**DevOps Capstone Project**

**DEPLOYING A WEBSITE (MOVIE LISTING) INTO AWS CLOUD THROUGH PROPER SCALING**

**A Project By:**

**GROUP-9**

B**.** Harsha srivalli-20A91A04D3

M.L. Kalpana chawla-20A91A04F3

A. suryakanthi-20A91A04J3

K. Yaswanth -21A95A0419

T. Ramesh-20P31A04B0

B. Sai vinay-20P31A04J1

G. Swamy-21MH5A0363

|  |  |
| --- | --- |
| **CHAPTER** | **PAGE NO** |
| **Scope** | **1** |
| **Technology description** | **2-3** |
| **Purpose & real-time usage** | **3-4** |
| **Process step by step** | **5-15** |
| **Conclusion** | **19** |

**SCOPE:**

* The goal of this project is to demonstrate the ability to deploy a full-stack web application on AWS, with proper scaling and redundancy, while also leveraging AWS services such as S3, EC2, and Load Balancer.

**Technologies used**

* Frontend: ReactJS
* Backend: NodeJS
* Database: MongoDB
* Amazon web services

**Tools used:**

* Docker hub
* Git hub

**Purpose:**

* The purpose of deploying the "Movie listing" website into the cloud infrastructure (AWS) with proper scaling is to ensure high availability, scalability, and reliability of the application, while also reducing the infrastructure management overhead. By deploying the application in the cloud, we can take advantage of the cloud provider's infrastructure, which includes features such as load balancing, auto-scaling, and managed database services. This will help to ensure that the application is highly available and can handle increasing traffic and workload as needed. Additionally, by using containers and Docker, we can simplify the deployment process and enable consistent deployments across different environments.
* **Real-time applications:** Netflix, Amazon prime, Zee5, hotstar. Etc,

**ReactJs:**

* ReactJS is a simple, feature rich, component-based JavaScript UI library. It can be used to develop small applications as well as big, complex applications. ReactJS provides minimal and solid feature set to kick-start a web application. React community compliments React library by providing large set of ready-made components to develop web application in a record time. React community also provides advanced concept like state management, routing, etc.,

**NodeJS:**

* Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a web browser. Node.js is a popular, lightweight web framework for beginners, and it is used by many big companies like Netflix and Uber.

**MongoDB:**

* MongoDB is a document database. It stores data in a type of JSON format called BSON.

**Amazon web services(aws):**

* AWS provides a highly reliable, scalable, low-cost infrastructure platform in the cloud that powers hundreds of thousands of businesses in 190 countries around the world.

**GitHub:**

* GitHub is an online software development platform. It's used for storing, tracking, and collaborating on software projects. Used for pulling and pushing the files

**Docker hub:**

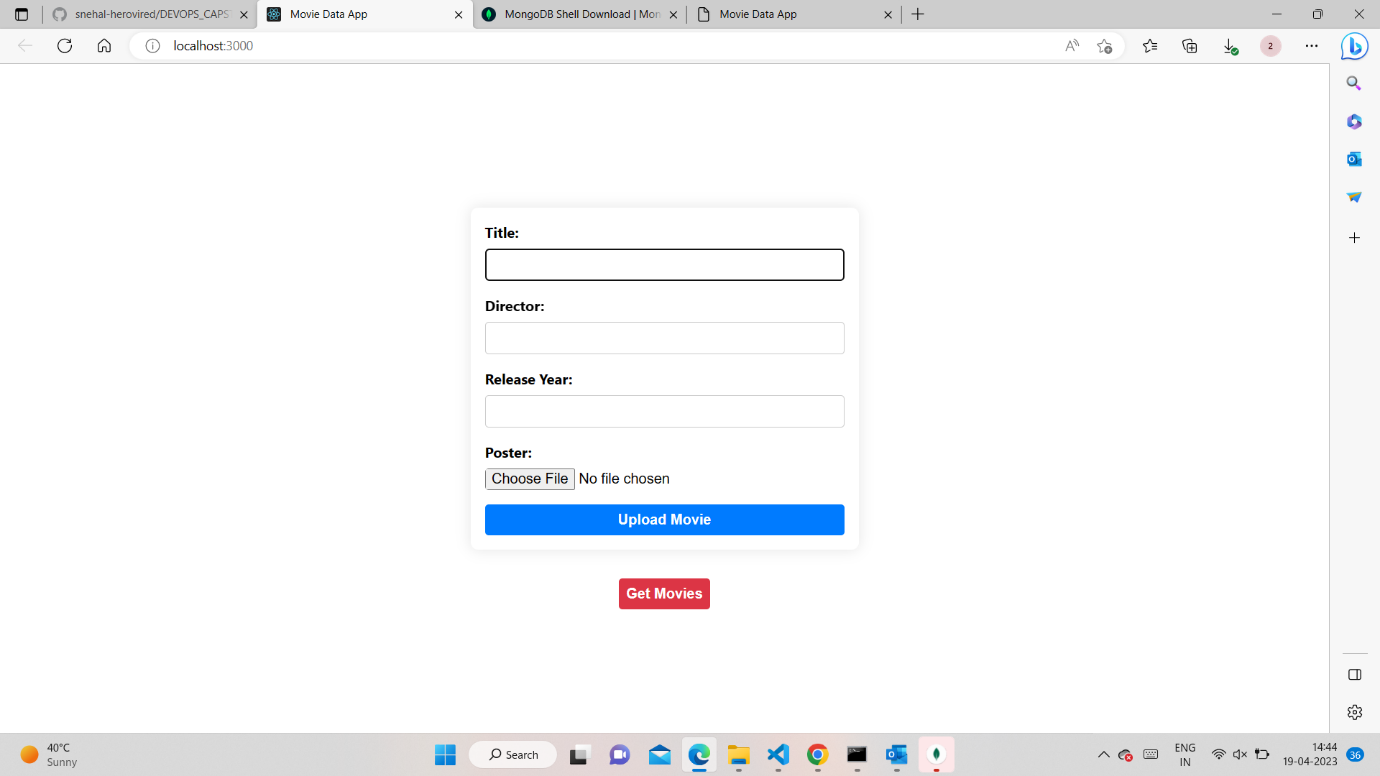
* Docker Hub is the central meeting place for container community developers, open-source projects and independent software vendors (ISV) building and distributing their code in containers.

**Process for hosting the movie listing application:**

* First, we clone the repository and open the vs code (visual studio) and click on 🡪open the terminal🡪install npm by using the command
* Npm install
* First do for client folder 🡪connect the client
* By using the commands:
* Npm start (or) node index.js

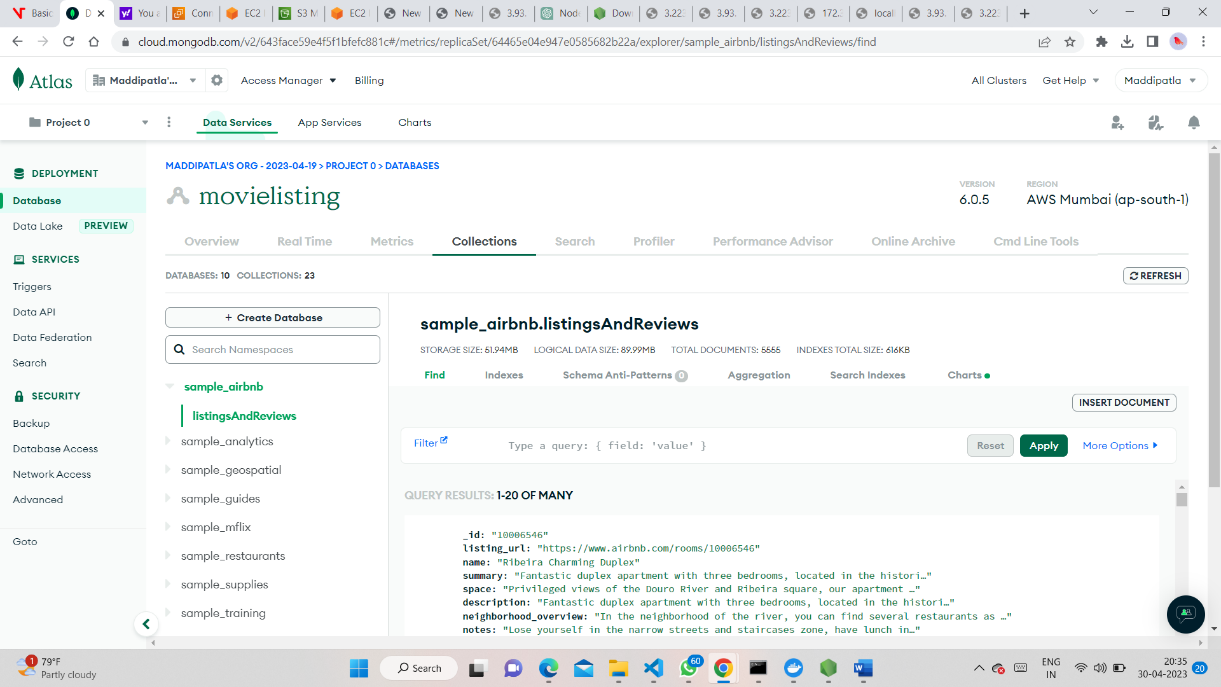
Check whether the movie listing page is opened or not by clicking the link after npm start.

**Proofs of concept:**

****

* How to connect the Mongo dB database to server
* First create an account on Mongo dB atlas.
* Download mongo dB for windows. (Mongodb compass)
* create a cluster (Create a new cluster: Once you have created an account, you can create a new cluster).
* After creating a new cluster, configure your database by setting up authentication, defining database users, and creating a new database.

