DevOps

CAPSTONE PROJECT

DEPLOYING A MOVIE LISTING WEBSITE INTO AWS CLOUD INFRASTRUCTURE WITH PROPER SCALING

Team Members: **Team No. 01**

21A950A516 – SIMHADRI HIMMASRI**(TL)**

20A91A0525 – KALAHASTHI SAHASRA

20p31a0592 – KANTIPUDI UMA PARVATHI DEVI

20A91A0519 – GANTI VENKATA PATTABHI

20A91A0517 – DEMANTH VAMSI KARTHIK REDDY CHINTAPALLI

20A91A0508 – BHUMIKA SREE SARELLA

20P31A0580 – YASHWANT SATYA VIGNESH DULAM

CONTENT

PURPOSE OF THE PROJECT

The main purpose of this project is to deploy a movie listing website that uses ReactJS as frontend, NodeJS as backend, and MongoDB as a database on an Amazon EC2 instance using Docker container technology. Docker is a software platform that allows you to build, test, and deploy applications quickly. Mongo dB to replace the local database so that the data is stored in our cluster and images are stored in S3.

Whereas EC2,aws service is easy to deploy and manage virtual servers. By the end of the project, we can host a movie listing web application on EC2 instances and balance the application load with the help of Elastic Load Balancer(ELB) and it can be maintained by Docker.

SCOPE OF THE PROJECT

The scope of the project is to deploy the web application for movie listing using EC2 instance and docker containerization tool. In this project, we are using ReactJS as the frontend, NodeJS as the backend, and MongoDB as the database. And this application load will be balanced with the help of one of the AWS services called Elastic Load Balancer.

In place of a local database, we use MongoDB Atlas for our data storing and S3 for storing the uploaded or existing images.S3, MongoDB, and EC2 plays a huge role in the whole project. At last, this whole web application can be accessed by giving a DNS for it.

TECHNOLOGIES USED

Aws is the world's most comprehensive and broadly adopted cloud, offering over 200 fully featured services from data centres globally. over 200 services, including computing, storage, databases, analytics, machine learning, IoT, and security. It is flexible, cost-effective, and reliable. And some popular AWS services include EC2 for computing, S3 for storage, RDS for databases, Lambda for serverless computing, DynamoDB for NoSQL databases, and ECS for container management.In this project, we used the following services:

Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed non-linear workflows. And we also used GitHub for cloning the repository and all. GitHub is a web-based version control and collaboration platform for software developers. Microsoft, the biggest single contributor to GitHub.

Docker is an open source platform that enables developers to build, deploy, run, update and manage containers—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Docker simplifies application development and deployment by enabling developers to package their applications in containers and run them on any system with Docker installed. This provides greater flexibility, portability, and scalability for applications.

MAJOR TOOLS THAT ARE UTILIZED IN THE PROCESS OF IMPLEMENTATION

Visual Studio Code also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux, and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

EC2 (Elastic Compute Cloud) is a web service provided by Amazon Web Services (AWS) that allows users to rent virtual computers, known as instances, on which they can run their own applications. EC2 provides a flexible, scalable, and cost-effective computing environment for developers and businesses to deploy and run applications in the cloud.

MongoDB Atlas is an integrated suite of data services centered around a cloud database designed to accelerate and simplify how you build with data.

Amazon S3 (Simple Storage Service) is a cloud-based storage service that enables users to store and retrieve data from anywhere on the internet. It has various storage classes fit for multiple and diverse purposes accordingly to their requirements. It is highly reliable, secure, and scalable, and supports various types of data, including text files, images, videos, and backups.

GitHub is a web-based platform that provides developers with a variety of tools for software development and collaboration. It includes a version control system called Git, which allows developers to track changes to their code and collaborate with others.

Identity and Access Management (IAM) is a web service provided by Amazon Web Services (AWS) that enables users to securely manage access to AWS resources. IAM allows users to control who can access the AWS resources and what actions they can perform on those resources.

Route 53 is a highly available and scalable cloud-based Domain Name System (DNS) service provided by Amazon Web Services (AWS). It helps developers and business owners by translating the domain name into an IP address.And it can be used to register and manage domain names, create and manage DNS records, and perform health checks on resources.

S3 Multer is a node. js middleware for handling multipart/form-data , which is primarily used for uploading files. It is written on top of busboy for maximum efficiency. multer-s3 : Streaming multer storage engine for AWS S3.

And some tools that are used for the implementation of the project are:

TECHNOLOGIES STACK

Frontend

Database

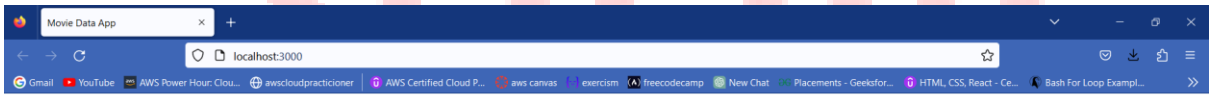
Backend

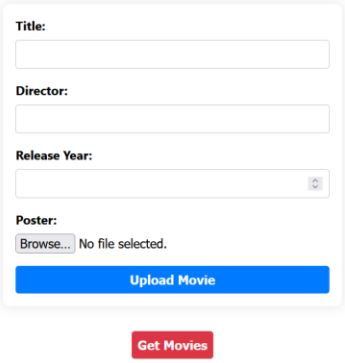
IMPLEMENTATION

Step 1 : Cloning the web application repository from the GitHub

Initially cloned the given Movie Listing web application repository. And deployed the frontend. And the repository link was:

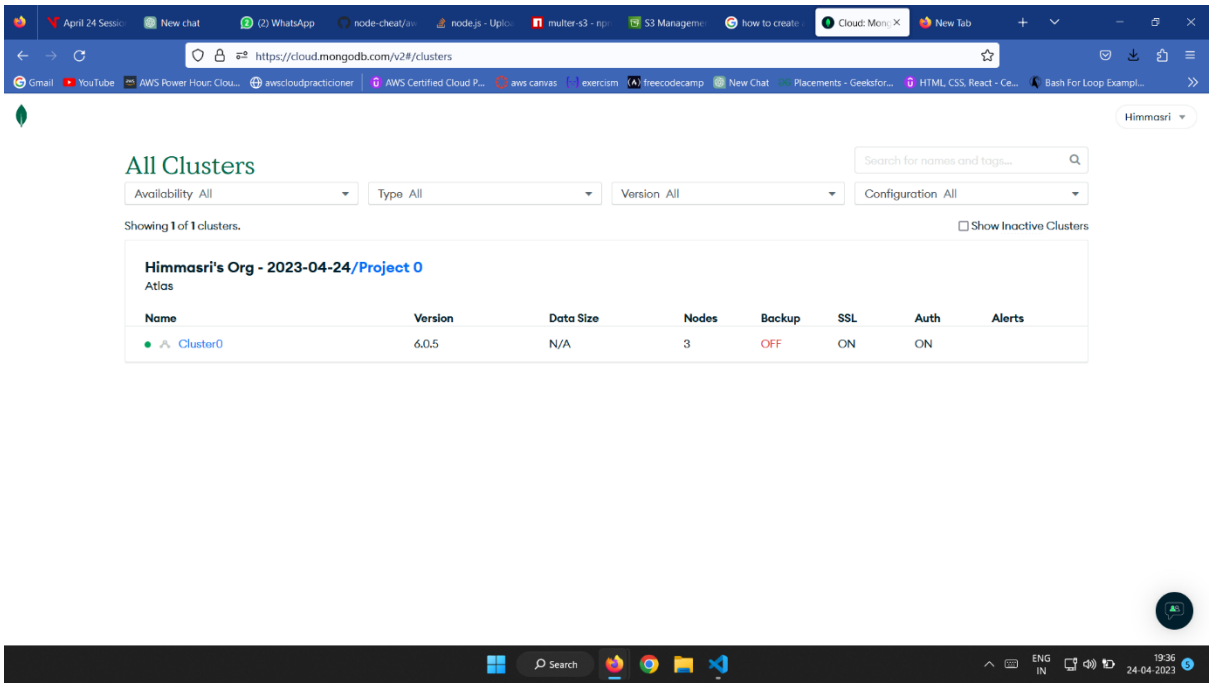
https://github.com/snehal-herovired/DEVOPS_CAPSTONE

