**DevOps**

**CAPSTONE PROJECT**

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

Team Members:

B. Harsha srivalli-20A91A04D3

g.swamy-21mh5a0363

M. Kalpana chawla-20A91A04F3

suryakanthi-20A91A04J3

K. Yaswanth -21A95A0419

T. Ramesh-20P31A04B0

Sai vinay-20P31A04J1

N.Shashank-20A91A04A1

**CONTENT**

|  |  |
| --- | --- |
| **CHAPTER** | **PAGE NO** |
| **PURPOSE OF THE PROJECT & SCOPE OF THE PROJECT** | **1** |
| **TECHNOLOGIES USED IN THIS PROJECT** | **2** |
| **MAJOR TOOLS THAT ARE UTILIZED IN THE PROCESS OF IMPLEMENTATION** | **3** |
| **IMPLEMENTATION** | **4** |
| **DEPLOYMENT DIAGRAM** | **16** |
| **CONCLUSION** | **17** |

**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.

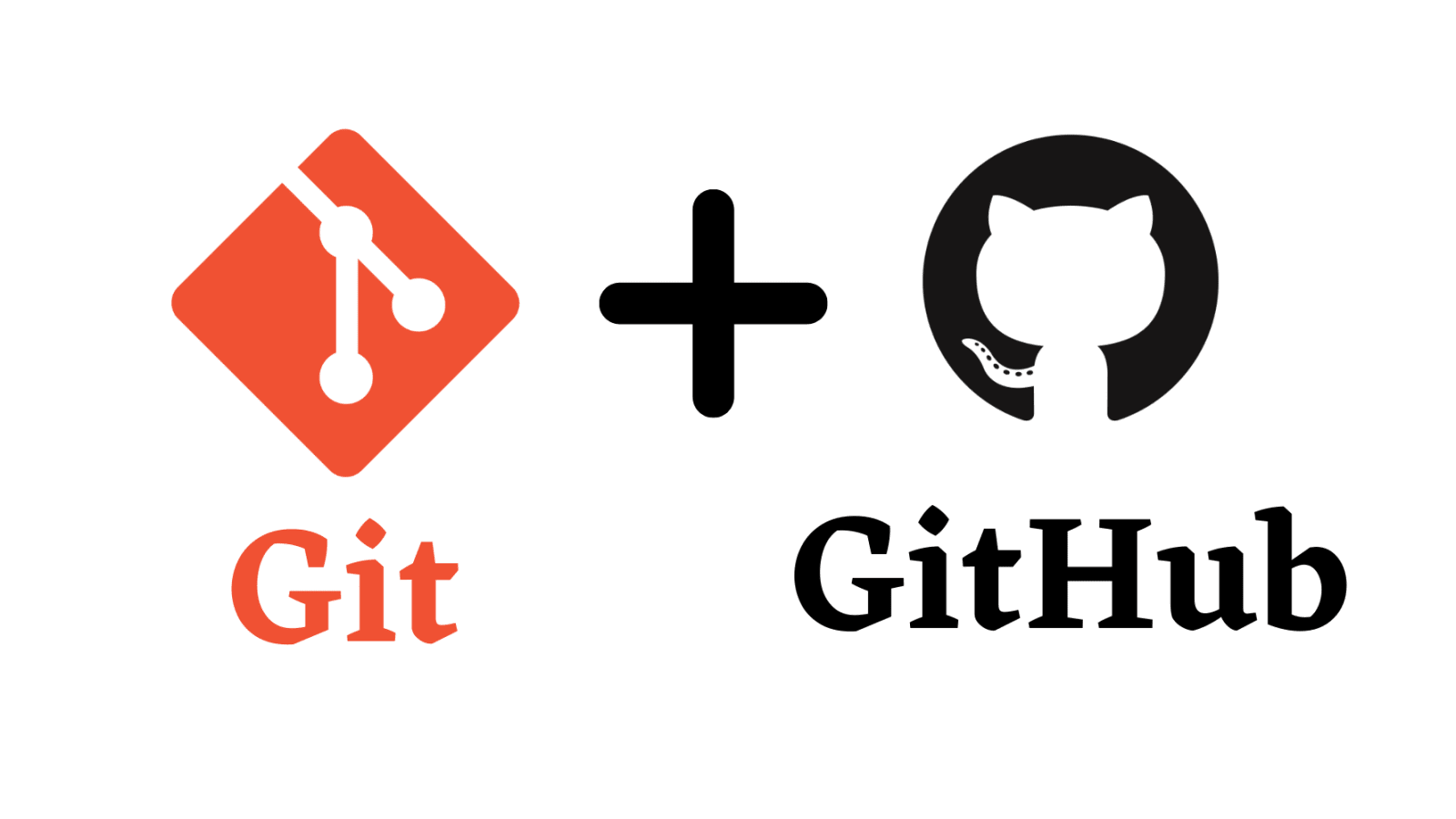
**TECHNOLOGIES USED**

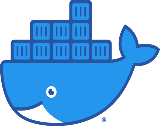
**Aws** is the world's most comprehensive and broadly adopted cloud, offering over 200 fully featured services from data centres globally. over 200 services, including computing, storage, databases, analytics, machine learning, IoT, and security. It is flexible, cost-effective, and reliable. And some 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.

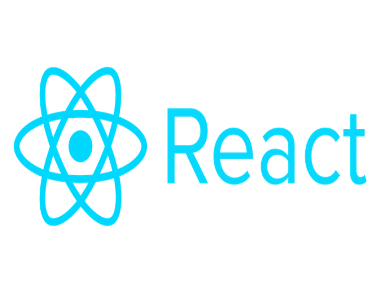
**S3 Multer** is a node. js middleware for handling multipart/form-data , which is primarily used for uploading files. It is written on top of busboy for maximum efficiency. multer-s3 : Streaming multer storage engine for AWS S3.