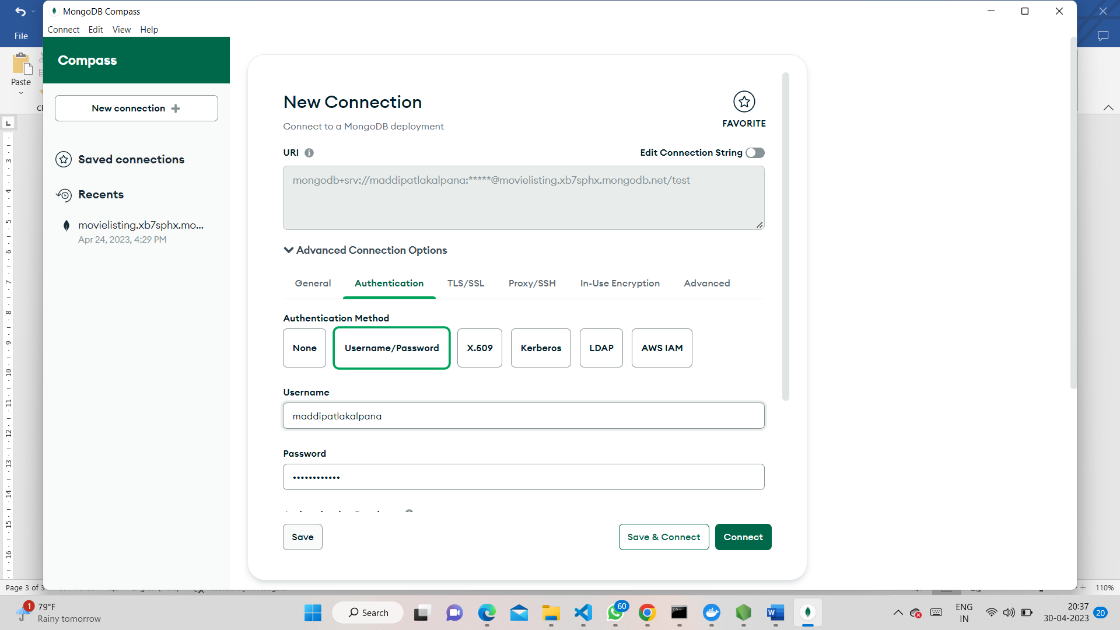
Proof:



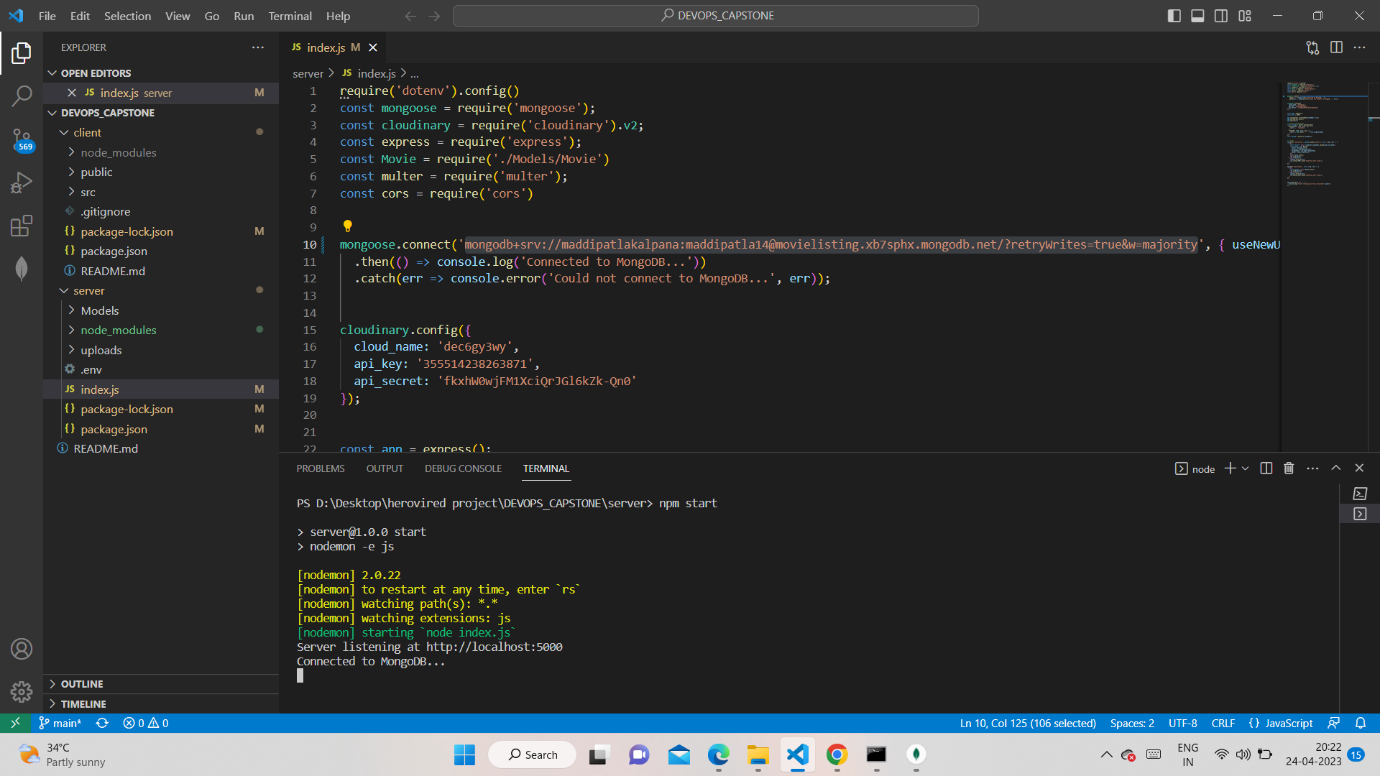
* Connect to the MongoDB cluster: Once the database is set up, you can obtain the connection string for your MongoDB cluster from the Atlas dashboard. Replace the existing connection string in your Node.js application with the new connection string.

Proofs:



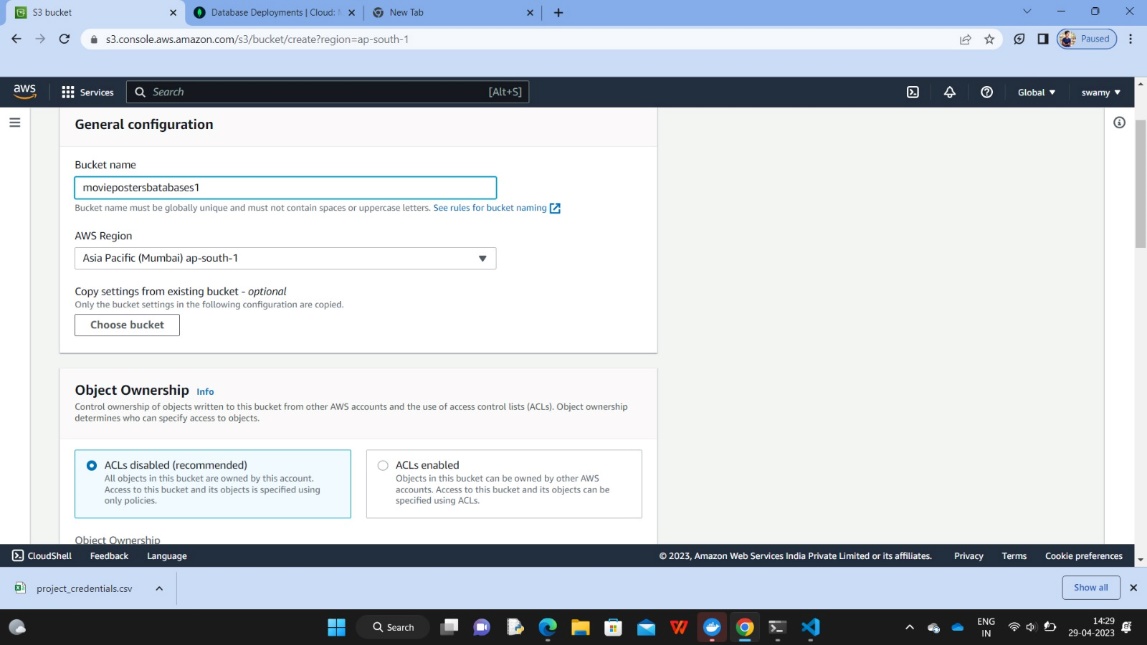


* Next connect the server and install npm and connect server backend to the database MongoDB

