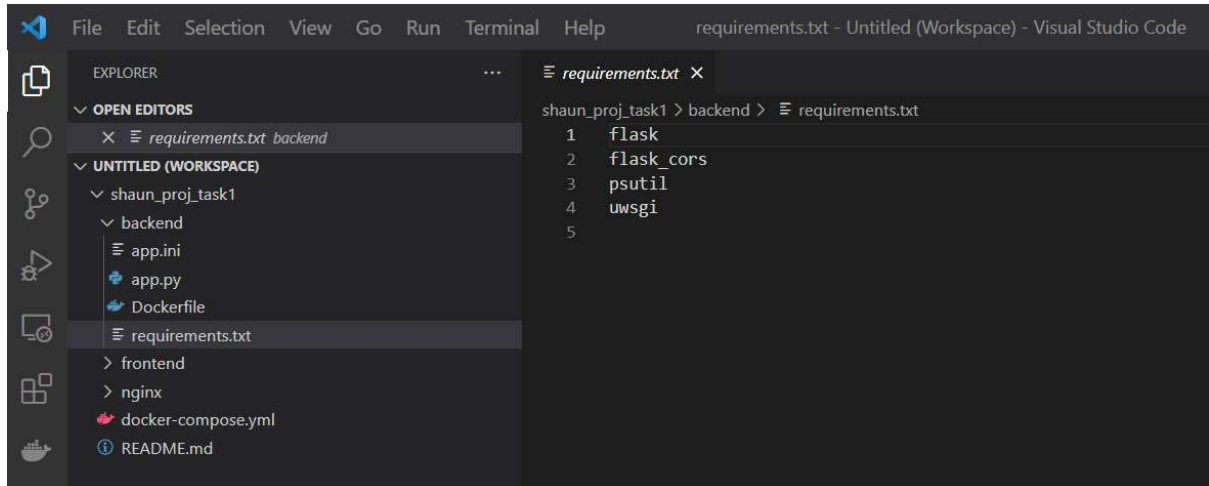


Task #1 Dockerize the Application

1. Setup backend container (flask).

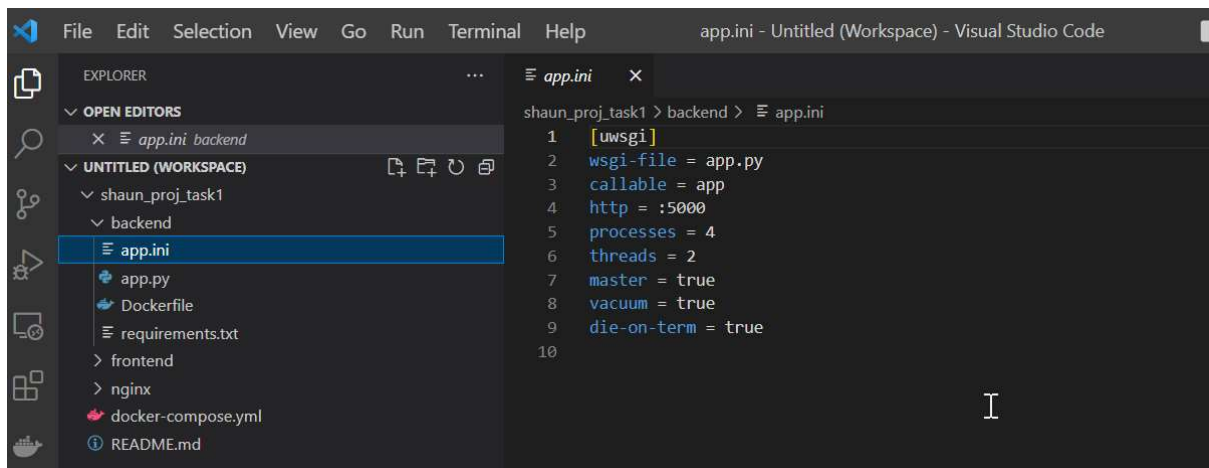
1a. Prepare requirement.txt for required modules:



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the project structure with the 'backend' directory expanded, containing 'app.ini', 'app.py', 'Dockerfile', and 'requirements.txt'. The Editor panel shows the 'requirements.txt' file with the following content:

```
1 flask
2 flask_cors
3 psutil
4 uwsgi
5
```