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

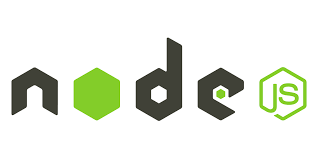
 

**TECHNOLOGIES STACK**

**Frontend Database**

 ****

**Backend**

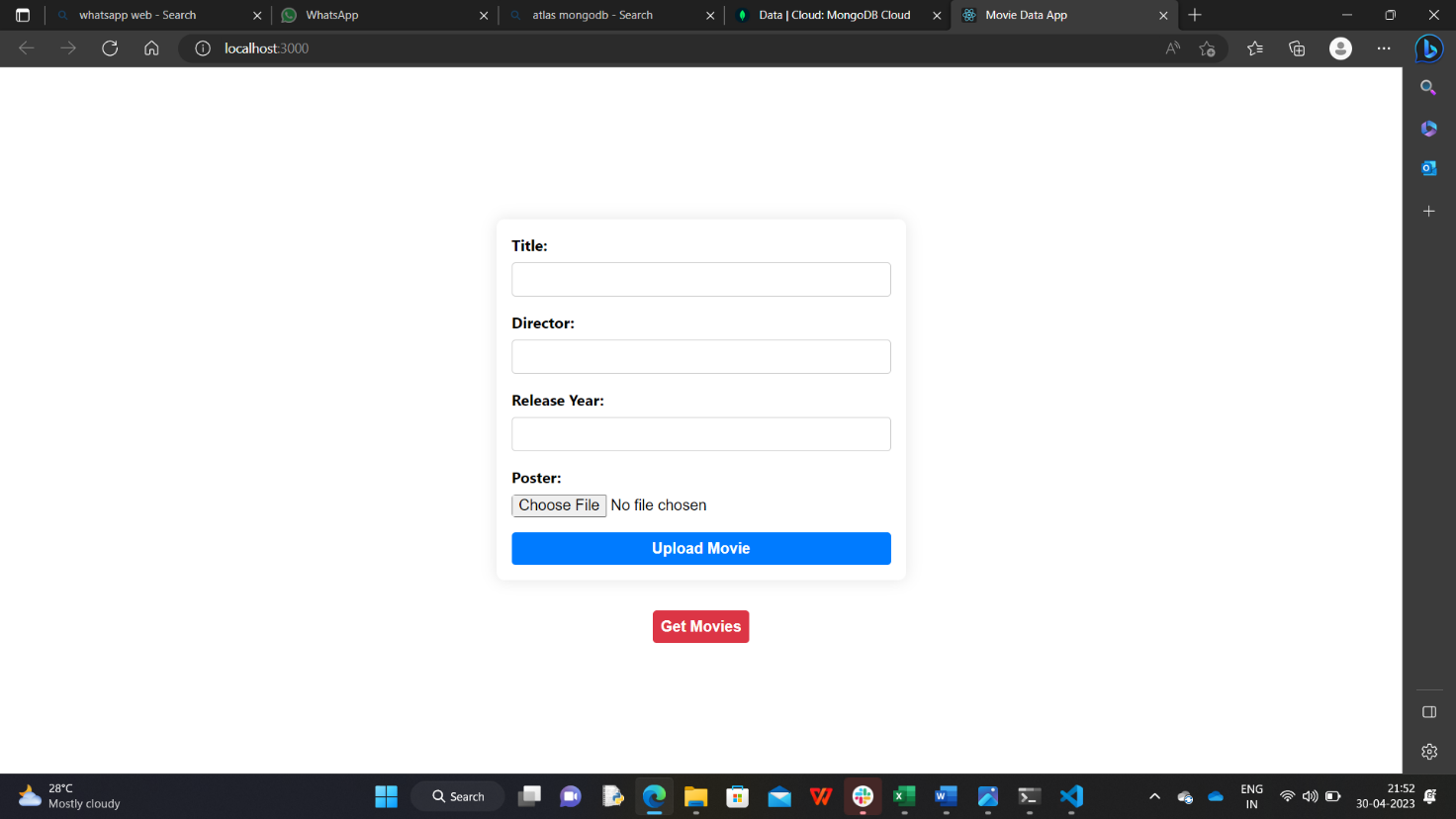


**IMPLEMENTATION**

**Step 1 : Cloning the web application repository from the GitHub**

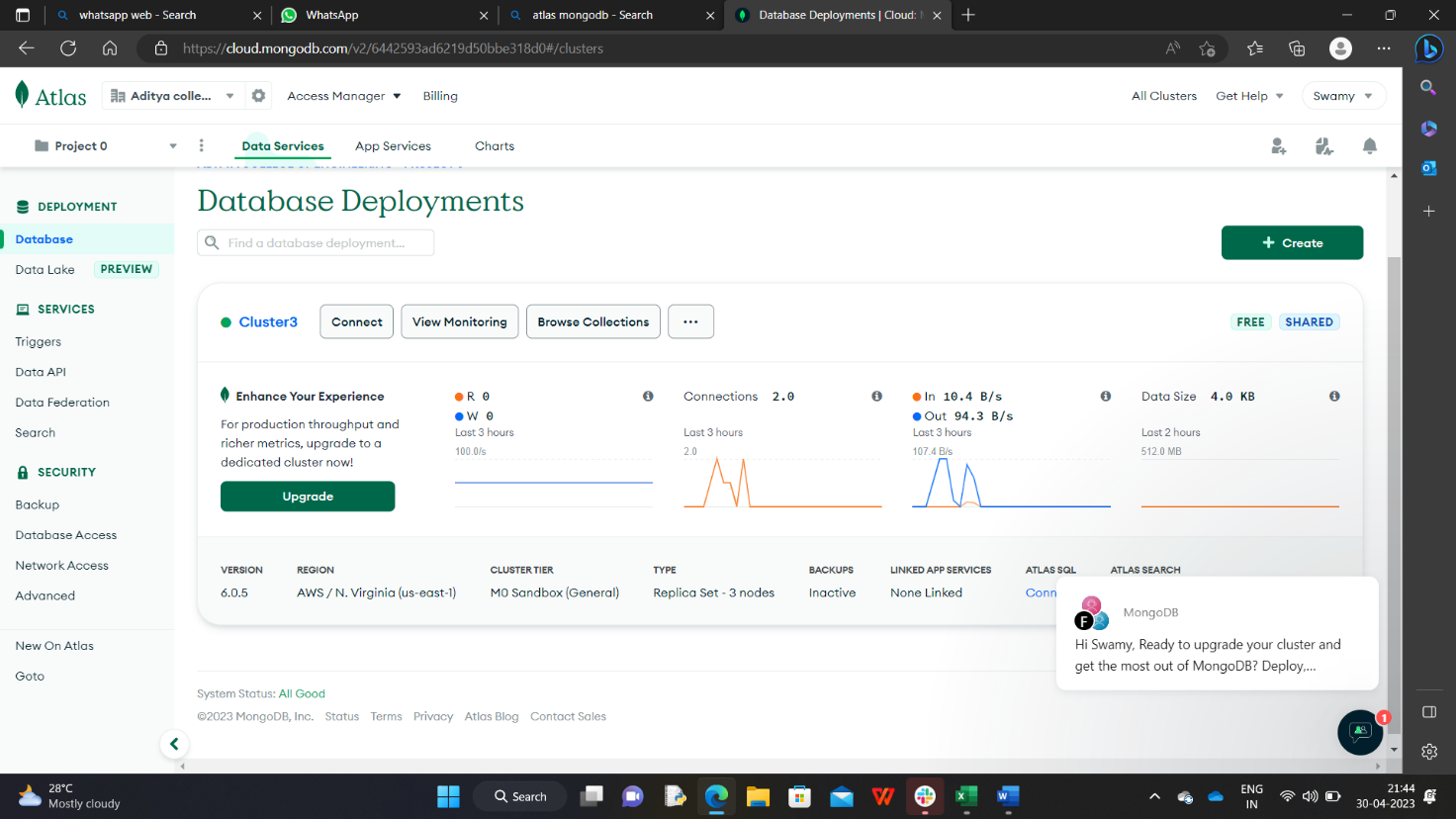
Initially cloned the given Movie Listing web application repository. And deployed the frontend. And the repository link was:

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

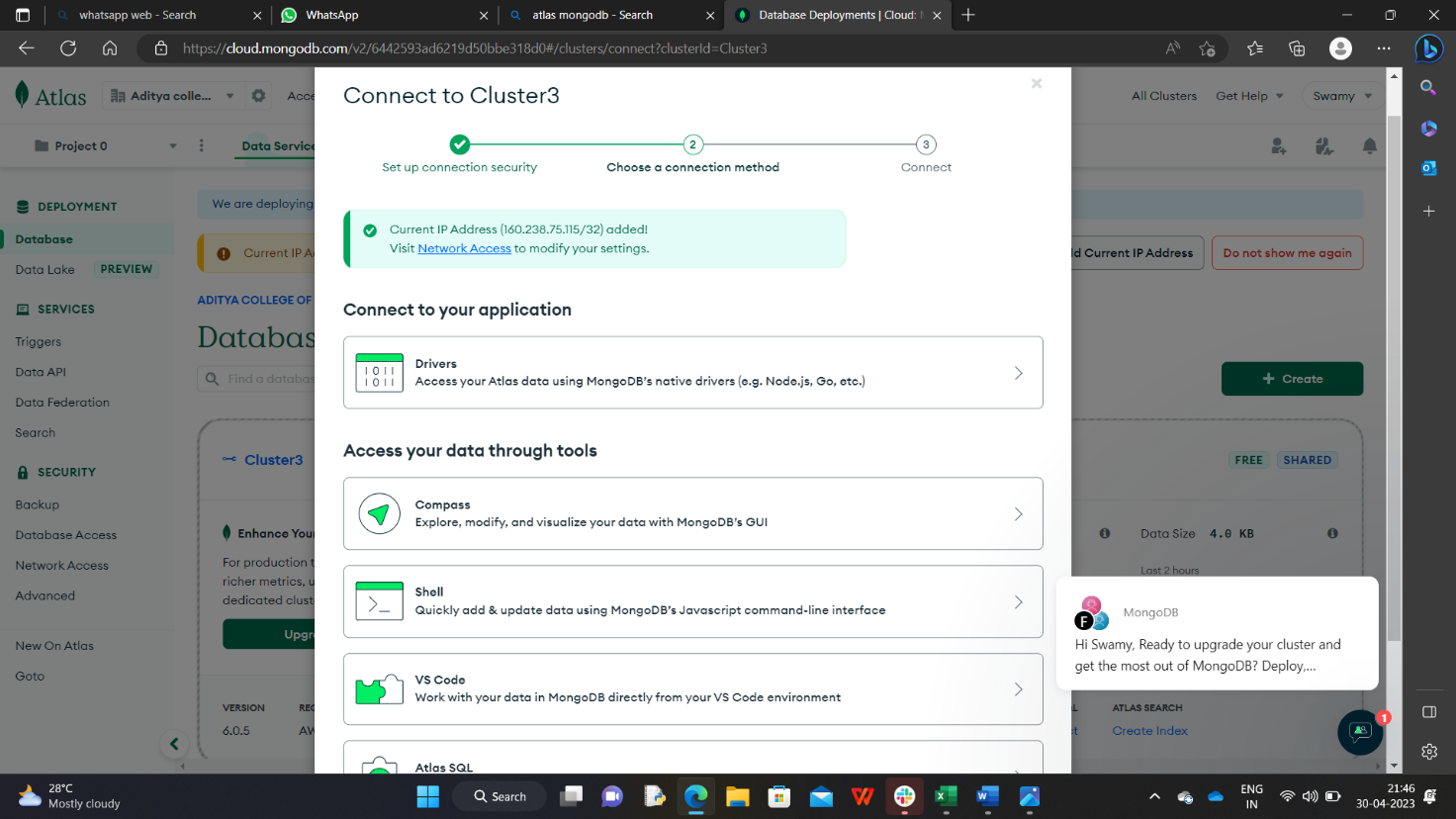


**Step 2**: **Create a cluster in mongoDB atlas to store the uploaded data and data collections that we have about the movies**