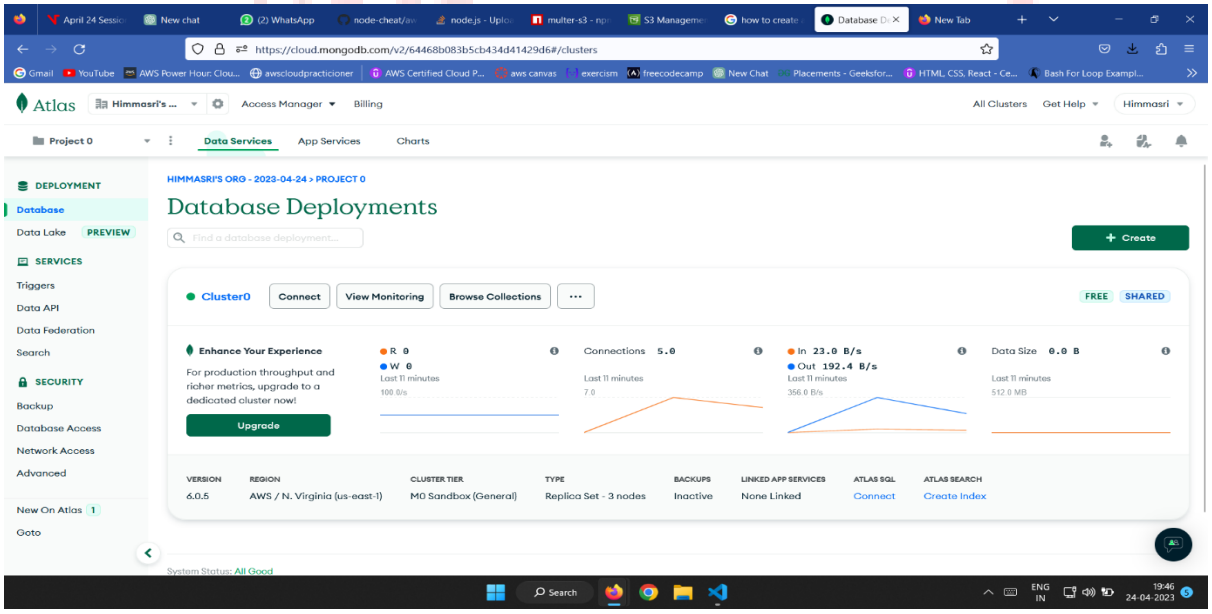




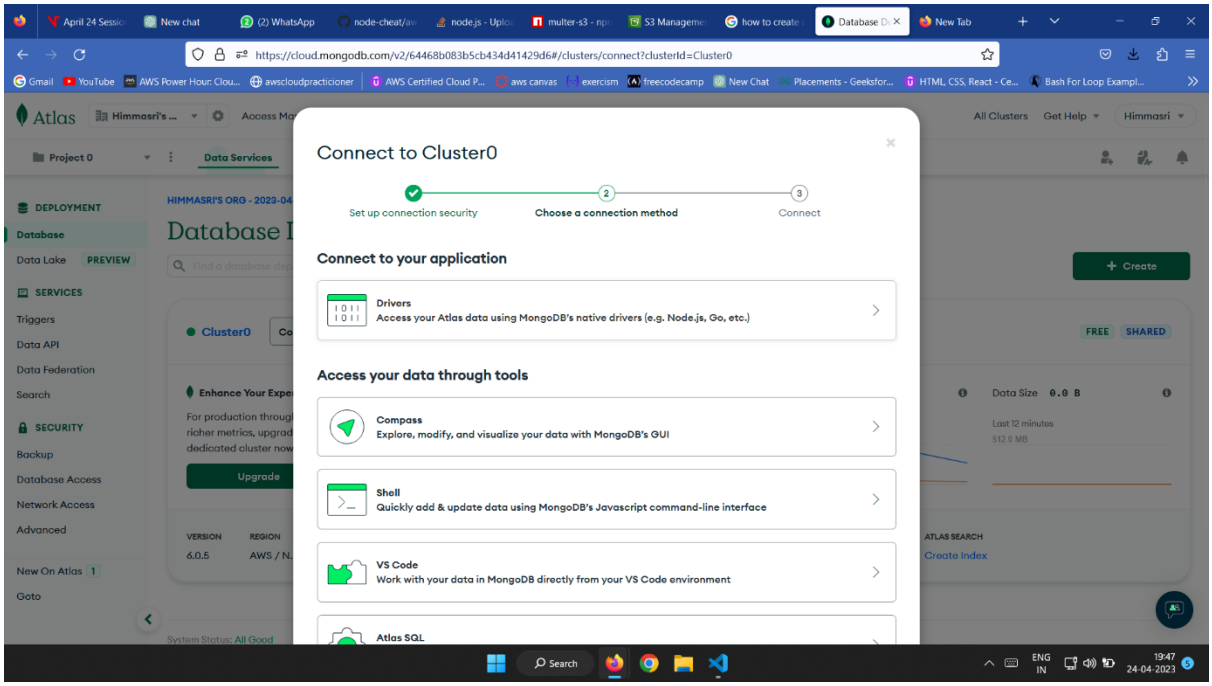
Step 2: **Create a cluster in mongoDB atlas to store the uploaded data and data collections that we have about the movies**

Here we are login into MongoDB Atlas account and created a cluster and given the network access asfds allow access from anywhere

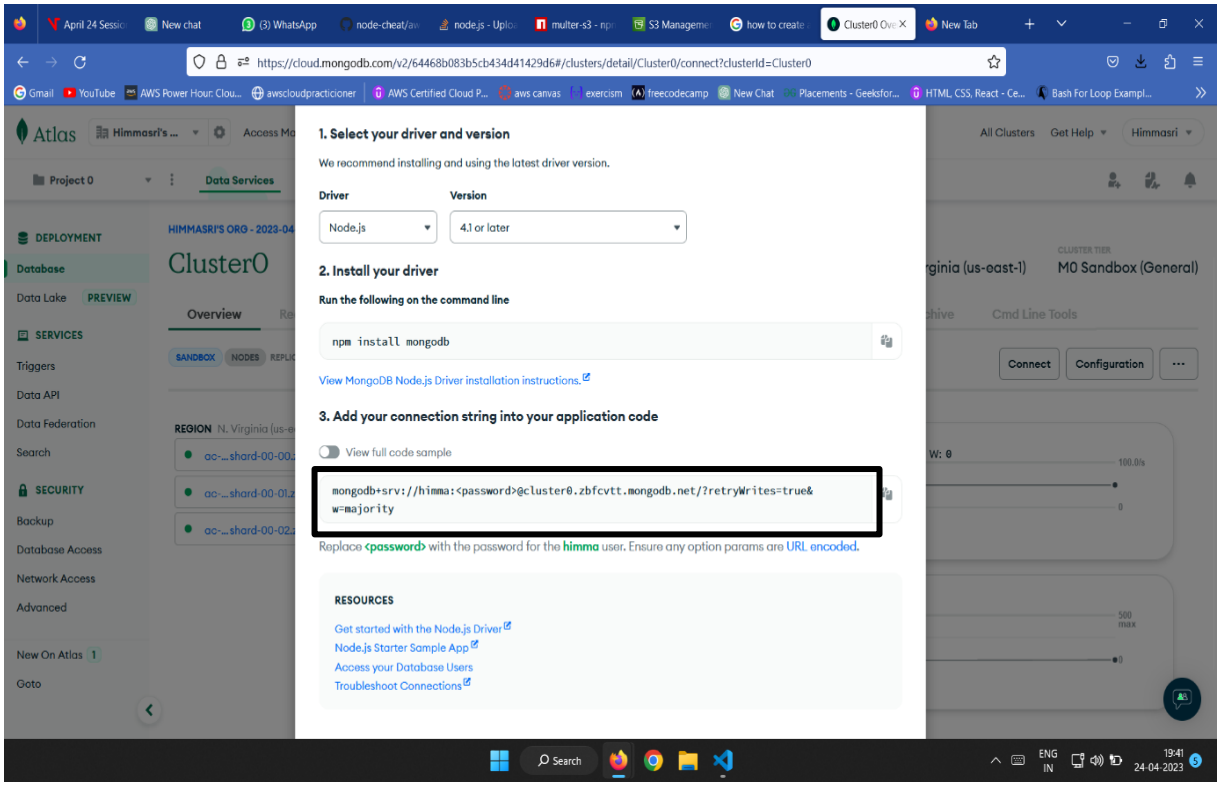




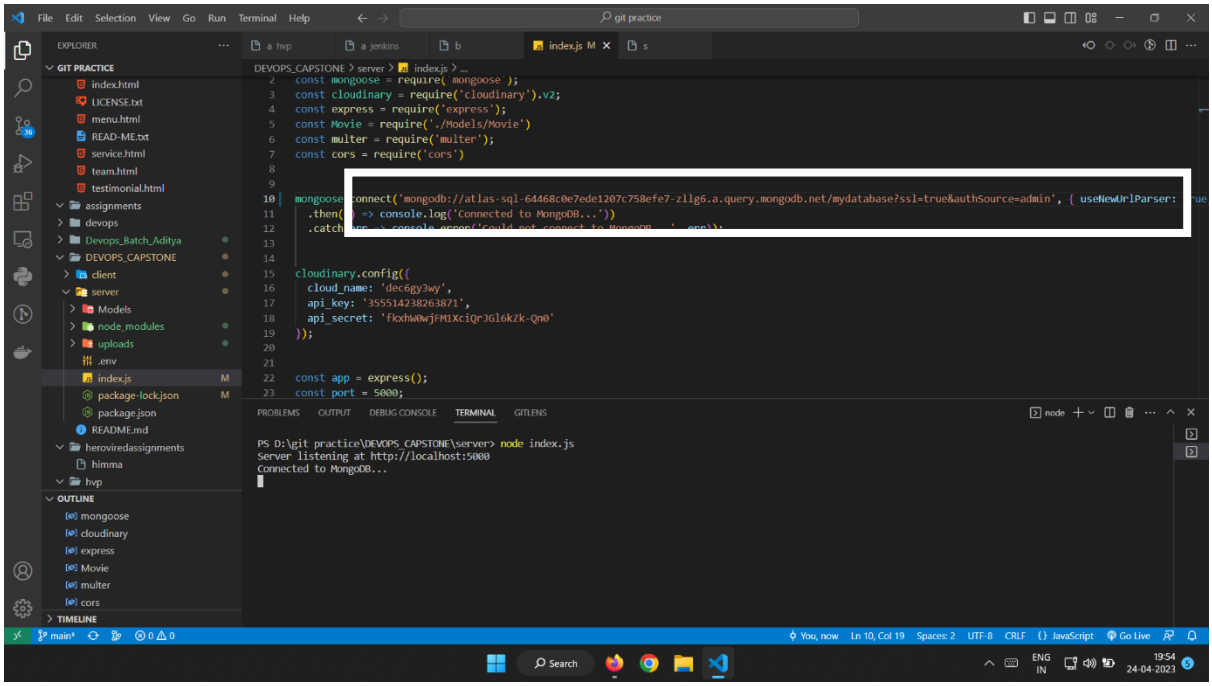
After creating the cluster successfully,then connect it using the link provided by the mongoDB Drivers.