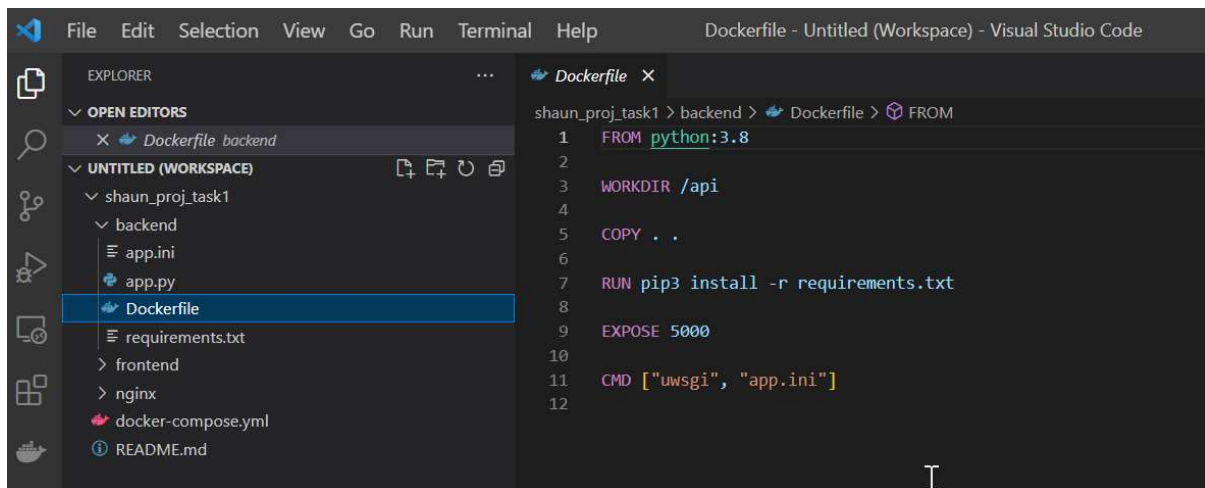
1b. Configure app.ini (use uWSGI Server to serve the flask application)



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the project structure with the 'backend' directory expanded, containing 'app.ini', 'app.py', 'Dockerfile', and 'requirements.txt'. The Editor panel shows the 'app.ini' file with the following content:

```
1 [uwsgi]
2 wsgi-file = app.py
3 callable = app
4 http = :5000
5 processes = 4
6 threads = 2
7 master = true
8 vacuum = true
9 die-on-term = true
10
```

