5.1P: Containerisation of a simple web application using Docker

1. Install Docker

Installed, screenshot –

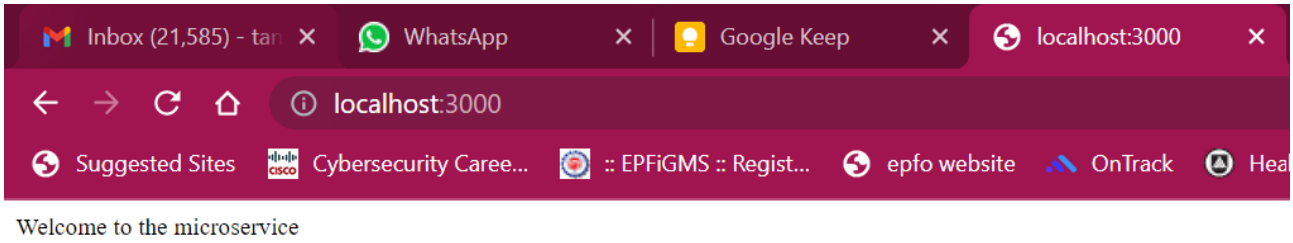
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL                                   powershell +

PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\practical\5.1P> git --version
git version 2.37.0.windows.1
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\practical\5.1P> docker --version
Docker version 20.10.17, build 100c701
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\practical\5.1P>
```

2. Clone the sample web application *(you can clone your app from the previous ontrack task, or if you are going to develop an application for HD tasks you can proceed with that one- there is no limitation for the app)*

The cloned application has very simple code –

```
Welcome    JS index.js ×
JS index.js > ...
const { json } = require('express');
const express = require('express');
const app = express();

// Home page request
app.get('/', (req, res) => {
  res.send("Welcome to the microservice");
});

app.listen(3000, () => {
  console.log('Server is listening on port 3000');
});
```

And runs with the output –



3. Create a Dockerfile

```
5.1P.code-workspace U    index.js M    Dockerfile U
5.1P > Dockerfile
#denotes base image
FROM node:14

#setting working directory
WORKDIR /usr/src/app

COPY package*.json index.js ./

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

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

4. Build the Docker image

Command used → docker build -t helloworld

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                                    powershell

PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.1P\5.1Prepo\5.1P> docker build -t helloworld .
[+] Building 180.5s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                                                    0.1s
 => => transferring dockerfile: 295B                                                                                    0.1s
 => [internal] load .dockerignore                                                                                       0.1s
 => => transferring context: 2B                                                                                         0.0s
 => [internal] load metadata for docker.io/library/node:14                                                              4.1s
 => [1/4] FROM docker.io/library/node:14@sha256:a97048059988c65f974b37dfe25a44327069a0f4f81133624871de0063b98075      170.2s
 => => resolve docker.io/library/node:14@sha256:a97048059988c65f974b37dfe25a44327069a0f4f81133624871de0063b98075        0.0s
 => => sha256:a97048059988c65f974b37dfe25a44327069a0f4f81133624871de0063b98075 776B / 776B                              0.0s
 => => sha256:7c865c7b34546b8756bab8a67927af6f09fdc0eeb76b67e1b5ef18a3f85ae574 7.52kB / 7.52kB                          0.0s
 => => sha256:792af667f62688dbef1f3ffeeca2daa6d448a62b6cacd604f1c4fc043d6cc2a6 7.86MB / 7.86MB                         11.4s
 => => sha256:3e37868ebf669334a2dbdb206ac7b84d8f8d184a40dfb8c9bc501b29cda12548 10.00MB / 10.00MB                       29.0s
 => => sha256:aeb3a59418bc20c7c340a3f8afb19c2258cff7d1bae213720ee12b3d885fd1b1 2.21kB / 2.21kB                          0.0s
 => => sha256:4e2befb7f5d18aa27b3619ddf1b93607e62ca82d0c627557537c149893346d86 50.45MB / 50.45MB                      101.6s
 => => sha256:591fe17e35ddac86d63495a7708b07af369e0d67d8742880f1e73cf1a205e028 51.87MB / 51.87MB                      107.5s
 => => sha256:b9cba6e3073a1f2eac2d50c107bba6dbb76de0325854a3a0e7c6f52ae8fc5521 191.85MB / 191.85MB                    157.7s
 => => sha256:5d54aff43b9d5a82025cbae68c26151d48ce3e061018e8547a567ee0de7ca4a3 4.20kB / 4.20kB                         102.9s
 => => extracting sha256:4e2befb7f5d18aa27b3619ddf1b93607e62ca82d0c627557537c149893346d86                               2.7s
 => => extracting sha256:5d54aff43b9d5a82025cbae68c26151d48ce3e061018e8547a567ee0de7ca4a3                               0.1s
 => => extracting sha256:c78fc09a7fc1f753c36b55417e23e15443df991b2b8d966cee79732b55a8cabc                               2.1s
 => => extracting sha256:15e2b631f375d86aa99b5a571b1b34d8ad9d55c61d1012c9f4db15a9f8bf1125                               0.2s
 => => extracting sha256:b3936a150fbdeb5b012d3274d3cfb32613cd991f13a3d88a785e7e609f4cfe01                               0.0s
```

