

Project Number - 1

Deploying Dynamic Web Application

2000031095

Section13

Cloud and Devops

Services I have used are



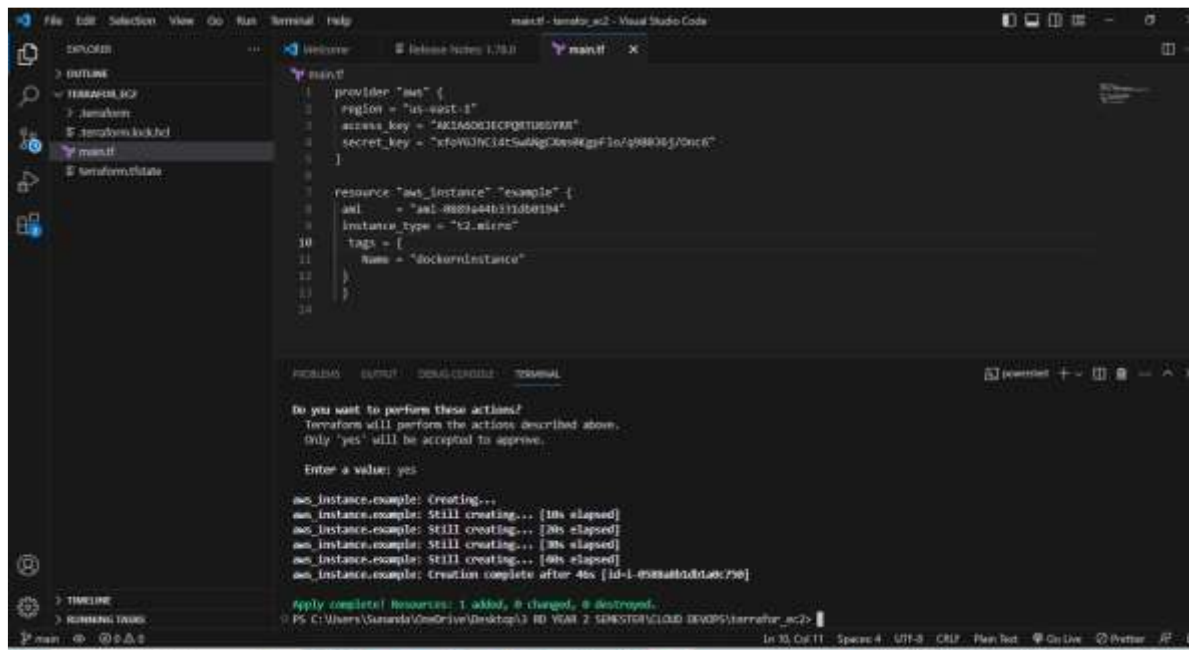
For GIT CLONE for ec2 automation EC2 Instances for computing