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



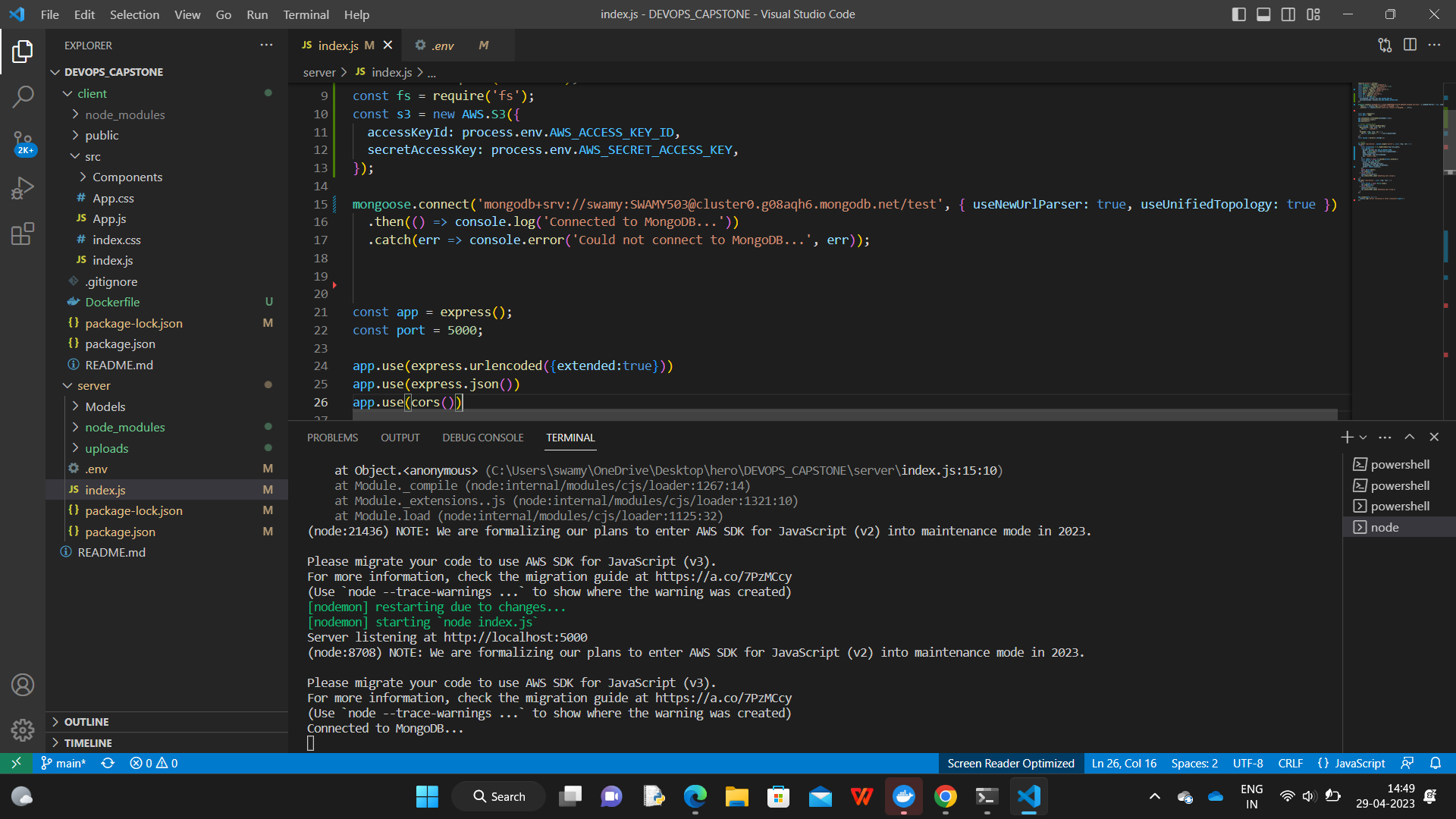
After creating the cluster successfully,then connect it using the link provided by the mongoDB Drivers.



Now,copied the url and connect it to the backend



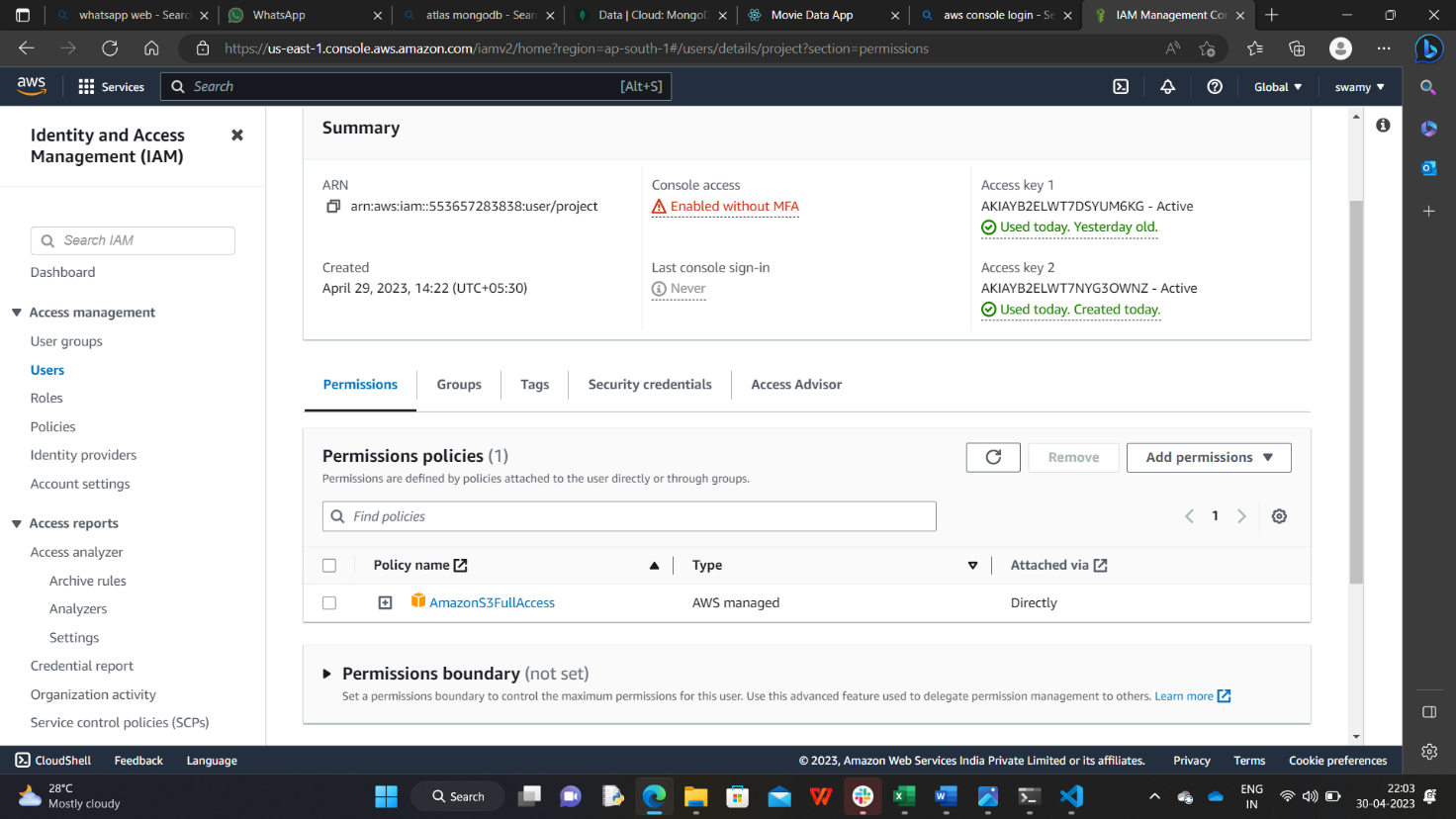
Now we have to connect the MongoDB Database to the backend using that link.



**`**

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

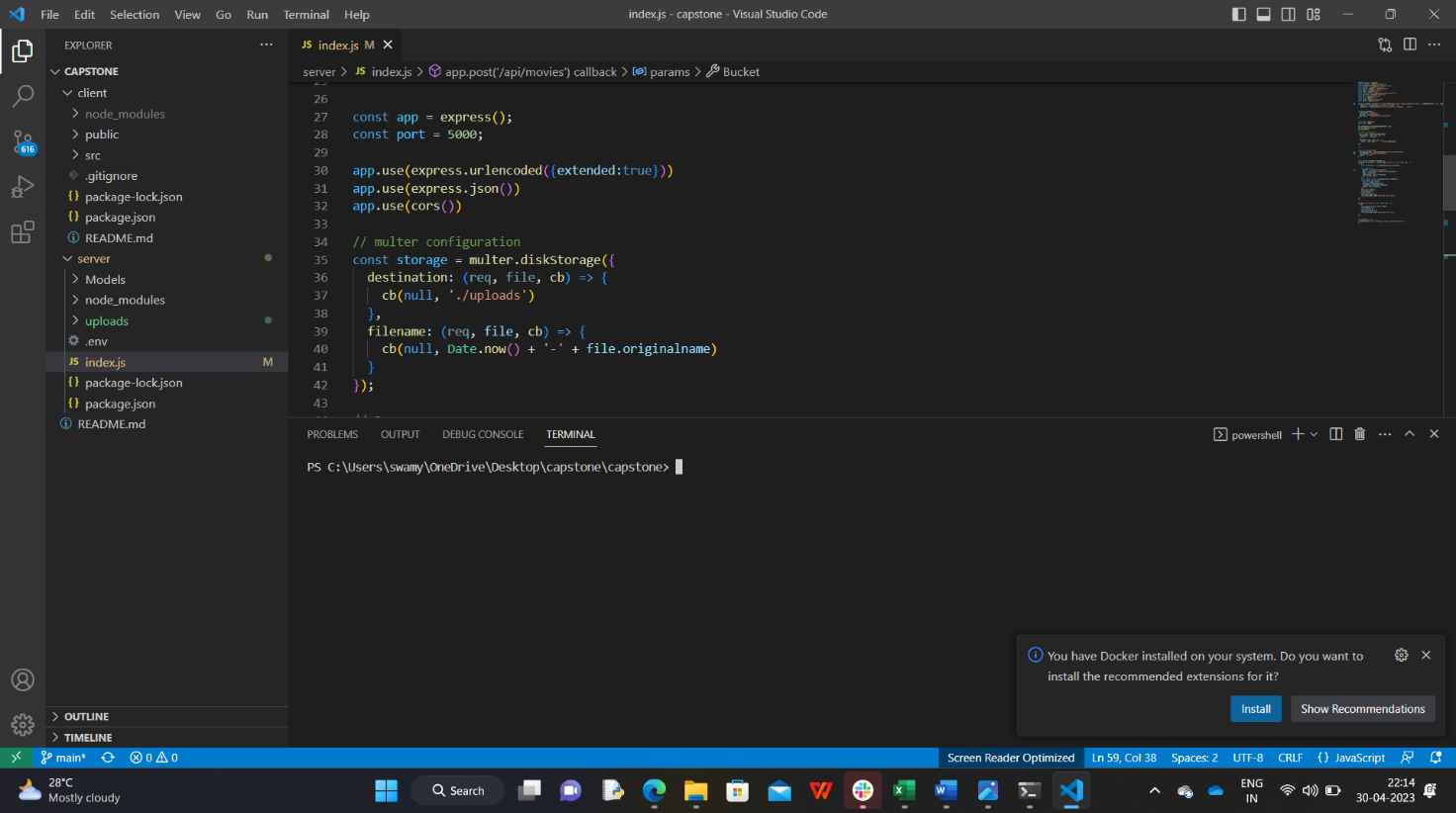
Before creating an S3 bucket, we have to create an IAM user and give S3 full access policy to that user.



