Deploying a Movie Listing Website on AWS Cloud

GROUP-6

TEAM MEMBERS:

- G.VAMSHIDHAR
- P.S.S.PRADEEP
- CH.SAI POOJITHA
- K.D.P.N.JYOTHI
- K.BHANU
- J.N.V.S.MAHALAKSHMI
- V.RAJA
- A.P.S.S.BHARGAVI

CONTRIBUTION OF TEAM MEMBERS

S.No	ROLES FOR DEVOPS_CAPSTONE	MEMBERS
1	Connecting the Server with Atlas MongoDB.	Vamshidhar, Sai Pradeep, Sai Poojitha
2	Creating IAM user and S3.	Jyothi, Mahalakshmamma
3	Configuring the Application Code with S3-multer.	Raja, Jyothi, Vamshidhar, Sai Pradeep
4	Containerization of the code using Docker file.	Bhanu, Sai Poojitha, Vamshidhar
5	Deploying server on EC2 using Docker.	Sai Pradeep, Raja, Bhargavi
6	Deploying client on EC2 using Docker.	Bhanu, Mahalakshmamma, Bhargavi
7	Creation of target group and Load Balancer.	Vamshidhar, Bhanu, Sai Poojitha
8	Creating AWS Deployment Diagram.	Vamshidhar, Sai Pradeep
9	Preparing Project Documentation.	Jyothi, Mahalakshmamma, Bhargavi, Raja

- **GitHub URL:**

  https://github.com/pradeep-pulaparthi/Capstone_project

*****This is the URL of the repository which contains the modified codes of the front end and back end.

- **Frontend URL:**

  http://44.204.124.118:6200/

- **Backend URL:**

  http://44.193.213.43:6200/

- **DNS URL:**

  http://myloadbalancer-d6377885a348cd6f.elb.us-east-1.amazonaws.com/

PURPOSE OF THE PROJECT:

The aim of the project is to deploy a movie listing web application into the cloud so that it will allow users to connect from anywhere and also allows them to upload movie posters and details.

- The website uses ReactJS at the front end, NodeJS at the back end, and MongoDB as the database.
- The main objective of the project is to provide a platform for users to explore and discover new movies, as well as share their favorite movies with others.
- The website is designed to be user-friendly and interactive, to help users easily find the movies they are interested in.
- By deploying the website on the AWS cloud infrastructure with proper scaling, the project aims to provide a seamless and reliable experience for users, even during periods of high traffic.
- One of the main advantages of this project is that it stores the data in multiple locations. So even when there is a loss of data at one place all the information will be made available at the other locations.
- We can find that the data is being stored in the MongoDB database, S3 buckets, and docker.

SCOPE OF THE PROJECT:

The scope of the project includes the development of a movie listing website that allows users to upload movie details and images.

- **Objectives:** The project's main objective is to deploy an existing "Movie listing" website on the cloud infrastructure of Amazon Web Services and ensure proper scaling.
- **Deliverables:** The deliverables of the project will be having a fully functional and scalable "Movie listing" website running on AWS infrastructure. The website should be deployed with all its components including the frontend, backend, database, and S3 bucket for images.

The project involves designing and implementing the following features:

- **Backend configuration:** The backend server is configured to use MongoDB as a database and multer for file upload.
- **AWS deployment:** The website is deployed on the AWS cloud infrastructure using S3, and EC2, and proper scaling has been established using the load balancers.
- **Image storage:** The website uses AWS S3 to store the images uploaded by the user.
- **Movie upload:** Users can upload their own movie details and images to the website.
- **Movie listing:** Users can view a list of movies uploaded by other users.

TECHNOLOGIES USED IN THIS PROJECT:

The following are the technologies that have been used in the project:

1)Frontend development:

The user interface is designed by using React JS for enhancing flexibility and performance.

2)Backend development:

NodeJS has been used for developing the back end of the movie listing website. NodeJS has the ability to handle multiple requests simultaneously.

3)Database management:

MongoDB has been used as the database for storing movie details, user information, and other relevant data. MongoDB is a NoSQL database that is highly flexible.

4)File Upload:

The multer-s3 package extends the functionality of multi by providing the ability to store uploaded files directly to Amazon S3 (Simple Storage Service) instead of storing them locally on the server.

5)Cloud infrastructure:

We have used Amazon Web Services (AWS) for deploying the movie listing website on the cloud infrastructure with proper scaling. We have used the following services of AWS:

- **EC2:** EC2 stands for Elastic Compute Cloud. It is a web service that helps to launch and manage virtual machines known as instances in the cloud. In this project, we used EC2 to host the “Movie Listing” website.
- **S3:** S3 stands for Simple Storage Service. It is a highly scalable, durable, and secure object storage service. S3 is used to store and retrieve any amount of data from anywhere on the web. The images related to the website will be stored in the S3 bucket whenever the movie details have been uploaded.
- **ELB:** ELB stands for Elastic Load Balancer. ELB automatically distributes the incoming traffic across multiple Amazon EC2 instances, helping to improve the availability and scalability of applications running on AWS. In our project, we have used a Network load balancer for the distribution of traffic to multiple EC2 instances.

MAJOR TOOLS AND SERVICES UTILIZED IN THIS PROJECT:

- **Visual Code:** Visual Studio Code, commonly referred to as VS Code, is a free, open-source code editor developed by Microsoft. It is a lightweight and powerful tool for coding, debugging, and building applications. We have used it to write the codes of the front end and back end.
- **MongoDB:** A NoSQL database used for storing movie details, user information, and other relevant data.
- **ReactJS:** A JavaScript library used for building user interfaces and developing the front end of the movie listing website.
- **NodeJS:** A JavaScript runtime environment used for developing the backend of the movie listing website.
- **Figma:** Figma is a cloud-based design and prototyping tool that allows designers and teams to collaborate on designing user interfaces, web pages, and mobile applications. It is used to design the architecture that has been followed.
- **GitHub:** GitHub is a web-based platform for version control and collaboration that allows developers to work together on software projects. It provides a range of features for software development, including code hosting, project management, issue tracking, and collaboration tools. In our project, it is used to host the front-end and back-end code of the movie listing website.
- **Docker:** Docker has been used to containerize the application by creating Docker images. We have built Docker images for both the front end and back end and they are made to run in the instances.
- **GIT**: Git is a version control system that is widely used for software development. Git stores the code and its history in a repository and allows developers to make changes to the code by creating new branches and committing changes to those branches.

IMPLEMENTATION:

1)Storing images:

Amazon S3(Simple Storage Service) provides a simple web services interface that can be used to store and retrieve data from anywhere on the web.

In this project, we will be using Multer-s3. Multer-s3 simplifies the process of uploading files to S3 by providing an easy-to-use middleware that integrates with your web application. It handles the process of creating and configuring an S3 bucket, setting up permissions and access control, and uploading files to S3.

*Open the AWS Management console and go to the IAM service. We need to create a new IAM user by providing the necessary permission to access the S3 service. Also, create an access key that will provide us with the access key and secret access key.

*We will be using these access key and secret access key in the server’s .env file.

*Go to S3 service and click on Create bucket.

*Give the bucket name and select a region.

*In the object ownership enable the ACLs.

*Uncheck block all the public access and enable the bucket versioning. Click on create a bucket.

*After creation of the bucket select the bucket and click on the “Permissions” tab. Edit the bucket policy that allows your backend server to access the bucket.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowUserToReadWriteObjects",
         "Effect": "Allow",
         "Principal": {
            "AWS": "arn:aws:iam::<ACCOUNTID>:user/<IAM USERNAME>"
         },
         "Action": [
            "s3:GetObject",
            "s3:PutObject"
         ],
         "Resource": "<BUCKETNAME>"
      }
   ]
}
```

*Modify the above code by giving the AWS account Id, IAM user name, and bucket name.

2) Cloud Database:

We will be using Atlas MongoDB cloud infrastructure to take the local database into the cloud. Atlas is a fully-managed cloud database service provided by MongoDB, Inc. that offers a global database deployment on AWS, Azure, and Google Cloud Platform.

*In the services of AWS search for MongoDB Atlas in the marketplace section.

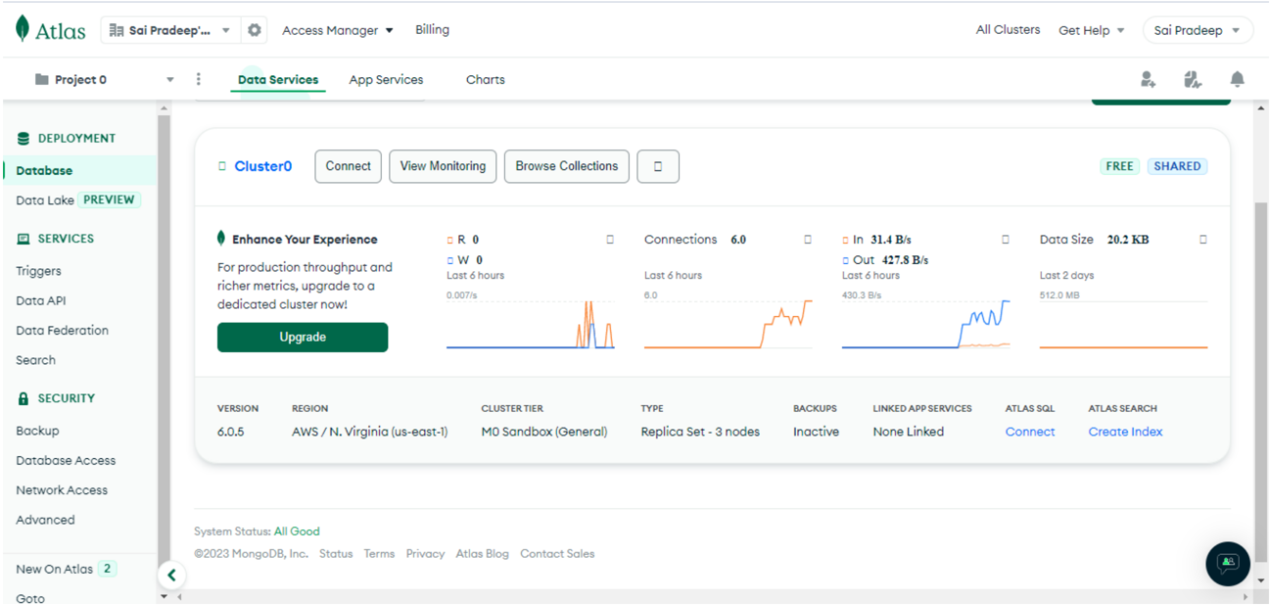
*Click on MongoDB Atlas(pay-as-you-go) which will redirect you to the MongoDB atlas page.

*If you don’t have an account already click on subscribe and then signup.

*Otherwise, log in to the MongoDB Atlas dashboard.

*Click on Build cluster and then provide the details required such as region, provider, cluster name, and the other settings required for your cluster and click on Create Cluster.

*Once your cluster is created, you can click on the "Connect" button to get the connection string needed for your application to connect to the database.



VERSION	REGION	CLUSTER TIER	TYPE	BACKUPS	LINKED APP SERVICES	ATLAS SQL	ATLAS SEARCH
6.0.5	AWS / N. Virginia (us-east-1)	M0 Sandbox (General)	Replica Set - 3 nodes	Inactive	None Linked	Connect	Create Index

*Select the option vs code in the column Access your data through and click on vs code.

*There will be a connection string copy it and open the vs code.

*In the index.js file of the server section, paste the above connection string in the Mongoose. connection().

*This helps us to connect the MongoDB using vs code.

3)Back-end deployment:

In this server, we have to install react-scripts, and then give the **npm start** command.

npm install react-scripts

npm start

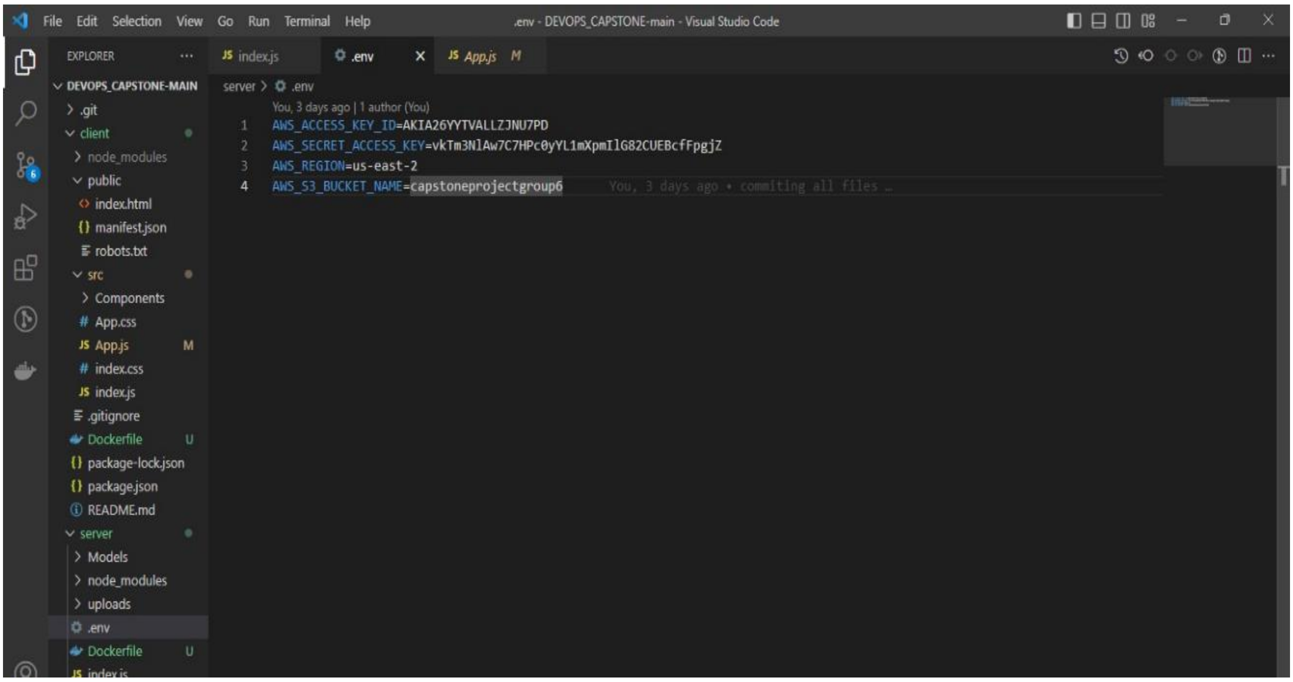
***In** the server, we have a .env file that needs to be modified with our access key, secret key, and bucket name.

AWS_ACCESS_KEY_ID=AKIA26YYTVALLZJNU7PD

AWS_SECRET_ACCESS_KEY=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

AWS_REGION=us-east-2

AWS_S3_BUCKET_NAME=capstoneprojectgroup6



