Deploying Node.js web app to AWS EC2 with Docker

Steps:

1. Launch AWS ubuntu ec2 instance.

2. Clone the repository from GitHub to your ubuntu server using command:

git clone <repository_URL>

```
ubuntu@ip-172-31-7-126:~/node-project$ git clone https://github.com/sayalishewale/node-js-project.git
Cloning into 'node-js-project'...
Username for 'https://github.com': sayalishewale
Password for 'https://sayalishewale@github.com':
remote: Enumerating objects: 140, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (69/69), done.
remote: Total 140 (delta 35), reused 32 (delta 8), pack-reused 62
Receiving objects: 100% (140/140), 7.28 MiB | 16.35 MiB/s, done.
Resolving deltas: 100% (39/39), done.
ubuntu@ip-172-31-7-126:~/node-project$ ls
node-js-project
ubuntu@ip-172-31-7-126:~/node-project$ cd node-js-project/
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$ ls
README.md  app.js  client-auth  db  logfile.log  middleware  package-lock.json  package.json  routes  server.js  startup  uncaughtExceptions.log
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$
```

3. Create a Dockerfile

```
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$ vi Dockerfile
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$ cat Dockerfile
FROM node:alpine
WORKDIR /app
COPY package.json /app
RUN npm install
COPY . /app
CMD ["npm", "start"]
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$
```

The first thing we need to do is define from which image we want to build from. Here we will use the node:alpine image available from the Docker Hub.

FROM node:alpine

Next we create a directory to hold the application code inside the image, this will be the working directory for your application:

WORKDIR /app

Then copy package.json file to the working directory using COPY instruction.

COPY package.json /app

This image comes with Node.js and NPM already installed so the next thing we need to do is to install your app dependencies using the npm binary.

RUN npm install

Copy your app's source code inside the Docker image, use the COPY instruction:

COPY . /app

define the command to run your app using CMD which defines your runtime.

CMD [“npm”,”start”]

4. Build an Image using Dockerfile

To build an image using Dockerfile, Go to the directory that has your Dockerfile and run the following command

docker build -t <image-name> .

Example: docker build -t node-app:v1 .

By using **docker images** command we can see the list of images

docker images

```
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$ docker build -t node-js-project .
Sending build context to Docker daemon  22.03MB
Step 1/7 : FROM node:alpine
 ---> 17299c0421ee
Step 2/7 : WORKDIR /app
 ---> Running in 2e4656dd04e5
Removing intermediate container 2e4656dd04e5
 ---> cf9e890f0b1d
Step 3/7 : COPY package.json /app
 ---> aac354e7bbec
Step 4/7 : RUN npm install
 ---> Running in 93b7194516c8

added 200 packages, and audited 201 packages in 17s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 9.2.0 -> 9.3.1
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v9.3.1>
npm notice Run `npm install -g npm@9.3.1` to update!
npm notice
Removing intermediate container 93b7194516c8
 ---> 13bdab4a51eb
Step 5/7 : COPY . /app
 ---> a4a695302e2b
Step 6/7 : EXPOSE 8000
 ---> Running in 18aed359751b
Removing intermediate container 18aed359751b
 ---> 28d7f93c34e0
Step 7/7 : CMD ["npm","start"]
 ---> Running in 7ed3a9221a4d
Removing intermediate container 7ed3a9221a4d
 ---> e31b335da881
Successfully built e31b335da881
Successfully tagged node-js-project:latest
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED         SIZE
node-js-project   latest    e31b335da881   6 seconds ago   241MB
node              alpine    17299c0421ee   8 days ago      176MB
nginx             alpine    c433c51bbd66   8 days ago      40.7MB
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$
```