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



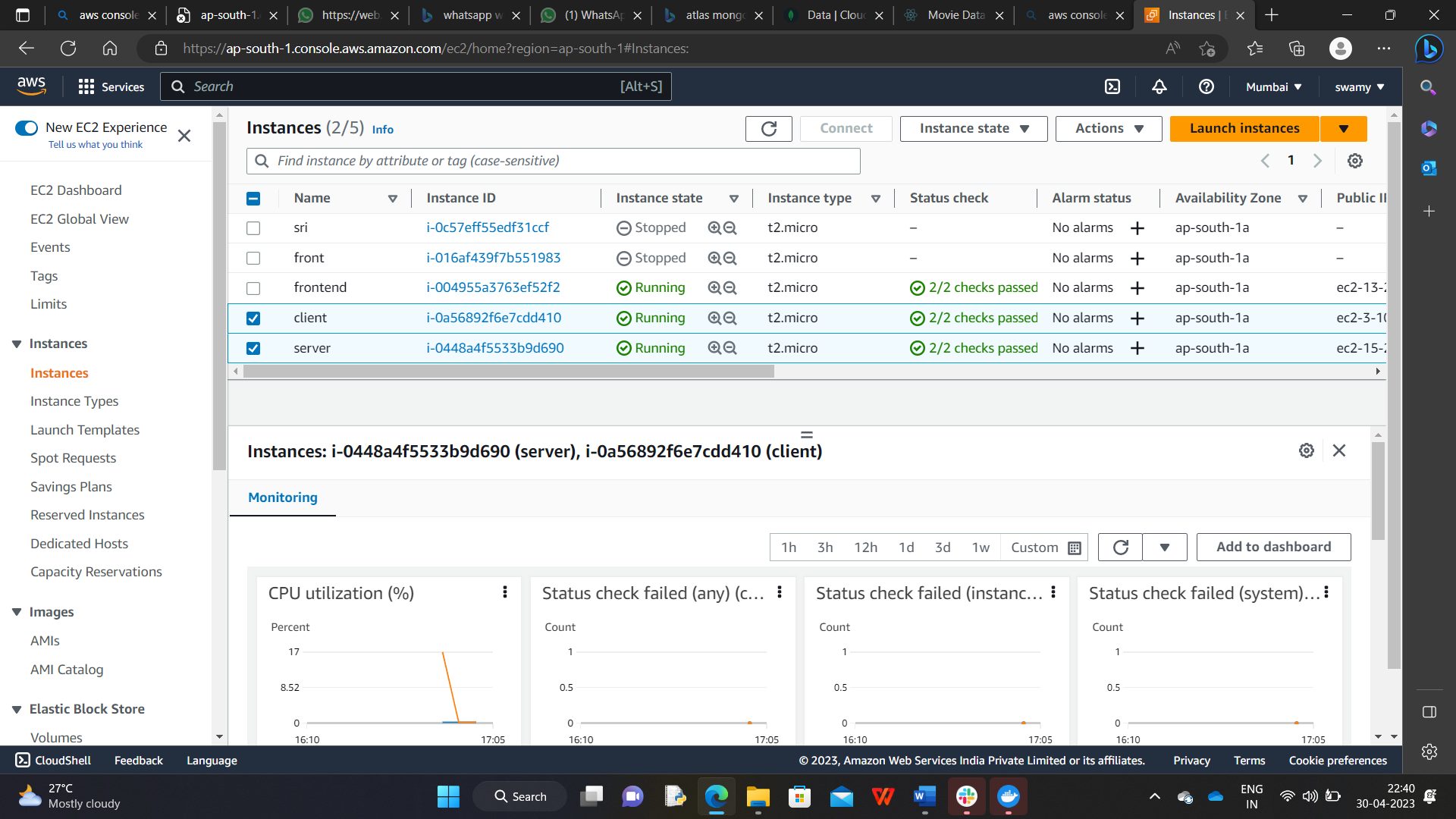
Changing the frontend and backend code to connect with the S3 bucket and MongoDB and installing AWS-SDK for supporting three runtimes: javascript for browser, Nodejs for server, React Native for mobile development, and installing Multer which is used to store uploaded files into the S3 bucket. It is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



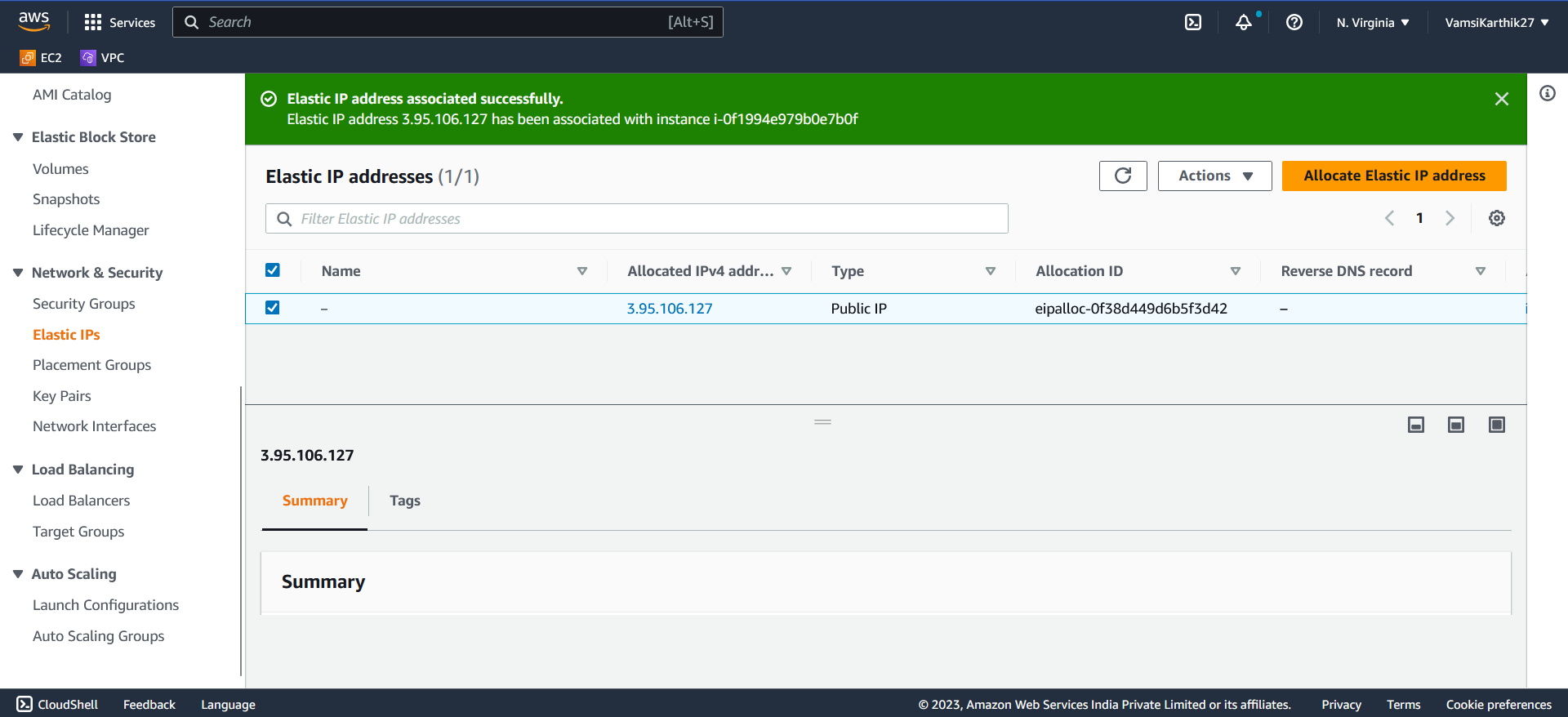


**Step 4 : Launching Instances for both frontend and backend**

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.