****

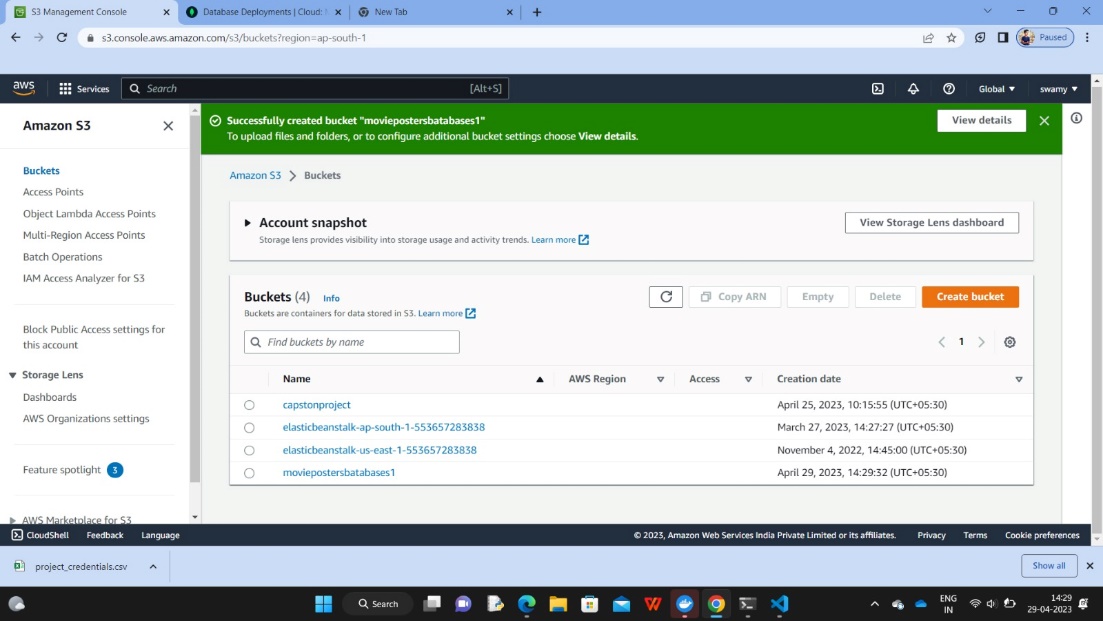
**STEP-1:**

* Use AWS S3 for storing images. You can use the multer-s3 library
* Create an S3 bucket in your AWS account. Install the multer-s3 library in your NodeJS backend application using the following command: npm install multer-s3

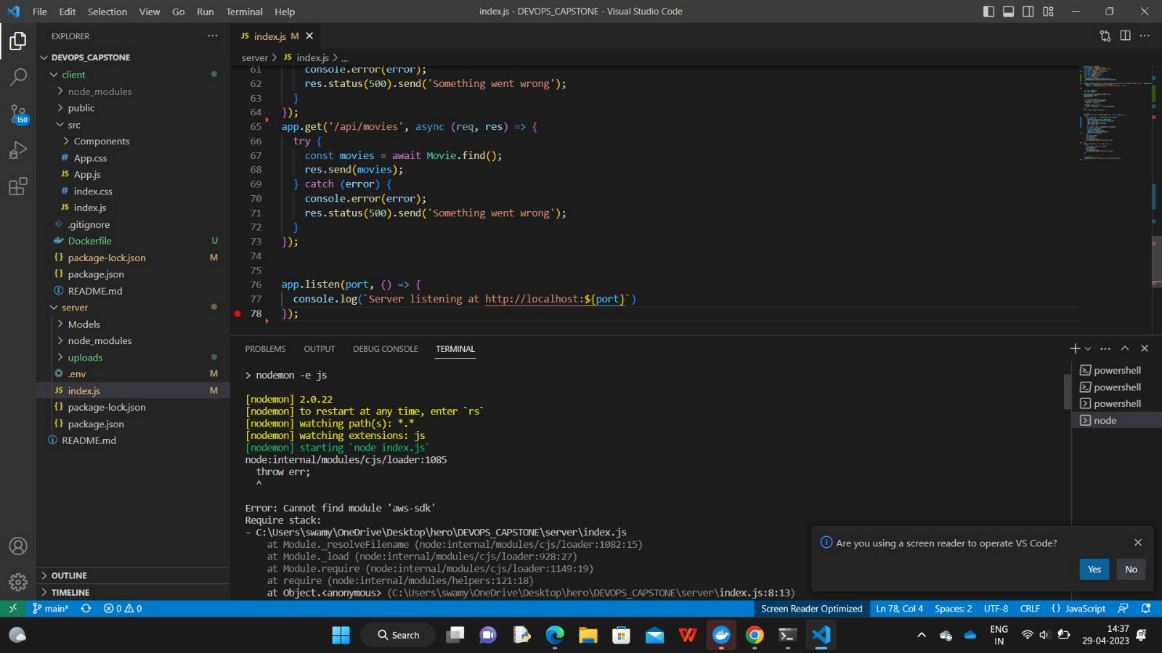


* Install the multer-s3 library in your NodeJS backend application using the

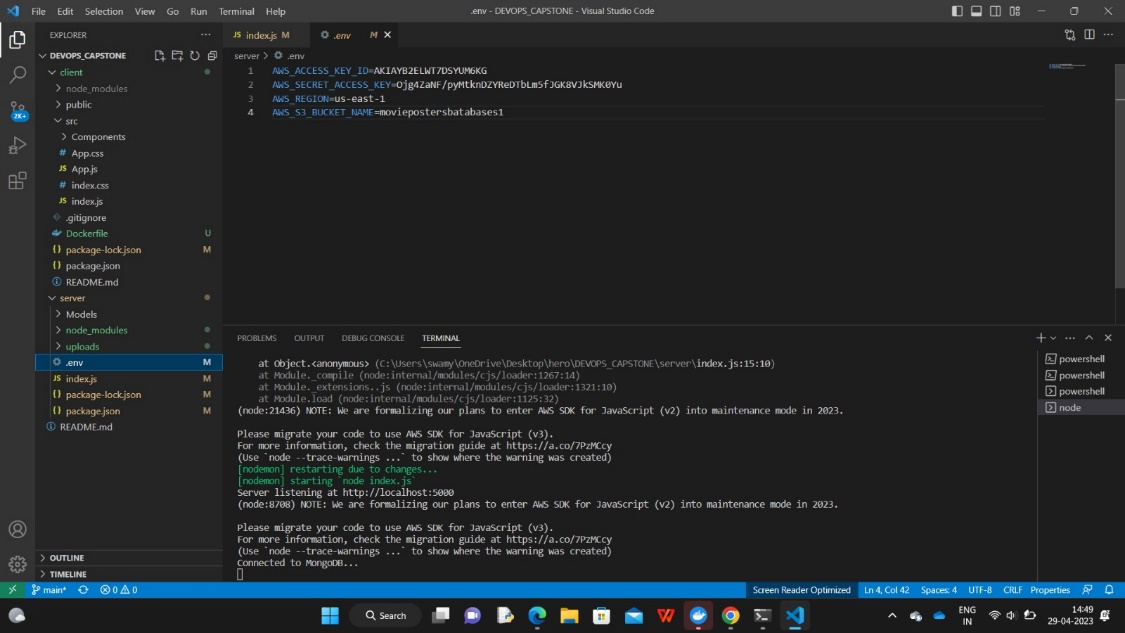
following command: npm install multer-s3



* Require multer-s3 in your backend code along with other required modules:

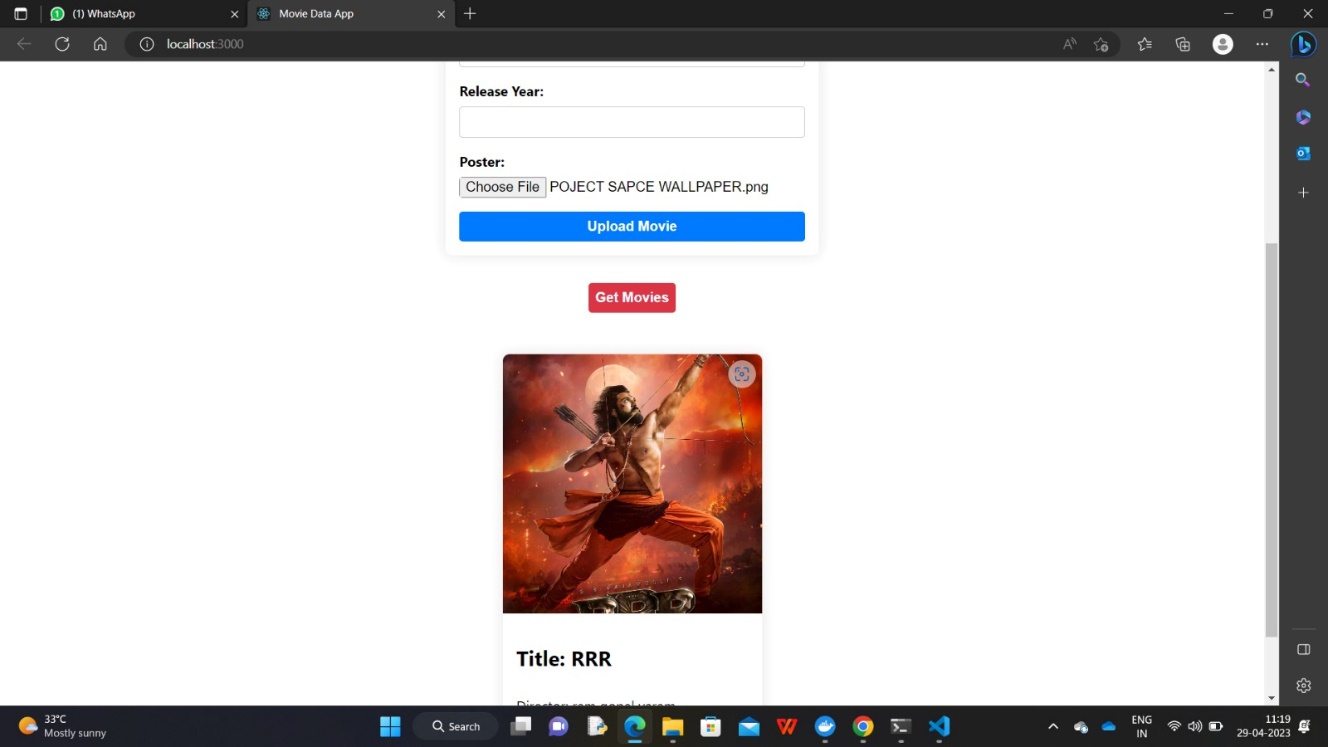


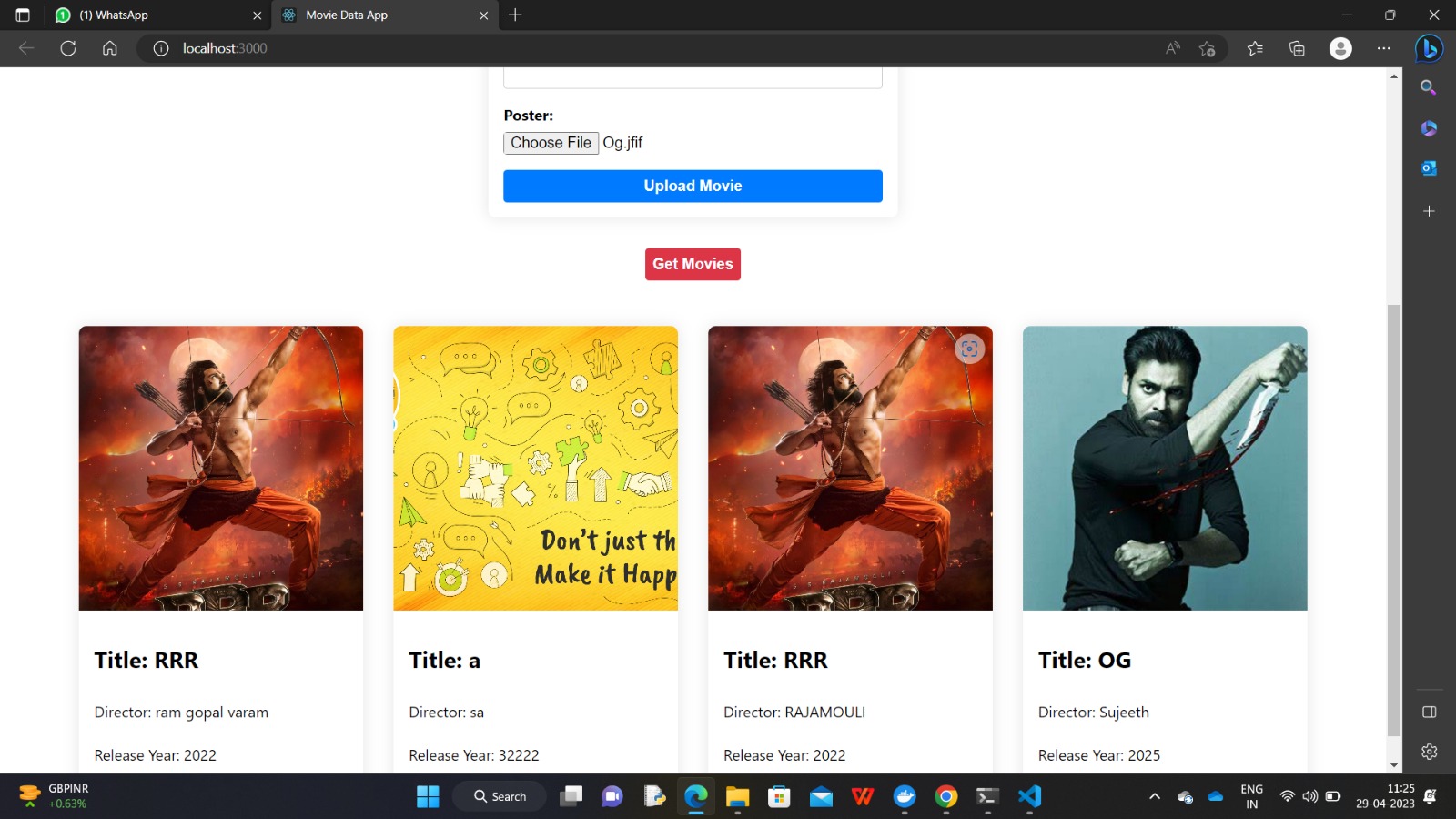
* Create an instance of the aws-sdk with your access key and secret key:

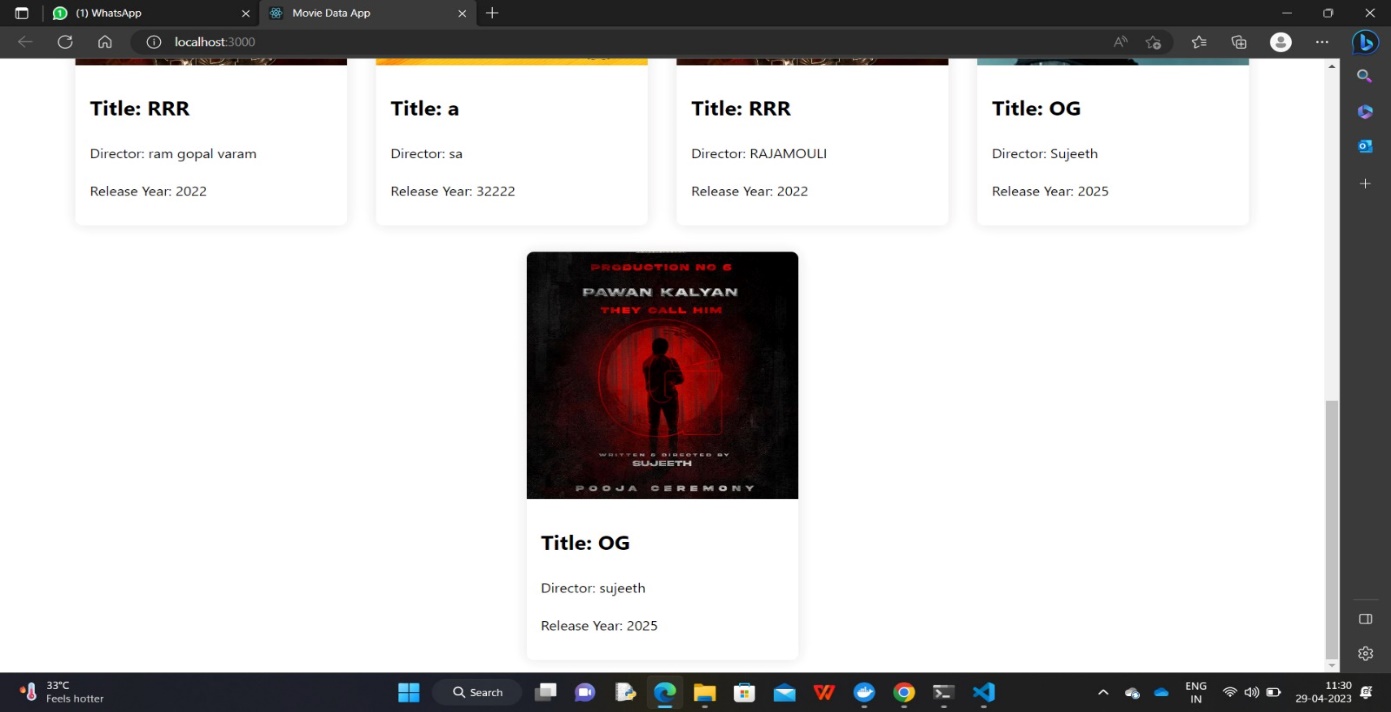


* Create a multer middleware that uses multer-s3 to upload the file to S3
* This will handle a single file upload and return the URL of the uploaded file in the response.