```
AWS_ACCESS_KEY_ID=AKIA26YYTVALLZJNU7PD
AWS_SECRET_ACCESS_KEY=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
AWS_REGION=us-east-2
AWS_S3_BUCKET_NAME=capstoneprojectgroup6
```

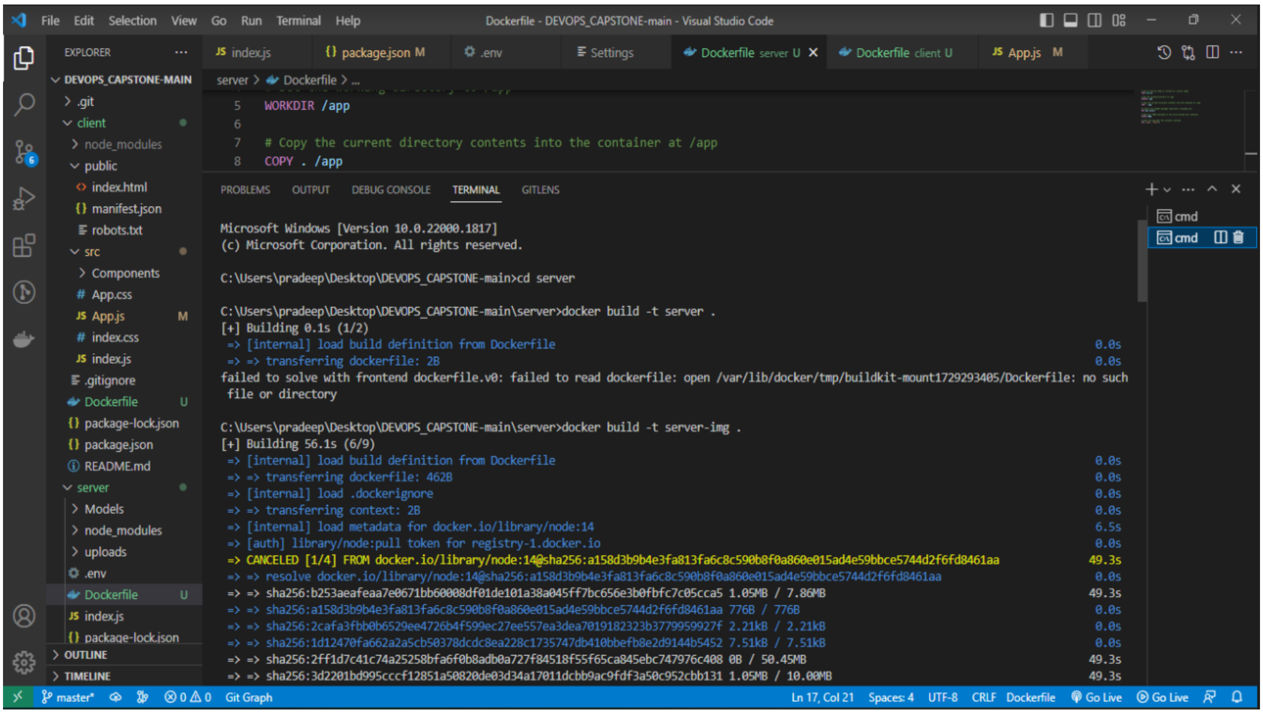
*We are using docker to deploy our backend application. We need to create a Docker file for the backend and build the image. Then, we will create an EC2 instance and install Docker on it. We will then copy the Docker image to the EC2 instance and run it. We will attach an Elastic IP to this instance so that the IP address remains static.

*Create a Docker file for the backend. This file contains a port number which will make it available to the world outside the container.

```
# Use an official Node.js runtime as a parent image

FROM node:14

# Set the working directory to /app

WORKDIR /ap

# Copy the current directory contents into the container at /app

COPY . /app

# Install any needed packages specified in package.json

RUN npm install

# Make port 5000 available to the world outside this container

EXPOSE 5000

# Start the app when the container launches

CMD ["npm", "start"]
```

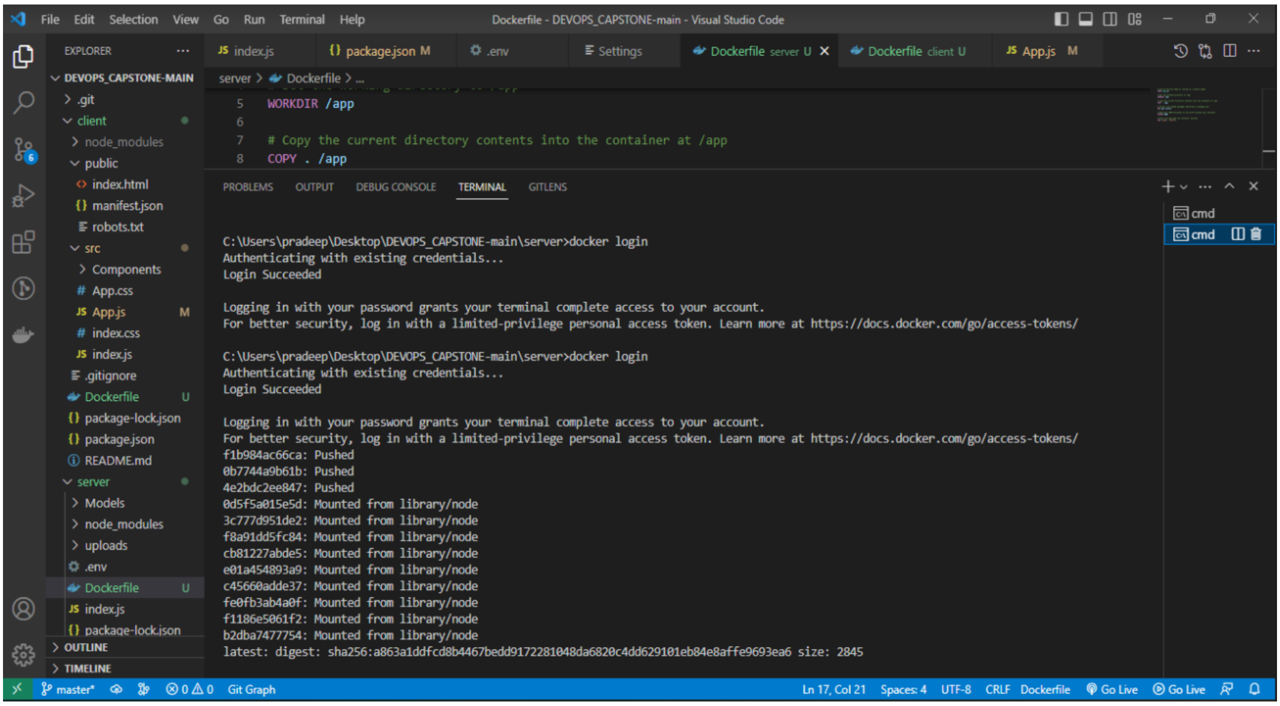
*Build a docker image named “server-img” in the directory containing the Docker file.

docker build -t server-img .



```
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app
```

```
Microsoft Windows [Version 10.0.22000.1817]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pradeep\Desktop\DEVOPS_CAPSTONE-main>cd server

C:\Users\pradeep\Desktop\DEVOPS_CAPSTONE-main\server>docker build -t server .
[+] Building 0.1s (1/2)
 => [internal] load build definition from Dockerfile                                                  0.0s
 => => transferring dockerfile: 2B                                                                    0.0s
failed to solve with frontend dockerfile.v0: failed to read dockerfile: open /var/lib/docker/tmp/buildkit-mount1729293405/Dockerfile: no such
 file or directory

C:\Users\pradeep\Desktop\DEVOPS_CAPSTONE-main\server>docker build -t server-img .
[+] Building 56.1s (6/9)
 => [internal] load build definition from Dockerfile                                                  0.0s
 => => transferring dockerfile: 462B                                                                  0.0s
 => [internal] load .dockerignore                                                                     0.0s
 => => transferring context: 2B                                                                       0.0s
 => [internal] load metadata for docker.io/library/node:14                                            6.5s
 => [auth] library/node:pull token for registry-1.docker.io                                           0.0s
 => CANCELED [1/4] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa    49.3s
 => => resolve docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa             0.0s
 => => sha256:b253aeafeaa7e0671bb60008df01de101a38a045ff7bc656e3b0fbfc7c05cca5 1.05MB / 7.86MB                                 49.3s
 => => sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa 776B / 776B                                     0.0s
 => => sha256:2cafa3fbb0b6529ee4726b4f599ec27ee557ea3dea7019182323b3779959927f 2.21kB / 2.21kB                                 0.0s
 => => sha256:1d12470fa662a2a5cb50378dcdc8ea228c1735747db410bbefb8e2d9144b5452 7.51kB / 7.51kB                                 0.0s
 => => sha256:2ff1d7c41c74a25258bfa6f0b8adb0a727f84518f55f65ca845ebc747976c408 0B / 50.45MB                                    49.3s
 => => sha256:3d2201bd995cccf12851a50820de03d34a17011dcbb9ac9fdf3a50c952cbb131 1.05MB / 10.00MB                                49.3s
```



```
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app
```

```
C:\Users\pradeep\Desktop\DEVOPS_CAPSTONE-main\server>docker login
Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/

C:\Users\pradeep\Desktop\DEVOPS_CAPSTONE-main\server>docker login
Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
f1b984ac66ca: Pushed
0b7744a9b61b: Pushed
4e2bdc2ee847: Pushed
0d5f5a015e5d: Mounted from library/node
3c777d951de2: Mounted from library/node
f8a91dd5fc84: Mounted from library/node
cb81227abde5: Mounted from library/node
e01a454893a9: Mounted from library/node
c45660adde37: Mounted from library/node
fe0fb3ab4a0f: Mounted from library/node
f1186e5061f2: Mounted from library/node
b2dba7477754: Mounted from library/node
latest: digest: sha256:a863a1ddfcd8b4467bedd9172281048da6820c4dd629101eb84e8affe9693ea6 size: 2845
```

*****This will create a new image with the name "docker-username/image-name" that points to the same image as "image-name". We can then use this new image name to push the image to a remote Docker registry or to run containers based on this image.

docker tag <image-name> <docker username>/<image-name>

*****Push the docker image into the docker hub using the following command:

docker push <docker username>/<image-name>

*****Go to the AWS Management Console and create an EC2 instance.

*After the successful creation of the EC2 instance select the instance and connect the instance.

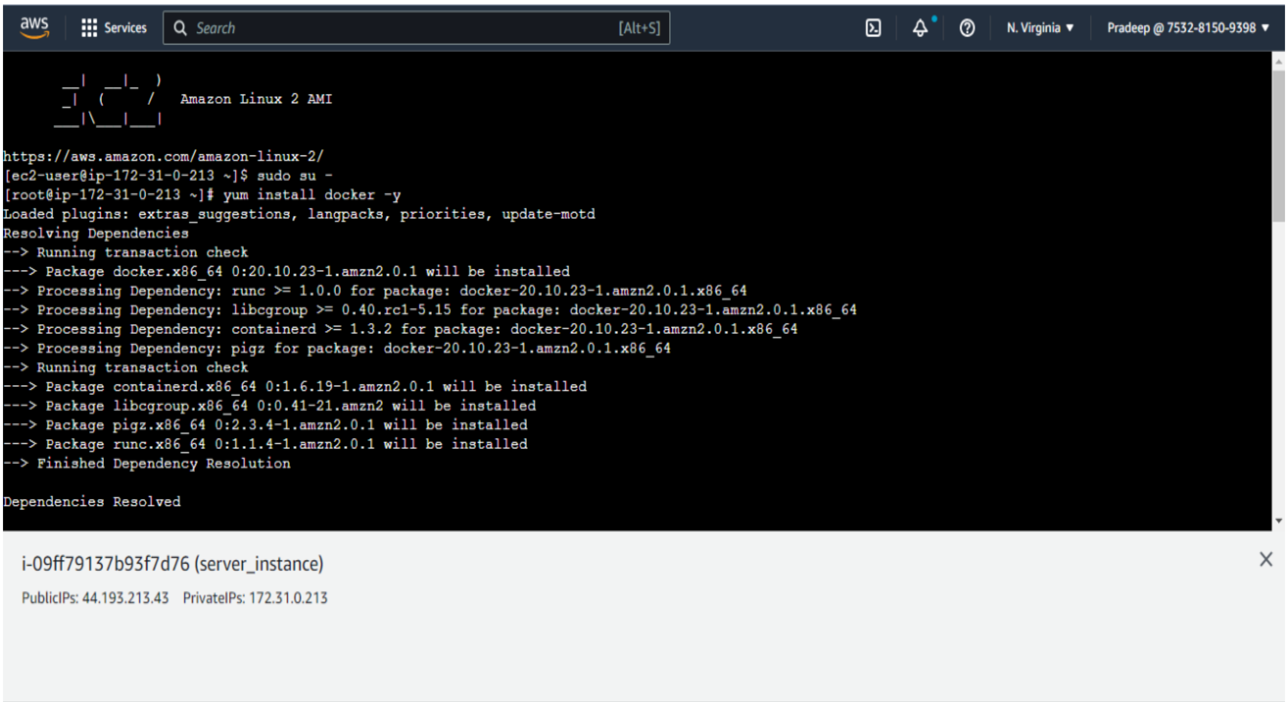
*After connecting the instance install docker and login to the docker.

*Then pull the docker image that has been created for the back end and run the image.

*Use the following commands to do the above steps:

sudo su –

yum install docker -y