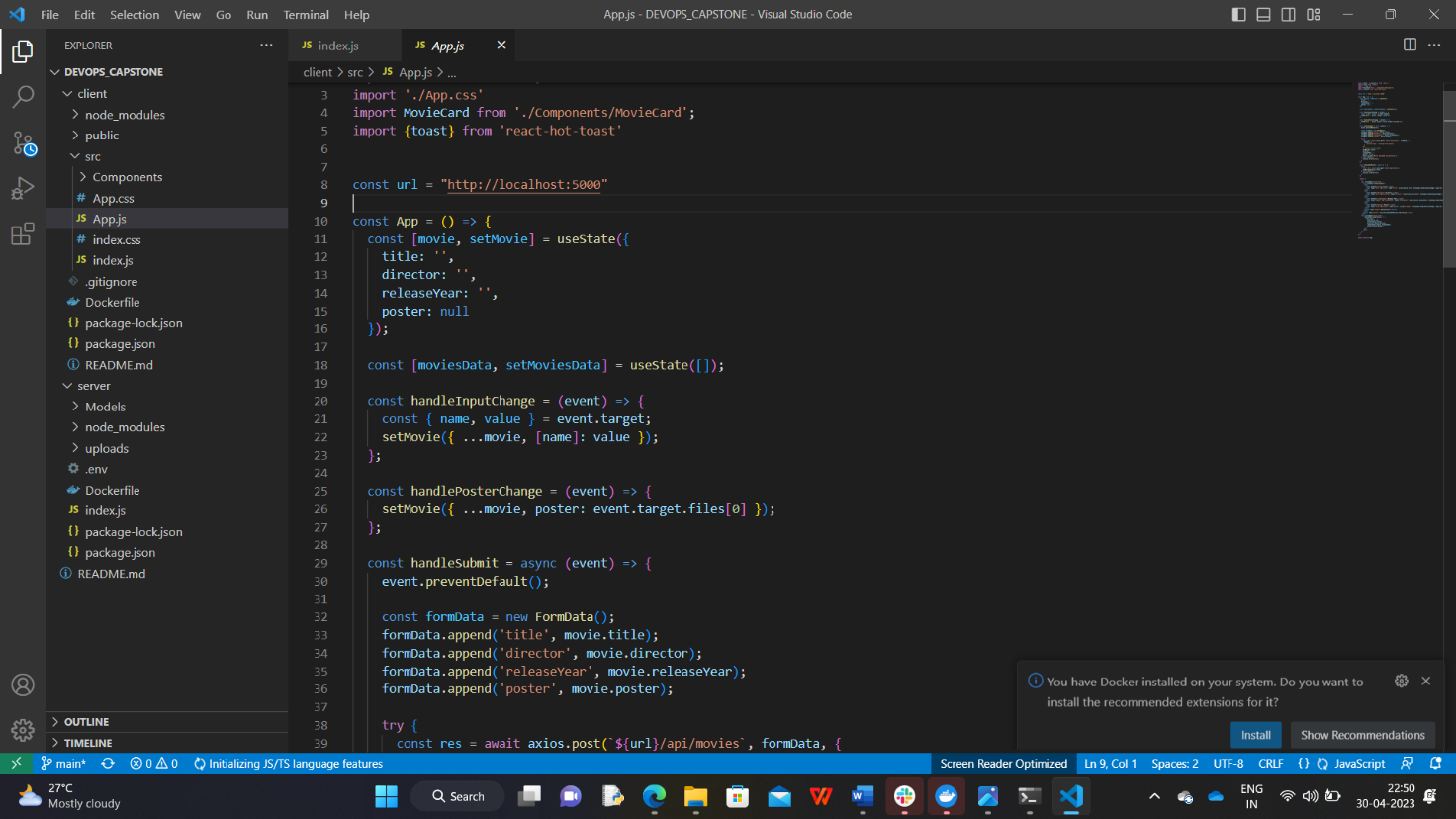


Providing Elastic IP addresses



**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*

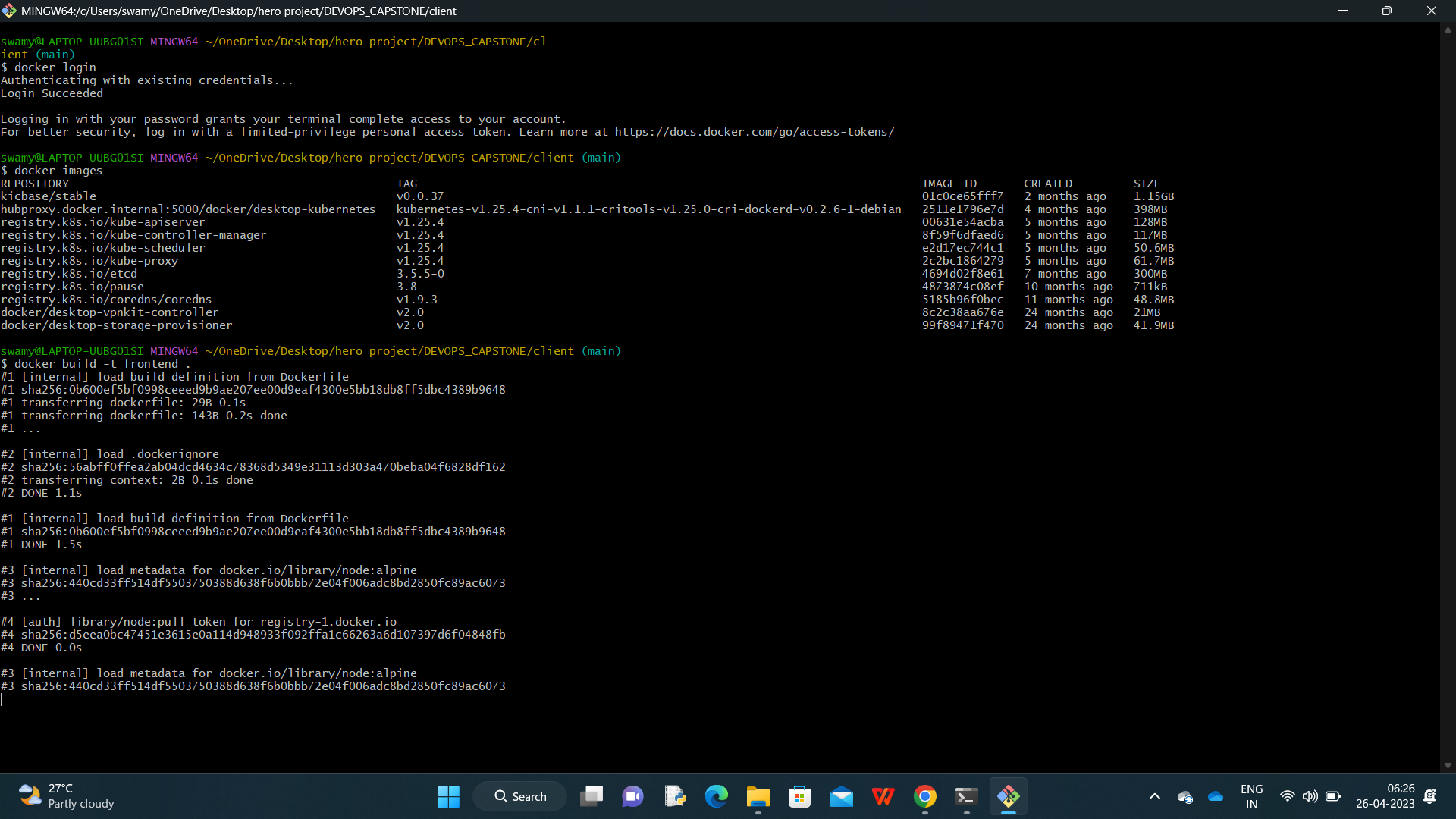
*EXPOSE 80*

*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.

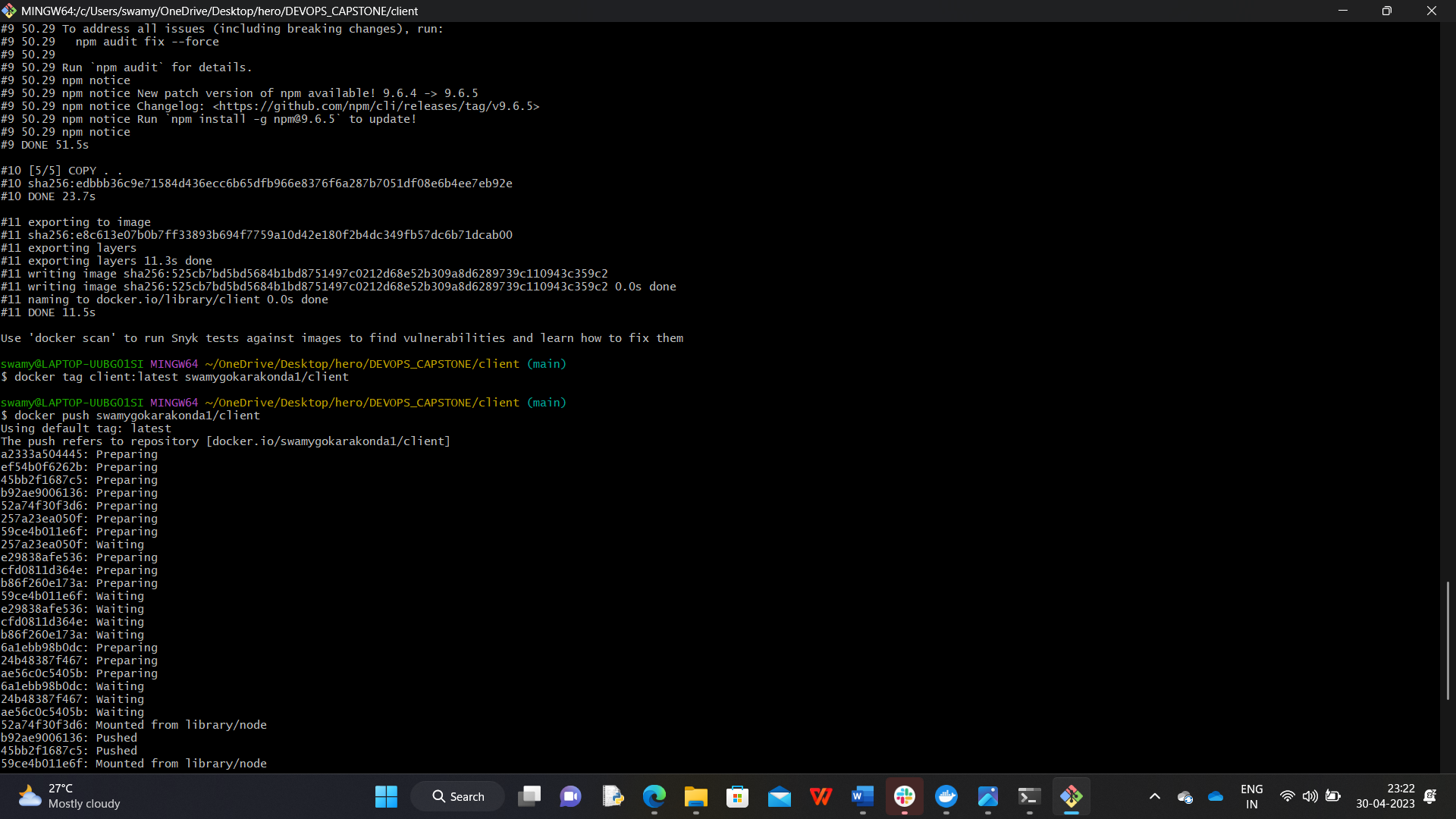
Building an images for backend and frontend