**Proofs for the concept:**

****



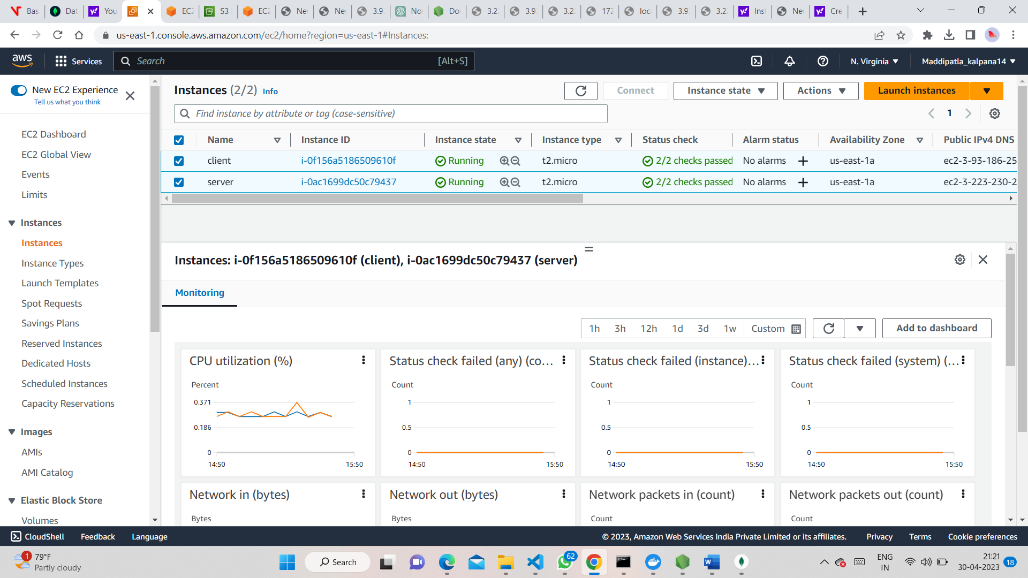
****

**Step-2:**

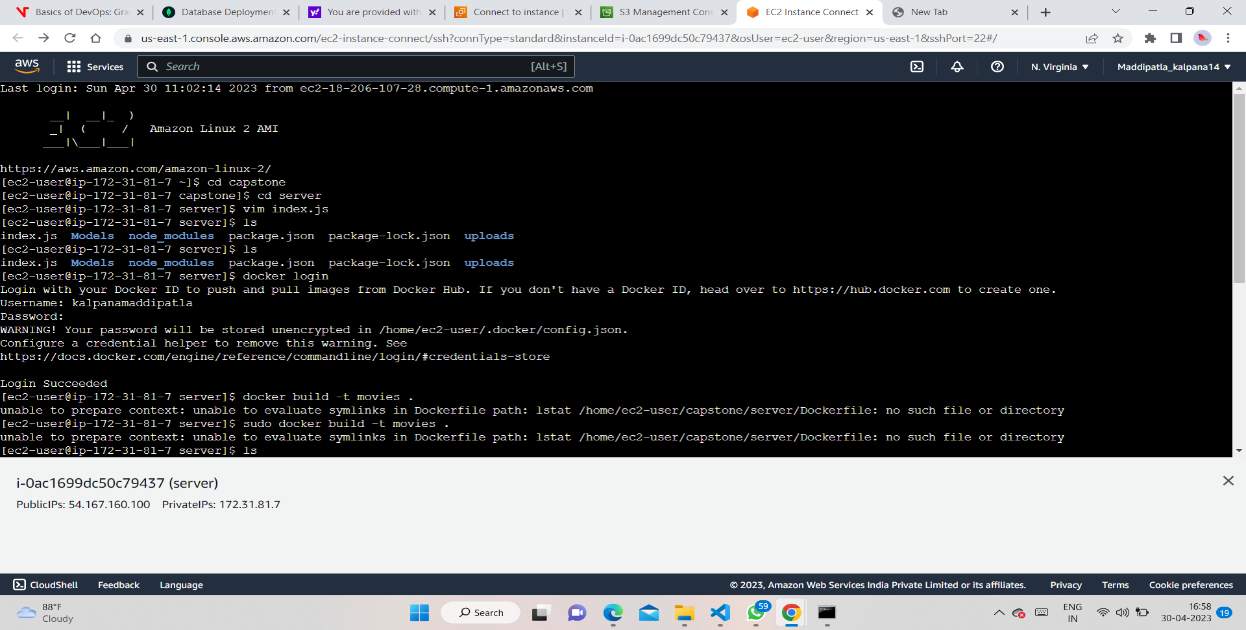
* Replace local database with Atlas MongoDB cloud infrastructure to take the database into the cloud.
* Create an Atlas account and create a new cluster. Atlas offers a free tier to get started.
* Create a new database user with proper privileges. Make sure to keep the password securely.
* Whitelist your IP address in the network access section to allow connections to the database from your development machine.
* In the Node.js backend code, replace the connection string with the connection string provided by Atlas.
* The connection string can be found in the Atlas dashboard under the "Connect" button for your cluster.
* Use the database user credentials you created in step 2. Test the backend code to ensure it can connect to the database successfully.
* Migrate the data from the local database to the Atlas database.
* Modify the frontend code to use the new database endpoints for fetching and displaying data deploy the updated code to your AWS infrastructure.

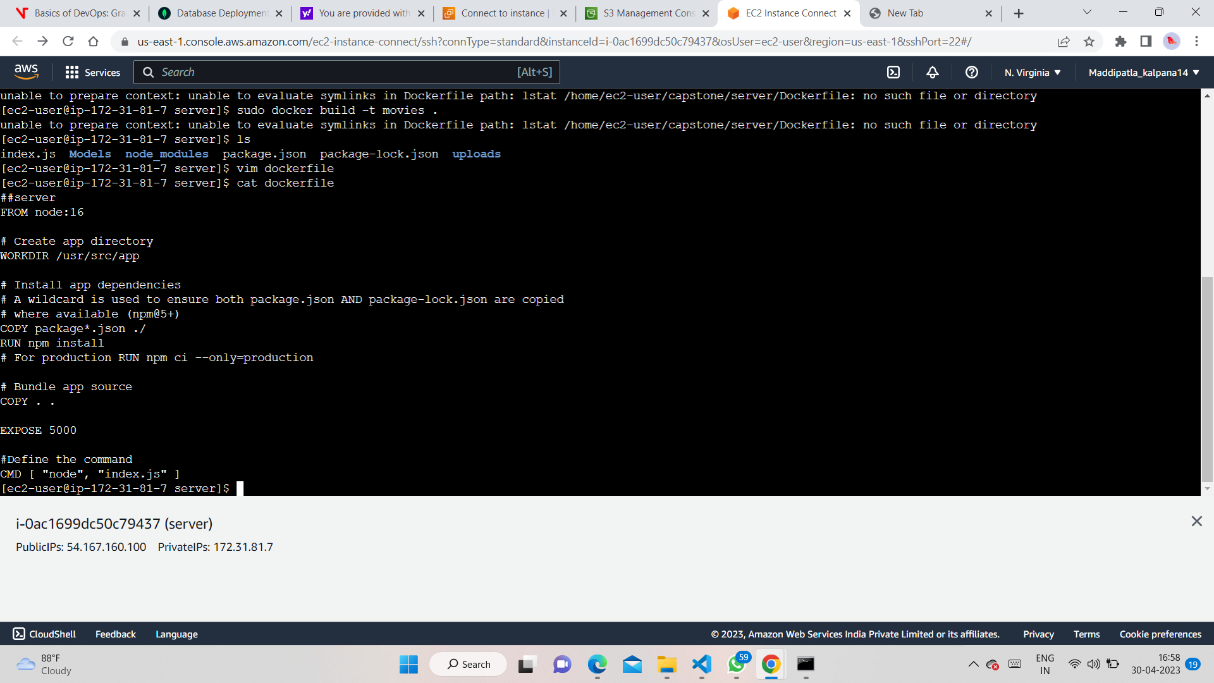
**STEP-3:**

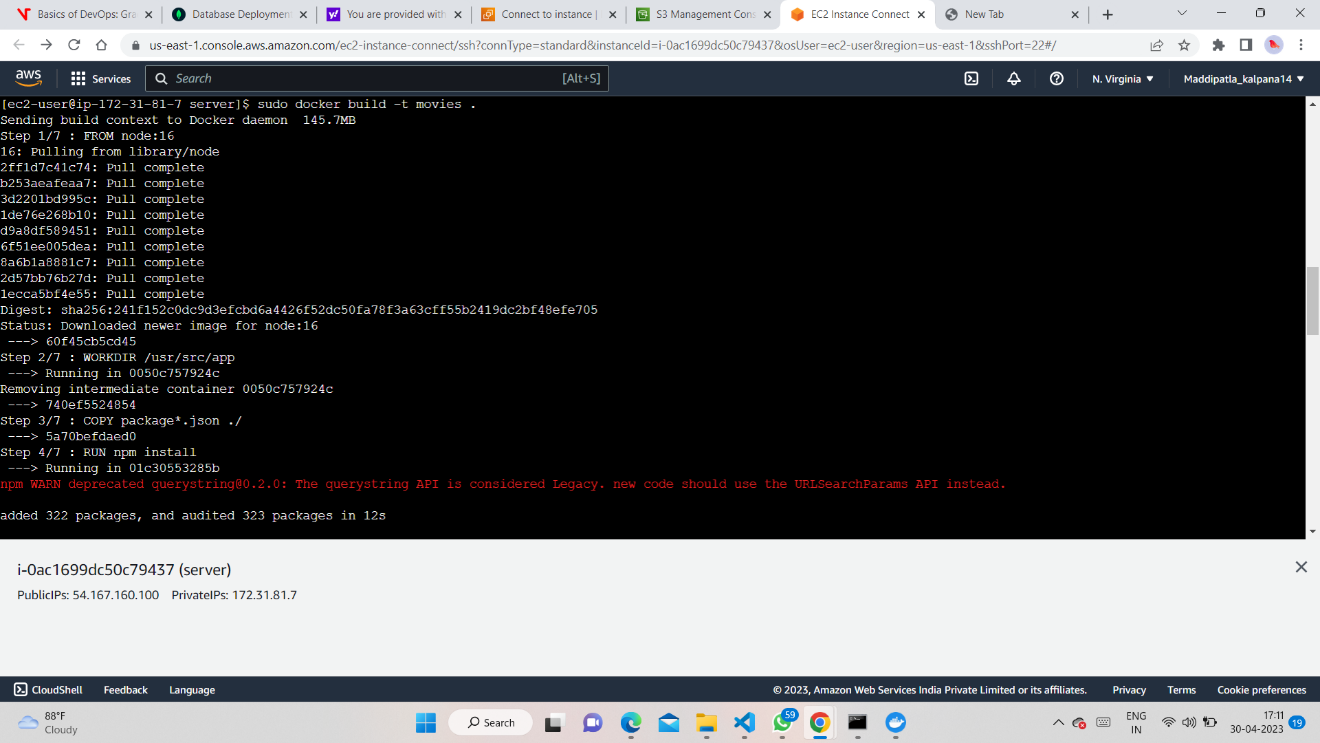
* Deploy Backend in EC2 instance and attach Elastic IP to this instance. Deployment should be done using Docker.
* Launch an ec2 instance by giving name (ex: myec2)🡪choose amazon machine image (AMI)🡪create a new keypair (. pem OR. ppk)🡪edit security groups🡪Launch instance.