1c. Configure backend(flask) dockerfile

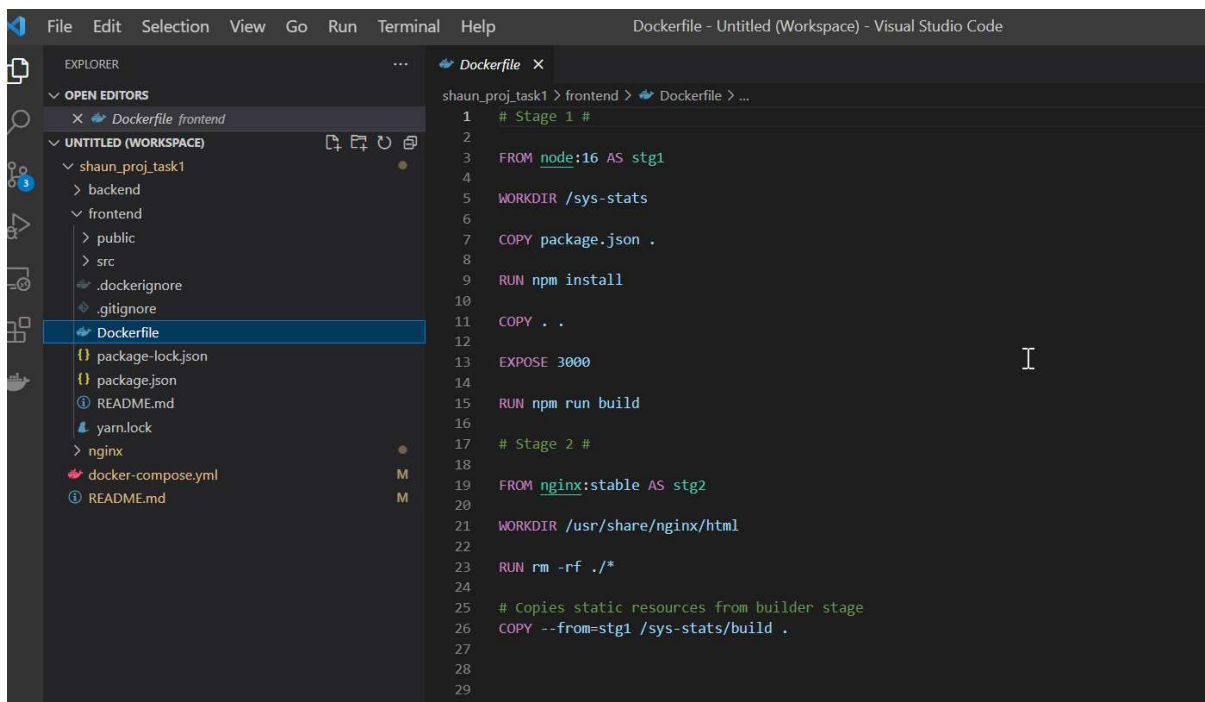


The screenshot shows the Visual Studio Code interface with a Dockerfile open in the editor. The Explorer sidebar on the left shows a project structure with a 'backend' directory containing 'app.ini', 'app.py', 'Dockerfile', and 'requirements.txt'. The Dockerfile in the editor contains the following instructions:

```
1 FROM python:3.8
2
3 WORKDIR /api
4
5 COPY . .
6
7 RUN pip3 install -r requirements.txt
8
9 EXPOSE 5000
10
11 CMD ["uwsgi", "app.ini"]
12
```

2. Setup frontend container (react)

2a. Setup multistage dockerfile for frontend.

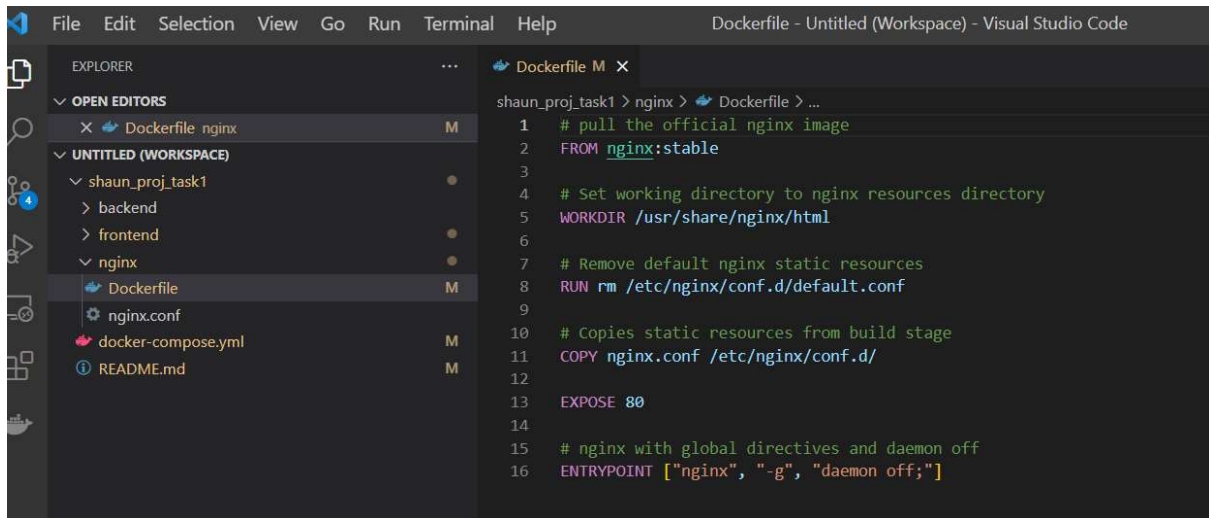


The screenshot shows the Visual Studio Code interface with a multistage Dockerfile open in the editor. The Explorer sidebar on the left shows a project structure with a 'frontend' directory containing 'public', 'src', '.dockerignore', '.gitignore', 'Dockerfile', 'package-lock.json', 'package.json', 'README.md', 'yarn.lock', 'nginx', 'docker-compose.yml', and 'README.md'. The Dockerfile in the editor contains the following instructions:

```
1 # Stage 1 #
2
3 FROM node:16 AS stg1
4
5 WORKDIR /sys-stats
6
7 COPY package.json .
8
9 RUN npm install
10
11 COPY . .
12
13 EXPOSE 3000
14
15 RUN npm run build
16
17 # Stage 2 #
18
19 FROM nginx:stable AS stg2
20
21 WORKDIR /usr/share/nginx/html
22
23 RUN rm -rf ./.*
24
25 # Copies static resources from builder stage
26 COPY --from=stg1 /sys-stats/build .
27
28
29
```