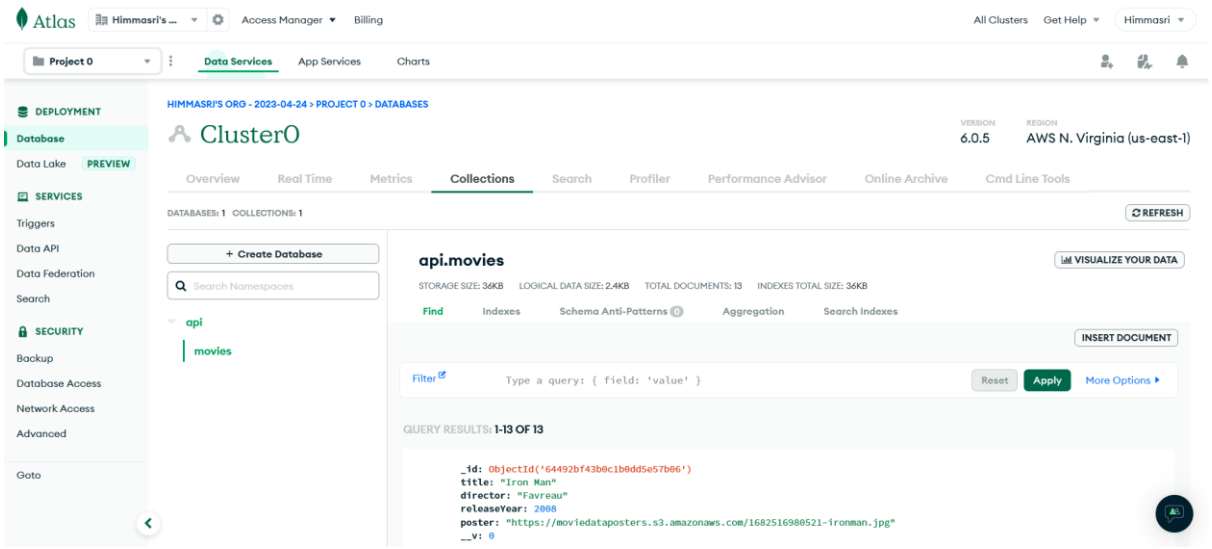


Now,copied the url and connect it to the backend



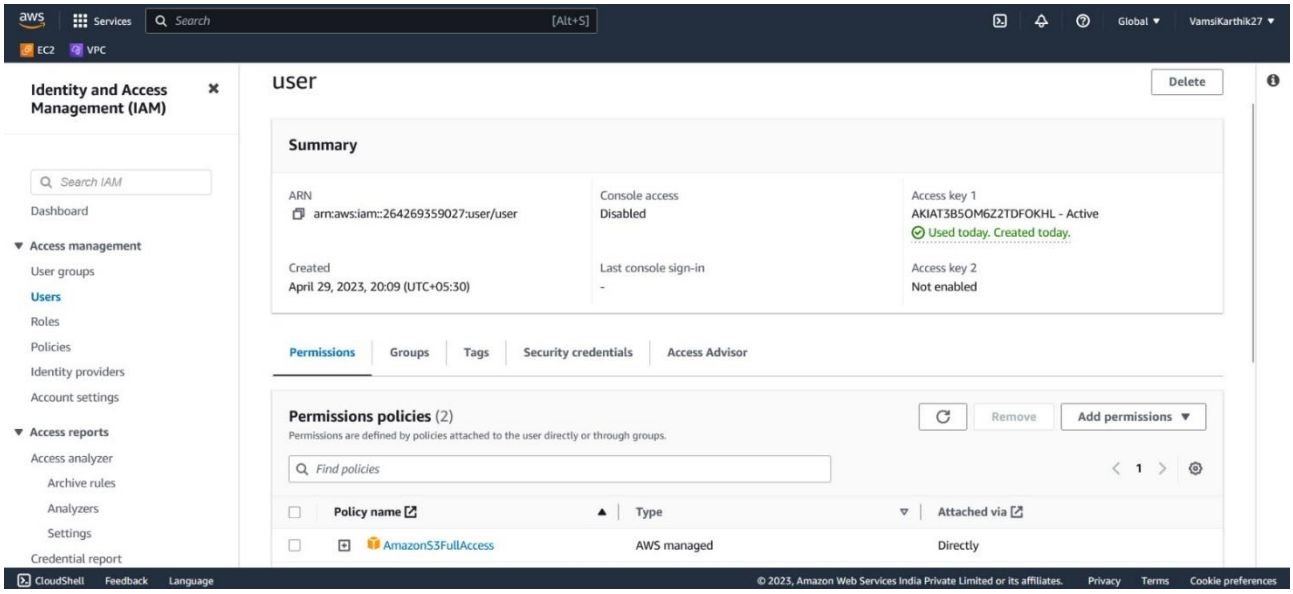
Now we have to connect the MongoDB Database to the backend using that link.



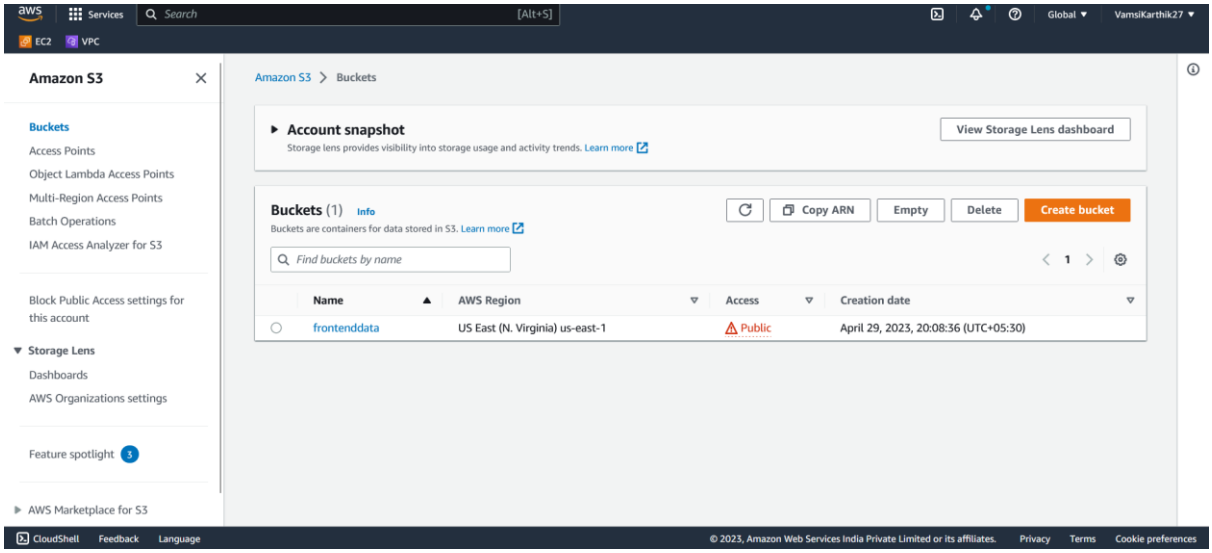


Step 3: Creating S3 bucket and connect to the bucket using S3 Multer library

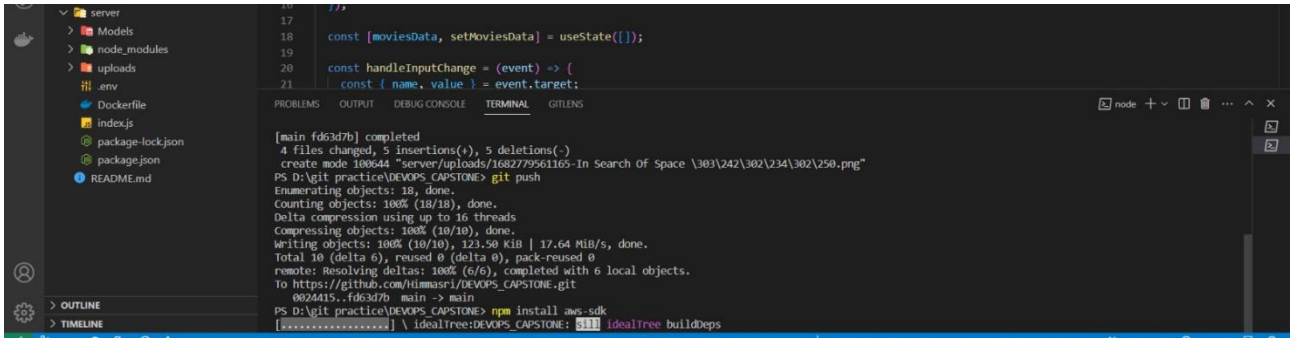
Before creating an S3 bucket, we have to create an IAM user and give S3 full access policy to that user.

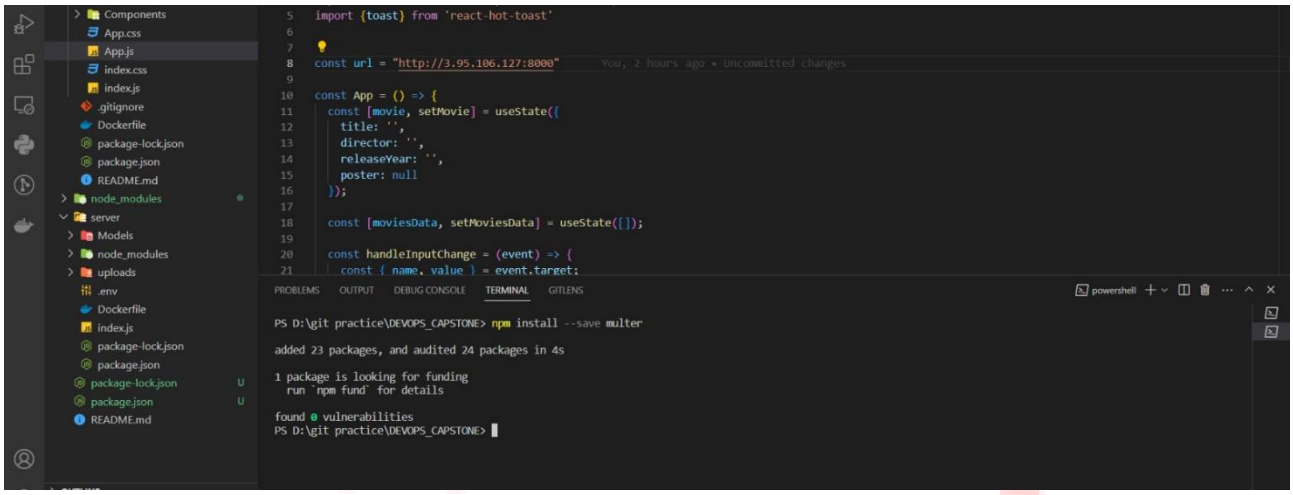


Create a S3 bucket to store the images which will be uploaded to the website.



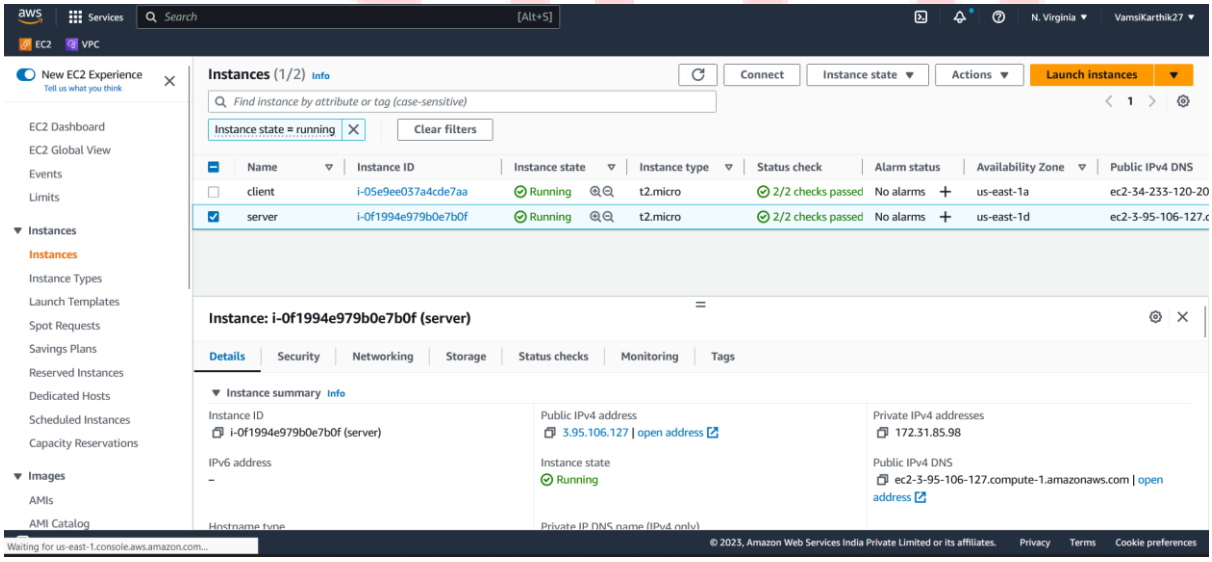
Changing the frontend and backend code to connect with the S3 bucket and MongoDB and installing AWS-SDK for supporting three runtimes: javascript for browser, Nodejs for server, React Native for mobile development, and installing Multer which is used to store uploaded files into the S3 bucket. It is a function that receives the request and response object to the request sends from the client side.