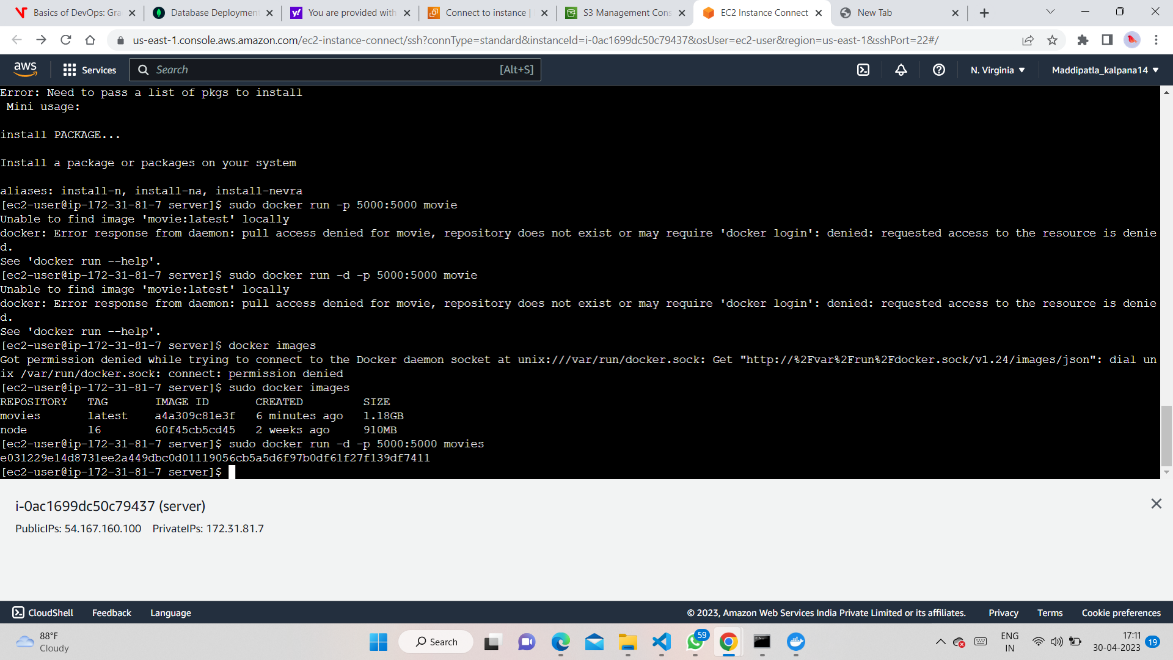
* Connect the instance and install node.js, npm, git, docker tools using necessary commands:
* Build the Docker image: Once Docker is installed, build the Docker image for your NodeJS backend. Create a Docker file in the root directory of your NodeJS application
* Proofs for concept:



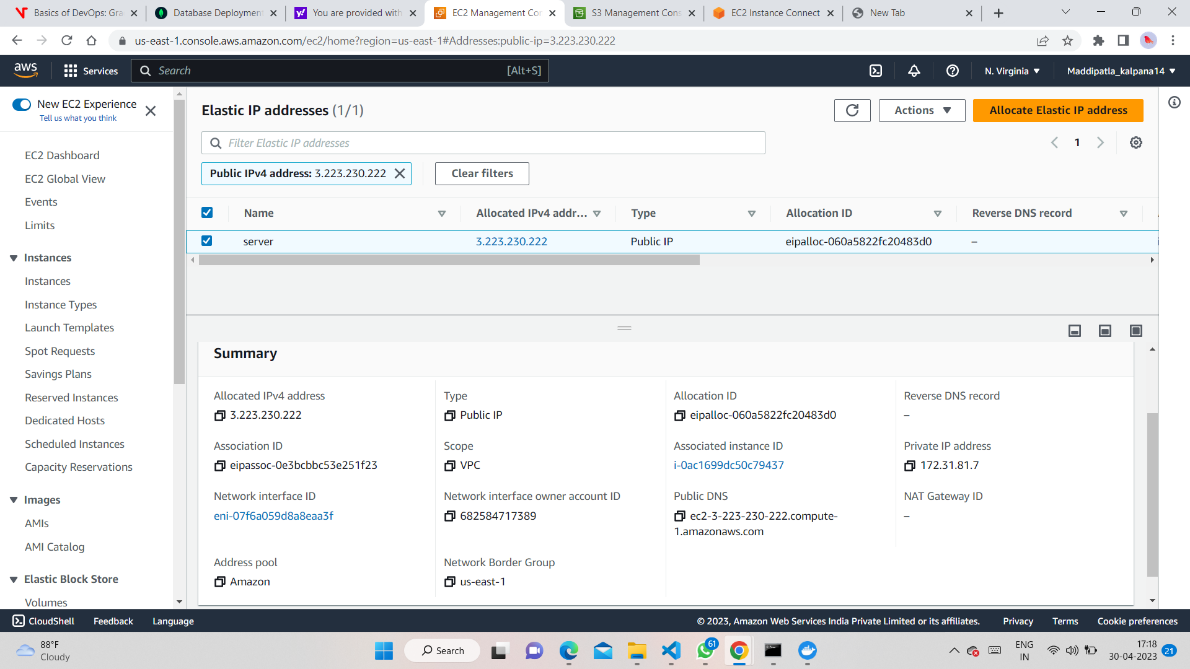


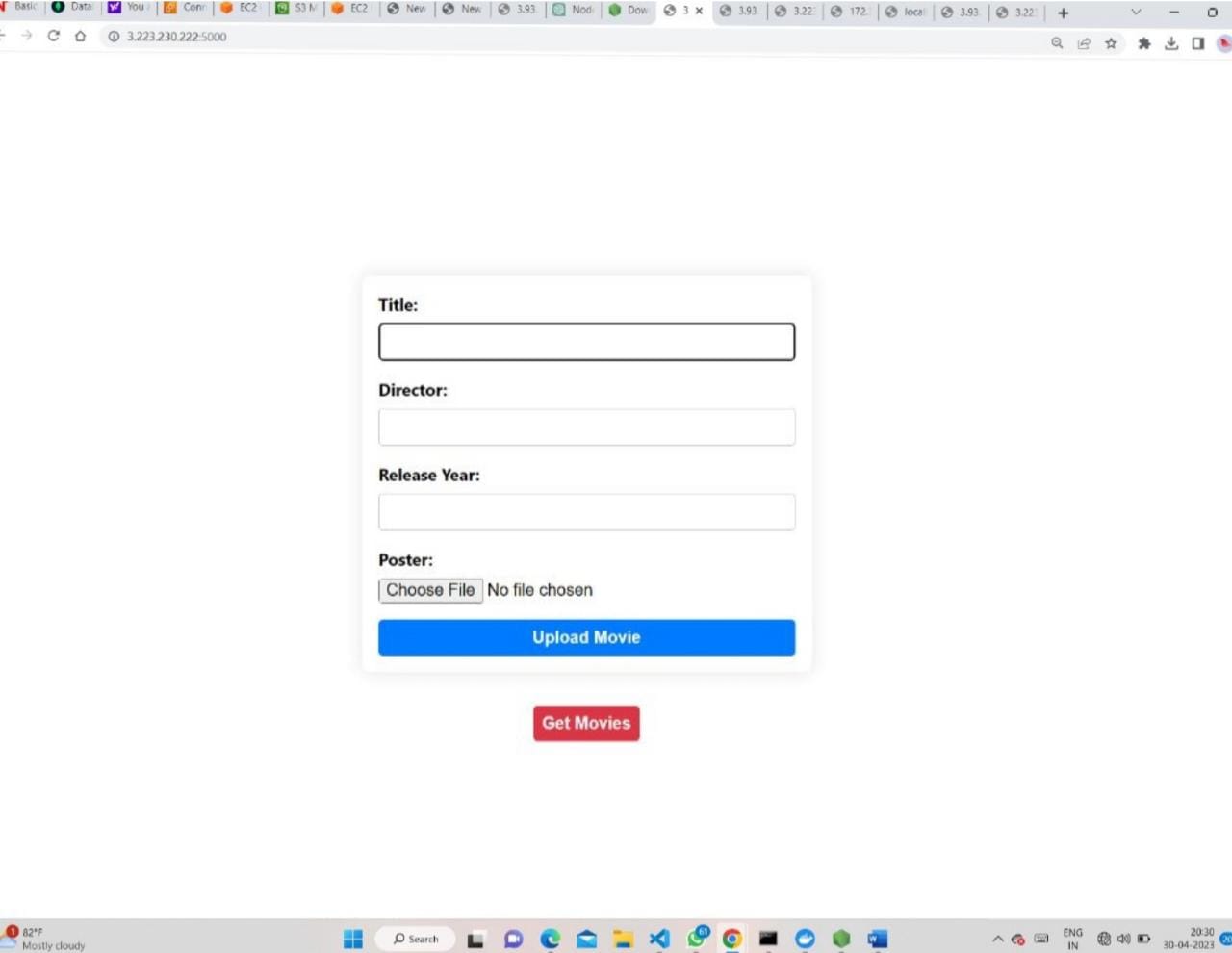


* Then, build the Docker image using the following command:
* sudo docker build -t image-name.
* Run the Docker container: Once the Docker image is built, you can run the container using the following command:
* sudo docker run -p port:num -d image-name