3. Setup nginx container

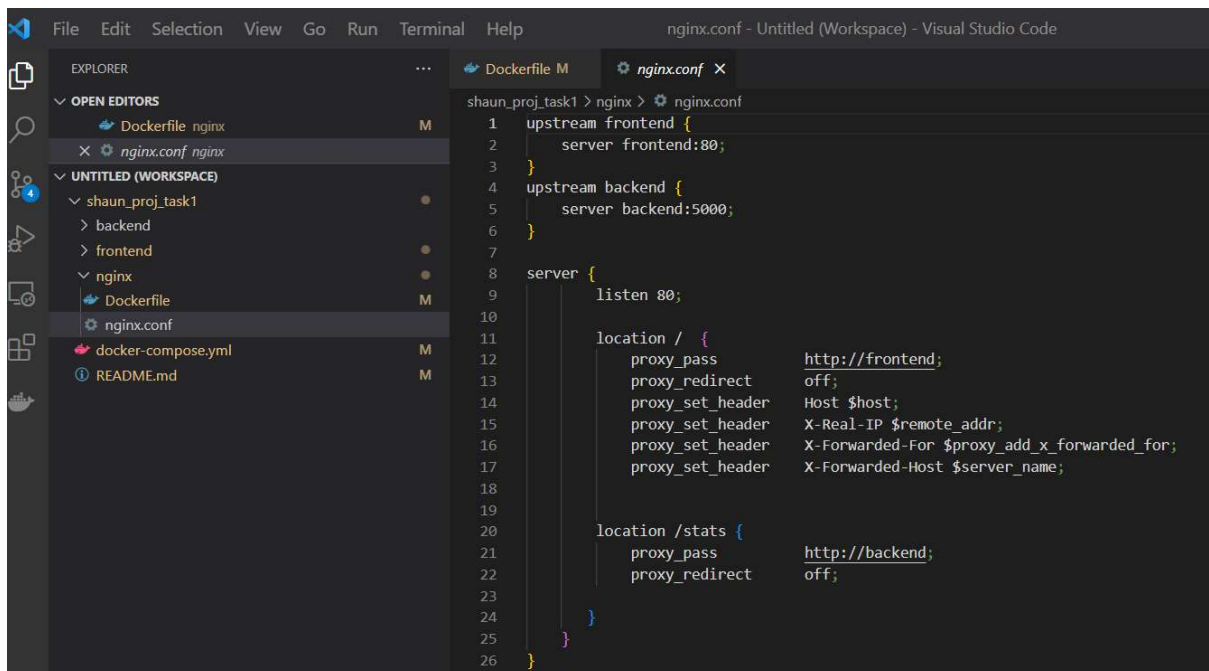
3a. Configure nginx dockerfile



The screenshot shows the Visual Studio Code interface with a Dockerfile open in the editor. The Explorer on the left shows the project structure with a folder named 'nginx' containing 'Dockerfile', 'nginx.conf', 'docker-compose.yml', and 'README.md'. The Dockerfile content is as follows:

```
1 # pull the official nginx image
2 FROM nginx:stable
3
4 # Set working directory to nginx resources directory
5 WORKDIR /usr/share/nginx/html
6
7 # Remove default nginx static resources
8 RUN rm /etc/nginx/conf.d/default.conf
9
10 # Copies static resources from build stage
11 COPY nginx.conf /etc/nginx/conf.d/
12
13 EXPOSE 80
14
15 # nginx with global directives and daemon off
16 ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

3b. Configure nginx.conf configuration file

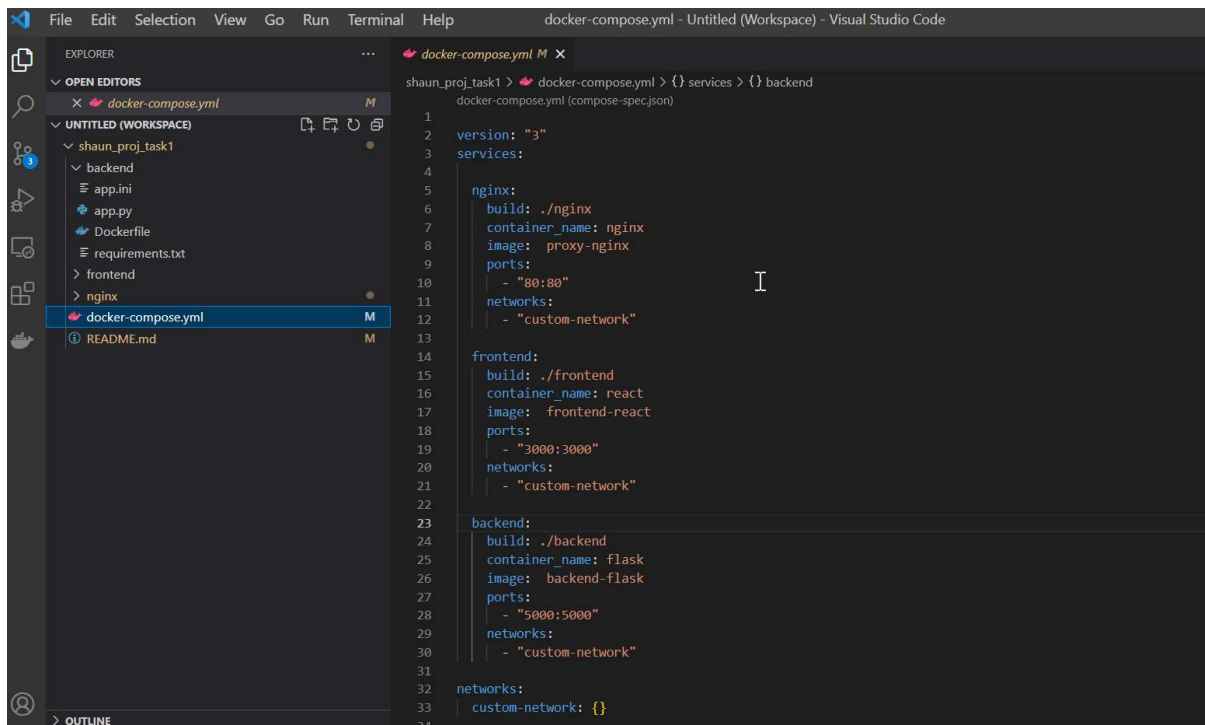


The screenshot shows the Visual Studio Code interface with the nginx.conf file open in the editor. The Explorer on the left shows the project structure with a folder named 'nginx' containing 'Dockerfile', 'nginx.conf', 'docker-compose.yml', and 'README.md'. The nginx.conf content is as follows:

