Virtual Machine Setup

Logging in through SSH without a password

Use the following command on a terminal to generate a SSH key. Replace they key name with something of your choice and, when prompted, create a passphrase.

ssh-keygen -f ~/.ssh/keyname -o -t ed25519 -a 200

This command generates a key using the Ed25519 algorithm which is pretty fast and secure. Here is a good summary of what you should be using, either way.

Use the following command to copy the public key to the remote computer. Replace the key name with the correct one, as well as username to login as and IP address of remote computer.

ssh-copy-id -i ~/.ssh/keyname <username>@<IP>

Enter the user password when prompted.

- If the system keeps asking for a password, using either root or with sudo privileges, check SSH config at /etc/ssh/sshd_config:

  PubkeyAuthentication should be uncommented and set to “yes”.

- To disable sign in with password, on the same file edit the following line:

  PasswordAuthentication can be uncommented, if so, and set to “no”.

  NOTE: Test beforehand that you are able to login without passwords to -at least- the root/sudo user, to avoid being locked out!

- After editing sshd_config, run sudo systemctl restart ssh

Setting up the web app

If we want to test the app in development mode, we need to install NPM using an account with sudo privileges. The commands to do so from NodeSource for Node v16.x are:

```
curl -fsSL https://deb.nodesource.com/setup_16.x | sudo -E bash -
sudo apt-get install -y nodejs
```

We can copy the files into a folder in our VM either by cloning the repo from GitHub or using the scp command from our computer. The command is this:

```
scp -r /local/path/ <remote-user>@<ip-address>:/remote/path/
```

Setting up the web server with nginx

In our project we are using docker to run an nginx container.

First, we need to build our app for production, which will be stored in a directory of the same name in our application folder. This can be done either in our own computers or straight in the VM. The command for doing so is:

```
npm run build
```

If done in our computers, the "/build" folder in our app must be copied into the VM. We use the same command as before:

```
scp -r /local/path/ <remote-user>@<ip-address>:/remote/path/
```

Using Docker

Install docker: https://docs.docker.com/engine/install/ubuntu/
Docker commands might need sudo privileges.

Scripts for installing docker and running/stopping the containers can be found inside the "/scripts" folder. They were written based on the following documentation by CSC: https://docs.csc.fi/support/tutorials/env-guide/linux-bash-scripts/

- Alternative 1, using bind mounts.

Nginx can be run with the following command:

```
docker run --name <container-name> -v
/static/content:/usr/share/nginx/html:ro -d -p 8686:80 nginx
```

Flags:

-p exposes the internal port 80 to external 8686 (in the example).

- Alternative 2, using a Dockerfile:

A Dockerfile is provided in the root directory of our app. Once we have it, along with the production build of our app, it is built from the same directory with:

```
docker build -t <build-name> .
```

For example: `docker build -t fe-app-build .`

Then run with:

```
docker run --name <container-name> -d -p 80:80  <build-name>
```

Here's another usable example:

```
docker run --name frontend-app -d -p 80:80 fe-app-build
```

- Custom config can be added with a Dockerfile in /host/path/:

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
```

Then built and run.

Example config file below:

```
server {
        listen 80 default server;
        root /var/www/html;
        index index.html index.htm;
        server_name <front-end-server-url>;
        location / {
                proxy_pass http://localhost:3000;
        }
        location /json {
                proxy_pass http://<backend-url>;
        }

}
```

- Other commands.

Docker has a pretty good documentation at https://docs.docker.com/ but these are some commands which are useful while we deploy our app:

docker container ls – lists running docker containers.

docker stop/restart/rm – stops, restarts or removes a container.

docker image ls – lists available images to run containers with.

docker image rm IMAGE img_id – removes an image from the system (the ID is obtained from the list)