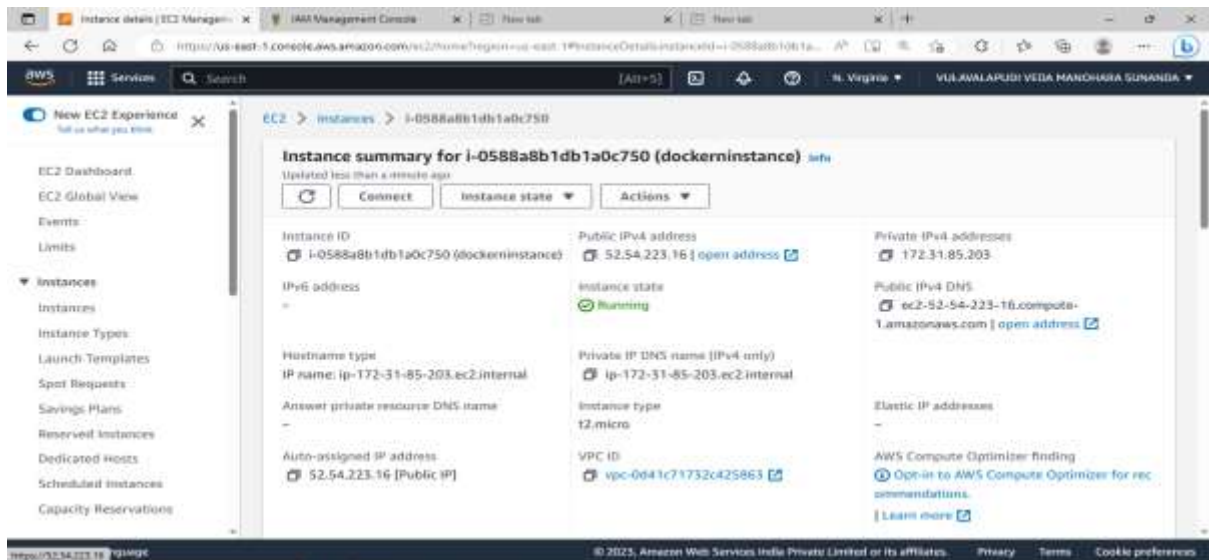
created by terraform



Docker Installed on EC2 Instance

Creating IAM user for Terraform automation





step1: create ec2 instance with linux ami

step2: use these commands

****Update the installed packages and package cache on your instance.**

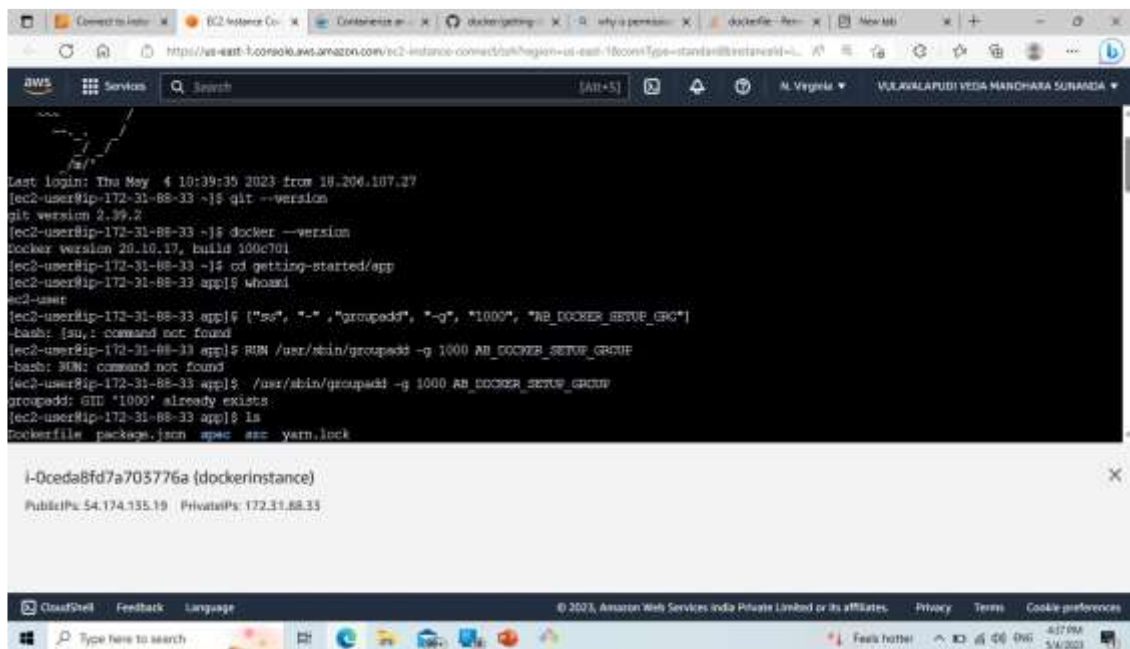
sudo yum update -y

****Install the most recent Docker Community Edition package.**

sudo yum install docker

****Check docker version docker version**

docker --version



```
Last login: Thu May 4 10:39:35 2023 from 18.204.107.27
[ec2-user@ip-172-31-88-33 ~]$ git --version
git version 2.39.2
[ec2-user@ip-172-31-88-33 ~]$ docker --version
Docker version 20.10.17, build 100c701
[ec2-user@ip-172-31-88-33 ~]$ cd getting-started/app
[ec2-user@ip-172-31-88-33 app]$ whoami
ec2-user
[ec2-user@ip-172-31-88-33 app]$ ["su", "-s", "groupadd", "-g", "1000", "AB_DOCKER_SETUP_GROUP"]
-bash: [su,: command not found
[ec2-user@ip-172-31-88-33 app]$ RUN /usr/sbin/groupadd -g 1000 AB_DOCKER_SETUP_GROUP
-bash: RUN: command not found
[ec2-user@ip-172-31-88-33 app]$ /usr/sbin/groupadd -g 1000 AB_DOCKER_SETUP_GROUP
groupadd: GID '1000' already exists
[ec2-user@ip-172-31-88-33 app]$ ls
Dockerfile package.json apoc src yarn.lock
```

i-0ceda8fd7a703776a (dockerinstance)
Public IP: 54.174.135.19 | Private IP: 172.31.88.33