```
1 upstream frontend {
2     server frontend:80;
3 }
4 upstream backend {
5     server backend:5000;
6 }
7
8 server {
9     listen 80;
10
11     location / {
12         proxy_pass http://frontend;
13         proxy_redirect off;
14         proxy_set_header Host $host;
15         proxy_set_header X-Real-IP $remote_addr;
16         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
17         proxy_set_header X-Forwarded-Host $server_name;
18     }
19
20     location /stats {
21         proxy_pass http://backend;
22         proxy_redirect off;
23     }
24 }
25
26 }
```

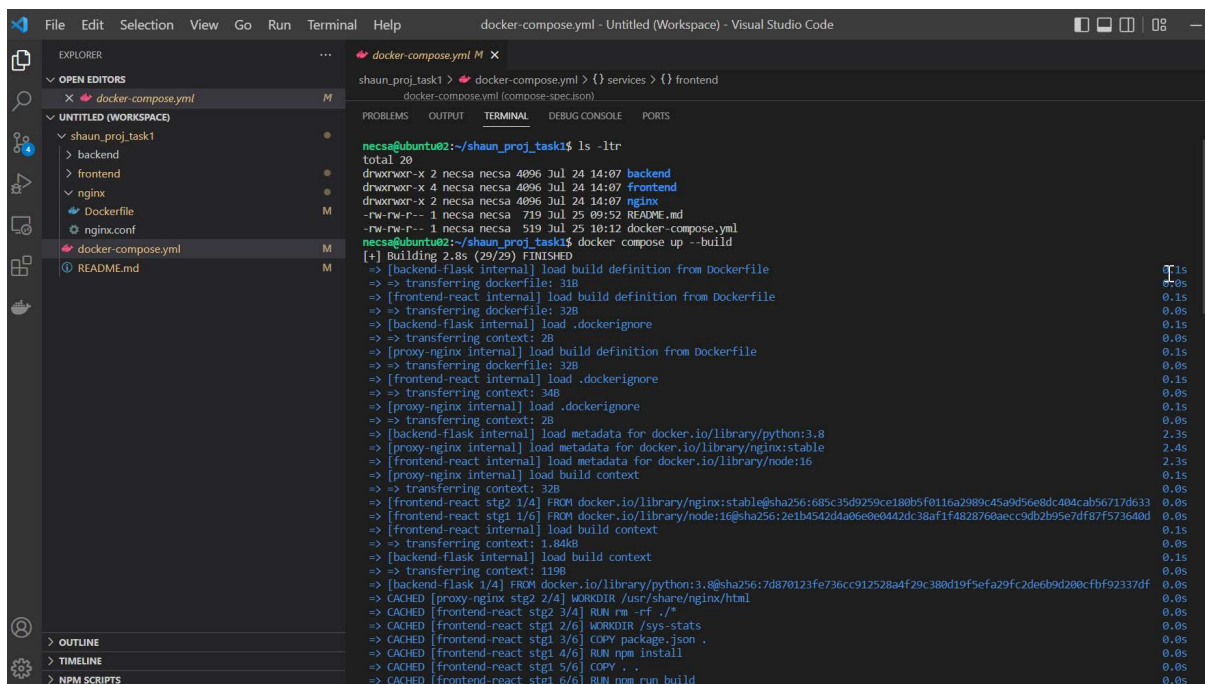
4. Deployment

4a. Prepare docker-compose.yml for the whole build.



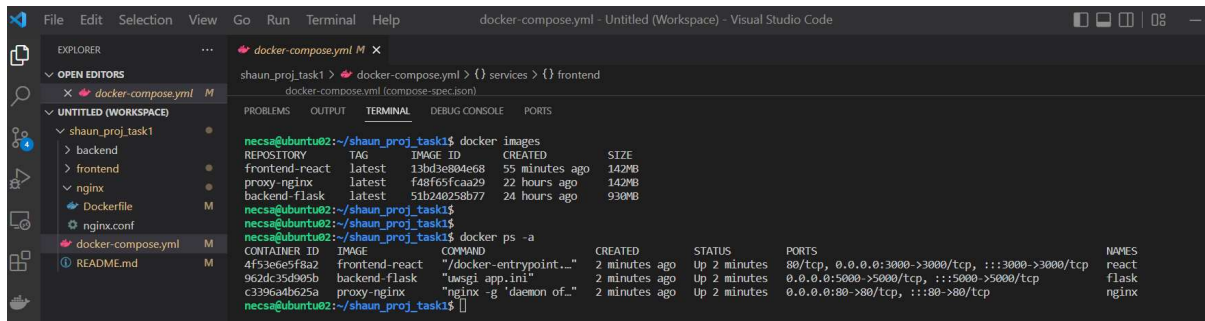
```
1 version: "3"
2 services:
3   nginx:
4     build: ./nginx
5     container_name: nginx
6     image: proxy-nginx
7     ports:
8       - "80:80"
9     networks:
10      - "custom-network"
11
12   frontend:
13     build: ./frontend
14     container_name: react
15     image: frontend-react
16     ports:
17       - "3000:3000"
18     networks:
19       - "custom-network"
20
21   backend:
22     build: ./backend
23     container_name: flask
24     image: backend-flask
25     ports:
26       - "5000:5000"
27     networks:
28       - "custom-network"
29
30 networks:
31   custom-network: {}
```

4b. Run deployment using docker compose up –build command.



