LAB 5 – DEPLOYING WEB APPLICATION WITH DOCKER

5.1 Deploying React Application as Front-end:

5.1.1 Crucial requirment and configuration before building app:

In any react application, one of the most common functions is API calls with the specific url that refers to Back-end application. However, if the url was hard code, administrator will not be able to change url at deployment time (or running time). Therefore, in order to make url dynamically at runtime, the recommended solution is as following:

Step 1: In public folder of react application, create a new file with name:

public/runtime-config.js

```
// values replaced at deploy time (CI/CD, Docker entrypoint, envsubst, etc.)
window.__RUNTIME_CONFIG__ = {
  API_BASE_URL: "%API_BASE_URL%"
};
```

Step 2: Edit public/index.html as:

```
<script src="%PUBLIC_URL%/runtime-config.js"></script>
<div id="root"></div>
```

Step 3: Create a new file config.ts in src folder:

```
export const API_BASE_URL =
  (window as any).__RUNTIME_CONFIG__?.API_BASE_URL ??
"http://localhost:3000";
```

Step 3: Using API_BASE_URL whenever make an API call:

```
fetch(`${API_BASE_URL}/v1/users`);
```

5.1.2 Building React application and deploying the built app with docker:

Lab 3 has refered that if you want to deploy a web site, you have to install web server including iis, apache or nginx. Therefore, the built application with react need to be run on docker, there must be nginx image integrated with built code javascript into the your application image, then create container to run it.

Step 1: Build the react application using yarn build command

After executing yarn build, in the project tree, there is a folder build as the image:

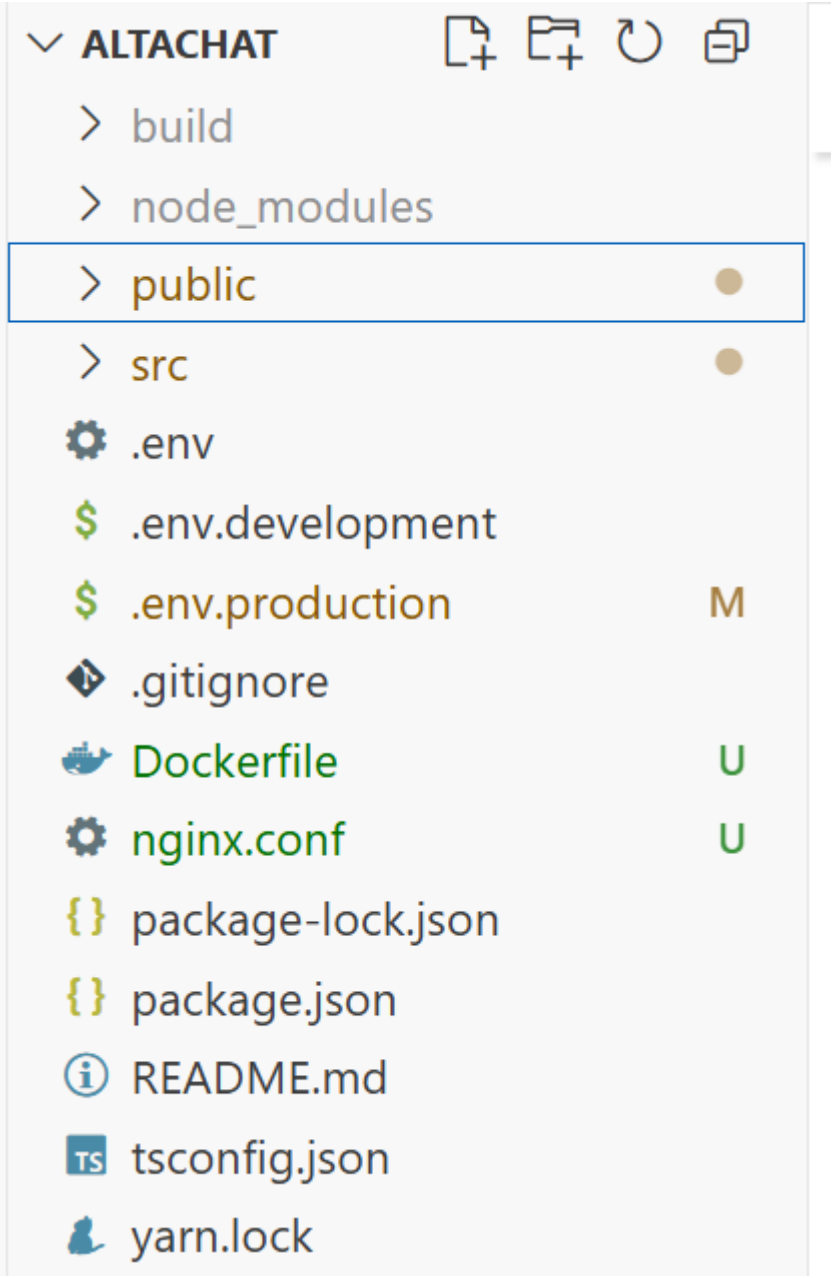

Image 5.1 Project tree structure with build folder created

Step 2: Create nginx.conf in the project folder:

```
server {
  listen 80;
  server_name _;

  root /usr/share/nginx/html;
```

```
  index index.html;

  # Cache hashed assets aggressively
  location ~* \.(js|css|png|jpg|jpeg|gif|svg|ico)$ {
    try_files $uri =404;
    access_log off;
    expires 1y;
    add_header Cache-Control "public, immutable";
  }

  # React Router fallback
  location / {
    try_files $uri /index.html;
  }
}
```

Step 3: Create Dockerfile in the project folder:

```
# Serve the already-built React app with Nginx

FROM nginx:alpine

COPY nginx.conf /etc/nginx/conf.d/default.conf

COPY build /usr/share/nginx/html

HEALTHCHECK CMD wget -qO- http://127.0.0.1/ || exit 1

EXPOSE 80
```

Step 4: copy build folder, Dockerfile, nginx.conf to ubuntu server.

Using scp -r command to copy build folder, Dockerfile and nginx.conf to ubuntu server.

Step 5: build the image and create container associated with this image and run the container.

5.2 Deploy .Net Core Restful API and integrated with database:

If you deploy DBMS as an image and your .net core api as another image. These images run on different containers. In order to integrate your api with database, you should put the two containers on the same docker network.

To show all available docker networks: **docker network ls**

The command: **docker network inspect appnet** will list all connected containers to the network **appnet**

To add an existing container to a network:

docker network connect <network_name> <container_name>

Example: add containers NvcmisCoffeeBk to network nvcnet by the command:

docker network connect nvcnet NvcmisCoffeeBk

When using command: docker run -d –name <container_name> -p outside_port:inside_port *-e ASPNETCORE_URLS=http://+:inside_port*

-e ASPNETCORE_URLS=http://+:inside_port to indicate that the container run with http rather than https.

Exercise: Lecturer provides three components:

- Backup PostGreSQL database file
- Built version of React Application
- Built version of .Net Core Restful API Application

Students have to deploy all of the three components with three images and associated containers. Finally, run the web on other machines in LAN.