****Add the ec2-user to the docker group so you can execute Docker commands without using sudo.**

****Exit the terminal and re-login to make the change effective**

```
sudo usermod -a -G docker ec2-user
exit
```

****Enable docker service**

```
sudo systemctl enable docker
```

****Start docker service**

```
sudo systemctl start docker
```

Check the Docker service.

```
sudo systemctl status docker
```

****Get docker details**

docker info

****Get docker command details**

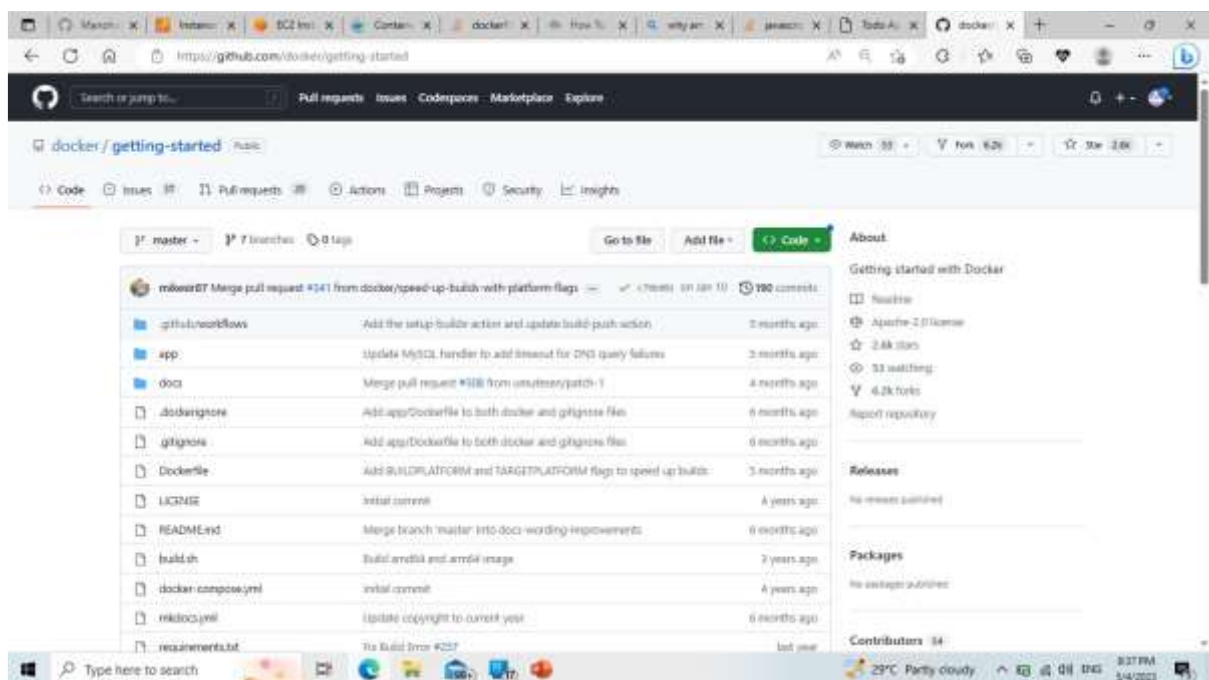
docker

****Get help on specific command**

docker --help

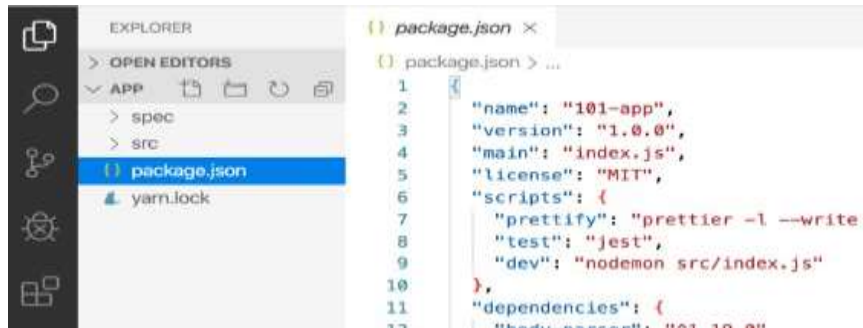
****Installing Git**

sudo yum install git



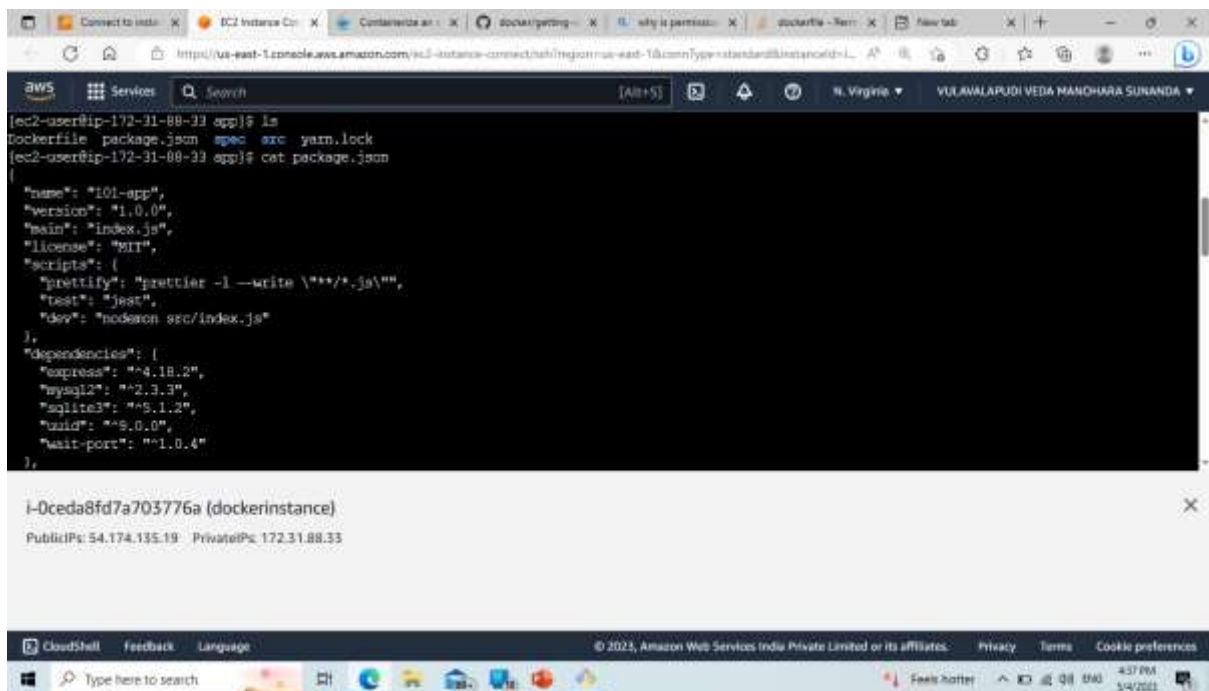
Working on project

git clone <https://github.com/docker/getting-started.git>
as below settings



The image shows a VS Code interface. On the left, the Explorer sidebar shows a file tree with 'APP' expanded, containing 'spec', 'src', 'package.json', and 'yarn.lock'. 'package.json' is selected. On the right, the editor shows the content of 'package.json'.

```
1  {} package.json > ...
2
3  "name": "101-app",
4  "version": "1.0.0",
5  "main": "index.js",
6  "license": "MIT",
7  "scripts": {
8    "prettify": "prettier -l --write \"**/*.js\"",
9    "test": "jest",
10   "dev": "nodemon src/index.js"
11 },
12 "dependencies": {
13   "express": "4.18.2",
14   "mysql2": "2.3.3",
15   "sqlite3": "5.1.2",
16   "uuid": "9.0.0",
17   "wait-port": "1.0.4"
18 }
```



The image shows an AWS CloudShell terminal window. The terminal output shows the user running 'ls' and 'cat package.json' commands. The output of 'cat package.json' is displayed. Below the terminal, the instance details for 'i-0ceda8fd7a703776a (dockerinstance)' are shown, including Public IP and Private IP.