5. Run the image to create a container

Running your image with -d runs the container in detached mode, leaving the container running in the background. The -p flag redirects a public port to a private port inside the container and –name flag is for container name.
Run the image you previously built:

docker run -d - -name <container-name> -p 8000:8000 <image-name>

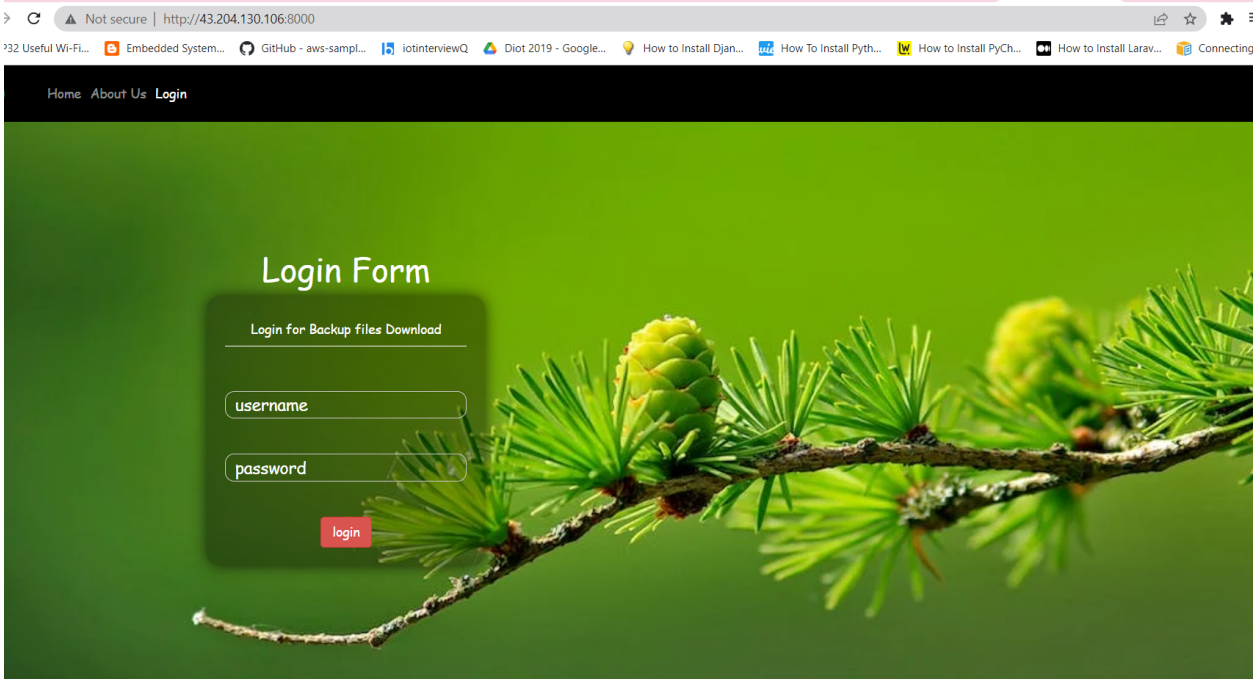
docker ps command is used to list all the containers.

```
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$ docker run -d --name node-project -p 8000:8000 node-js-project:latest
371d1fd7c96b89882ae10c283beb869757a5feda71ae32dbfb3bc56143643f5b
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS         PORTS                                       NAMES
371d1fd7c96b   node-js-project:latest   "docker-entrypoint.s…"   4 seconds ago   Up 3 seconds   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp   node-project
645a1a3864ad   nginx:alpine             "/docker-entrypoint.…"   6 hours ago     Up 6 hours     80/tcp                                      nginx-container
d5c0a8262c6f   nginx:alpine             "/docker-entrypoint.…"   6 hours ago     Up 6 hours     80/tcp                                      festive_rubin
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$
ubuntu@ip-172-31-7-126:~/node-project/node-js-project$
```

6. Go to your ec2 instance security groups and check whether port 8000 is opened or not. If not, then edit the security group and add the port no. 8000 to the security group.

7. Open the public IPv4 address of your ec2 instance with port no. 8000

http://Public-ip:8000



Thank you for reading! Hope it helps!