```
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-0-213 ~]$ sudo su -
[root@ip-172-31-0-213 ~]# yum install docker -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
---> Package docker.x86_64 0:20.10.23-1.amzn2.0.1 will be installed
--> Processing Dependency: runc >= 1.0.0 for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Processing Dependency: libcgroup >= 0.40.rc1-5.15 for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Processing Dependency: containerd >= 1.3.2 for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Processing Dependency: pigz for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Running transaction check
---> Package containerd.x86_64 0:1.6.19-1.amzn2.0.1 will be installed
---> Package libcgroup.x86_64 0:0.41-21.amzn2 will be installed
---> Package pigz.x86_64 0:2.3.4-1.amzn2.0.1 will be installed
---> Package runc.x86_64 0:1.1.4-1.amzn2.0.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved
```

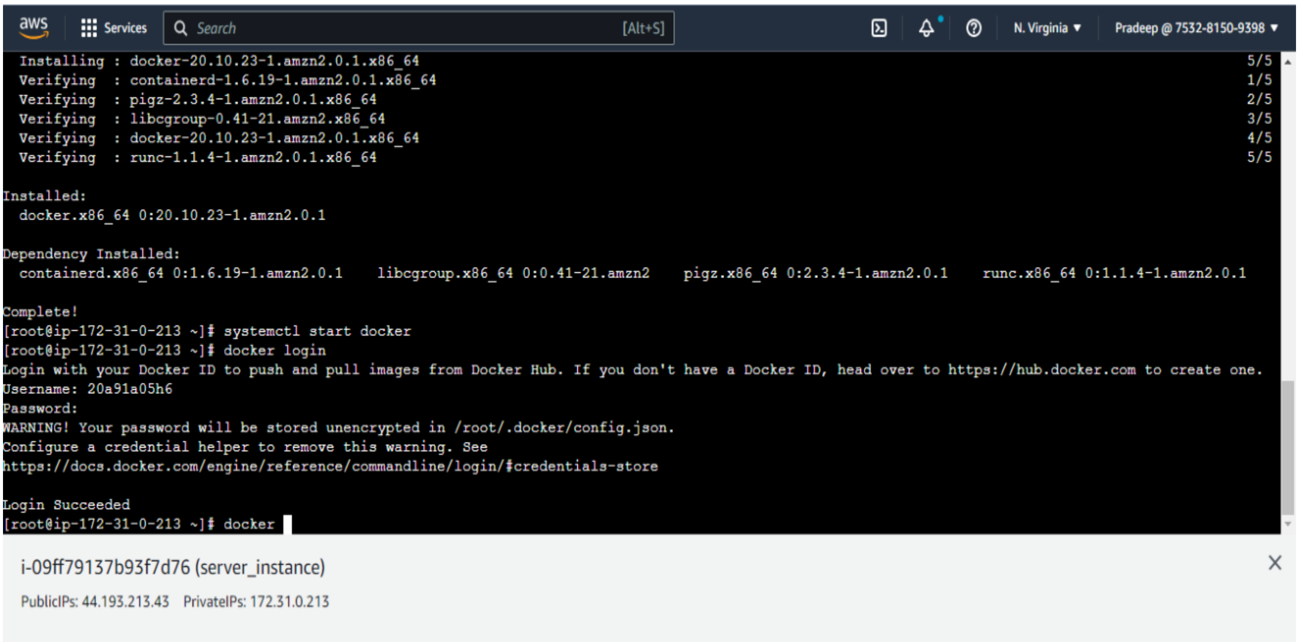
i-09ff79137b93f7d76 (server_instance)

PublicIPs: 44.193.213.43 PrivateIPs: 172.31.0.213

*Use the below commands to start the docker and login to docker.

systemctl start docker

docker login



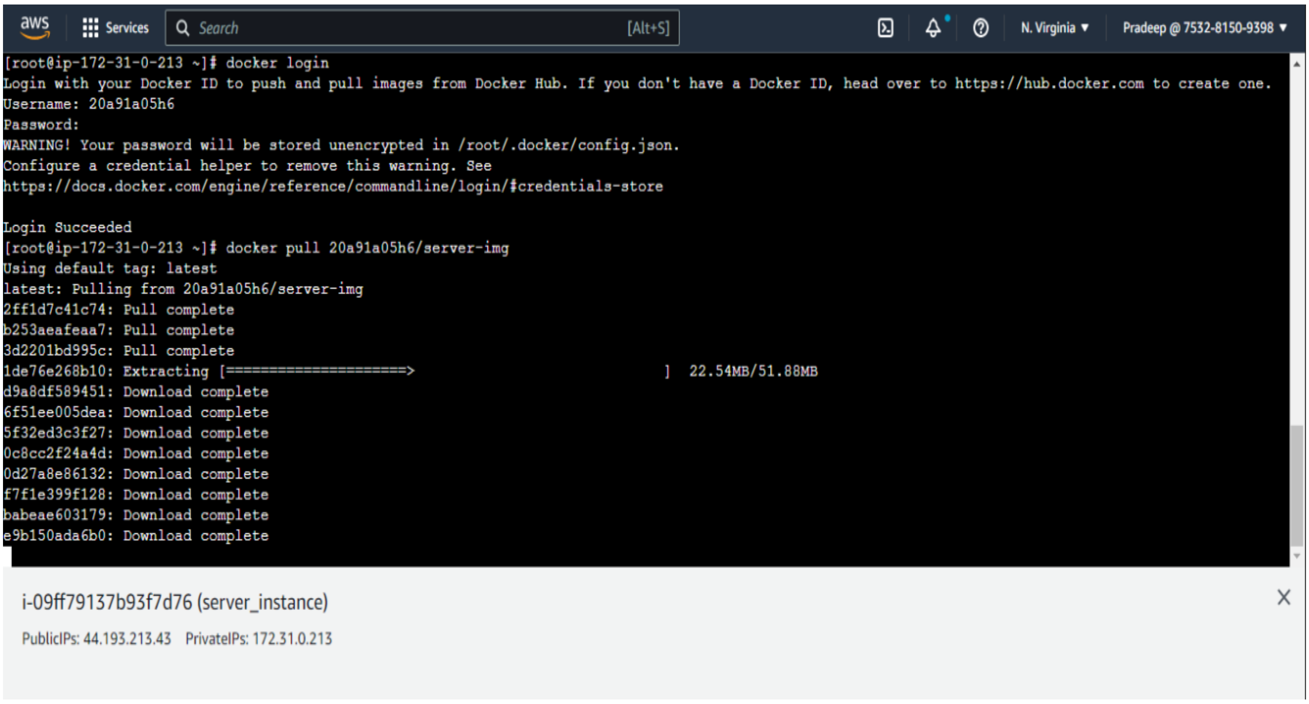
```
  Installing : docker-20.10.23-1.amzn2.0.1.x86_64                                                                          5/5
  Verifying  : containerd-1.6.19-1.amzn2.0.1.x86_64                                                                        1/5
  Verifying  : pigz-2.3.4-1.amzn2.0.1.x86_64                                                                               2/5
  Verifying  : libcgroup-0.41-21.amzn2.x86_64                                                                              3/5
  Verifying  : docker-20.10.23-1.amzn2.0.1.x86_64                                                                          4/5
  Verifying  : runc-1.1.4-1.amzn2.0.1.x86_64                                                                               5/5

Installed:
  docker.x86_64 0:20.10.23-1.amzn2.0.1

Dependency Installed:
  containerd.x86_64 0:1.6.19-1.amzn2.0.1    libcgroup.x86_64 0:0.41-21.amzn2    pigz.x86_64 0:2.3.4-1.amzn2.0.1    runc.x86_64 0:1.1.4-1.amzn2.0.1

Complete!
[root@ip-172-31-0-213 ~]# systemctl start docker
[root@ip-172-31-0-213 ~]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: 20a91a05h6
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@ip-172-31-0-213 ~]# docker
```

i-09ff79137b93f7d76 (server_instance)

PublicIPs: 44.193.213.43 PrivateIPs: 172.31.0.213

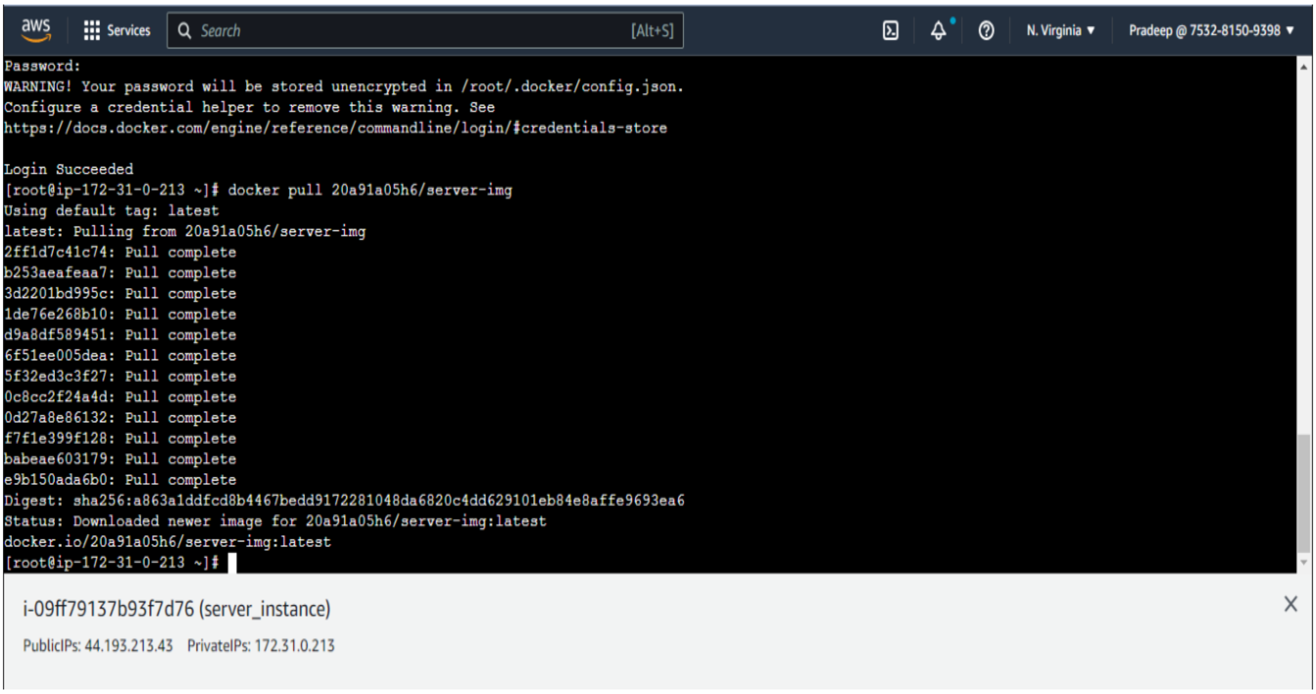
*****To pull the docker image, use the command-

docker pull <docker username>/<image name>