Creating a repository to store the images.

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

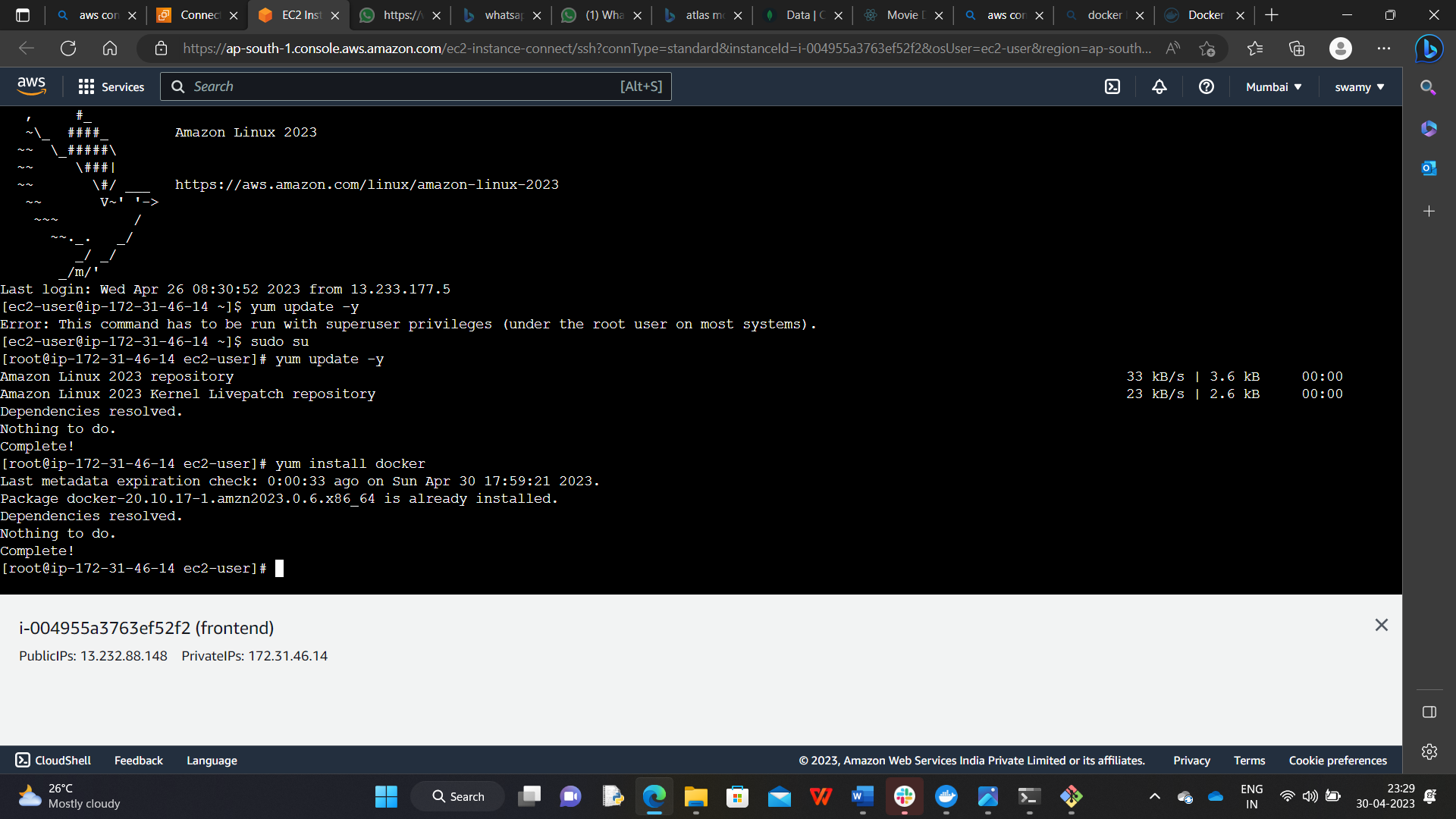
* **Docker push**: It is used to push or share a local Docker image or a repository to a central repository; it might be a public registry like https://hub.docker.com or a private registry or a self-hosted registry.



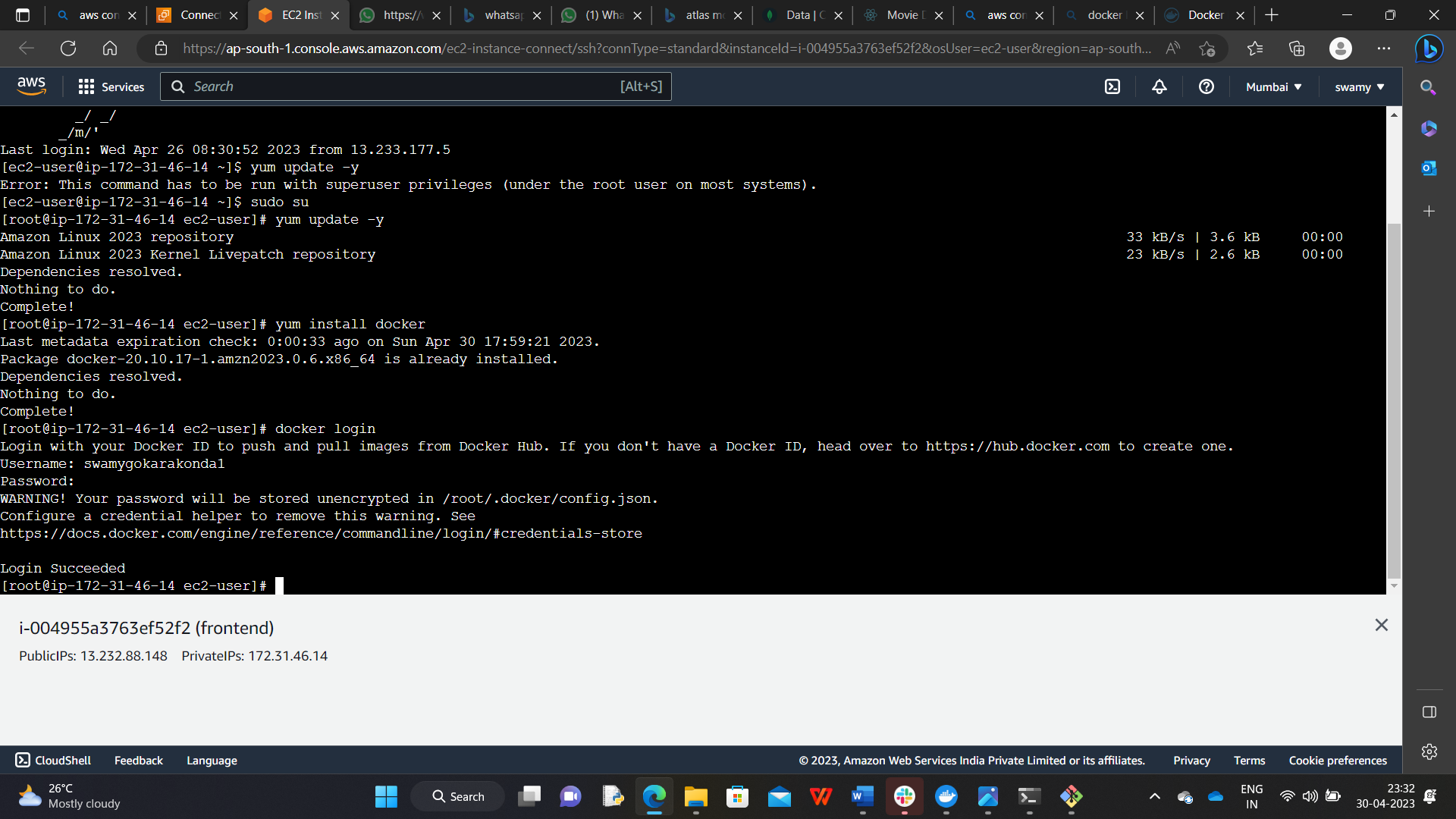
Now we will list all the images that we have still now using the command **docker ps**