```
[ec2-user@ip-172-31-88-33 app]$ ls
Dockerfile package.json spec src yarn.lock
[ec2-user@ip-172-31-88-33 app]$ cat package.json
{
  "name": "101-app",
  "version": "1.0.0",
  "main": "index.js",
  "license": "MIT",
  "scripts": {
    "prettify": "prettier -l --write \"**/*.js\"",
    "test": "jest",
    "dev": "nodemon src/index.js"
  },
  "dependencies": {
    "express": "4.18.2",
    "mysql2": "2.3.3",
    "sqlite3": "5.1.2",
    "uuid": "9.0.0",
    "wait-port": "1.0.4"
  }
}
```

i-0ceda8fd7a703776a (dockerinstance)
PublicIPs: 54.174.135.19 PrivateIPs: 172.31.88.33

```
cd /path/to/app [cd getting-started/app]
```

```
touch Dockerfile
```

```
nano Dockerfile
```

```
# syntax=docker/dockerfile:1
```

```
FROM node:18-alpine
```

```
WORKDIR /app
```

```
COPY . .
```

```
RUN yarn install --production
```

```
CMD ["node", "src/index.js"] -> to run index.js file
```

```
EXPOSE 3000 -> at port number 3000
```

```

}
[ec2-user@ip-172-31-88-33 app]$ cat Dockerfile
# syntax=docker/dockerfile:1

FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000

[ec2-user@ip-172-31-88-33 app]$ docker build -t getting-started .
Sending build context to Docker daemon 4.62MB
Step 1/6 : FROM node:18-alpine
18-alpine: Pulling from library/node
f56be85fc22e: Pull complete
931b0e865bc2: Pull complete
60542df8b663: Pull complete
062e26bc2446: Pull complete
Digest: sha256:1ccc70acda680aa4ba47f53e7c40b2d4d6892de74817128e0662d32647dd7f4d

i-0ceda8fd7a703776a (dockerinstance)
PublicIPs: 54.174.135.19 · PrivateIPs: 172.31.88.33

CloudShell Feedback Language © 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences
Type here to search
```