```
[root@ip-172-31-0-213 ~]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: 20a91a05h6
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@ip-172-31-0-213 ~]# docker pull 20a91a05h6/server-img
Using default tag: latest
latest: Pulling from 20a91a05h6/server-img
2ff1d7c41c74: Pull complete
b253aeafeaa7: Pull complete
3d2201bd995c: Pull complete
1de76e268b10: Extracting [====================>                              ]  22.54MB/51.88MB
d9a8df589451: Download complete
6f51ee005dea: Download complete
5f32ed3c3f27: Download complete
0c8cc2f24a4d: Download complete
0d27a8e86132: Download complete
f7f1e399f128: Download complete
babeae603179: Download complete
e9b150ada6b0: Download complete
```

i-09ff79137b93f7d76 (server_instance)
PublicIPs: 44.193.213.43 PrivateIPs: 172.31.0.213
X

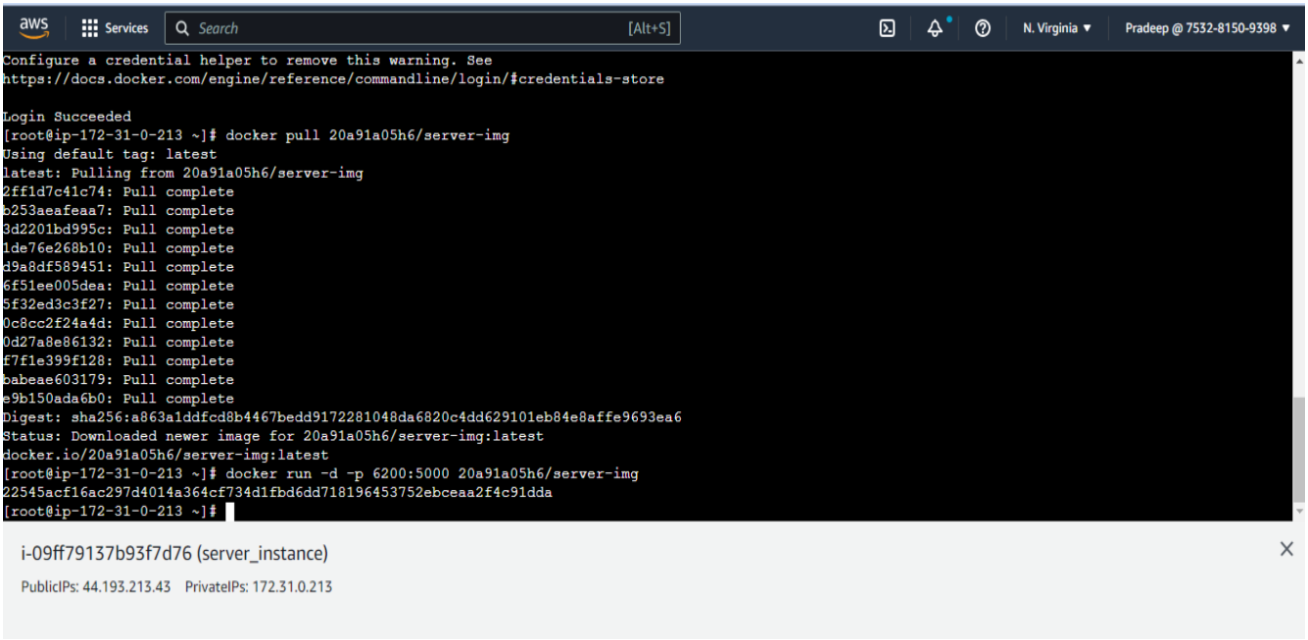

aws
Services
Search
[Alt+S]
N. Virginia ▾
Pradeep @ 7532-8150-9398 ▾

```
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@ip-172-31-0-213 ~]# docker pull 20a91a05h6/server-img
Using default tag: latest
latest: Pulling from 20a91a05h6/server-img
2ff1d7c41c74: Pull complete
b253aeafeaa7: Pull complete
3d2201bd995c: Pull complete
1de76e268b10: Pull complete
d9a8df589451: Pull complete
6f51ee005dea: Pull complete
5f32ed3c3f27: Pull complete
0c8cc2f24a4d: Pull complete
0d27a8e86132: Pull complete
f7f1e399f128: Pull complete
babeae603179: Pull complete
e9b150ada6b0: Pull complete
Digest: sha256:a863a1ddfcd8b4467bedd9172281048da6820c4dd629101eb84e8affe9693ea6
Status: Downloaded newer image for 20a91a05h6/server-img:latest
docker.io/20a91a05h6/server-img:latest
[root@ip-172-31-0-213 ~]#
```

i-09ff79137b93f7d76 (server_instance)
PublicIPs: 44.193.213.43 PrivateIPs: 172.31.0.213
X

*****To run the docker image

docker run -d -p Default_port:Expose_port <docker username>/<image name>



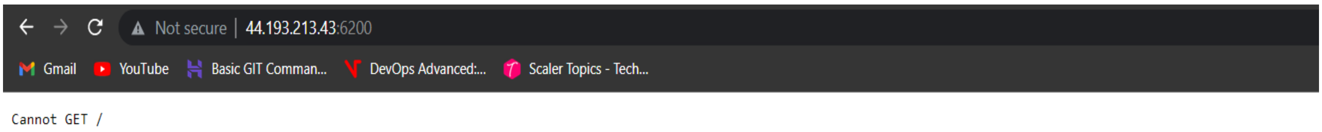
```
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@ip-172-31-0-213 ~]# docker pull 20a91a05h6/server-img
Using default tag: latest
latest: Pulling from 20a91a05h6/server-img
2ff1d7c41c74: Pull complete
b253aeafeaa7: Pull complete
3d2201bd995c: Pull complete
1de76e268b10: Pull complete
d9a8df589451: Pull complete
6f51ee005dea: Pull complete
5f32ed3c3f27: Pull complete
0c8cc2f24a4d: Pull complete
0d27a8e86132: Pull complete
f7f1e399f128: Pull complete
babeae603179: Pull complete
e9b150ada6b0: Pull complete
Digest: sha256:a863a1ddfcd8b4467bedd9172281048da6820c4dd629101eb84e8affe9693ea6
Status: Downloaded newer image for 20a91a05h6/server-img:latest
docker.io/20a91a05h6/server-img:latest
[root@ip-172-31-0-213 ~]# docker run -d -p 6200:5000 20a91a05h6/server-img
22545acf16ac297d4014a364cf734d1fbd6dd718196453752ebceaa2f4c91dda
[root@ip-172-31-0-213 ~]# 
```

i-09ff79137b93f7d76 (server_instance)

PublicIPs: 44.193.213.43 PrivateIPs: 172.31.0.213

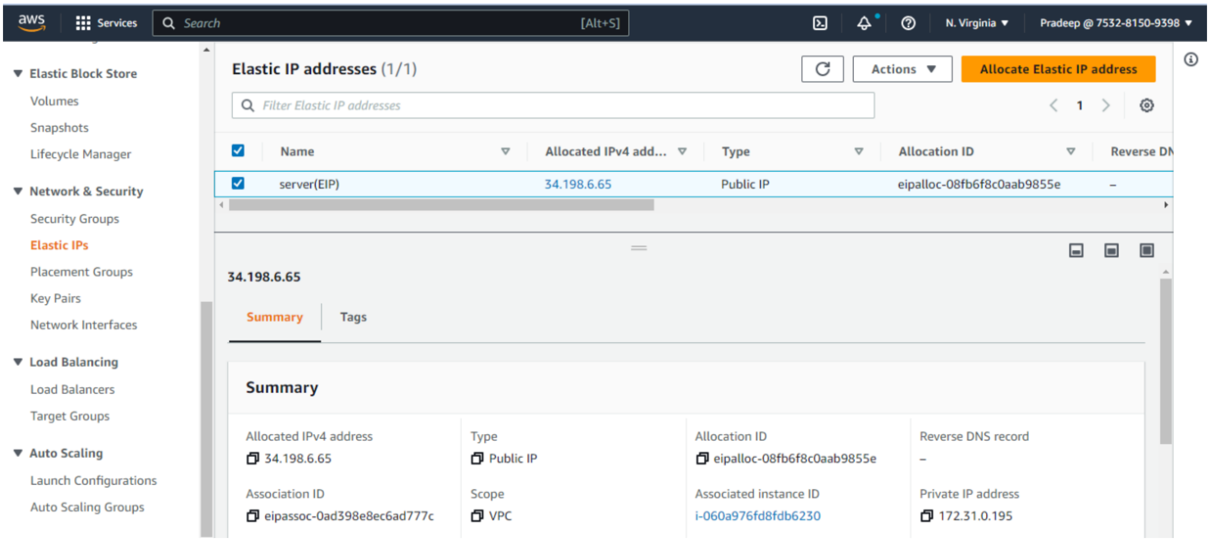
***Copy the public IP of the instance. Open a new tab and then give **PublicIP:Default_port** as the URL for connecting the backend.

*We will be then getting an error page which was the output of our backend code.



```
Cannot GET /
```

***** Attach an Elastic IP to the server instance which will give your instance a static IP address that doesn’t change when we start or stop the instance.



Elastic IP addresses (1/1)

☑	Name	Allocated IPv4 add...	Type	Allocation ID	Reverse DN
☑	server(EIP)	34.198.6.65	Public IP	eipalloc-08fb6f8c0aab9855e	–

34.198.6.65

Summary

Allocated IPv4 address
34.198.6.65

Type
Public IP

Allocation ID
eipalloc-08fb6f8c0aab9855e

Reverse DNS record
–

Association ID
eipassoc-0ad398e8ec6ad777c

Scope
VPC

Associated instance ID
i-060a976fd8fdb6230

Private IP address
172.31.0.195

4)Frontend Deployment:

***In the client, we need to install the packages like Mongoose, express, cors, and cloudinary which are used in the index.js file using the following command:**

```
npm i mongoose cloudinary express multer cors aws-sdk
```

*In client, we will have a package.json file that contains the installed packages, when we give the command **npm run build**, all the necessary packages will be installed.

***We need to make the front-end fetch data from the back end. In order to do this we need to make changes in the **App.js** file of the front-end code.

*We need to change the URL with the URL of the backend code output.

const url = "http://localhost:5000"

***const url needs to be modified to

const url= http://serverpublicip:assignedportnumber

const url=http://34.198.6.65:6200

*Create a Docker file for the front end. This file contains a port number which will make it available to the world outside the container.

```
# Use an official Node.js runtime as a parent image

FROM node:14-alpine


# Set the working directory to /app

WORKDIR /


# Copy package.json and package-lock.json to the container

COPY package*.json ./


# Install dependencies

RUN npm install


# Copy the rest of the application code to the container

COPY . .


# Build the application

RUN npm run build


# Serve the application with a lightweight HTTP server

FROM nginx:alpine
```

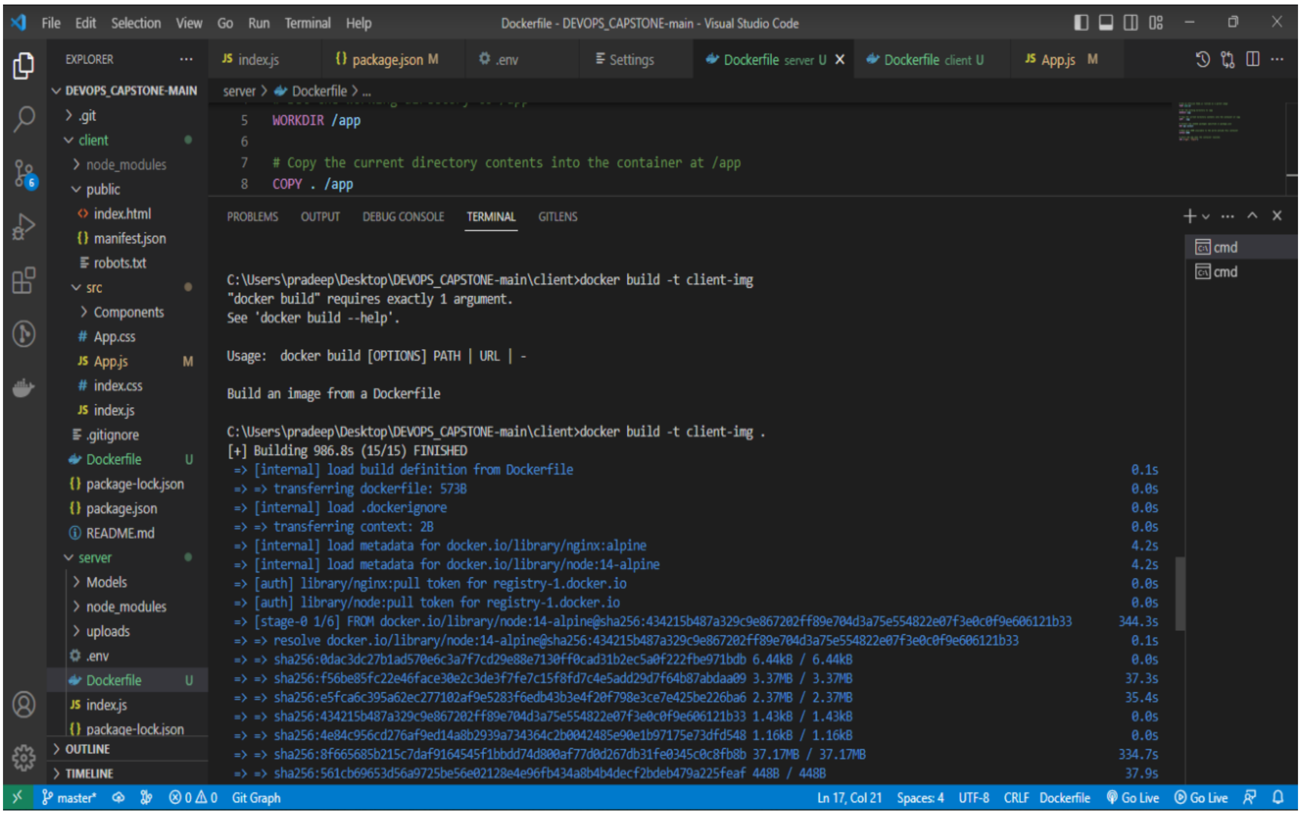
COPY --from=0 /build /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

*Build a docker image named “client-img” in the directory containing the Docker file.

docker build -t client-img .