```
 => => extracting sha256:b3936a150fbdeb5b012d3274d3cfb32613cd991f13a3d88a785e7e609f4cfe01                               0.0s
 => => transferring context: 22.55kB                                                                                    0.0s
 => [2/4] WORKDIR /usr/src/app                                                                                          0.4s
 => [3/4] COPY package*.json index.js ./                                                                                0.1s
 => [4/4] RUN npm install                                                                                               5.2s
 => exporting to image                                                                                                  0.2s
 => => exporting layers                                                                                                 0.2s
 => => writing image sha256:69271a9a6eb2ef00417aff0d1e04a7f17fd76a8a7776ca35e9b9a16c80e043b1                            0.0s
 => => naming to docker.io/library/helloworld                                                                           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\5.1P\5.1Prepo\5.1P>
```

5. Create a Docker Compose file

Content added -

```
docker-compose.yml > {} services > {} app > {} build > dockerfile
all.json
services:
  app:
    image: helloworld_image
    build:
      context: ./5.1P
      dockerfile: Dockerfile
    container_name: helloworld_container
    restart: always
    ports:
      - "4026:3000"
```

6. Start the Docker Compose environment
docker-compose up --build

51prepo 1 container	-	Running (1/1)	-		Open
helloworld_container 434fb024249e	helloworld_image:latest	Running	4026	35 seconds ag	

7. Test the application

localhost:4026

Suggested Sites Cybersecurity Caree... :: EPFiG

Welcome to the microservice

8. Push the Docker image to a registry

- enabled the container registry in project

```
C:\Program Files (x86)\Google\Cloud SDK>gcloud services enable containerregistry.googleapis.com
Operation "operations/acf.p2-374820615943-b955585b-3fe1-410c-9801-334993dcea3b" finished successfully.

C:\Program Files (x86)\Google\Cloud SDK>
```

- Tagged the image using → docker tag <image_name> gcr.io/<project_id>/<image_name>
- docker tag helloworld_image gcr.io/sit737-23t1-gujral-6790fc3/helloworld_image

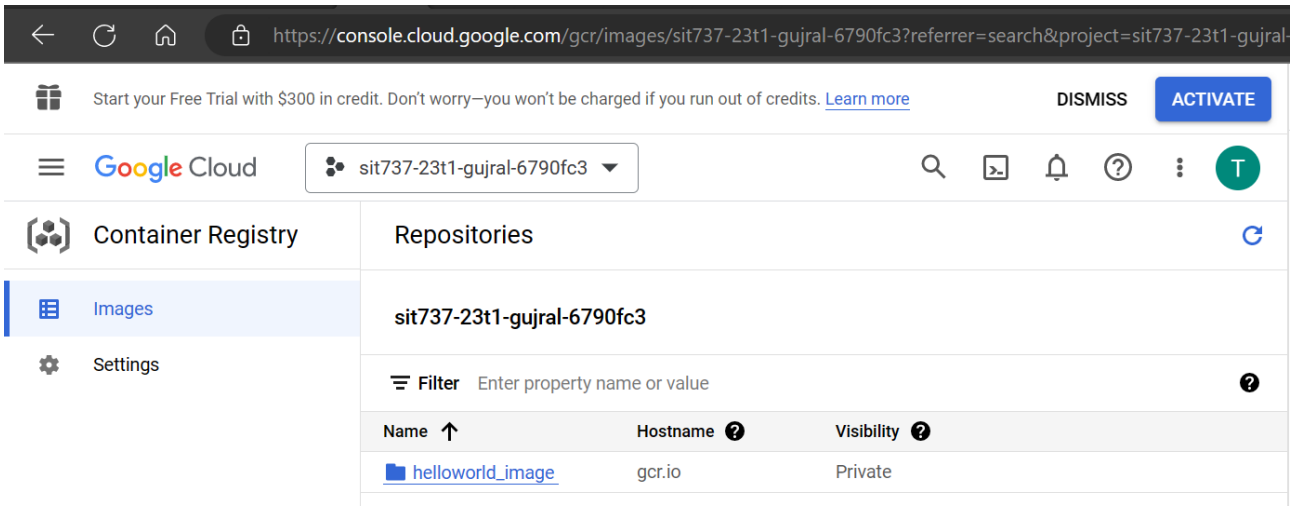
```
C:\Program Files (x86)\Google\Cloud SDK>docker tag helloworld_image gcr.io/sit737-23t1-gujral-6790fc3/helloworld_image

C:\Program Files (x86)\Google\Cloud SDK>
```

- Authenticate Docker with google cloud account
  Command used → gcloud auth configure-docker

```
Administrator: Command Prompt

C:\Windows\System32>gcloud auth configure-docker
Adding credentials for all GCR repositories.
WARNING: A long list of credential helpers may cause delays running 'docker build'. We recommend passing the registry na
me to configure only the registry you are using.
After update, the following will be written to your Docker config file located at [C:\Users\glkar\.docker\config.json]:
 {
  "credHelpers": {
    "gcr.io": "gcloud",
    "us.gcr.io": "gcloud",
    "eu.gcr.io": "gcloud",
    "asia.gcr.io": "gcloud",
    "staging-k8s.gcr.io": "gcloud",
    "marketplace.gcr.io": "gcloud"
  }
}