If “build” unsuccessful then use command

/usr/sbin/groupadd -g 1000 AB_DOCKER_SETUP_GROUP

and continue to work on same “build” command

```

[ec2-user@ip-172-31-88-33 app]$ docker build -t getting-started .
Sending build context to Docker daemon 4.62MB
Step 1/6 : FROM node:18-alpine
18-alpine: Pulling from library/node
f56be85fc22e: Pull complete
931b0e865bc2: Pull complete
60542df8b663: Pull complete
062e26bc2446: Pull complete
Digest: sha256:1ccc70acda680aa4ba47f53e7c40b2d4d6892de74817128e0662d32647dd7f4d
```



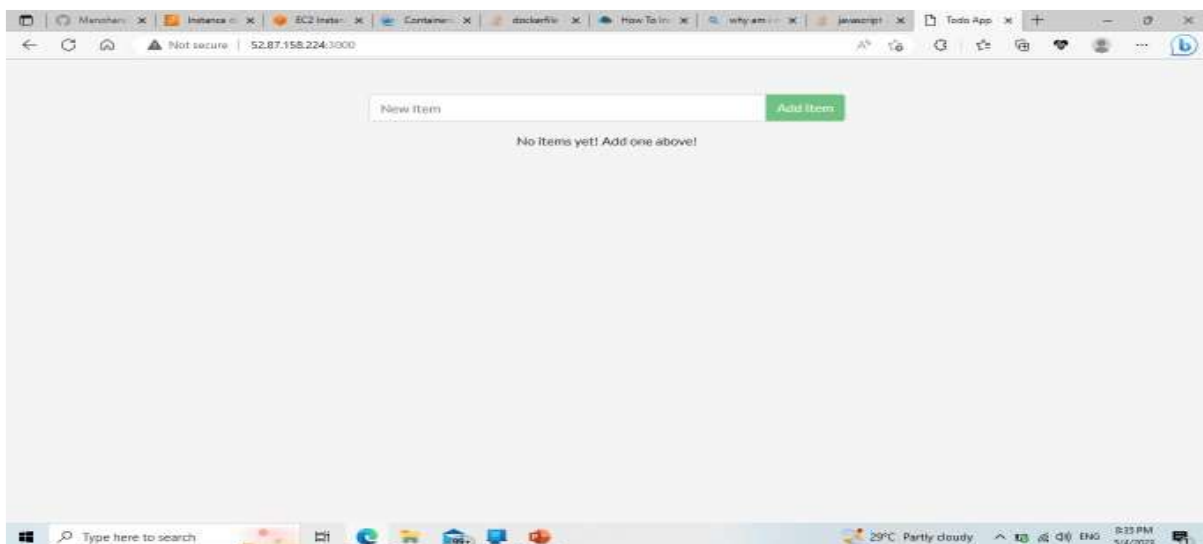
```
Step 5/6 : CMD ["node", "src/index.js"]
--> Running in 30fb467b4324
Removing intermediate container 30fb467b4324
--> 8c6179d1292f
Step 6/6 : EXPOSE 3000
--> Running in 4c26c287e921
Removing intermediate container 4c26c287e921
--> 5b230dcdf19b
Successfully built 5b230dcdf19b
Successfully tagged getting-started:latest
(ec2-user@ip-172-31-88-33 app) $ docker run -dp 3000:3000 getting-started
a5440aa8e844b0928a591a26034a4c6375b333384536f5c68f7a4b1679d
(ec2-user@ip-172-31-88-33 app) $
(ec2-user@ip-172-31-88-33 app) $
(ec2-user@ip-172-31-88-33 app) $ docker image ls
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
getting-started      latest          5b230dcdf19b    6 minutes ago   265MB
node                 18-alpine       6f44d13d1258    2 weeks ago     175MB
(ec2-user@ip-172-31-88-33 app) $
```

i-0ceda8fd7a703776a (dockerinstance)
PublicIPs: 54.174.155.19 PrivateIPs: 172.31.88.33