```
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app
```

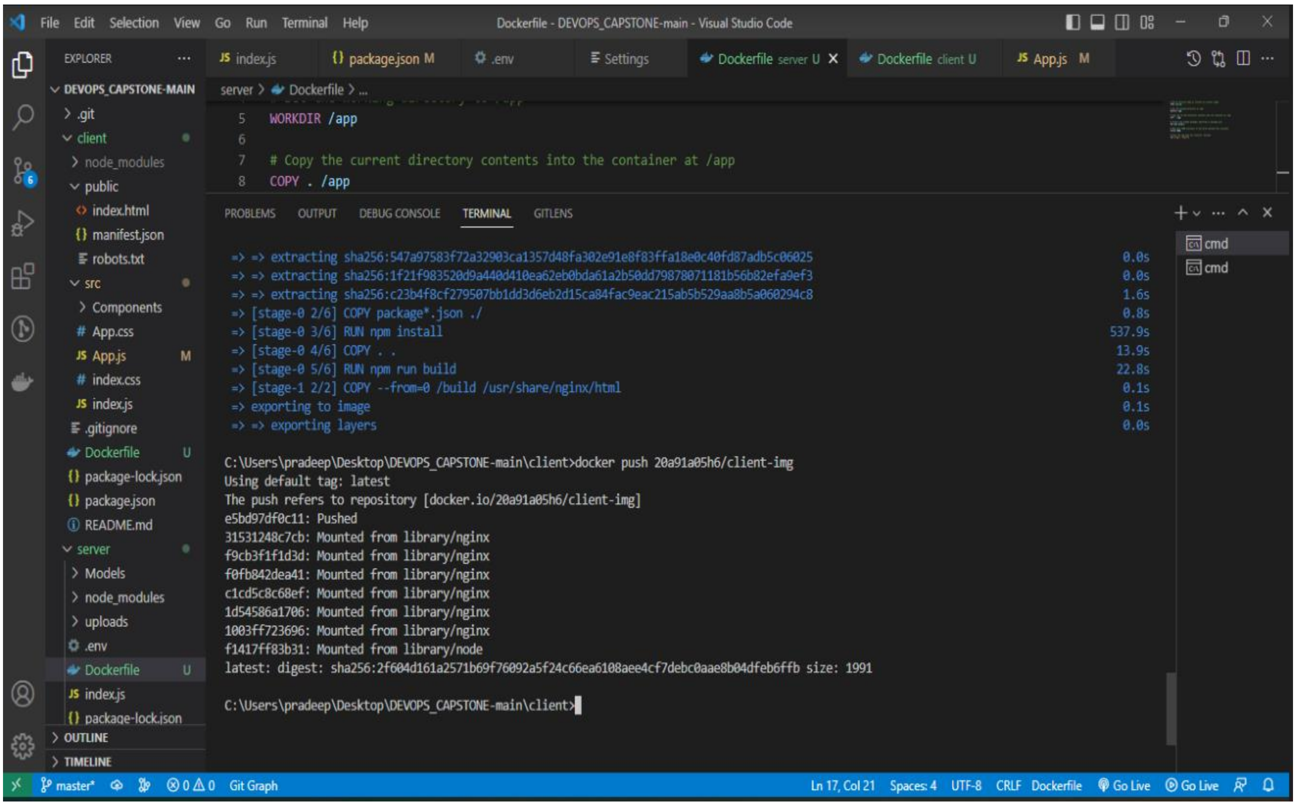
```
C:\Users\pradeep\Desktop\DEVOPS_CAPSTONE-main\client>docker build -t client-img
"docker build" requires exactly 1 argument.
See 'docker build --help'.

Usage:  docker build [OPTIONS] PATH | URL | -

Build an image from a Dockerfile

C:\Users\pradeep\Desktop\DEVOPS_CAPSTONE-main\client>docker build -t client-img .
[+] Building 986.8s (15/15) FINISHED
 => [internal] load build definition from Dockerfile                                                                  0.1s
 => => transferring dockerfile: 573B                                                                                  0.0s
 => [internal] load .dockerignore                                                                                     0.0s
 => => transferring context: 2B                                                                                       0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine                                                       4.2s
 => [internal] load metadata for docker.io/library/node:14-alpine                                                     4.2s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                          0.0s
 => [auth] library/node:pull token for registry-1.docker.io                                                           0.0s
 => [stage-0 1/6] FROM docker.io/library/node:14-alpine@sha256:434215b487a329c9e867202ff89e704d3a75e554822e07f3e0c0f9e606121b33   344.3s
 => => resolve docker.io/library/node:14-alpine@sha256:434215b487a329c9e867202ff89e704d3a75e554822e07f3e0c0f9e606121b33   0.1s
 => => sha256:0dac3dc27b1ad570e6c3a7f7cd29e88e7130ff0cad31b2ec5a0f222fbe971bdb 6.44kB / 6.44kB                          0.0s
 => => sha256:f56be85fc22e46face30e2c3de3f7fe7c15f8fd7c4e5add29d7f64b87abdaa09 3.37MB / 3.37MB                          37.3s
 => => sha256:e5fca6c395a62ec277102af9e5283f6edb43b3e4f20f798e3ce7e425be226ba6 2.37MB / 2.37MB                          35.4s
 => => sha256:434215b487a329c9e867202ff89e704d3a75e554822e07f3e0c0f9e606121b33 1.43kB / 1.43kB                          0.0s
 => => sha256:4e84c956cd276af9ed14a8b2939a734364c2b0042485e90e1b97175e73dfd548 1.16kB / 1.16kB                          0.0s
 => => sha256:8f665685b215c7daf9164545f1bbdd74d800af77d0d267db31fe0345c0c8fb8b 37.17MB / 37.17MB                        334.7s
 => => sha256:561cb69653d56a9725be56e02128e4e96fb434a8b4b4decf2bdeb479a225feaf 448B / 448B                              37.9s
```

*This will create a new image with the name "dockerusername/image-name" that points to the same image as "image-name". We can then use this new image name to push the image to a remote Docker registry or to run containers based on this image.

docker tag <image-name> <docker username>/<image-name>

*Push the docker image into the docker hub using the following command:

docker push <docker username>/<image-name>



```
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app
```

```
=> => extracting sha256:547a97583f72a32903ca1357d48fa302e91e8f83ffa18e0c40fd87adb5c06025        0.0s
=> => extracting sha256:1f21f983520d9a440d410ea62eb0bda61a2b50dd79878071181b56b82efa9ef3        0.0s
=> => extracting sha256:c23b4f8cf279507bb1dd3d6eb2d15ca84fac9eac215ab5b529aa8b5a060294c8        1.6s
=> [stage-0 2/6] COPY package*.json ./                                                          0.8s
=> [stage-0 3/6] RUN npm install                                                              537.9s
=> [stage-0 4/6] COPY . .                                                                      13.9s
=> [stage-0 5/6] RUN npm run build                                                             22.8s
=> [stage-1 2/2] COPY --from=0 /build /usr/share/nginx/html                                     0.1s
=> exporting to image                                                                           0.1s
=> => exporting layers                                                                          0.0s

C:\Users\pradeep\Desktop\DEVOPS_CAPSTONE-main\client>docker push 20a91a05h6/client-img
Using default tag: latest
The push refers to repository [docker.io/20a91a05h6/client-img]
e5bd97df0c11: Pushed
31531248c7cb: Mounted from library/nginx
f9cb3f1f1d3d: Mounted from library/nginx
f0fb842dea41: Mounted from library/nginx
c1cd5c8c68ef: Mounted from library/nginx
1d54586a1706: Mounted from library/nginx
1003ff723696: Mounted from library/nginx
f1417ff83b31: Mounted from library/node
latest: digest: sha256:2f604d161a2571b69f76092a5f24c66ea6108aee4cf7debc0aae8b04dfeb6ffb size: 1991

C:\Users\pradeep\Desktop\DEVOPS_CAPSTONE-main\client>
```

*****Go to the AWS Management Console and create 3 EC2 instances.

*After the successful creation of the EC2 instances select the instance and connect the instance.

*After connecting the instance install docker and login to the docker.

*Then pull the docker image that has been created for the front end and run the image.

*Use the following commands to do the above steps:

sudo su –

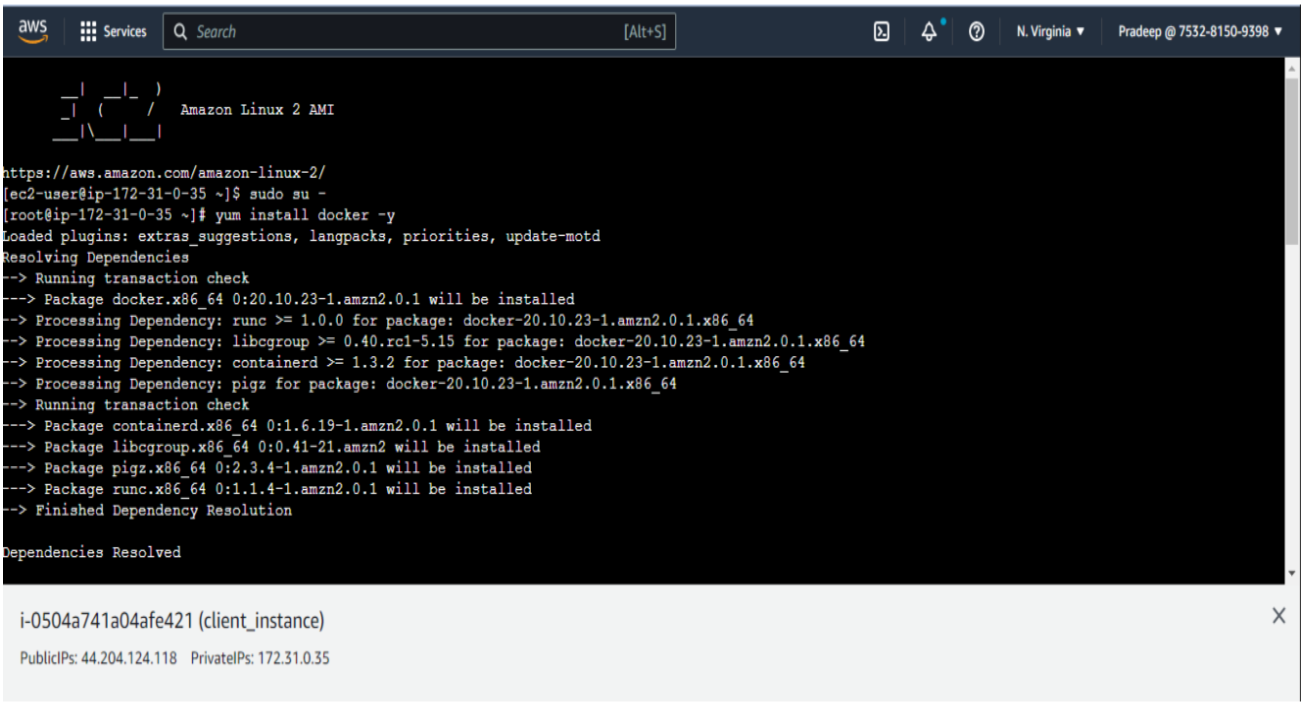
yum install docker -y

systemctl start docker

docker login

docker pull <docker username>/<image name>

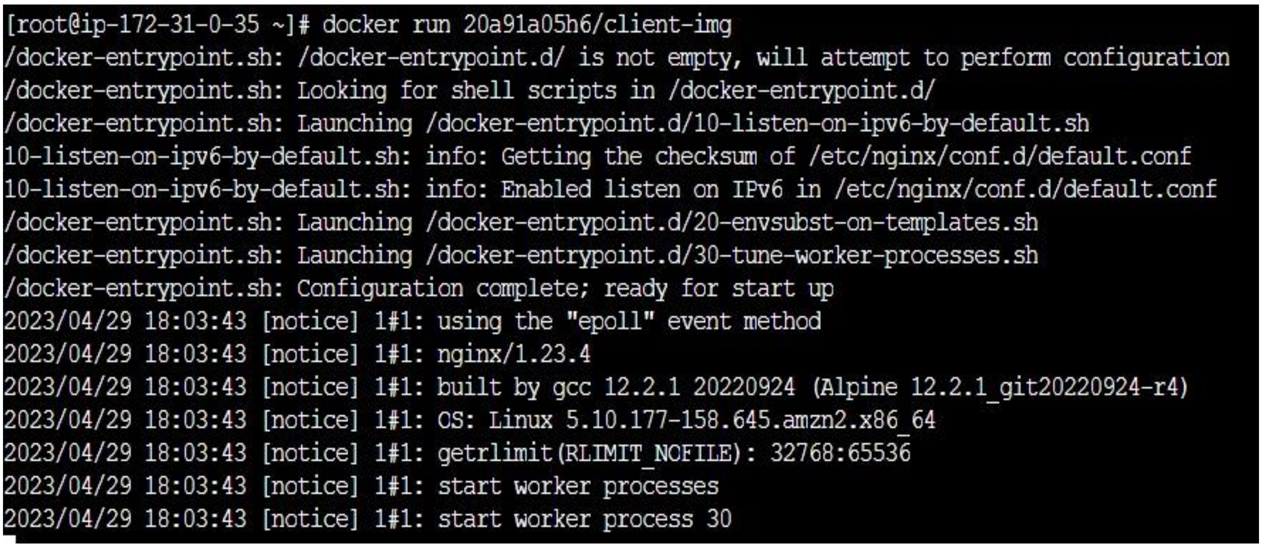
docker run -d -p Default_port:Expose_port <docker username>/<image name>