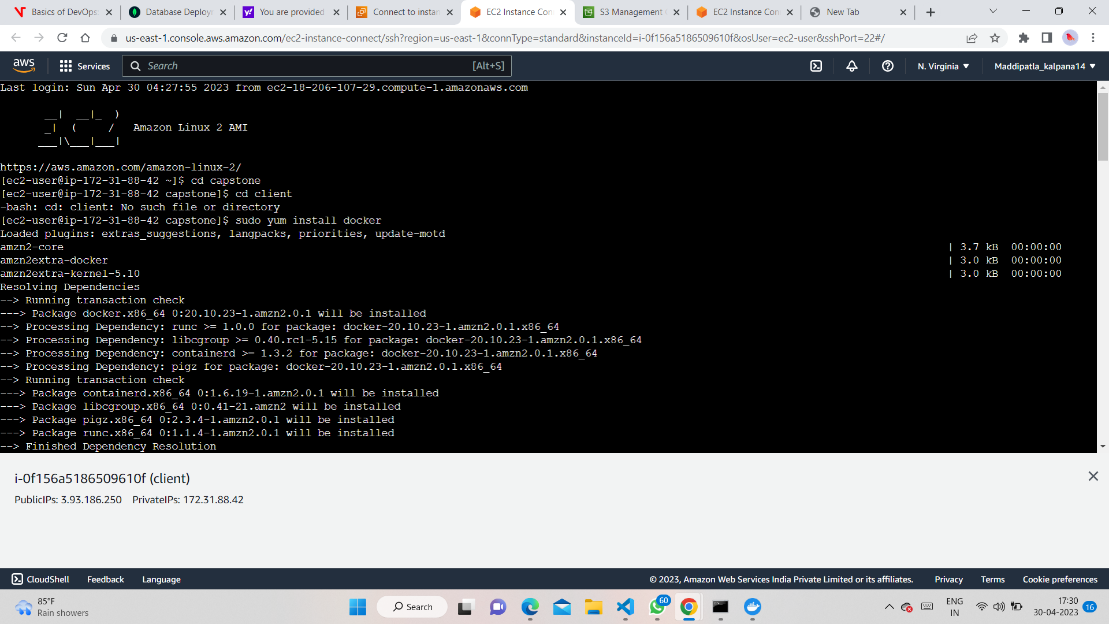
* Attach Elastic IP: To attach an Elastic IP to the EC2 instance, go to the EC2 dashboard, select the instance and click on the "Actions" dropdown menu. From there, select "Associate Elastic IP address" and follow the prompts to associate an Elastic IP with the instance.

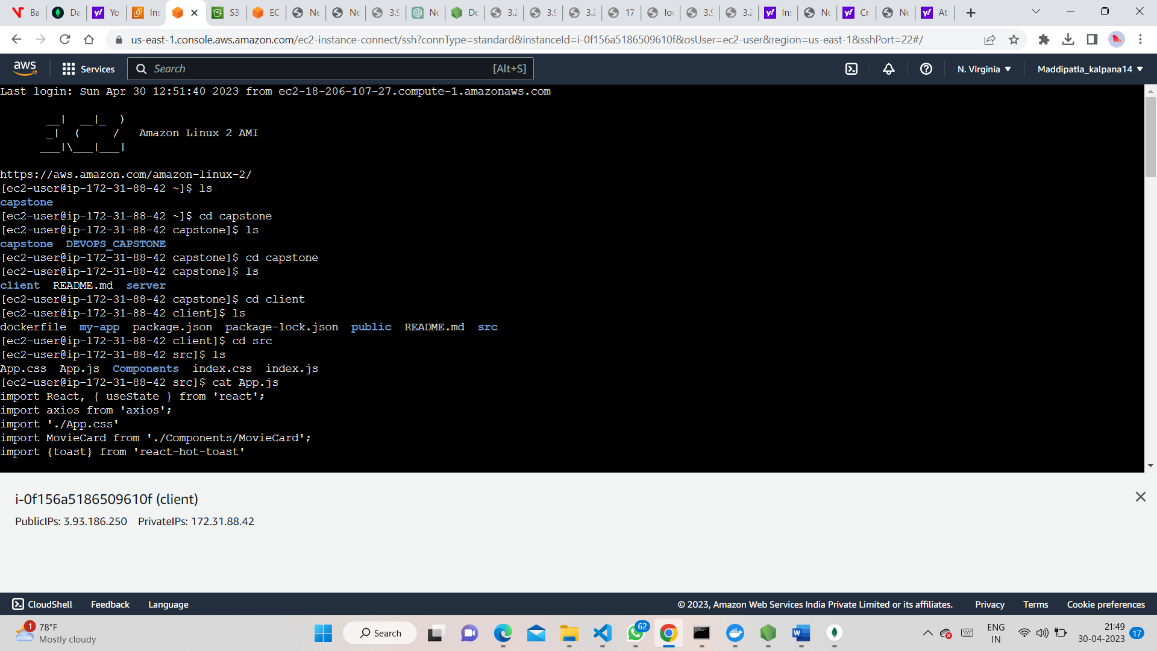




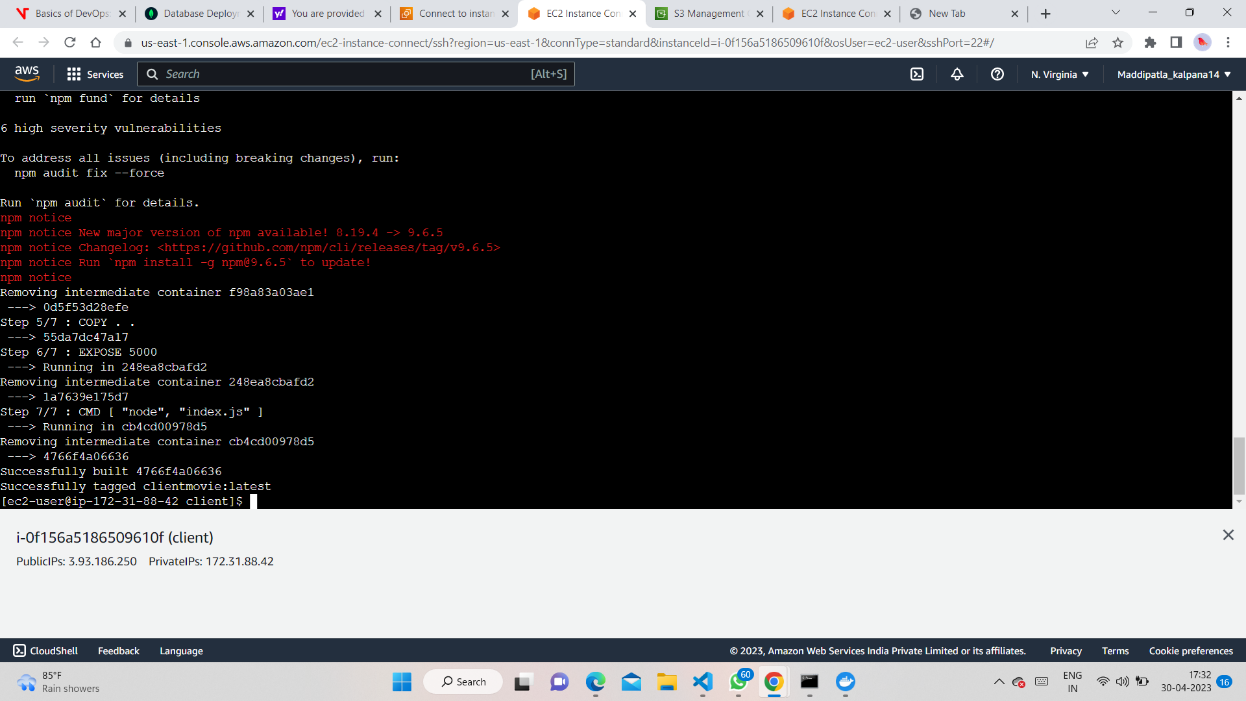
**Step-4:**

* Modify the Frontend code to be able to fetch data from Backend. Finally, deploy frontend using docker into EC2 instance.
* Update the Frontend code to fetch data from the Backend API instead of a local API. You can update the API endpoint in the src/App.js file or wherever the API calls are being made.