**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

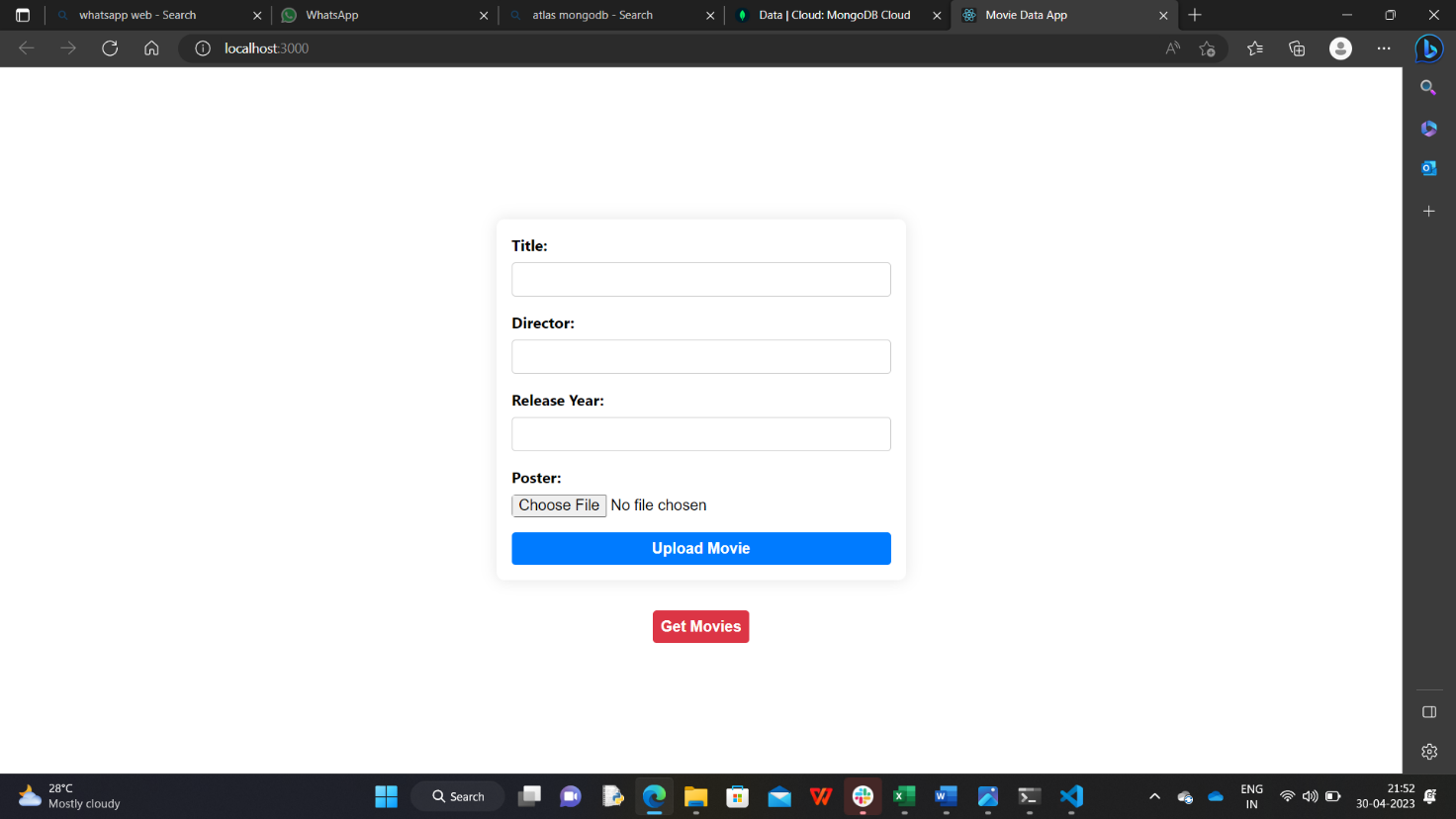


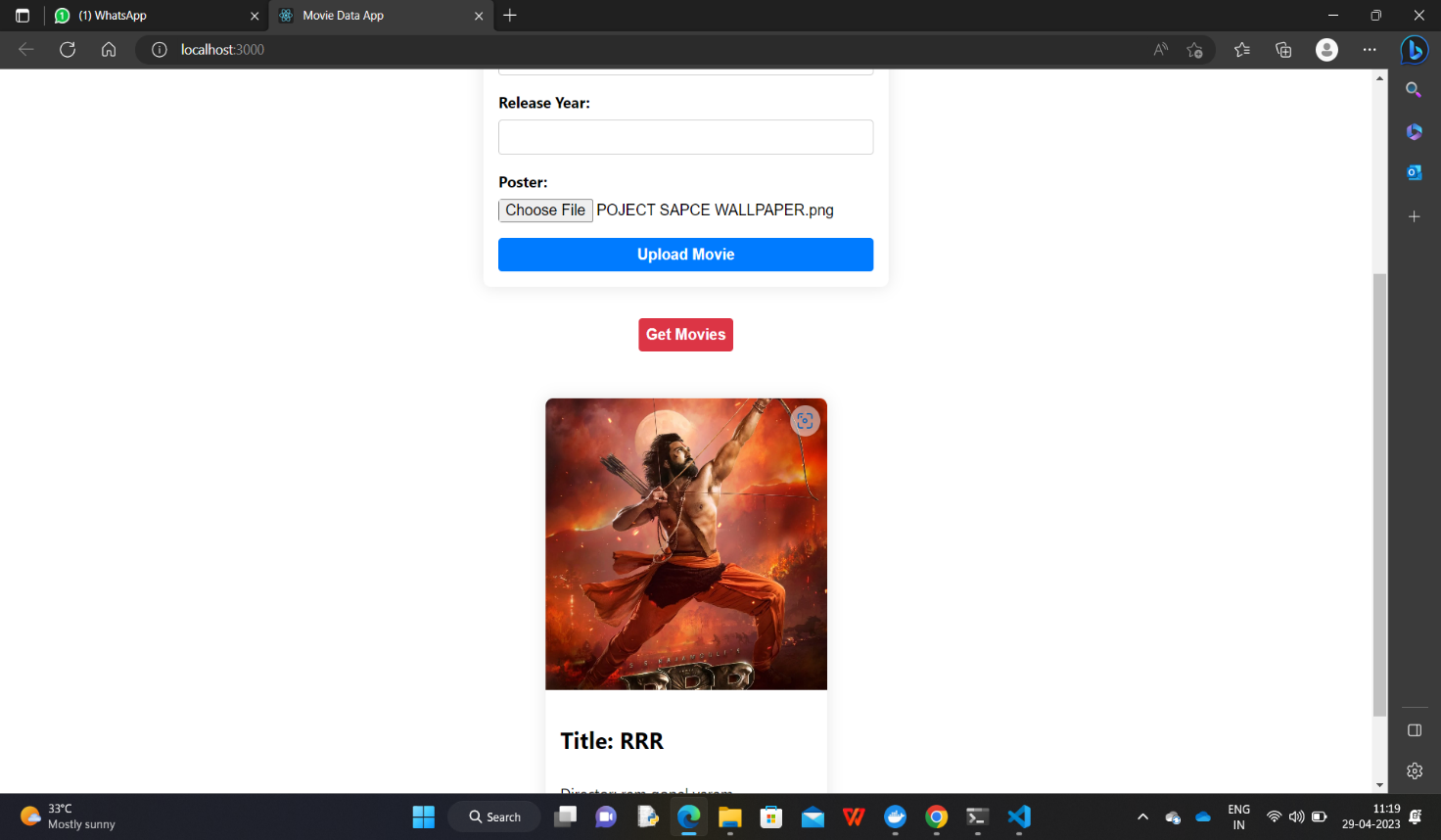
Now run the docker images in the corresponding instances and it shows that it was connecting to MongoDB