```
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-0-35 ~]$ sudo su -
[root@ip-172-31-0-35 ~]# yum install docker -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
---> Package docker.x86_64 0:20.10.23-1.amzn2.0.1 will be installed
--> Processing Dependency: runc >= 1.0.0 for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Processing Dependency: libcgroup >= 0.40.rc1-5.15 for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Processing Dependency: containerd >= 1.3.2 for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Processing Dependency: pigz for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Running transaction check
---> Package containerd.x86_64 0:1.6.19-1.amzn2.0.1 will be installed
---> Package libcgroup.x86_64 0:0.41-21.amzn2 will be installed
---> Package pigz.x86_64 0:2.3.4-1.amzn2.0.1 will be installed
---> Package runc.x86_64 0:1.1.4-1.amzn2.0.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved
```

i-0504a741a04afe421 (client_instance)

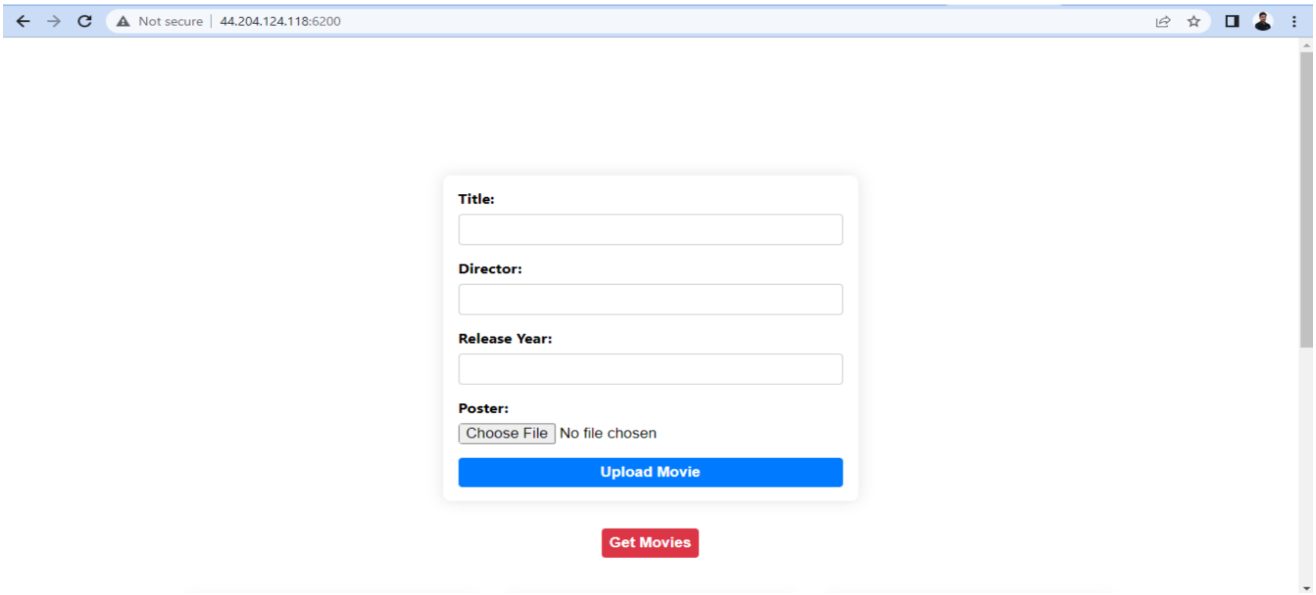
PublicIPs: 44.204.124.118   PrivateIPs: 172.31.0.35