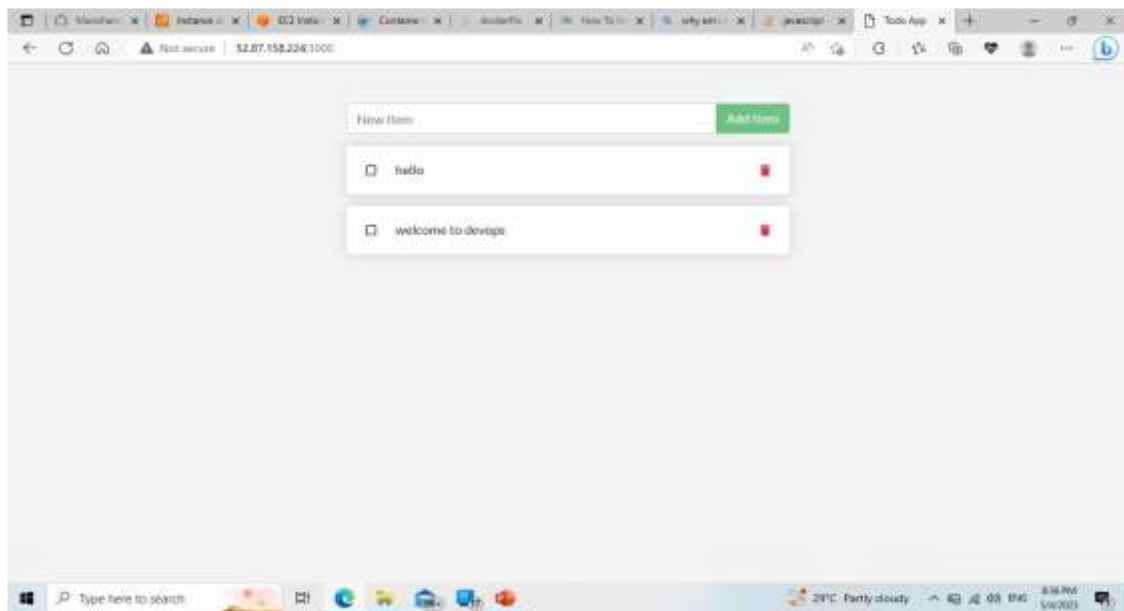
Work with the tags

docker run-dp 3000:3000 gettingstarted

The IP Addresses of the may be varied as I have terminated an instance due to billing issue and created the same in other instance with same procedure



Dynamic addition of Items in “POST METHOD”

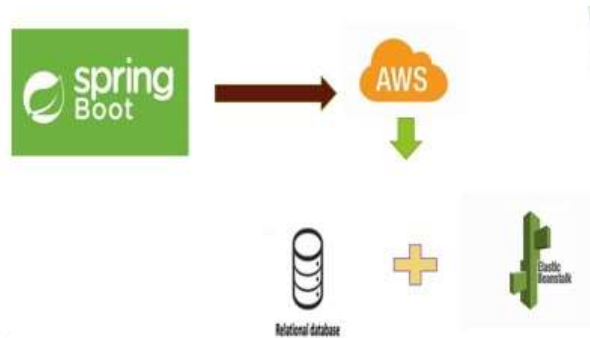


Project Number 2

**Deploying Spring boot Web Application ,
Automation via AWS Cloud**

Services used are

Git bash to push my files to my Git account +
Elastic Beanstalk +
RDS for configuring Application backend



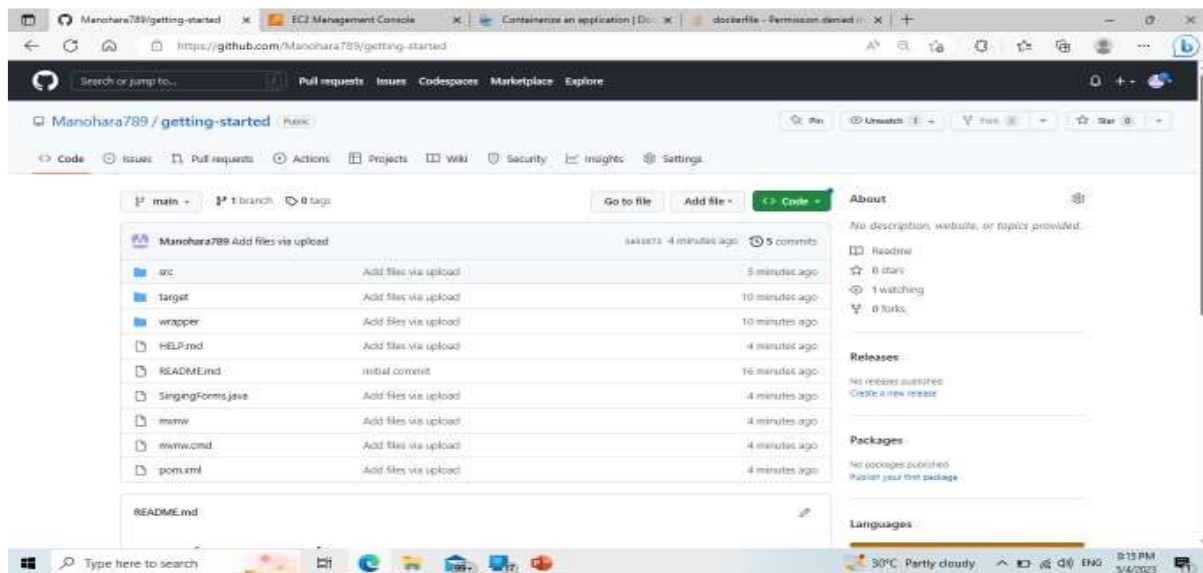
- Step1 : create and SD Project of your own choice.