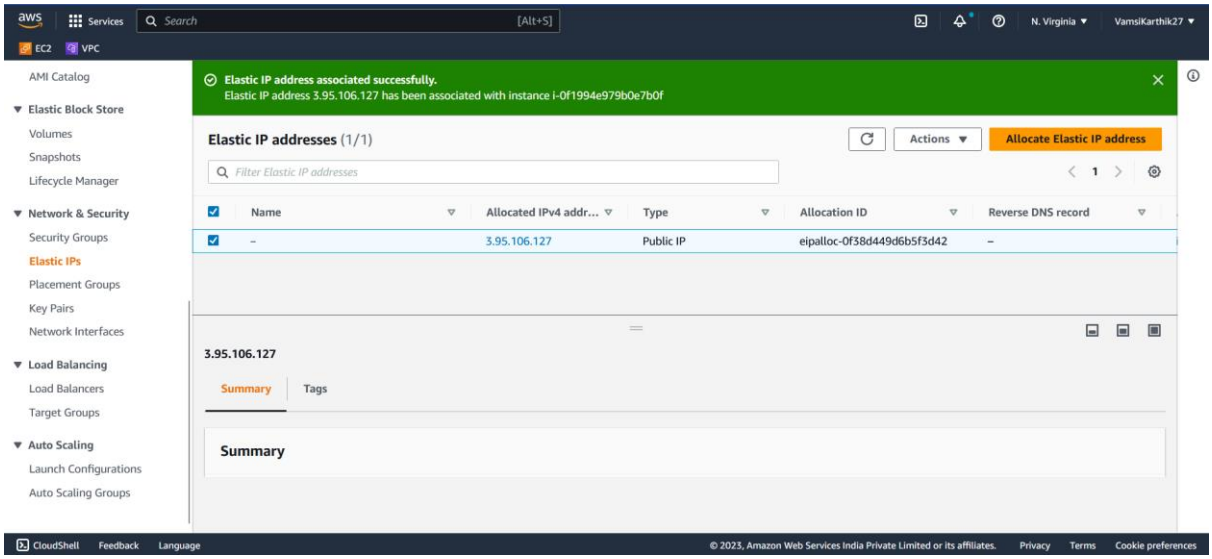


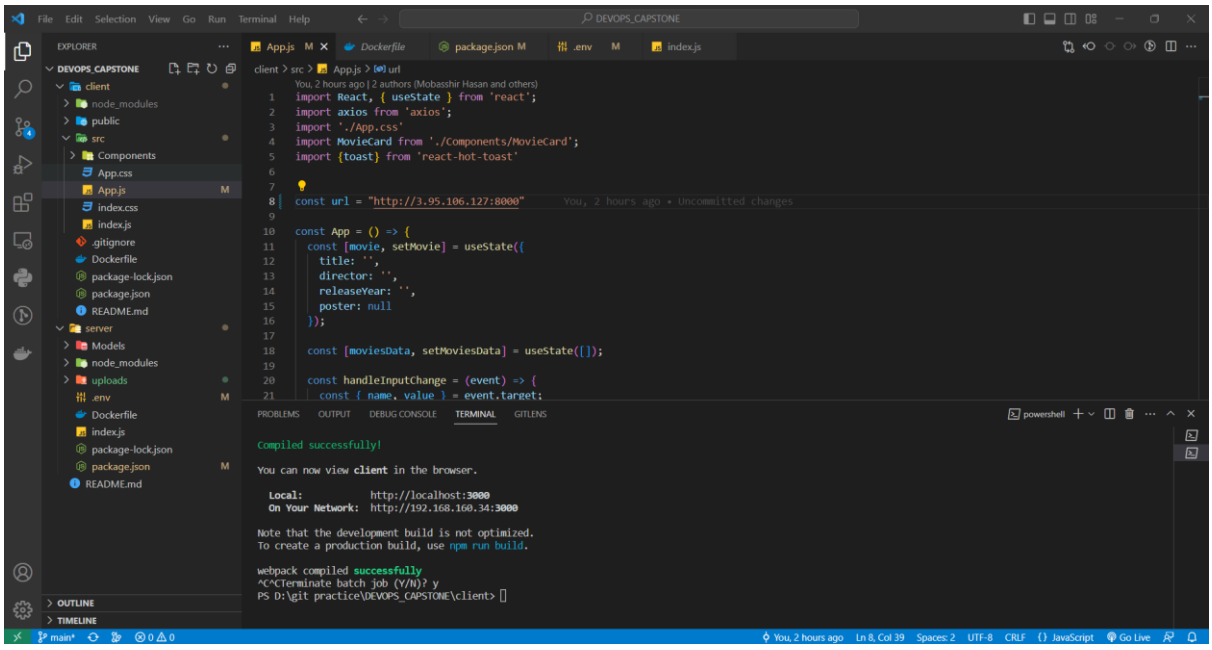
Step 4 : Launching Instances for both frontend and backend

Before containerizing we have to create instances for the server and client.Launch two instances for the server and client to deploy and connect them.



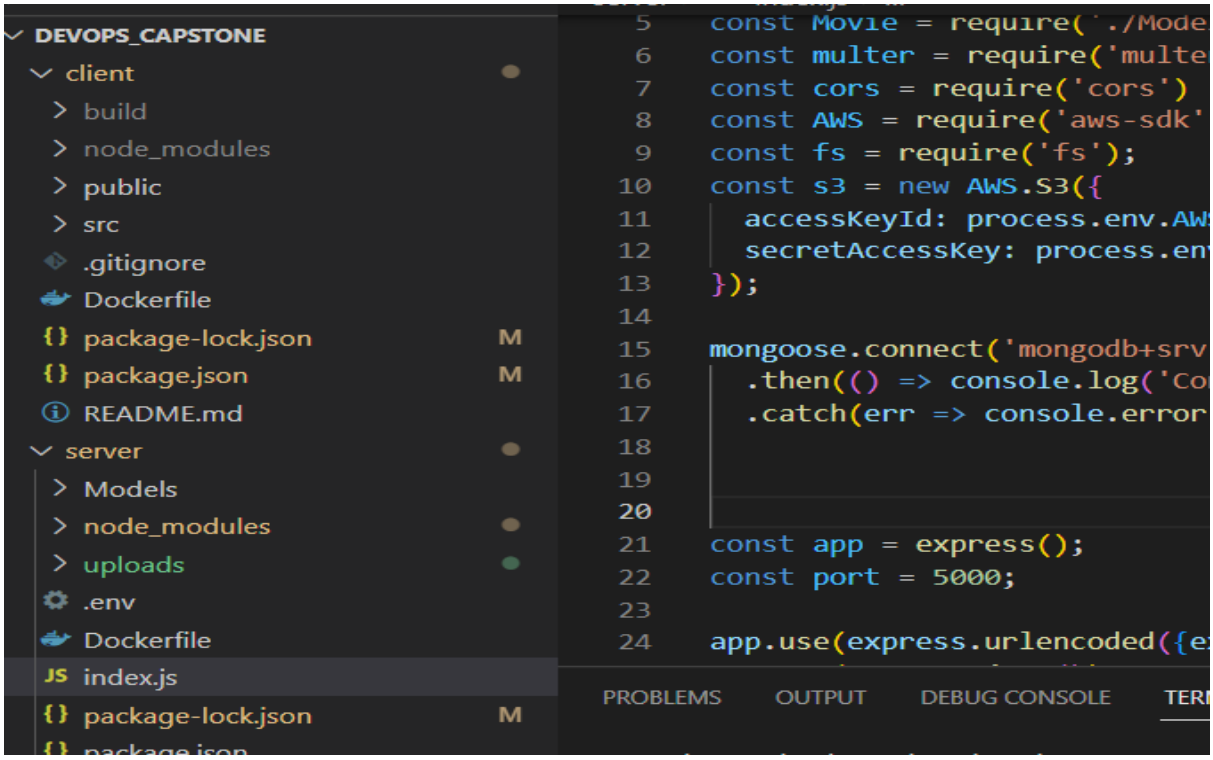
Providing Elastic IP addresses





Step 5 : Containerizing of the frontend and backend code using Dockerfile

Here we are creating docker files in both frontend and backend to convert it them into docker images. Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Docker can automatically build images by reading those instructions present in the docker file.



Server side DockerFile:

```
# Use an official Node.js runtime as a parent image
FROM node:14
# Set the working directory to /app
WORKDIR /app
# 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"]
```

Client side DockerFile:

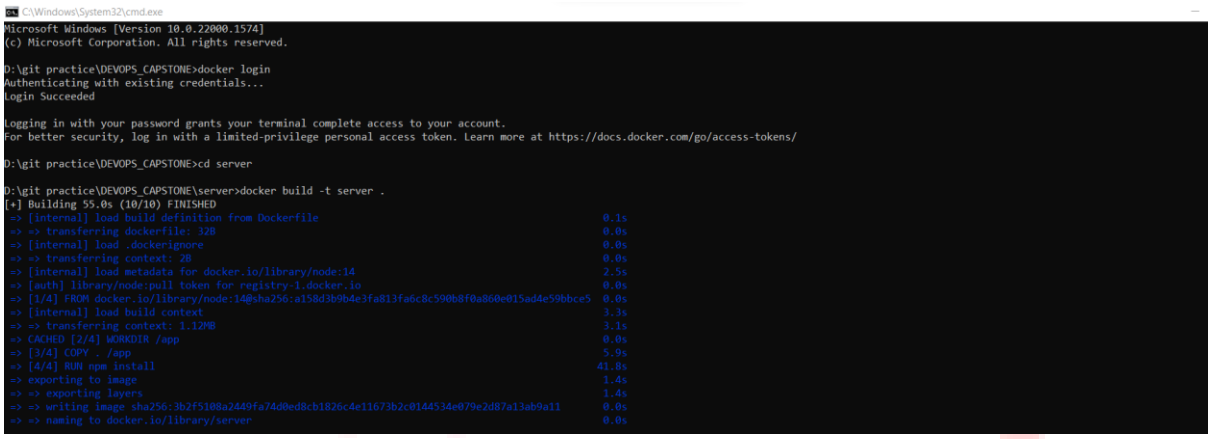
```
# Use an official Node.js runtime as a parent image
FROM node:14-alpine
# Set the working directory to /app
WORKDIR /
# Copy package.json and package-lock.json to the container
COPY package*.json ./
# Install dependencies
RUN npm install
# Copy the rest of the application code to the container
COPY . .
# Build the application
RUN npm run build
# Serve the application with a lightweight HTTP server
```

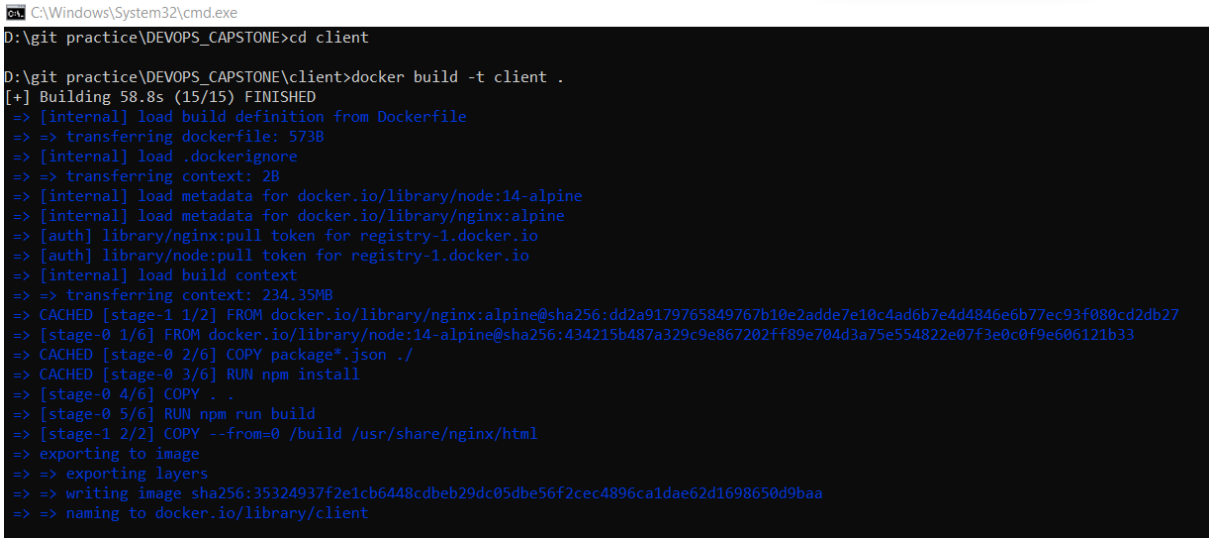
FROM nginx:alpine
COPY --from=0 /build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