```
[root@ip-172-31-0-35 ~]# docker run 20a91a05h6/client-img
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/04/29 18:03:43 [notice] 1#1: using the "epoll" event method
2023/04/29 18:03:43 [notice] 1#1: nginx/1.23.4
2023/04/29 18:03:43 [notice] 1#1: built by gcc 12.2.1 20220924 (Alpine 12.2.1_git20220924-r4)
2023/04/29 18:03:43 [notice] 1#1: OS: Linux 5.10.177-158.645.amzn2.x86_64
2023/04/29 18:03:43 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 32768:65536
2023/04/29 18:03:43 [notice] 1#1: start worker processes
2023/04/29 18:03:43 [notice] 1#1: start worker process 30
```

*Copy the public IP of the instance. Open a new tab and then give **PublicIP:Default_port** as the URL.

*We will be then getting the “Movie listing” website page.



*The .jpg ,.png ,.jpeg ,etc formats files are accepted .These files are stored with a unique object id in the mondodb, It helps us to easily retrieve data from the database.

5)Load balancing

Elastic Load Balancer (ELB) automatically distributes the incoming traffic across multiple Amazon EC2 instances, helping to improve the availability and scalability of applications running on AWS. In our project, we have used a Network load balancer for the distribution of traffic to multiple EC2 instances.

*Go to AWS Management Console.

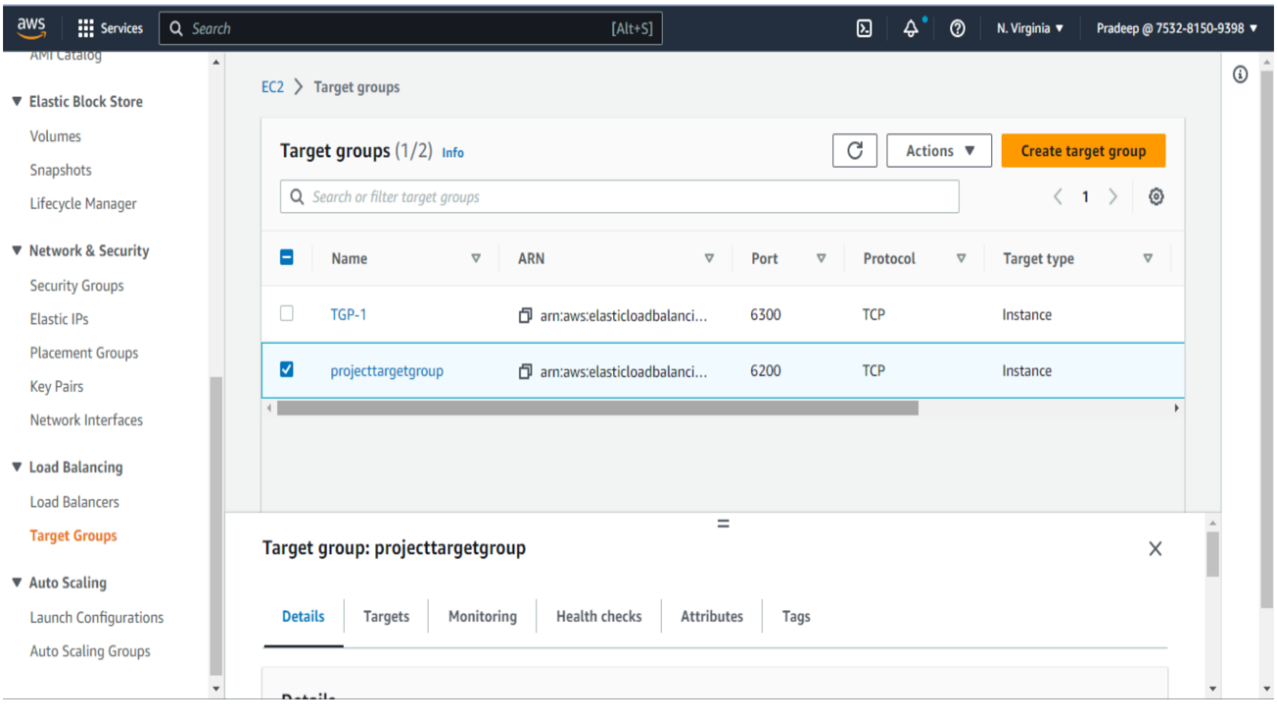
*Select the EC2 and go to the EC2 dashboard.

*In the load balancing section, click on target groups.

*Click on create target group. Choose instances as the type of target group.

*Give a target group name and select protocol as TCP and port as Default port used in the Docker file of front end and back end.

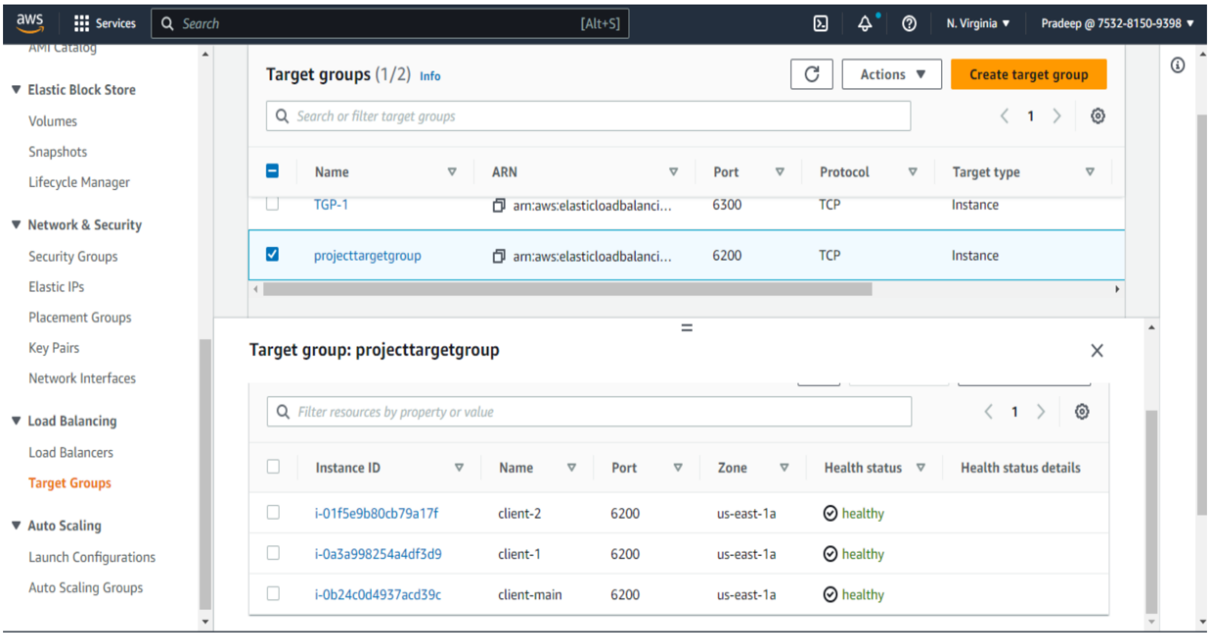
*Register all the client servers as the target groups and click on Create target group.



Target groups (1/2)

	Name	ARN	Port	Protocol	Target type
□	TGP-1	arn:aws:elasticloadbalanci...	6300	TCP	Instance
☑	projecttargetgroup	arn:aws:elasticloadbalanci...	6200	TCP	Instance

Target group: projecttargetgroup

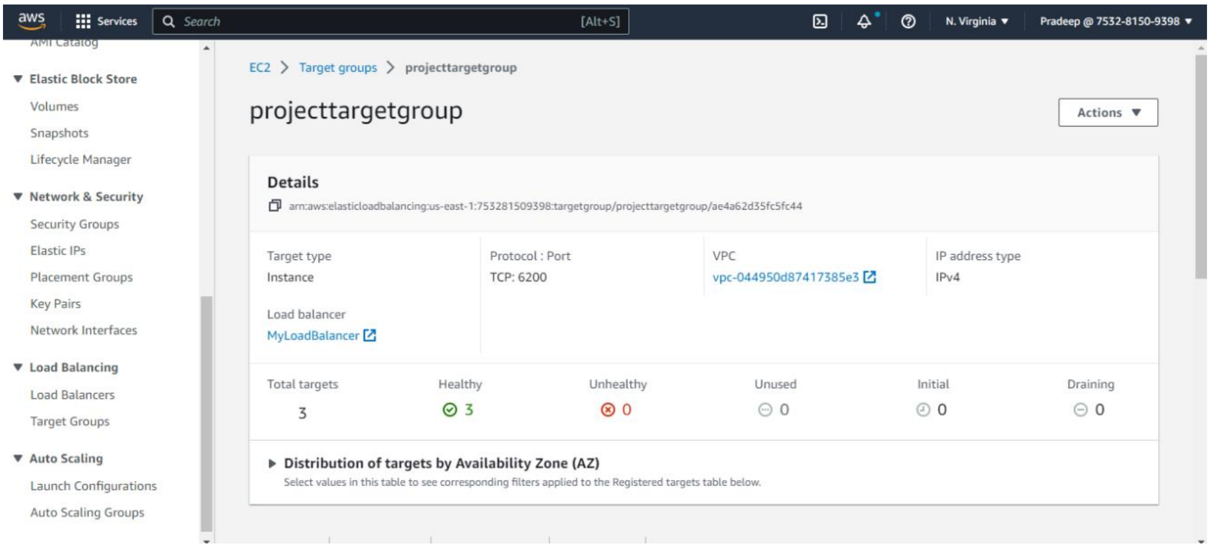


Target groups (1/2) Info

	Name	ARN	Port	Protocol	Target type
□	TGP-1	arn:aws:elasticloadbalanci...	6300	TCP	Instance
☑	projecttargetgroup	arn:aws:elasticloadbalanci...	6200	TCP	Instance

Target group: projecttargetgroup

	Instance ID	Name	Port	Zone	Health status	Health status details
□	i-01f5e9b80cb79a17f	client-2	6200	us-east-1a	✓ healthy	
□	i-0a3a998254a4df3d9	client-1	6200	us-east-1a	✓ healthy	
□	i-0b24c0d4937acd39c	client-main	6200	us-east-1a	✓ healthy	



projecttargetgroup

Details

arn:aws:elasticloadbalancing:us-east-1:753281509398:targetgroup/projecttargetgroup/ae4a62d35fc5fc44

Target type: Instance

Protocol : Port: TCP: 6200

VPC: vpc-044950d87417385e3

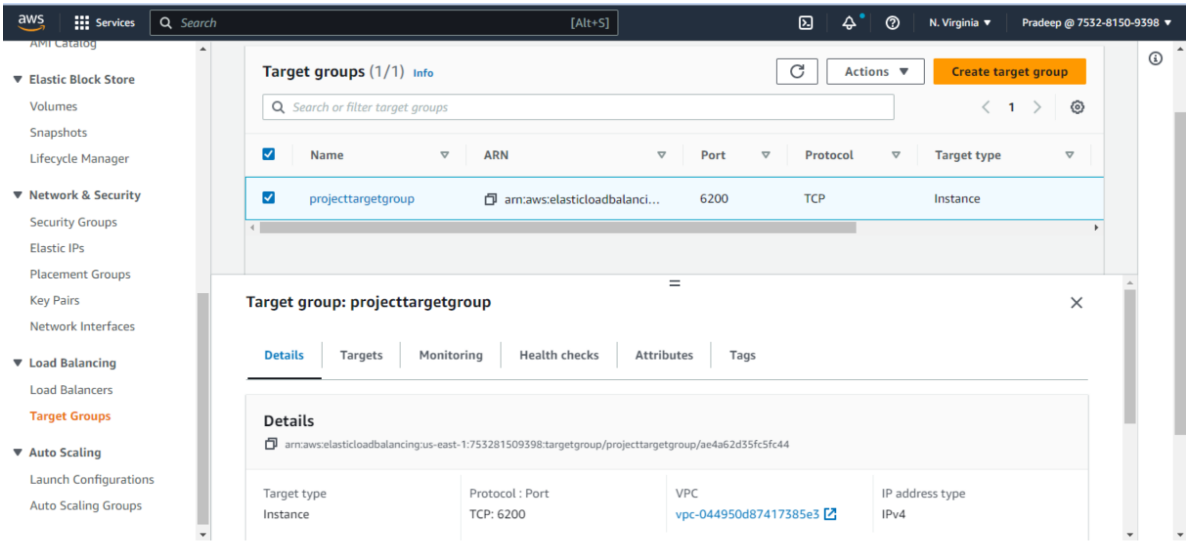
IP address type: IPv4

Load balancer: MyLoadBalancer

Total targets	Healthy	Unhealthy	Unused	Initial	Draining
3	✓ 3	☒ 0	0	0	0

▶ Distribution of targets by Availability Zone (AZ)

Select values in this table to see corresponding filters applied to the Registered targets table below.



Target groups (1/1) Info

	Name	ARN	Port	Protocol	Target type
☑	projecttargetgroup	arn:aws:elasticloadbalanci...	6200	TCP	Instance

Target group: projecttargetgroup

Details

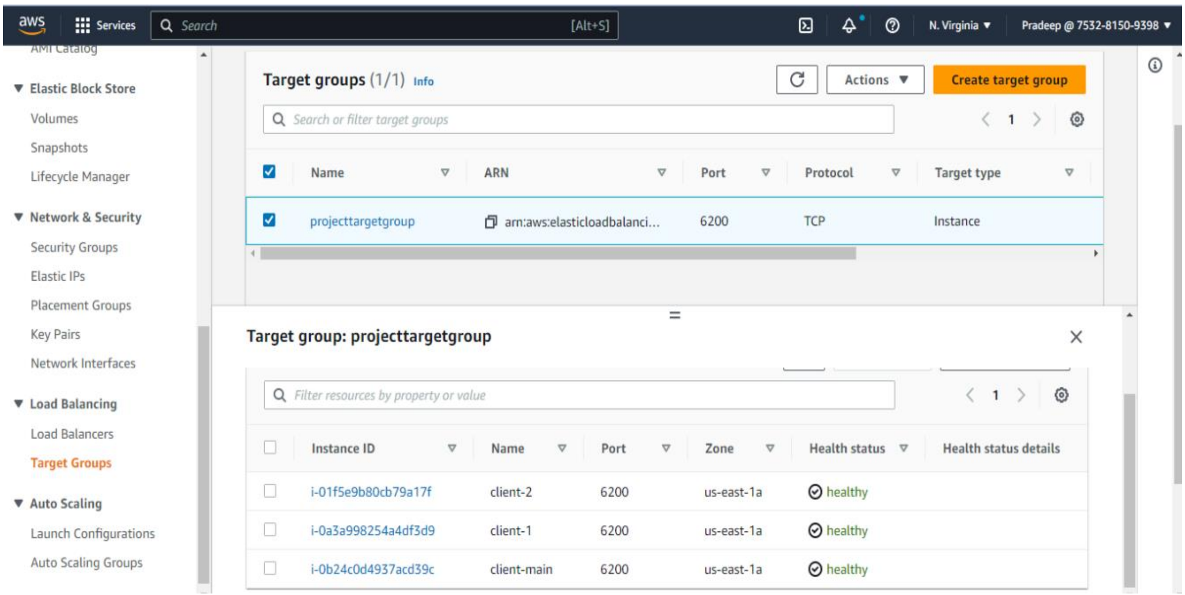
arn:aws:elasticloadbalancing:us-east-1:753281509398:targetgroup/projecttargetgroup/ae4a62d35fc5fc44

Target type: Instance

Protocol : Port: TCP: 6200

VPC: vpc-044950d87417385e3

IP address type: IPv4



Target groups (1/1) Info

☑	Name	ARN	Port	Protocol	Target type
☑	projecttargetgroup	arn:aws:elasticloadbalanci...	6200	TCP	Instance

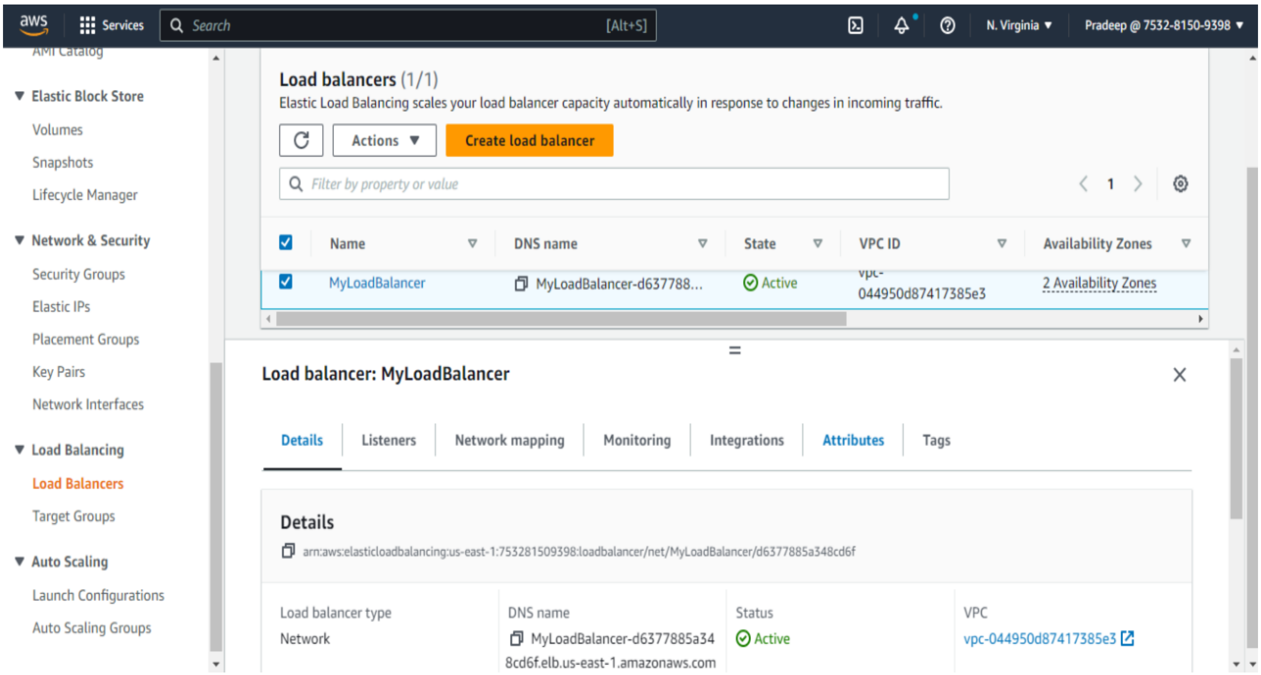
Target group: projecttargetgroup

□	Instance ID	Name	Port	Zone	Health status	Health status details
□	i-01f5e9b80cb79a17f	client-2	6200	us-east-1a	✓ healthy	
□	i-0a3a998254a4df3d9	client-1	6200	us-east-1a	✓ healthy	
□	i-0b24c0d4937acd39c	client-main	6200	us-east-1a	✓ healthy	

*Go to the Load balancer section and select the Network load balancer as the load balancer type.

*Click on Create. Give a load balancer name and select all the availability zones as mappings.

*In the Listener's and routing section, select the target group that was created earlier. Click on Create the load balancer.



Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

☑	Name	DNS name	State	VPC ID	Availability Zones
☑	MyLoadBalancer	MyLoadBalancer-d637788...	✓ Active	vpc-044950d87417385e3	2 Availability Zones

Load balancer: MyLoadBalancer

Details

arn:aws:elasticloadbalancing:us-east-1:753281509398:loadbalancer/net/MyLoadBalancer/d6377885a348cd6f

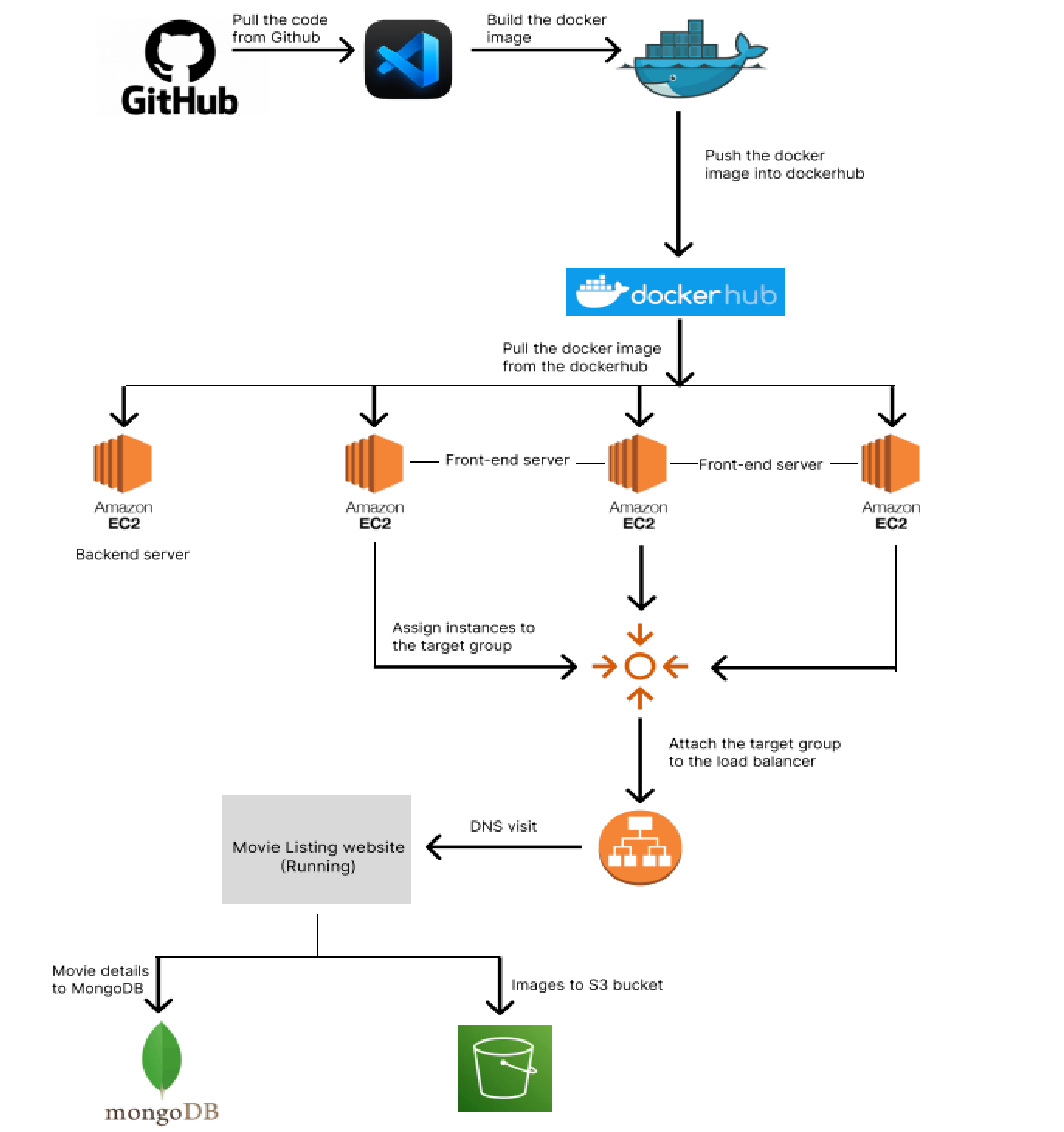
Load balancer type	DNS name	Status	VPC
Network	MyLoadBalancer-d6377885a348cd6f.elb.us-east-1.amazonaws.com	✓ Active	vpc-044950d87417385e3

AWS DEPLOYMENT DIAGRAM:

Figma:

Figma is a cloud-based design and prototyping tool that allows designers and teams to collaborate on designing user interfaces, web pages, and mobile applications.

As a part of this project we have used Figma tool to design the architecture of the entire deployment process which makes everyone to easily understand the process and helps in every phase to understand the tasks needs to be done.

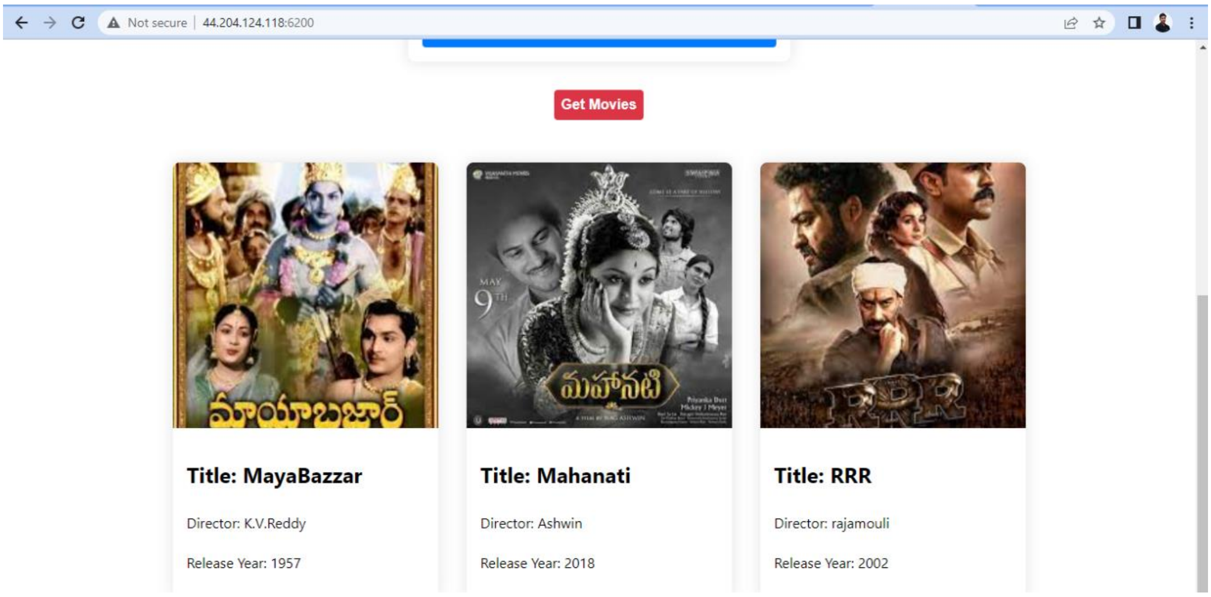

GitHub
Pull the code from Github
Build the docker image
Push the docker image into dockerhub
docker hub
Pull the docker image from the dockerhub
Amazon EC2
Backend server
Amazon EC2
Front-end server
Amazon EC2
Front-end server
Amazon EC2
Assign instances to the target group
Attach the target group to the load balancer
DNS visit
Movie Listing website (Running)
Movie details to MongoDB
Images to S3 bucket
mongoDB

OBSERVATIONS:

From the project " Movie Listing " we have observed,

- **Technology Stack:** The project " Movie Listing " uses popular and robust technologies such as ReactJS, NodeJS, and MongoDB. These technologies are well-suited for developing modern and scalable web applications.
- **Local Storage for Images**: The project uses local storage to store images, which can be a limitation as local storage has limited storage capacity. Using a cloud-based storage solution such as Amazon S3 would be a better option.
- We have used Multer S3 which provides a simple and easy-to-use interface to upload files to S3. It integrates well with Express.js, a popular Node.js web application framework, and can be used with any other Node.js framework as well.
- **Cloud Infrastructure**: The movie listing project requires a cloud infrastructure that is reliable, scalable, and secure. AWS provides a wide range of services that can be used to build and deploy the project. Deploying the website to AWS is a good choice as AWS provides a wide range of services to build, deploy, and scale web applications.
- **Containerization using Docker**: Docker containers provide a consistent runtime environment, which helps to eliminate issues related to differences in dependencies and configurations across various development, testing, and production environments.
- This ensures that the application runs consistently across different environments. We have used the docker hub to push and pull the images that we have built.
- **Proper Auto Scaling**: Scaling is an essential aspect of any web application, and it is crucial to design the architecture in such a way that it can scale horizontally or vertically. Using AWS Auto Scaling can help in scaling the application efficiently.
- **Security:** It is essential to ensure that the application is secure, and user data is protected. Using AWS Identity and Access Management (IAM), VPC can help in securing the application.

Overall, using AWS to build and deploy the movie listing project can provide a reliable, scalable, and secure cloud infrastructure. It simplifies the deployment process, reduces infrastructure management overhead, and enables efficient use of cloud resources.



Title: MayaBazzar

Director: K.V.Reddy

Release Year: 1957

Title: Mahanati

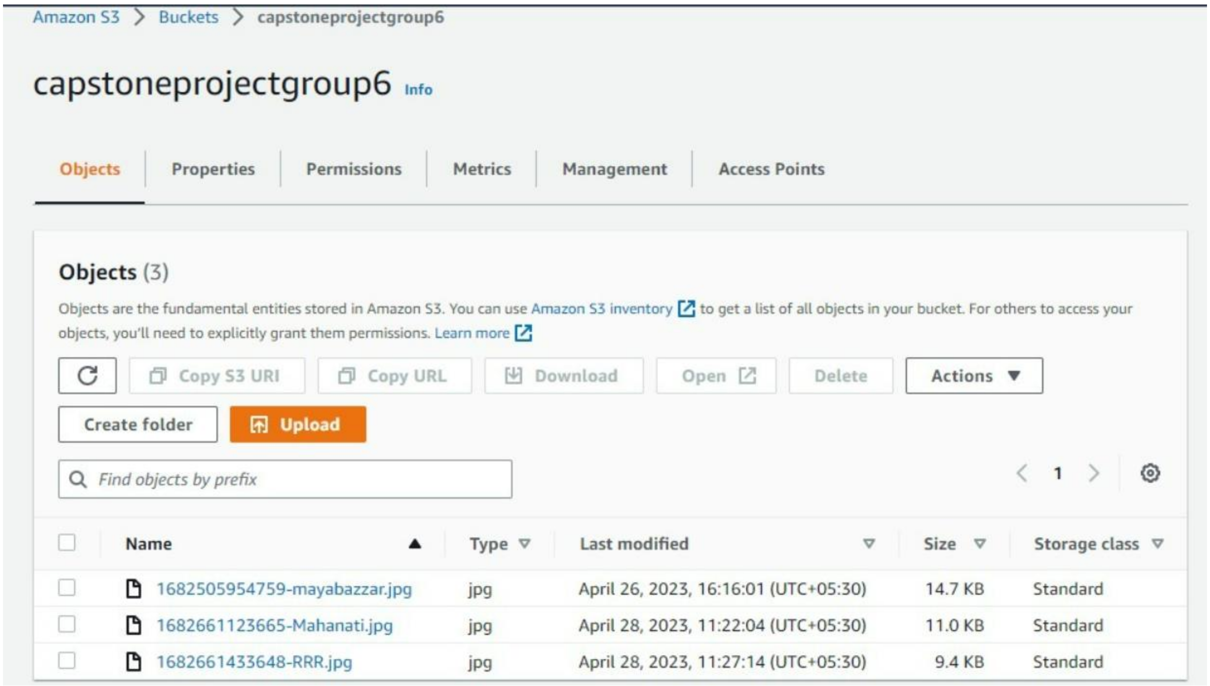
Director: Ashwin

Release Year: 2018

Title: RRR

Director: rajamouli

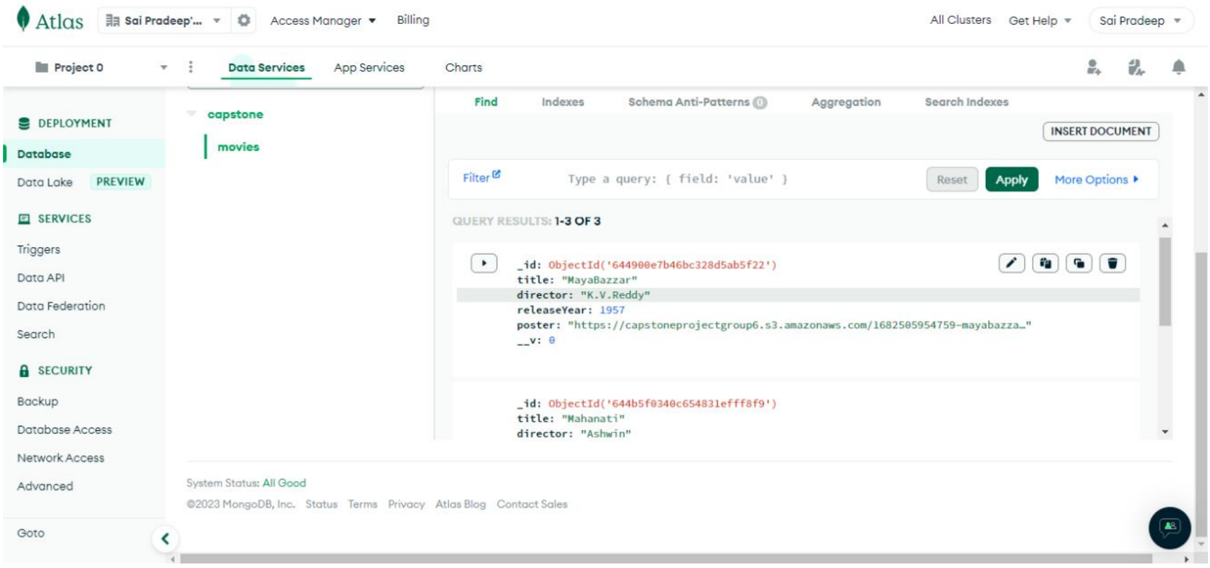
Release Year: 2002



Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name ▲	Type ▽	Last modified ▽	Size ▽	Storage class ▽
□	1682505954759-mayabazzar.jpg	jpg	April 26, 2023, 16:16:01 (UTC+05:30)	14.7 KB	Standard
□	1682661123665-Mahanati.jpg	jpg	April 28, 2023, 11:22:04 (UTC+05:30)	11.0 KB	Standard
□	1682661433648-RRR.jpg	jpg	April 28, 2023, 11:27:14 (UTC+05:30)	9.4 KB	Standard



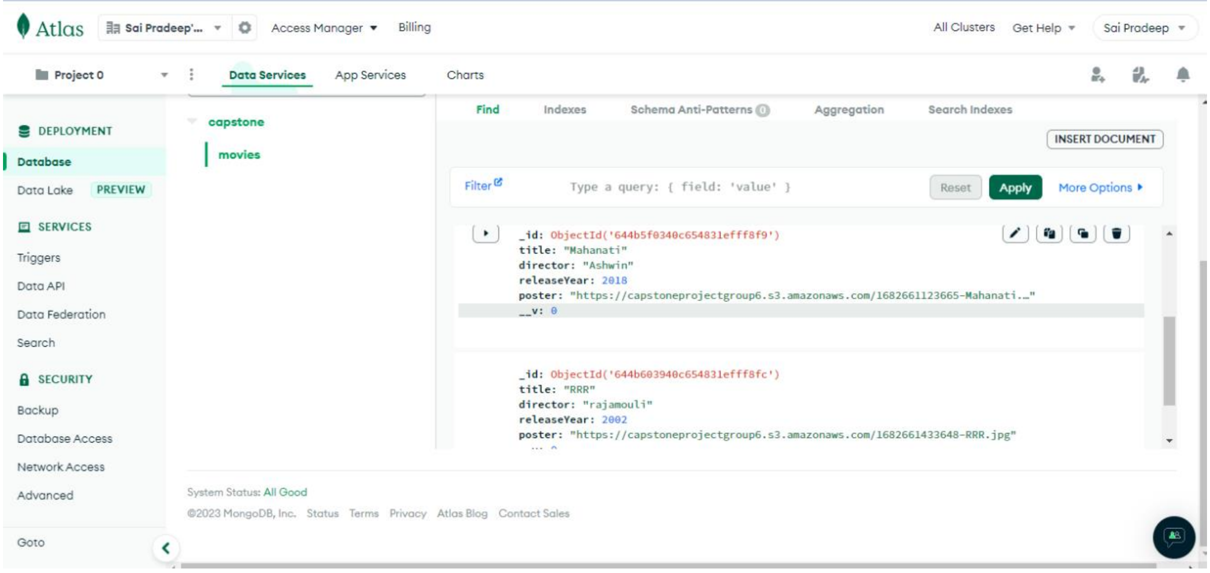
```
Type a query: { field: 'value' }
```

QUERY RESULTS: 1-3 OF 3