We have created Indian Culture Information System Project

Tools we used are : Spring boot, JavaScript for frontend

Hibernate, JPA Mapping, MySQL for backend

This is my git with spring boot so you can also work on this in the same way



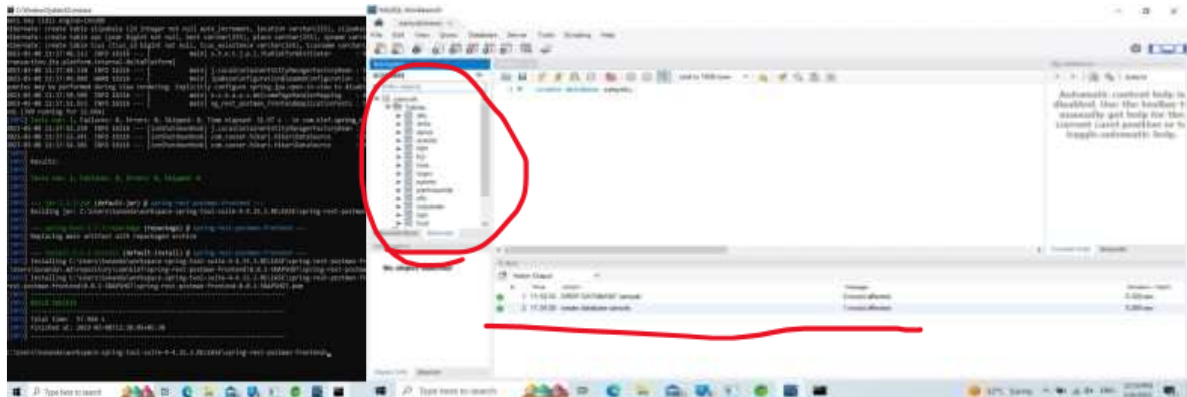
Output in the localhost machine port 8080



Steps I have followed to automate my application via EBS instance

Step1 : Create an RDS with required username, password and using its endpoint create a connection

Step2 : After establishing connection you **copy endpoint to your “project application.properties”** and open command prompt from the project location and enter **“mvn clean install”** to transfer tables to RDS database



Step 3 : Apply “mvn clean package” to get a snapshot of your project and then upload it to your EBS by selecting JAVA as your runtime and