Now we have to build images for both server and client using the “**docker build”** command

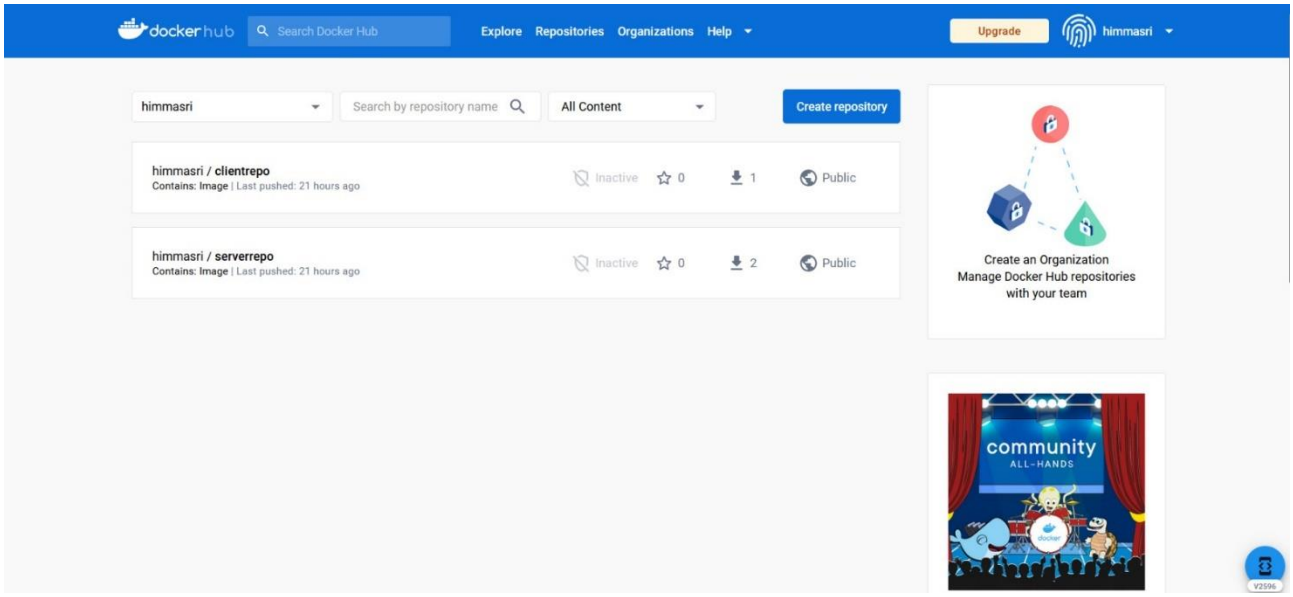
- **Docker build**: It builds Docker images from a Dockerfile and a “context”. A build's context is the set of files located in the specified PATH or URL.

Building an images for backend and frontend





Creating a repository to store the images.



Now we will push the images to the repository that we have created.

- **Docker push**: It is used to push or share a local Docker image or a repository to a central repository; it might be a public registry like https://hub.docker.com or a private registry or a self-hosted registry.

```
D:\git practice\DEVOPS_CAPSTONE\server>docker tag server:latest himmasri/serverrepo:latest

D:\git practice\DEVOPS_CAPSTONE\server>docker push himmasri/serverrepo:latest
The push refers to repository [docker.io/himmasri/serverrepo]
85319e113b20: Pushed
1d409ea365aa: Pushed
50b414c052eb: Pushed
0d5f5a015e5d: Pushed
3c777d951de2: Mounted from library/node
f8a91dd5fc84: Mounted from library/node
cb81227abde5: Mounted from library/node
e01a454893a9: Mounted from library/node
c45660adde37: Pushed
fe0fb3ab4a0f: Pushed
f1186e5061f2: Pushed
b2dba7477754: Pushed
latest: digest: sha256:1a20ef5c0a89d9180c7e440ceb63783338d4ad2db649983cf7d8196d0504014b size: 2845
```

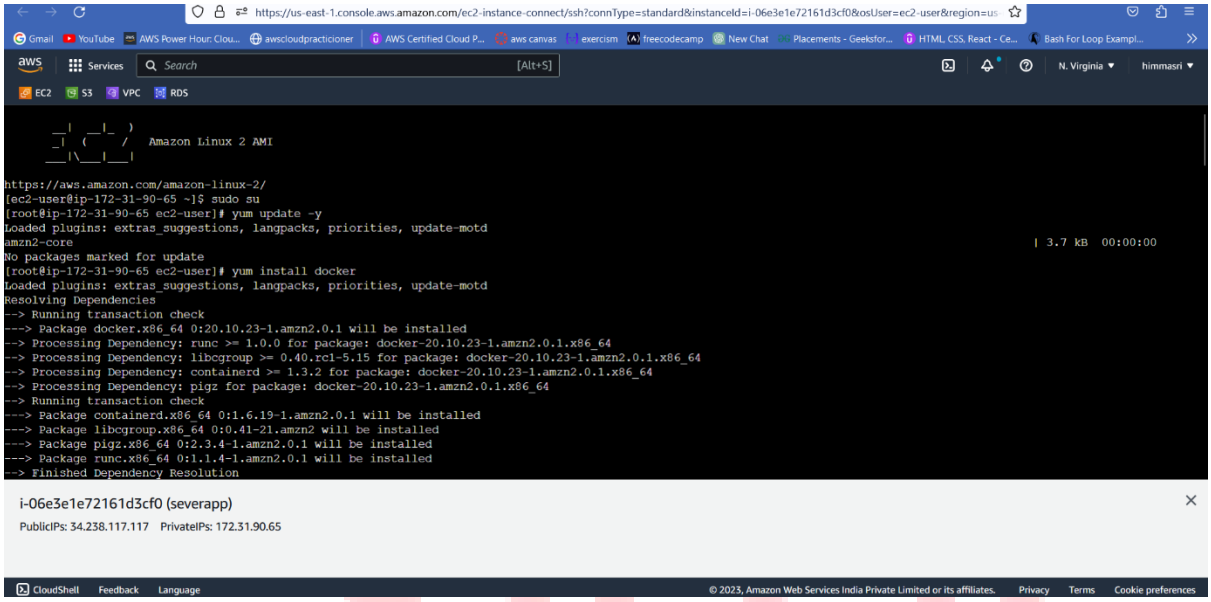
```
D:\git practice\DEVOPS_CAPSTONE\client>docker tag client:latest himmasri/clientrepo:latest

D:\git practice\DEVOPS_CAPSTONE\client>docker push himmasri/clientrepo:latest
The push refers to repository [docker.io/himmasri/clientrepo]
15f8768e751c: Pushed
31531248c7cb: Mounted from library/nginx
f9cb3f1f1d3d: Pushed
f0fb842dea41: Pushed
c1cd5c8c68ef: Pushed
1d54586a1706: Pushed
1003ff723696: Mounted from library/nginx
f1417ff83b31: Mounted from library/nginx
latest: digest: sha256:b9fe3baa3318cac04d70f735f2548f15c87dfac2e50de04b128279585a974fa7 size: 1991

D:\git practice\DEVOPS_CAPSTONE\client>docker images
REPOSITORY            TAG       IMAGE ID       CREATED         SIZE
himmasri/clientrepo   latest    35324937f2e1   2 hours ago     41.8MB
client                latest    35324937f2e1   2 hours ago     41.8MB
himmasri/serverrepo   latest    3b2f5108a244   2 hours ago     1.14GB
server                latest    3b2f5108a244   2 hours ago     1.14GB
hello-world           latest    feb5d9fea6a5   19 months ago   13.3kB
```

Now we will list all the images that we have still now using the command **docker ps**

```
D:\git practice\DEVOPS_CAPSTONE\client>docker images
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
himmasri/clientrepo     latest    35324937f2e1   2 hours ago      41.8MB
client                  latest    35324937f2e1   2 hours ago      41.8MB
himmasri/serverrepo     latest    3b2f5108a244   2 hours ago      1.14GB
server                  latest    3b2f5108a244   2 hours ago      1.14GB
hello-world             latest    feb5d9fea6a5   19 months ago    13.3kB

D:\git practice\DEVOPS_CAPSTONE\client>
```

Step 6 : Deploying client and server on EC2 instances

We have to pull images from the corresponding repositories. So before pulling the images, we have to update the system and install git and docker.



Now, login into docker through the username and password using the docker login command and then pull the images into the instances using the docker pull

Now run the docker images in the corresponding instances and it shows that it was connecting to MongoDB

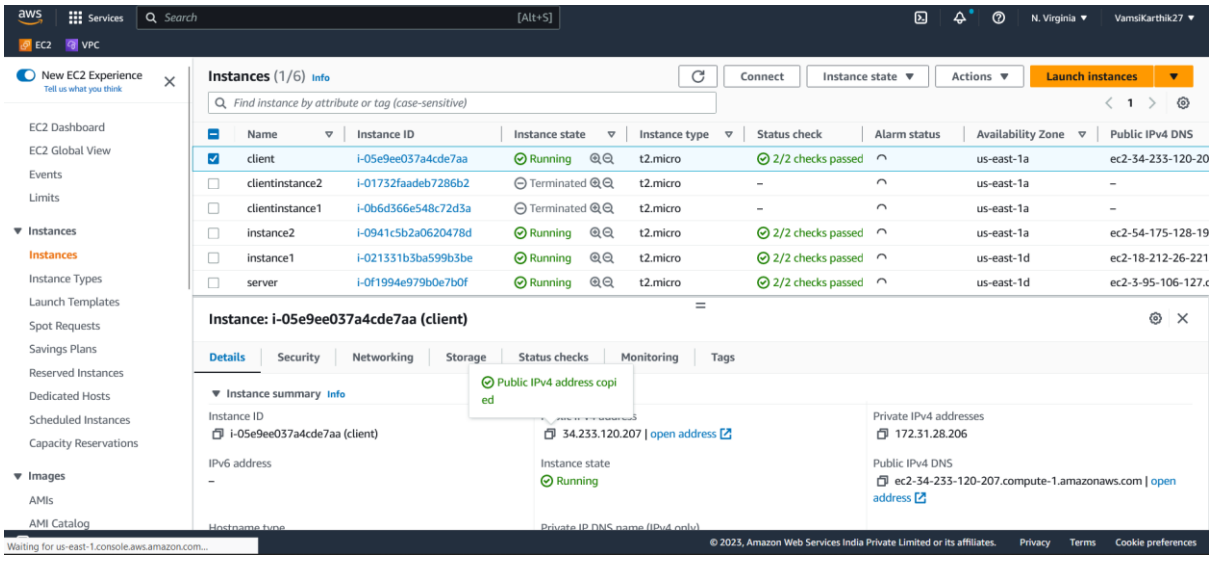
```
EC2  S3  VPC  RDS

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

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-88-78 ~]$ sudo su
[root@ip-172-31-88-78 ec2-user]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ip-172-31-88-78 ec2-user]# docker login
Authenticating with existing credentials...
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-172-31-88-78 ec2-user]# docker pull himmasri/client
Using default tag: latest
Error response from daemon: manifest for himmasri/client:latest not found: manifest unknown: manifest unknown
[root@ip-172-31-88-78 ec2-user]# docker pull himmasri/clientrepo
Using default tag: latest
latest: Pulling from himmasri/clientrepo
f56be85fc22e: Pull complete
2ce963c369bc: Pull complete
59b9d2200e63: Pull complete
3e1e579c95fe: Pull complete
```

