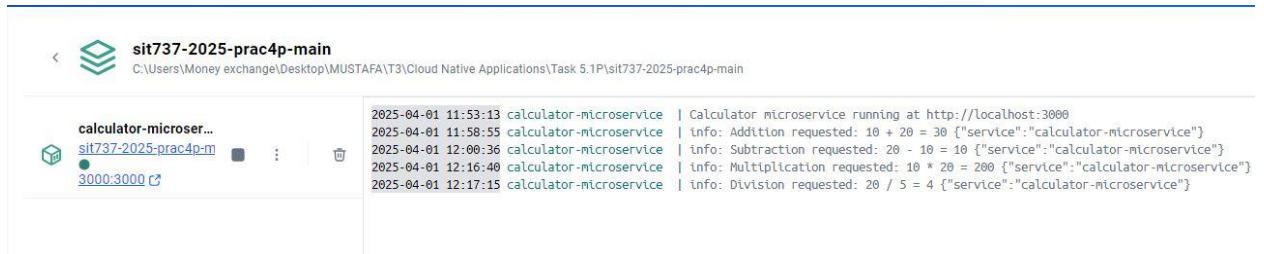
- Start all the containers defined in the *docker-compose.yml* file:
- docker-compose up -d*
- The *-d* flag runs the containers in detached mode, so they run in the background.

## 7. Test the Application

- To test if the application is running, open a browser to access the web application:

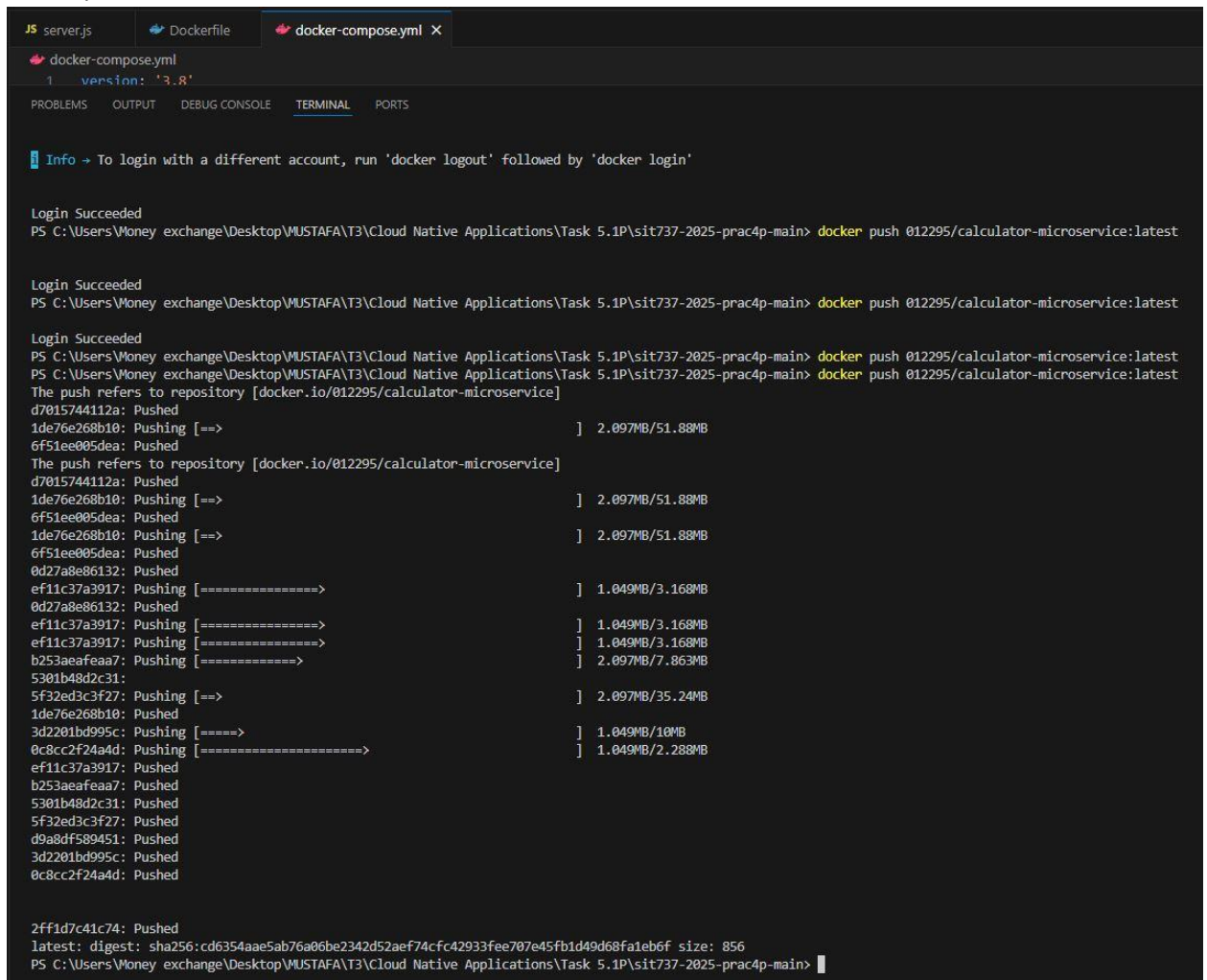


- Container logs



## 8. Push the Docker Image to a Registry

- Tag the image to push it to a Docker registry
- `docker push 012295/calculator-microservice:latest`



## Part II: Health Check Implementation

### 1. Add a Health Endpoint

- In the *server.js* file, add a */health* route to return a healthy status:

```
app.get('/health', (req, res) => {  
  res.status(200).send('OK');  
})
```

- This endpoint responds with *200 OK* when the application is healthy.

### 2. Modify Docker Compose File to Include Health Check

- Update the *docker-compose.yml* file to include the health check:

```
healthcheck:  
  test: ["CMD", "curl", "-f", "http://localhost:3000/health"]  
  interval: 30s  
  retries: 3  
  start_period: 10s  
  timeout: 10s
```

- This configures the health check to call the */health* endpoint every 30 seconds, retrying up to 3 times if it fails.

### 3. Start or Restart the Docker Compose Environment

- Restart containers to apply the changes:

```
docker-compose down  
docker-compose up -d
```

### 4. Testing the health check

- Use the following command to check the health status of the container:

```
docker inspect --format='{{json .State.Health}}' calculator-microservice
```

- To test the health check, stop the container:

```
docker stop calculator-microservice
```

- After stopping, check the health status again:

```
docker inspect --format='{{json .State.Health}}' calculator-microservice
```

- The health status will now show as "unhealthy"

- Restart the container:  
`docker start calculator-microservice`
- After restarting, check the health status once more:  
`docker inspect --format='{{json .State.Health}}' calculator-microservice`
- It should show the container as "healthy"

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
-> => exporting manifest list sha256:d82e6777b0ba25af6d9a28544c706678102330018f9b1461e1f9fabbb6122a2 0.0s
-> => naming to docker.io/library/sit737-2025-prac4p-main-calculator:latest 0.0s
-> => unpacking to docker.io/library/sit737-2025-prac4p-main-calculator:latest 0.3s
-> [calculator] resolving provenance for metadata file 0.0s
[*] Running 3/3
  ✓ calculator Built 0.0s
  ✓ Network sit737-2025-prac4p-main-app-network Created 0.1s
  ✓ Container calculator-microservice Started 1.5s
PS C:\Users\Voney\exchange\Desktop\MUSTAFA\T3\Cloud Native Applications\Task 5.1P\sit737-2025-prac4p-main> docker inspect --format='{{json .State.Health}}' calculator-microservice
{"Status":"healthy","FailingStreak":0,"Log":[{"Start":"2025-04-01T02:04:57.419012287Z","End":"2025-04-01T02:04:57.479060504Z","ExitCode":0,"Output":""}]}
PS C:\Users\Voney\exchange\Desktop\MUSTAFA\T3\Cloud Native Applications\Task 5.1P\sit737-2025-prac4p-main> docker stop calculator-microservice
calculator-microservice
PS C:\Users\Voney\exchange\Desktop\MUSTAFA\T3\Cloud Native Applications\Task 5.1P\sit737-2025-prac4p-main> docker inspect --format='{{json .State.Health}}' calculator-microservice
{"Status":"unhealthy","FailingStreak":0,"Log":[{"Start":"2025-04-01T02:04:57.419012287Z","End":"2025-04-01T02:04:57.479060504Z","ExitCode":0,"Output":""}]}
PS C:\Users\Voney\exchange\Desktop\MUSTAFA\T3\Cloud Native Applications\Task 5.1P\sit737-2025-prac4p-main> docker start calculator-microservice
calculator-microservice
PS C:\Users\Voney\exchange\Desktop\MUSTAFA\T3\Cloud Native Applications\Task 5.1P\sit737-2025-prac4p-main> docker inspect --format='{{json .State.Health}}' calculator-microservice
{"Status":"healthy","FailingStreak":0,"Log":[{"Start":"2025-04-01T02:04:57.419012287Z","End":"2025-04-01T02:04:57.479060504Z","ExitCode":0,"Output":""}]}

```