Edit : 1. Environment variables PORT_NUMBER : 5000

2. Database Settings USERNAME : root

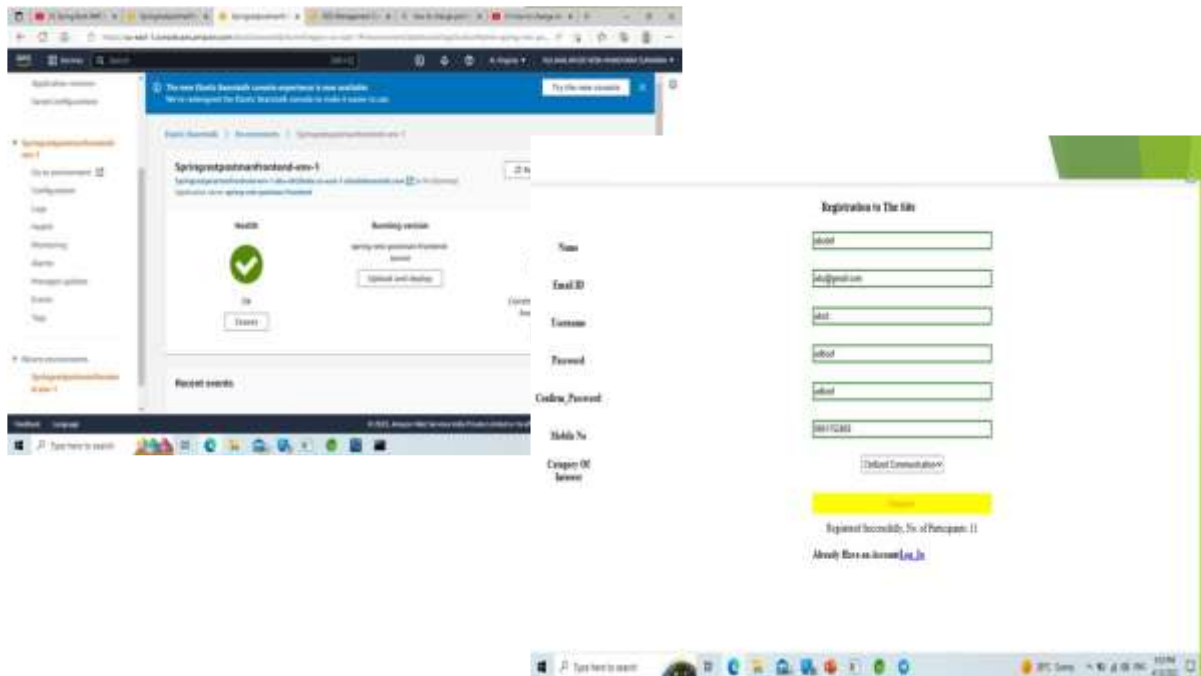
PASSWORD

Step 4 : Edit Auto-created EC2 Instance Security group inbound rules to allow backend data also

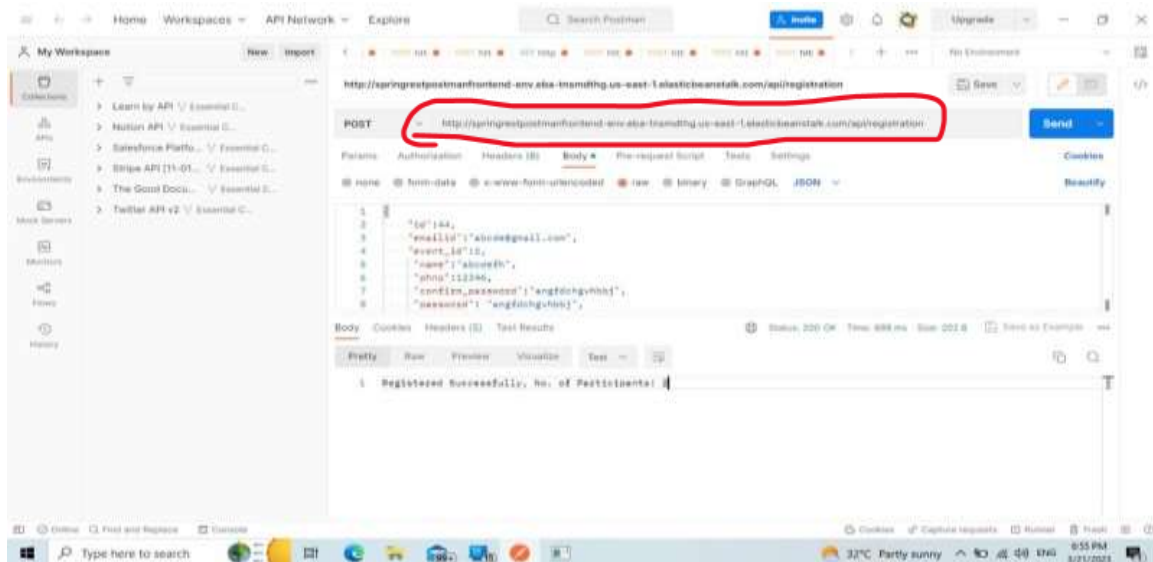
Step 5 : Take the help of EBS Application -> Environment URL and paste it in the POST MAN to check the request is working or not

► Step 6 : We tried the registration page URL “POST MAPPING” and finally it worked.

► We then analyzed that it has been hosted successfully.



Post mapping into the Registration page been successful. Observe the URL in POSTMAN App



Conclusion :

I have done 2 projects

1. Using Devops Services like - **Git, Terraform, AWS Client Manageable Compute Services, Docker on EC2**
2. Using AWS Cloud Services like - Elastic Beanstalk, RDS [MySQL]

Observations :

These **both** belongs to **Deployment of Dynamic Web Application** but they differ in Client-Side Automation.

This Client-Side Automation **can be possible using services like Terraform, Docker Orchestration etc...**

So, to show this difference **I have done 2, 3-Tier Architecture Projects.**

THANK YOU