```
ncsa@ubuntu02:~/shaun_proj_task1$ ls -ltr
total 20
drwxrwxr-x 2 ncsa ncsa 4096 Jul 24 14:07 backend
drwxrwxr-x 4 ncsa ncsa 4096 Jul 24 14:07 frontend
drwxrwxr-x 2 ncsa ncsa 4096 Jul 24 14:07 nginx
-rw-rw-r-- 1 ncsa ncsa 719 Jul 25 09:52 README.md
-rw-rw-r-- 1 ncsa ncsa 519 Jul 25 10:12 docker-compose.yml
ncsa@ubuntu02:~/shaun_proj_task1$ docker compose up --build
[+] Building 2.8s (29/29) FINISHED
=> [backend-flask internal] load build definition from Dockerfile
=> transferring dockerfile: 31B
=> [frontend-react internal] load build definition from Dockerfile
=> transferring dockerfile: 32B
=> [backend-flask internal] load .dockerignore
=> transferring context: 2B
=> [proxy-nginx internal] load build definition from Dockerfile
=> transferring dockerfile: 32B
=> [frontend-react internal] load .dockerignore
=> transferring context: 34B
=> [proxy-nginx internal] load .dockerignore
=> transferring context: 2B
=> [backend-flask internal] load metadata for docker.io/library/python:3.8
=> [proxy-nginx internal] load metadata for docker.io/library/nginx:stable
=> [frontend-react internal] load metadata for docker.io/library/node:16
=> [proxy-nginx internal] load build context
=> transferring context: 32B
=> [frontend-react stg2 1/4] FROM docker.io/library/nginx:stable@sha256:685c35d9259ce180b5f0116a2989c45a9d56e8dc404cab56717d633
=> [frontend-react stg1 1/6] FROM docker.io/library/node:16@sha256:2e1b4542d4a06e0e0442dc38af1f4828760aecd9b2b95e7df87f573640d
=> [frontend-react internal] load build context
=> transferring context: 1.84kB
=> [backend-flask internal] load build context
=> transferring context: 119B
=> [backend-flask 1/4] FROM docker.io/library/python:3.8@sha256:7d870123fe736cc912528a4f79c380d19f5efa29fc2de6bd9f280c6fbf92337df
=> CACHED [proxy-nginx stg2 2/4] WORKDIR /usr/share/nginx/html
=> CACHED [frontend-react stg2 3/4] RUN rm -rf /
=> CACHED [frontend-react stg1 2/6] WORKDIR /sys-stats
=> CACHED [frontend-react stg1 3/6] COPY package.json .
=> CACHED [frontend-react stg1 4/6] RUN npm install
=> CACHED [frontend-react stg1 5/6] COPY . .
=> CACHED [frontend-react stg1 6/6] RUN npm run build
```

4c. Verify resources created.



The screenshot shows the Visual Studio Code interface with a Docker Compose file open. The terminal output displays the following information:

```
shaun_proj_task1 > docker-compose.yml { } services > { } frontend
docker-compose.yml (compose-spec:1.0)