```
_id: ObjectId('644900e7b46bc328d5ab5f22')
title: "MayaBazzar"
director: "K.V.Reddy"
releaseYear: 1957
poster: "https://capstoneprojectgroup6.s3.amazonaws.com/1682505954759-mayabazza…"
__v: 0

_id: ObjectId('644b5f0340c654831efff8f9')
title: "Mahanati"
director: "Ashwin"
```

System Status: All Good

©2023 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales



```
Type a query: { field: 'value' }
```

```
_id: ObjectId('644b5f0340c654831efff8f9')
title: "Mahanati"
director: "Ashwin"
releaseYear: 2018
poster: "https://capstoneprojectgroup6.s3.amazonaws.com/1682661123665-Mahanati.…"
__v: 0

_id: ObjectId('644b603940c654831efff8fc')
title: "RRR"
director: "rajamouli"
releaseYear: 2002
poster: "https://capstoneprojectgroup6.s3.amazonaws.com/1682661433648-RRR.jpg"
```

System Status: All Good

©2023 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

CONCLUSION:

In conclusion, deploying the "Movie listing" website onto the AWS cloud infrastructure provides several benefits, including improved scalability, reliability, security, and resource utilization. By deploying the NodeJS application, Amazon S3 for storing images, and MongoDB Atlas for database management, we can ensure that the application runs smoothly and efficiently. Additionally, by using Docker containers, we can further simplify the deployment process, improve portability, and ensure consistency across environments. This enables us to easily deploy and scale the application without compromising performance or security. Overall, deploying the "Movie listing" website onto the AWS cloud infrastructure using Docker containers provides a robust and efficient solution for managing and scaling the application. It helps to reduce infrastructure management overhead, improve application performance and scalability, and ensure that the application runs consistently across different environments.

- **GitHub URL:**

  https://github.com/pradeep-pulaparthi/Capstone_project

*****This is the URL of the repository which contains the modified codes of the front end and back end.

- **Frontend URL:**

  http://44.204.124.118:6200/

- **Backend URL:**

  http://44.193.213.43:6200/

- **DNS URL:**

  http://myloadbalancer-d6377885a348cd6f.elb.us-east-1.amazonaws.com/

THANK YOU