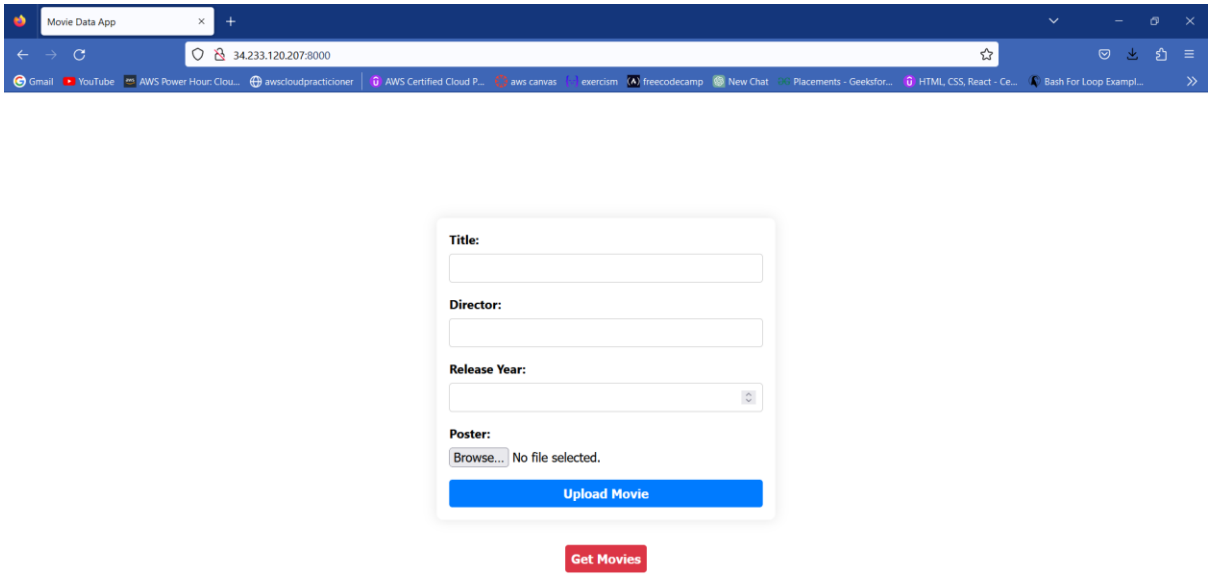
```
[root@ip-172-31-22-96 ec2-user]# docker pull himmasri/serverrepo
Using default tag: latest
latest: Pulling from himmasri/serverrepo
2ff1d7c41c74: Already exists
b253aeafeaa7: Already exists
3d2201bd995c: Already exists
1de76e268b10: Already exists
d9a8df589451: Already exists
6f51ee005dea: Already exists
5f32ed3c3f27: Already exists
0c8cc2f24a4d: Already exists
0d27a8e86132: Already exists
ded92201424e: Pull complete
0e764d15e393: Pull complete
9dbe5c31b7e8: Pull complete
Digest: sha256:1a20ef5c0a89d9180c7e440ceb63783338d4ad2db649983cf7d8196d0504014b
Status: Downloaded newer image for himmasri/serverrepo:latest
docker.io/himmasri/serverrepo:latest
[root@ip-172-31-22-96 ec2-user]# docker images
REPOSITORY                  TAG       IMAGE ID       CREATED        SIZE
himmasri/serverrepo         latest    3b2f5108a244   16 hours ago   1.14GB
umakantipudi/capserver-repo   latest    f4a7228e5e18   25 hours ago   1.14GB
[root@ip-172-31-22-96 ec2-user]# docker run -d -p 8000:5000 himmasri/serverrepo
7897d75402ac386f9d833382f13bbdec693787d13f742bfb17b8da60cd1103af
[root@ip-172-31-22-96 ec2-user]#
```

i-0beb909a16f42fd83 (server)

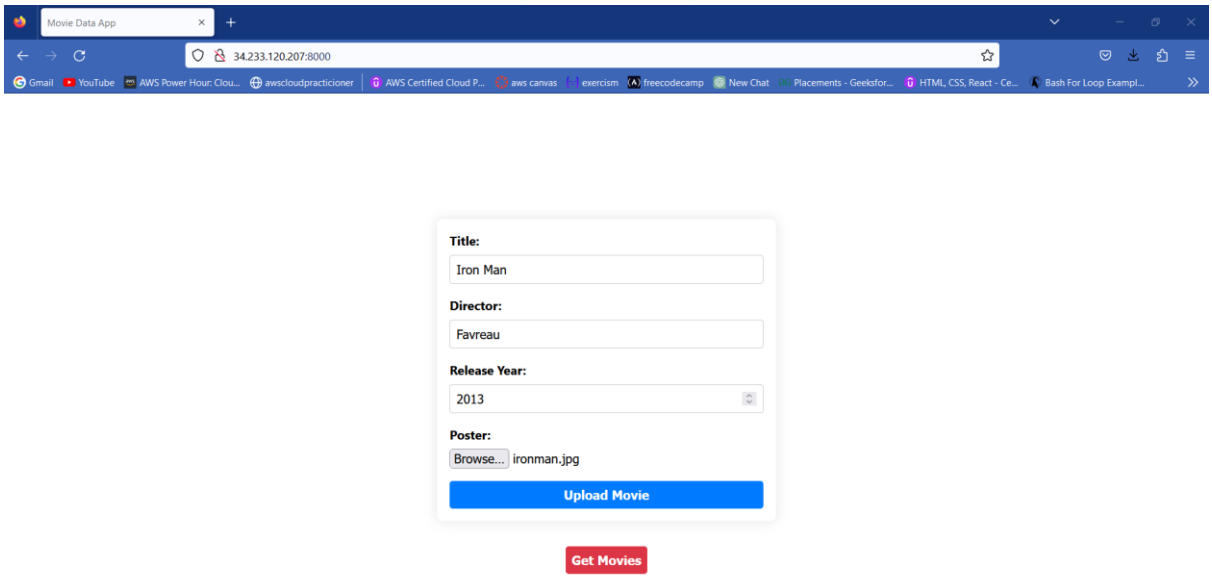
Step 7: Checking whether movies are uploading or not and also getting movies when we click on get movies button or not

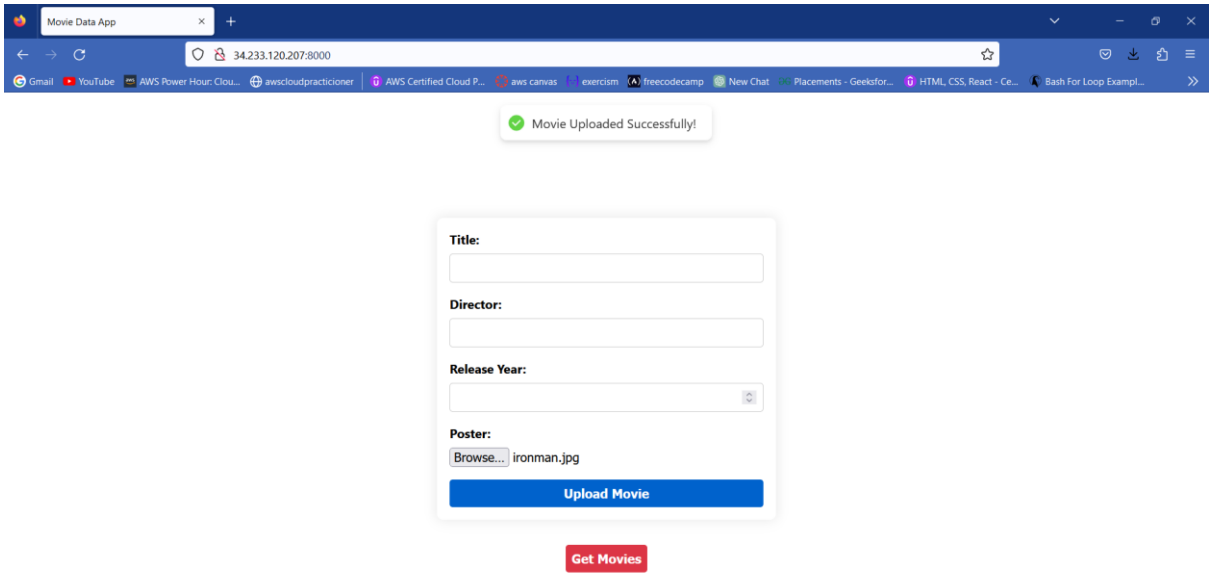
After deploying to check the website we use the public IP address of the Client Instance along with the listening port number.