Do you want to continue (Y/n)?  Y

Docker configuration file updated.
```

- Pushed the image using → docker push gcr.io/<project_id>/<image_name>
- Command used → docker push gcr.io/sit737-23t1-gujral-6790fc3/helloworld_image

```
C:\Windows\System32>docker push gcr.io/sit737-23t1-gujral-6790fc3/helloworld_image
Using default tag: latest
The push refers to repository [gcr.io/sit737-23t1-gujral-6790fc3/helloworld_image]
78737201d7d7: Pushed
2f8d9537fe34: Pushed
5057de0c6dbb: Pushed
abc13cdab6f5: Pushed
a3da022a2808: Layer already exists
51035f2c0064: Layer already exists
928cd3b0404b: Layer already exists
cbc8f3f1dc11: Layer already exists
c58e75992d51: Layer already exists
847345bfcef4: Layer already exists
41523268899a: Layer already exists
be70b6dc0e0e: Layer already exists
f50705e98bf5: Layer already exists
latest: digest: sha256:daf7d85daa998232ceb12988f7c837e94e6421ded6fe3cac0e90a2d10d8047ba size: 3048
```



- Verified the successful image upload -
  Command →gcloud container images list-tags gcr.io/sit737-23t1-gujral-6790fc3/helloworld_image
  Screenshot –

```
C:\Windows\System32>gcloud container images list-tags gcr.io/sit737-23t1-gujral-6790fc3/helloworld_image
DIGEST: daf7d85daa99
TAGS: latest
```

9. Implement container health check

You can continue this task and modify the Docker Compose file to include container health checks that monitor the status of the application and its dependencies. If a container fails a health check, the container should be restarted automatically

This has been implemented using the following in the docker compose yaml file –

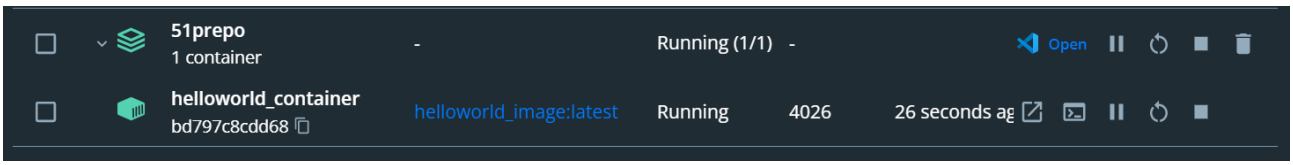
```
docker-compose.yml > {} services > {} app > {} healthcheck > [ ] test
all.json
services:
  app:
    image: helloworld_image
    build:
      context: ./5.1P
      dockerfile: Dockerfile
    container_name: helloworld_container
    restart: on-failure
    ports:
      - "4026:3000"
    healthcheck:
      test: ["CMD-SHELL", "curl --fail http://localhost:3000/ || exit 1"]
      interval: 30s
      timeout: 10s
      retries: 3
```

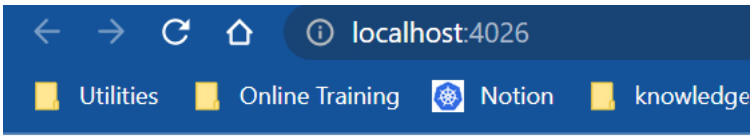
- The restart policy is set to on-failure
- Added healthcheck command in the docker compose file
- Added the properties of test, interval, timeout and retries in a way that http://localhost:4026/ is tested every 1 minute to check if the server is up.
- Timeout 10 sec is the duration it takes to wait for health check to complete
- If it is down, docker automatically starts the container.
- Hence, by adding the health check above, docker keeps monitoring the health of the containers.

Testing this new addition –

- Modified the docker compose yaml
- Performed docker compose up

```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.1P\5.1Prepo> docker-compose up
Recreating helloworld_container ... done
Attaching to helloworld_container
helloworld_container | Running on http://${HOST}:${PORT}
```





Welcome to the microservice

- The container was running at this time as seen in the docker-compose ps output

```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.1P\5.1Prepo> docker-compos
e ps
        Name                      Command                      State                  Ports
-----------------------------------------------------------------------------------------------------
helloworld_container   docker-entrypoint.sh node  ...   Up (health: starting)   0.0.0.0:4026->3000/tcp
```

- Stopped the container explicitly and and then checked after a few seconds to see that it was up again.

```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.1P\5.1Prepo> docker stop h
elloworld_container
helloworld_container
```

```
PS C:\Tanya\DEAKIN\T1 2023\SIT737 Cloud Native Application Development\tasks\5.1P\5.1Prepo> docker-compos
e ps
        Name                      Command                      State                  Ports
-----------------------------------------------------------------------------------------------------
helloworld_container   docker-entrypoint.sh node  ...   Up (health: starting)   0.0.0.0:4026->3000/tcp
```