**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.

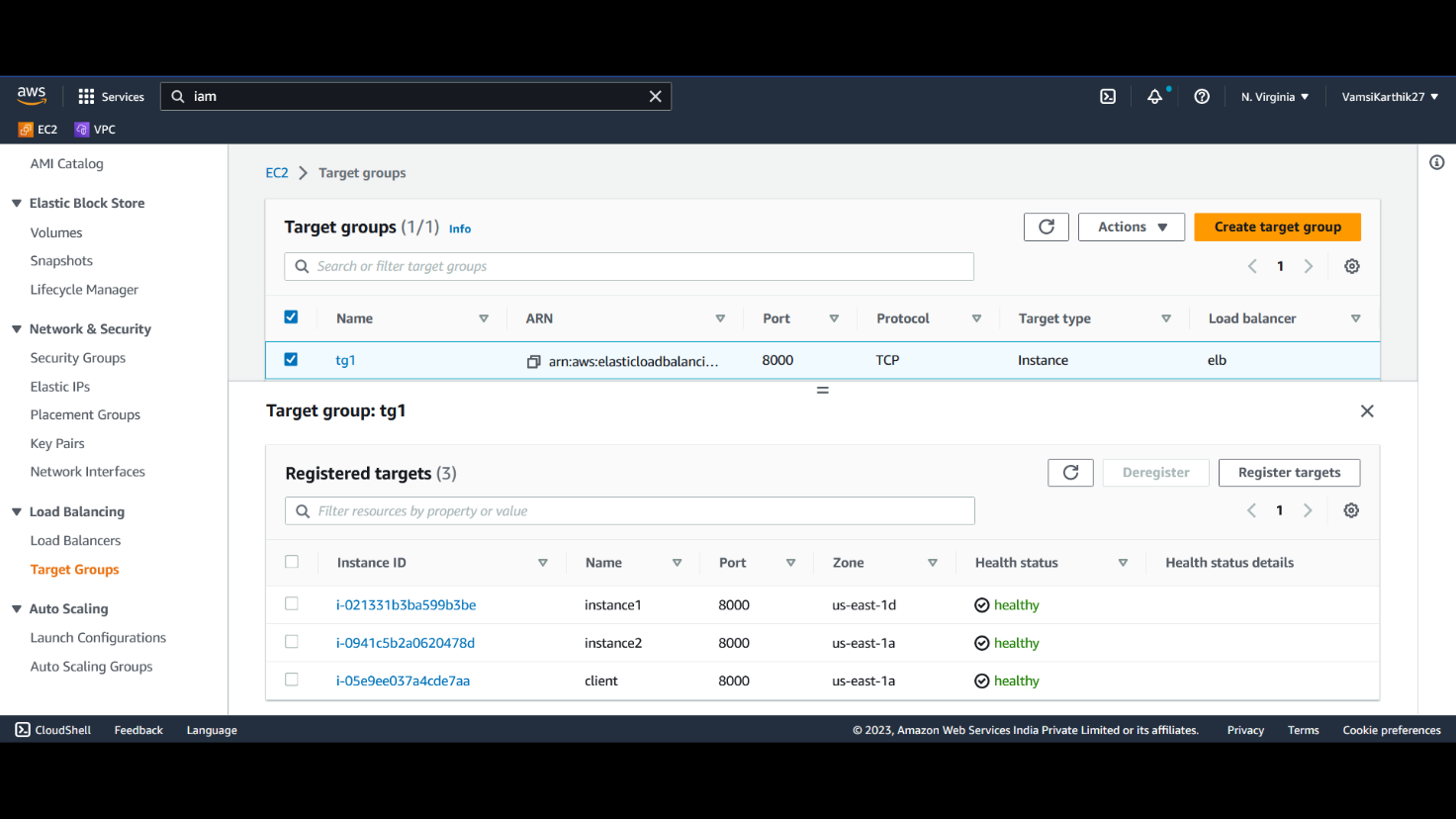




Here the uploaded images are reflected in the S3 bucket and also in the MongoDB Atlas

**Step 8: Creating of Target group and Load Balancer**

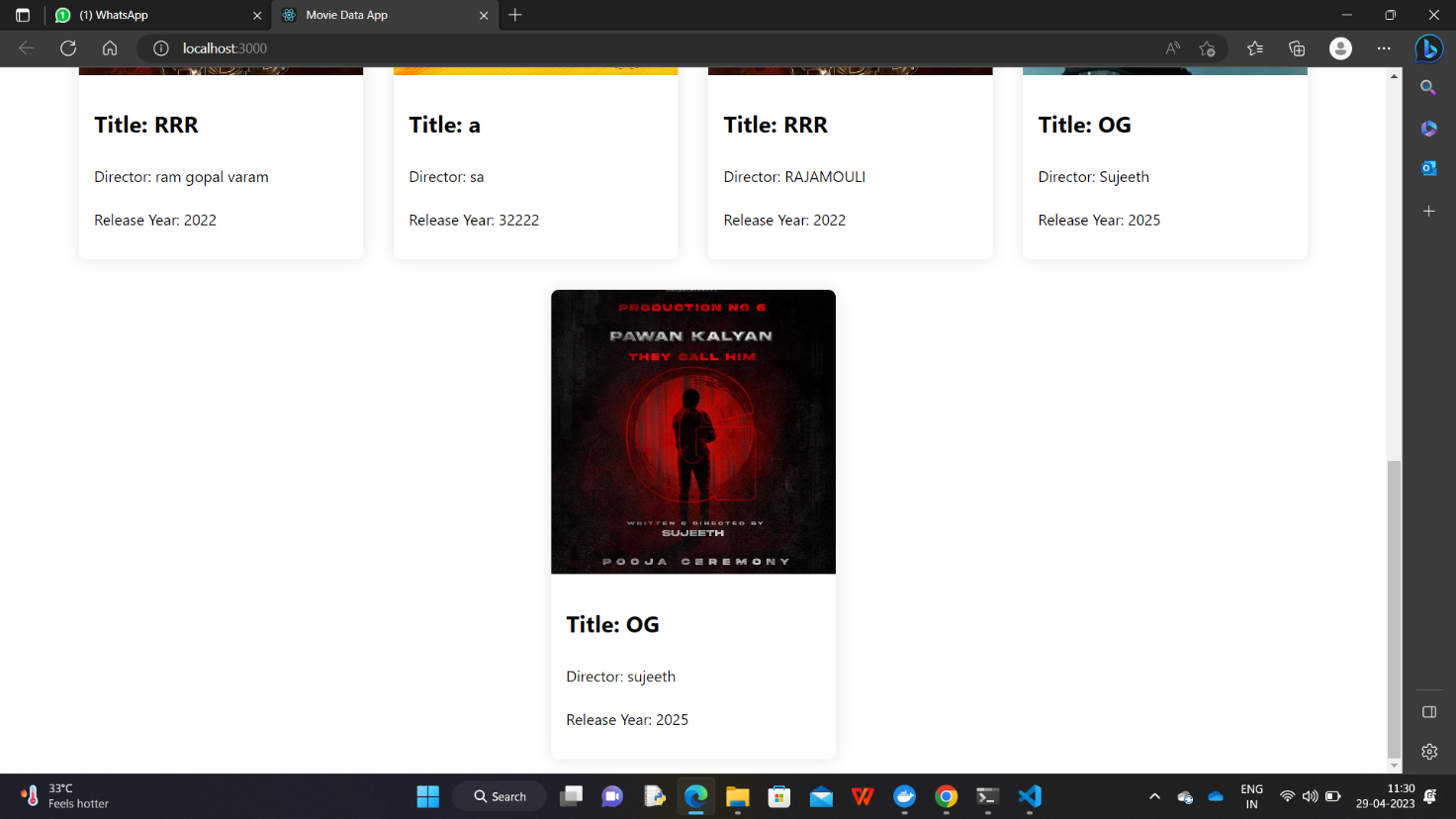
Now, create a target group that is used to tell the load balancer where to direct traffic to instances, fixed IP addresses, and AWS Lambda functions among others.



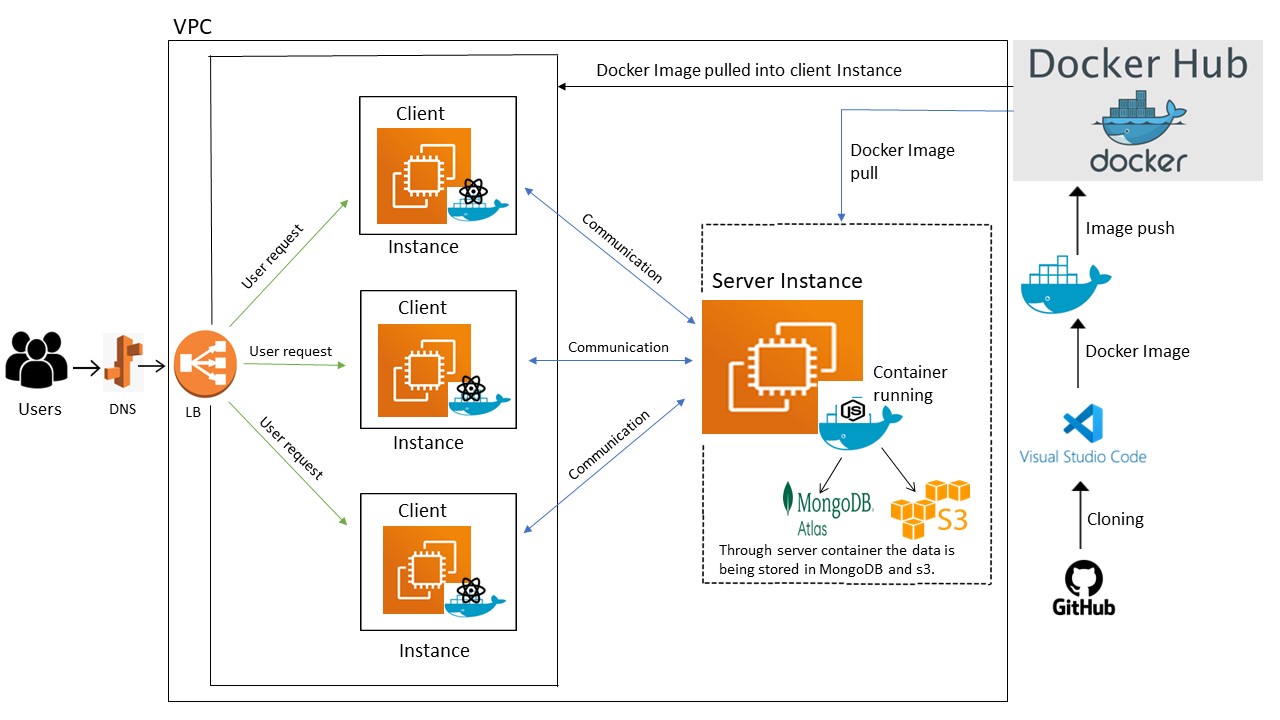
Create an Elastic Load balancer which is used to manage the load on the instances and it will increase the capacity based and reliability of applications. Creating a load balancer we can create one or more listeners and configure their rules to direct the traffic to the target group.

**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.



**DEPLOYMENT DIAGRAM**



**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