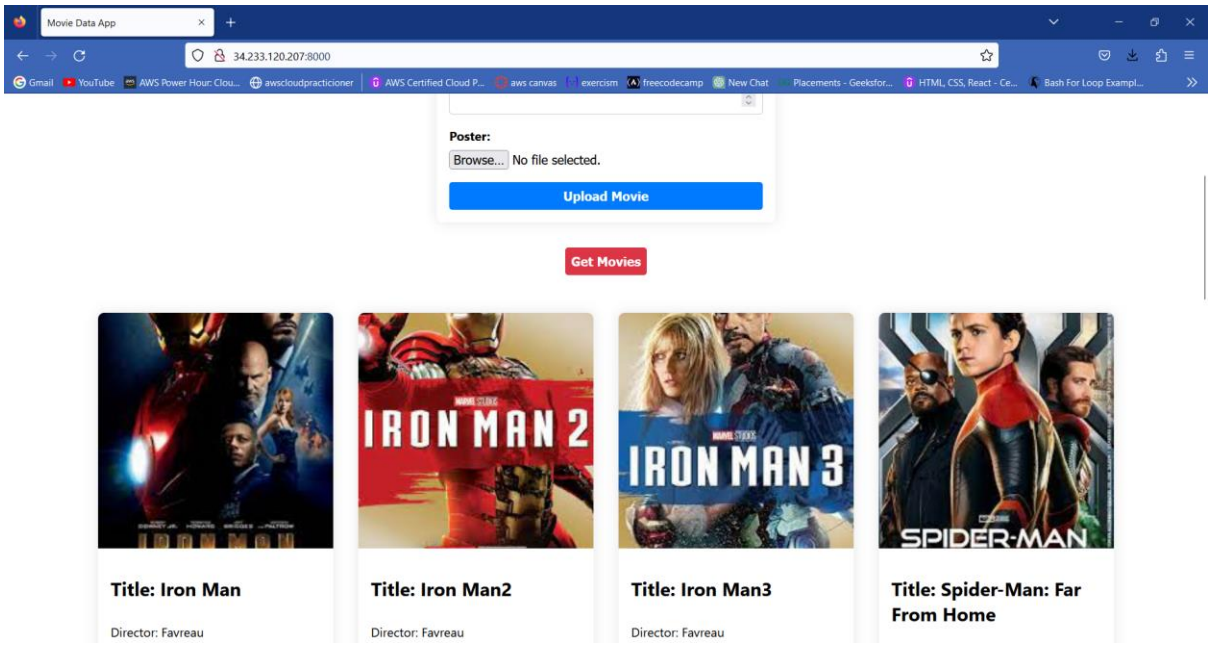




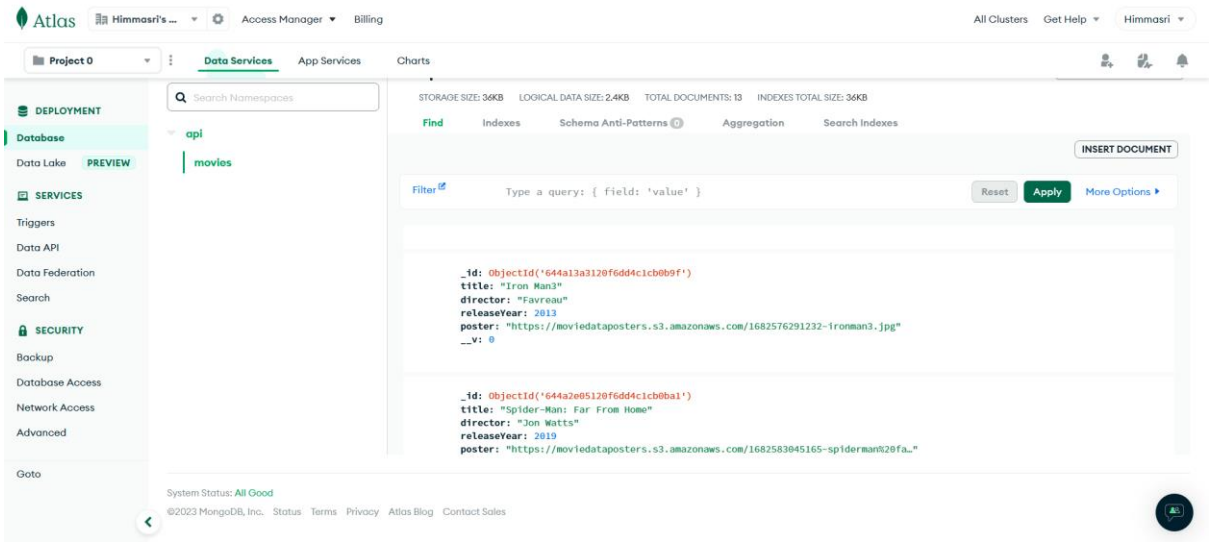
Now we will enter the details in the displayed web page and submit the details. The details are uploaded to the MongoDB server and photos are uploaded to the attached S3 bucket.





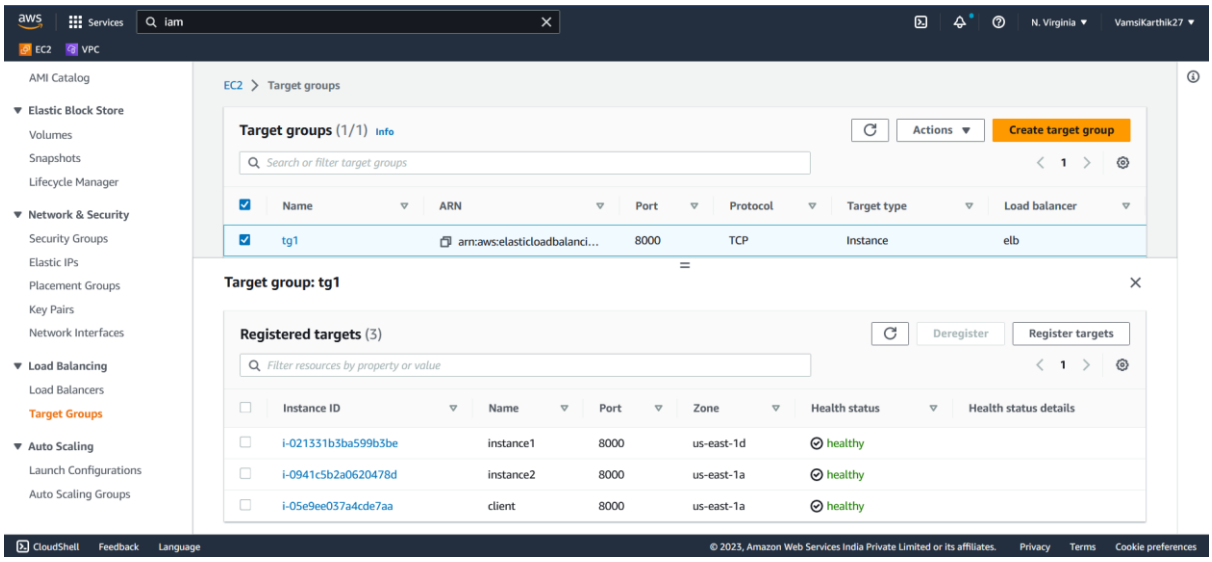


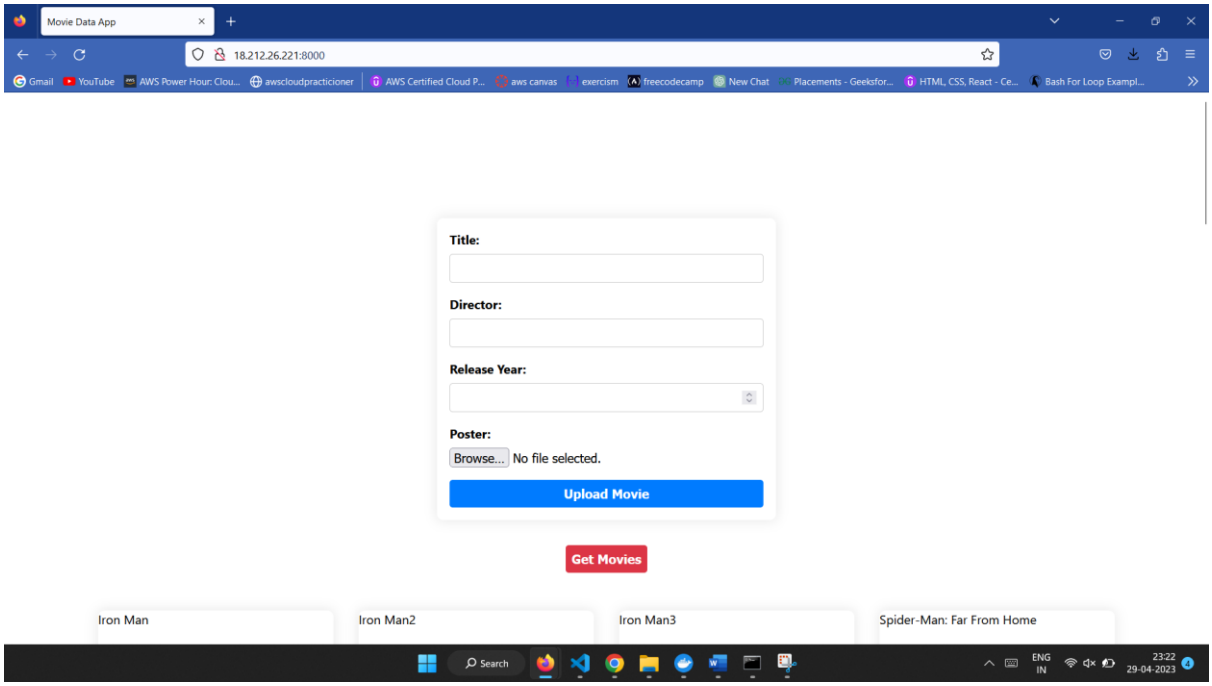
Here the uploaded images are reflected in the S3 bucket and also in the MongoDB Atlas



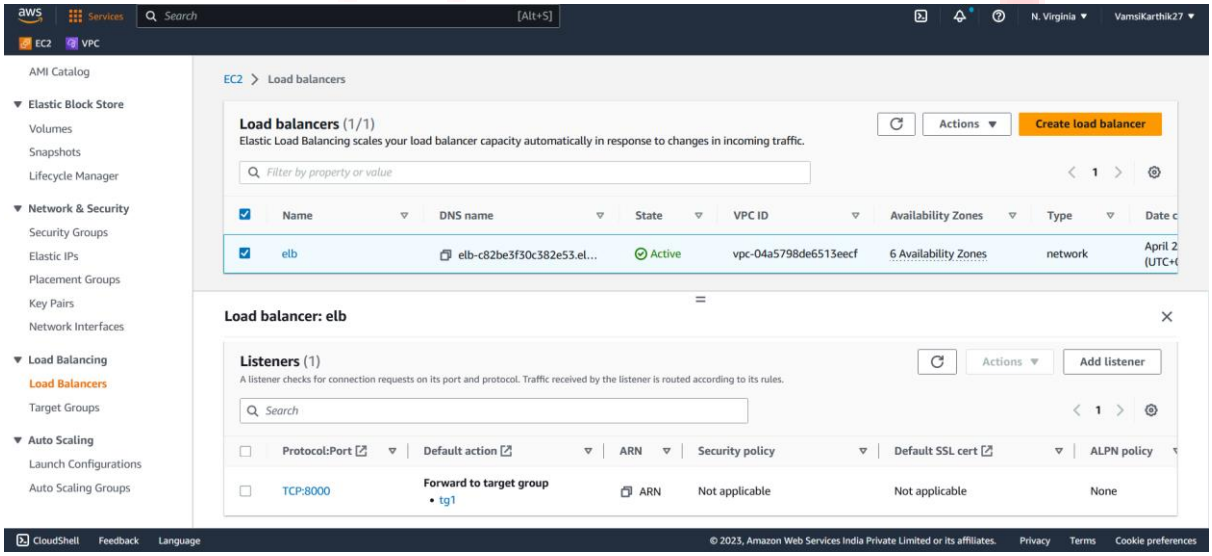
Step 8: Creating of Target group and Load Balancer

Now, create a target group that is used to tell the load balancer where to direct traffic to instances, fixed IP addresses, and AWS Lambda functions among others.





Create an Elastic Load balancer which is used to manage the load on the instances and it will increase the capacity based and reliability of applications. Creating a load balancer we can create one or more listeners and configure their rules to direct the traffic to the target group.



Title:

Director:

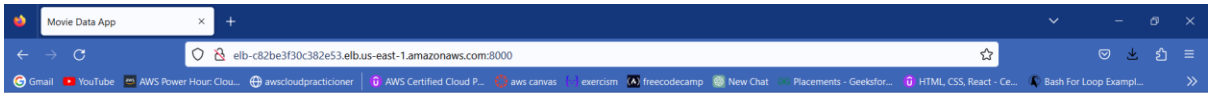
Release Year:

Poster:

Browse... No file selected.