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

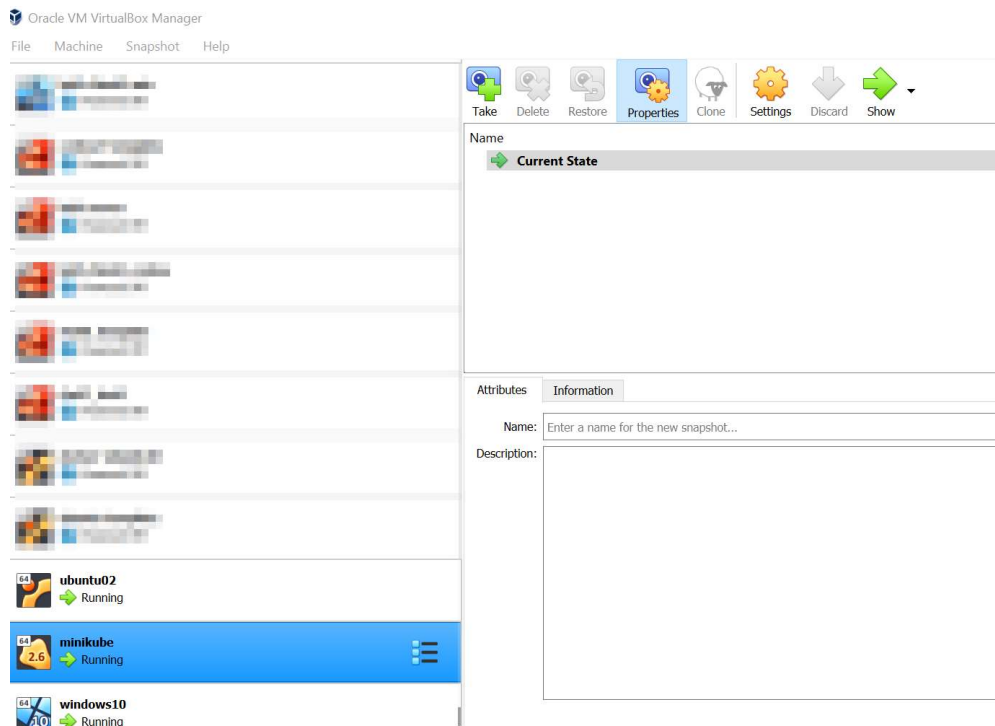
necsa@ubuntu02:~/shaun_proj_task1$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
frontend-react latest 13b3e80ae68 55 minutes ago 142MB
proxy-nginx latest f48f65fcaa29 22 hours ago 142MB
backend-flask latest 51b240258b77 24 hours ago 930MB
necsa@ubuntu02:~/shaun_proj_task1$
necsa@ubuntu02:~/shaun_proj_task1$
necsa@ubuntu02:~/shaun_proj_task1$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
4f33e5f8a2 frontend-react "/docker-entrypoint..." 2 minutes ago Up 2 minutes 80/tcp, 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp react
9e2dc35d985b backend-flask "uwsgi app.ini" 2 minutes ago Up 2 minutes 0.0.0.0:5000->5000/tcp, :::5000->5000/tcp flask
c3396a4b625a proxy-nginx "nginx -g 'daemon of..." 2 minutes ago Up 2 minutes 0.0.0.0:80->80/tcp, :::80->80/tcp nginx
necsa@ubuntu02:~/shaun_proj_task1$
```

5. Testing

5a. Environment:

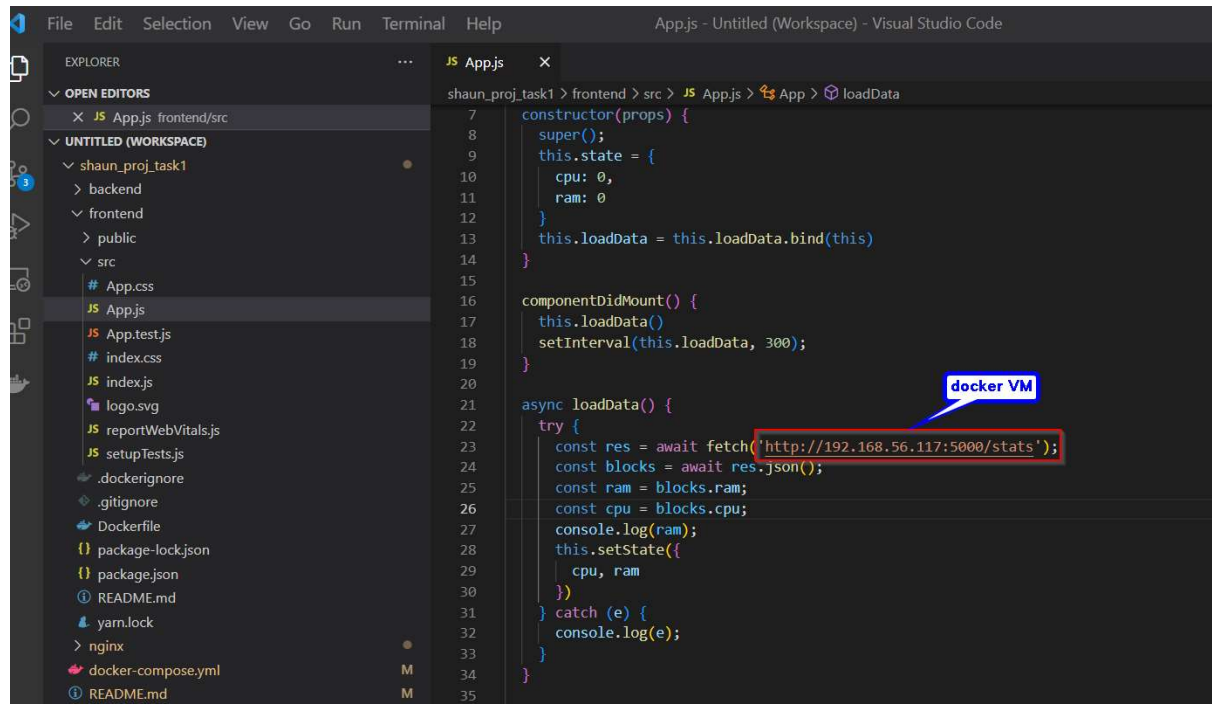
Ubuntu02 VM (192.168.56.117) -> VM machine where the dockers are being run and where the local dockers images are located.

Window10 VM (192.168.56.118) -> this is where to test the application is being tested via browser



For testing purposes, I have modified the src code to fetch the data from the docker VM itself (ubuntu02), where it will be reachable by windows VM via browser.

(note: unable to run docker desktop on my windows due to hardware settings limitation.



```
App.js
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

constructor(props) {
  super();
  this.state = {
    cpu: 0,
    ram: 0
  }
  this.loadData = this.loadData.bind(this)
}

componentDidMount() {
  this.loadData()
  setInterval(this.loadData, 300);
}

async loadData() {
  try {
    const res = await fetch('http://192.168.56.117:5000/stats');
    const blocks = await res.json();
    const ram = blocks.ram;
    const cpu = blocks.cpu;
    console.log(ram);
    this.setState({
      cpu, ram
    })
  } catch (e) {
    console.log(e);
  }
}
```

5b. Use Windows VM to test the application output via browser.