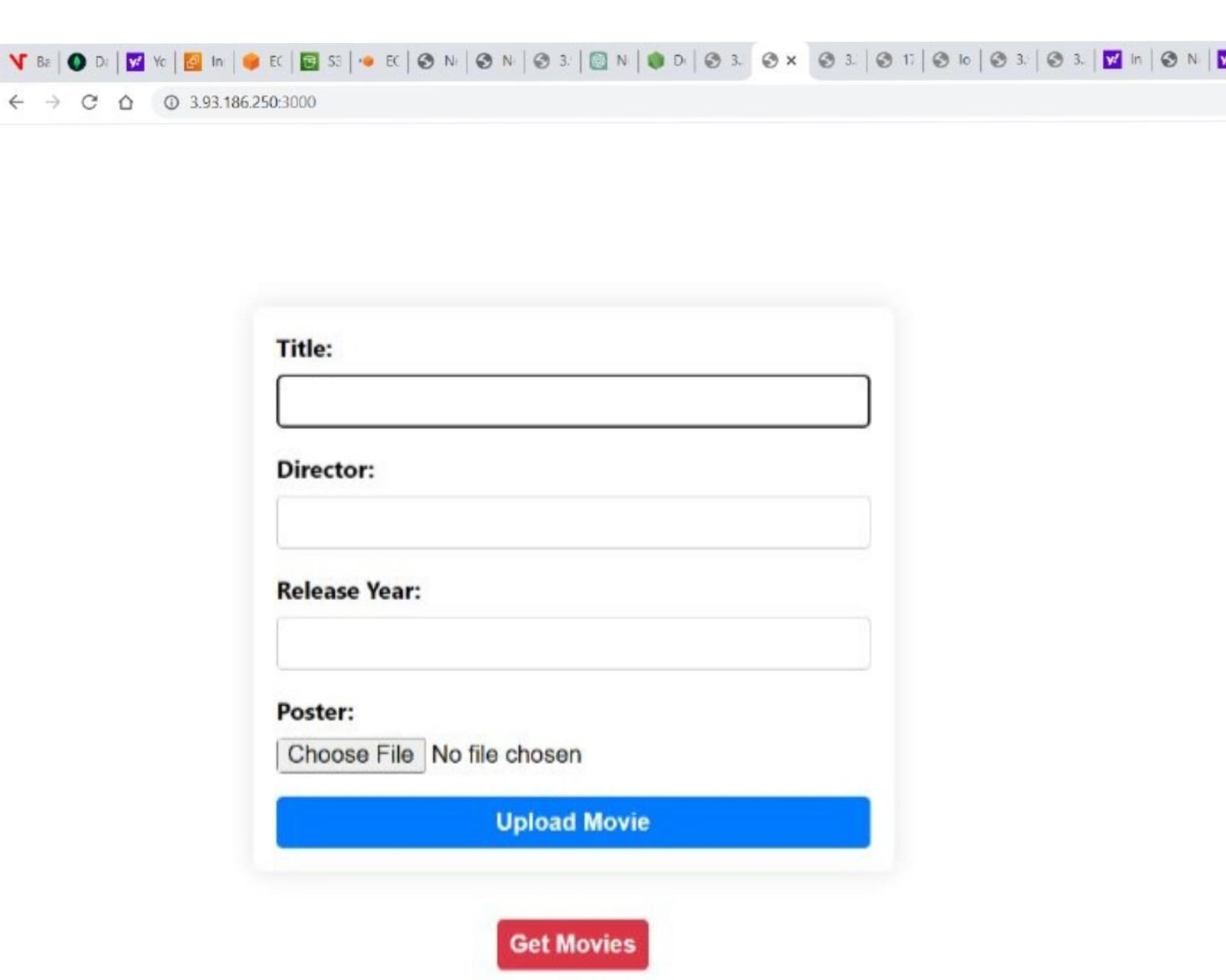
* Build a Docker image for the Frontend application. You can use a Docker file to define the image build process.



* Deploy the Docker image to an EC2 instance using a Docker container

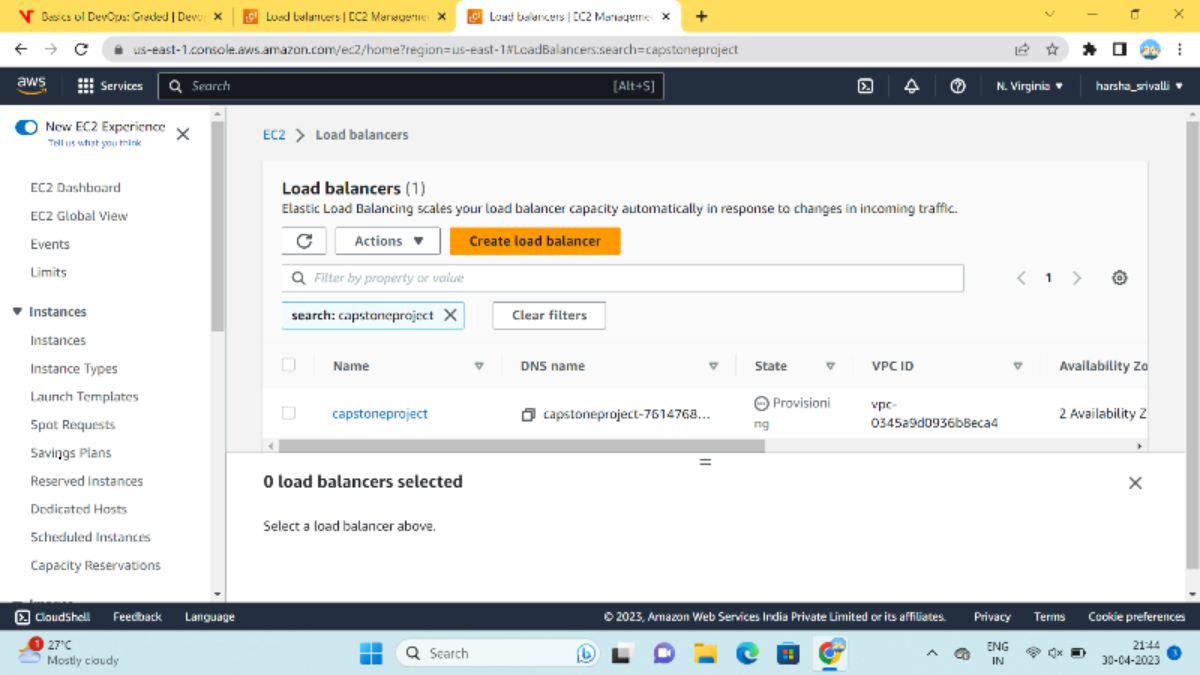


* You can now access the deployed Frontend application by visiting the EC2 instance's public IP address or the domain name associated with the instance



**Step-4:**

* Create an Elastic Load Balancer (ELB) in the AWS console. Select the appropriate type of load balancer based on your requirements.

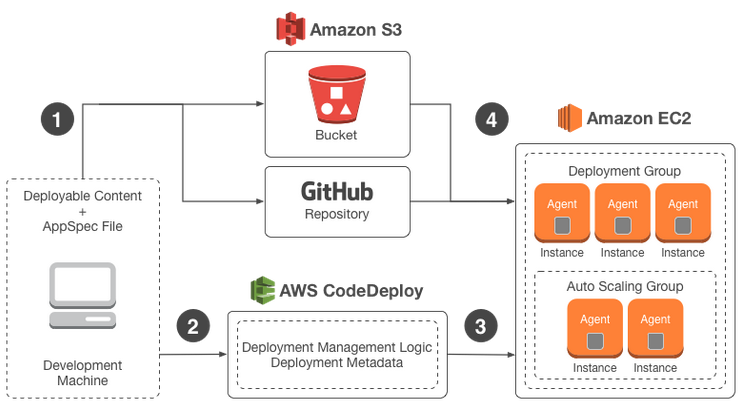


* Configure the load balancer to listen on the appropriate ports and forward traffic to our backend instances.
* Register your backend instances with the load balancer. Set up a target group to group our backend instances and configure the load balancer to use this target group.
* Test the load balancer by accessing your website through the load balancer's DNS name. Point your website's domain name to the load balancer's DNS name using a CNAME record. Monitor the traffic on our website and adjust the load balancer settings as necessary to optimize performance and availability.

**Step-5:**

* Amazon S3 bucket to store static assets such as images, videos, and audio files.
* Elastic Load Balancer (ELB) to distribute incoming traffic to the backend servers running in the Auto Scaling Group.
* EC2 instances to host the Node.js application servers in Docker containers. The Docker images can be stored in Amazon ECR for easy deployment to multiple regions.

**AWS deployment diagram**



**Step-6:**

* Host docker images into AWS ECR/Docker hub.
* Create an ECR repository: In the AWS Management Console, navigate to the Elastic Container Registry service and create a new repository. Authenticate Docker client to your registry: Run the aws ecr get-login-password command to retrieve a token that can be used to authenticate your Docker client to your registry. Build and tag your Docker image: Use the. docker build command to build your Docker image, and tag it with the ECR repository URI. Push the Docker image to the ECR repository: Use the docker push command to push the Docker image to the ECR repository
* Use the docker push command to push the Docker image to Docker Hub.
* **CHALLENGES ARISED DURING PROJECT TIME:**
* We face most of the issues at the time of ec2 connect ,due to lack of internet.
* We face most of the issues while modifiying dockerfiles, the server code for configuration with s3-multer.
* We facing some error while connecting to MongoDB database.
* **REFERENCE:**
* **Mongodb:** <https://www.mongodb.com/cloud/atlas>.
* **AWS:** <https://docs.aws.amazon.com/>
* **S3-multer:** <https://github.com/expressjs/multer-s3>

**TASK ASSIGNED TO TEAM MEMBERS**

|  |  |  |
| --- | --- | --- |
| **SNO** | **TASKS ASSIGNED** | **TEAM MEMBERS** |
| 1 | Deploying client on EC2 using Docker. | Kalpana, Srivalli |
| 2 | Connecting the Server with Atlas MongoDB. | Yaswanth, swamy |
| 3 | Deploying server on EC2 using Docker. | Yaswanth, Sai vinay |
| 4 | Creation a target group and Load Balancer. | Sai vinay, suryakanthi |
| 5 | Creating IAM user and Configuring S3-bucket. | Swamy, Ramesh |
| 6 | Host docker images into AWS ECR/Docker hub. | Suryakanthi, Ramesh |
| 7 | Project documentation | Kalpana,Srivalli |

**CONCLUSION-**

Given Capstone project involves ReactJs, NodeJS, MongoDB, S3-multer and AWS. Our frontend application allows us to upload movies along with posters, Title, Director name and Movie released year and also, we have 2 features i.e., upload movies and get movies. We used S3 to store Posters. We deployed frontend and backend using Docker. User can get movies and the details of movies.