Upload Movie

Get Movies



Title:

Avengers: Endgame

Director:

anthony russo

Release Year:

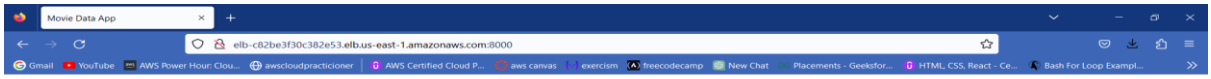
2020

Poster:

Browse... avengers end game.jpg

Upload Movie

Get Movies

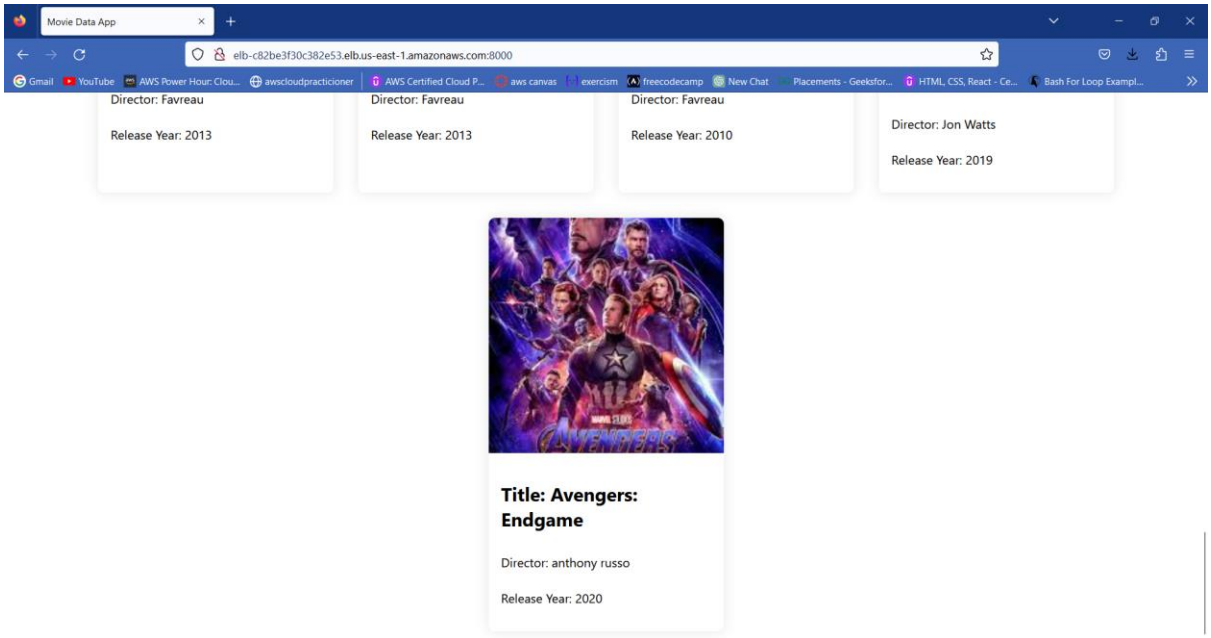


Movie Uploaded Successfully!

Title:

Director:

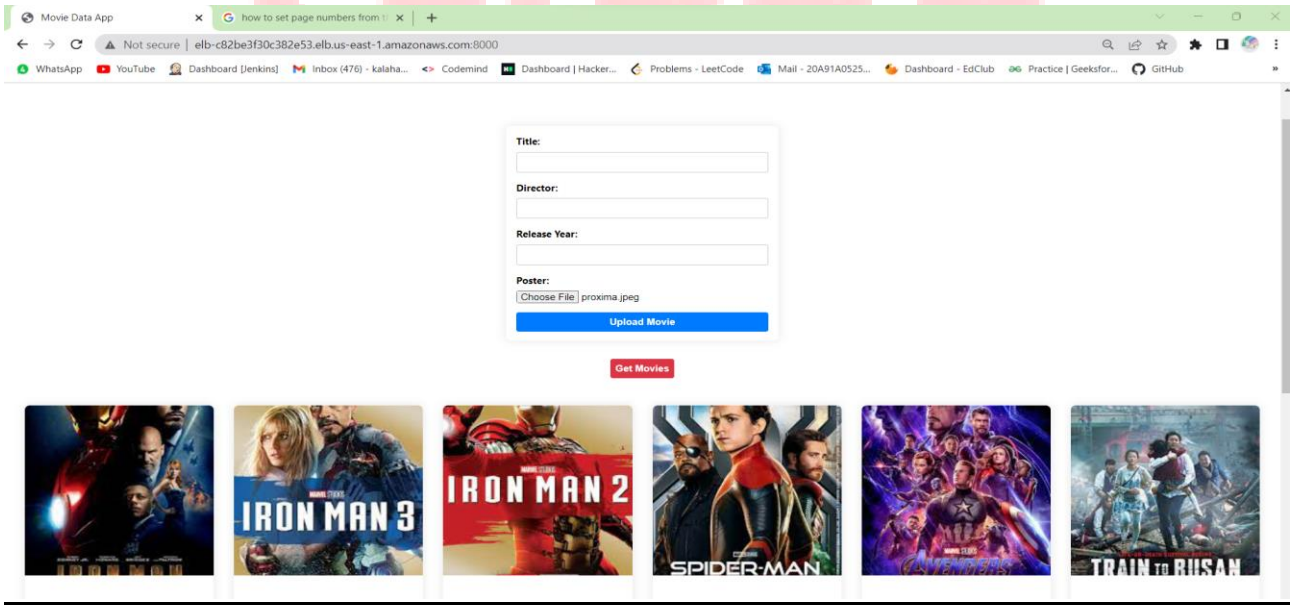
Release Year:



Step 9 : Providing DNS

After the creation of the load balancer, we are able to get a DNS name which is used to turn domain names into IP addresses, which allow browsers to get to websites and other internet resources.

"http://elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000/"



REFERENCE LINK

Frontend

34.233.120.207:8000

Backend

3.95.106.127:8000

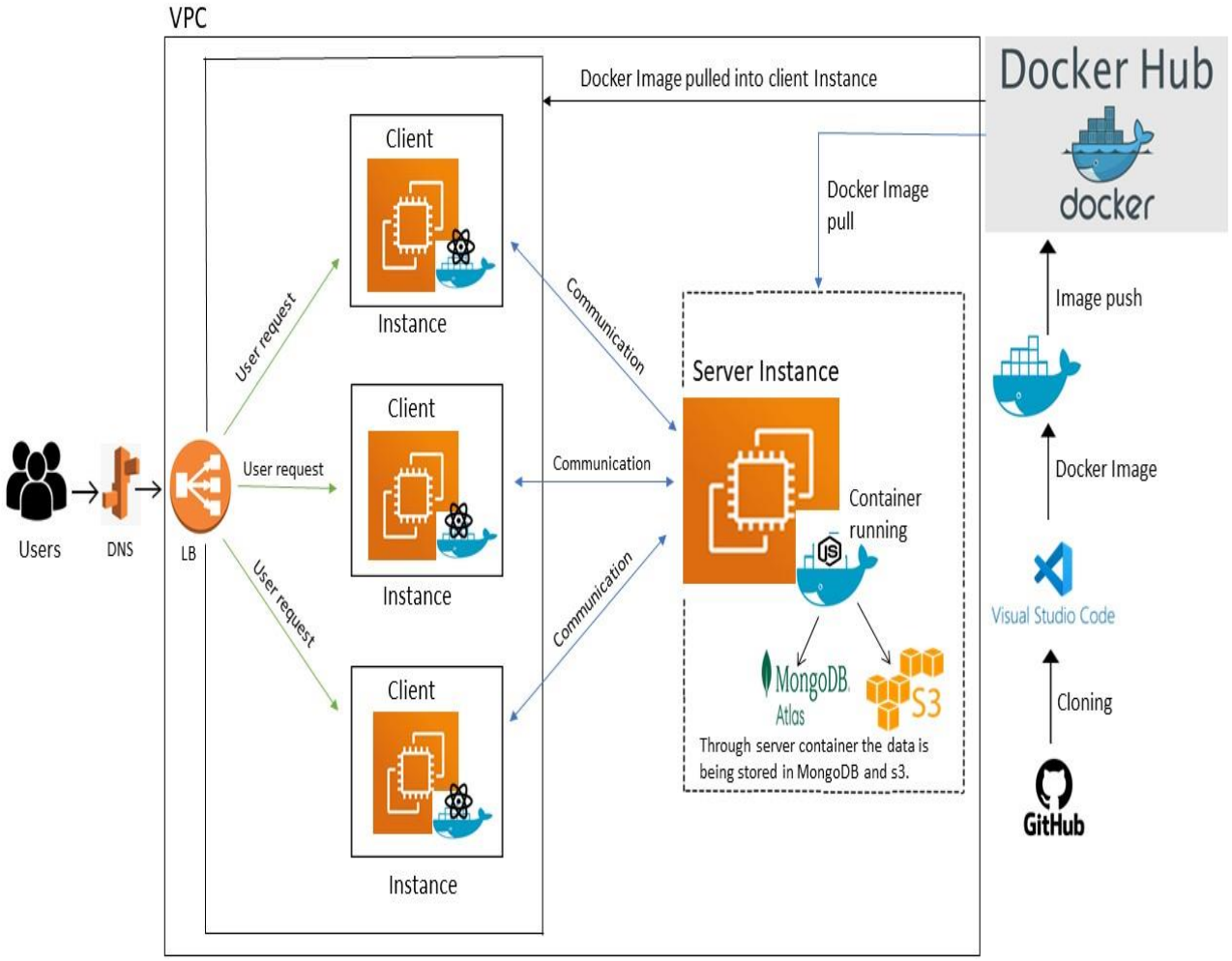
DNS

elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000/

Git-Repository Link

https://github.com/Himmasri/DEVOPS_CAPSTONE

DEPLOYMENT DIAGRAM



CONTRIBUTION OF TEAM MEMBERS

Himma	• Creation of IAM role and configuring the S3 bucket in Aws, Database creation in MongoDB Atlas
Venkat & Karthik	• Building the multer S3 code snippet to store poster into bucket and connecting the server to the MongoDB cluster to store data
Sahasra	• Docker file creation and Image build
Uma	• Launching of Ec2 Instances and deployed front and back end containers
Yeswanth	• Providing proper load balance to the application
Uma & Himma	• Deployment of "Movie Listing" application Architecture
Bhumika & Sahasra	• Project Documentation

CONCLUSION

As a conclusion of this project,we successfully deployed the Movie Listing web application using docker and EC2 instances.This web application includes ReactJS as a frontend and NodeJS as a backend and used multer for uploading files into S3 bucket and used AWS services to deploy the application.This application gives the movie details that are present in the database and also we can upload movies into it.

For uploading movies we used MongoDB Atlas as a database and to store the posters we used S3 multer to upload it to the S3 buckets.Deployed frontend and backend using EC2 instances using Docker and attached an Elastic IP address to that instances and modified the code in the frontend to fetch the data from the backend.In the frontend,we have title,director,movie released year and poster fields.And it has two functions upload movies and get movies.Using upload movies,user can able to upload movies into it.And when the user clicks on Get Movies button the user will get details of all the movies which are stored in mongoDB database and images are fetched from S3 bucket.

Those uploaded poster and movie details will be reflected in S3 bucket and also MongoDB Atlas database.All the details that are present in MongoDB and S3 bucket will be reflected on to the frontend pages as