

DEVOPS CAPSTONE PROJECT

GROUP – 8 TEAM

- | | |
|-----------------------|---------------------|
| 1. ANNAPURNA | (20A91A0572) |
| 2. RAMALAKSHMI | (20A91A0579) |
| 3. RAJESWARI | (20A91A0590) |
| 4. HARSHITA | (20A91A05A4) |
| 5. SRI DEVI | (20A91A05C2) |
| 6. PRIYANKA | (20MH1A05D3) |

INDEX

- 1) ABOUT THE PROJECT
- 2) TECHNOLOGIES USED
- 3) PROCESS
- 4) DNS NAME
- 5) AWS DEPLOYMENT DIAGRAM
- 6) CONTRIBUTION OF TEAM MEMBERS

ABOUT THE PROJECT:

The project is about **Deploying an Movie Ticket Application Using Docker , Ec2, Load Balancer.**

WHY DOCKER?

1. The docker has its own advantages of deploying the applications in an efficient way.

They are :

- Portability
 - Consistency
 - Scalability
 - Resource efficiency
 - Security
2. Overall, using Docker for deploying applications can greatly simplify the deployment process and improve the reliability, scalability, and efficiency of the application. With Docker, developers can focus on writing code and building features, without worrying about the underlying infrastructure or deployment process.

WHY LOADBALANCER AND EC2?

A load balancer is a networking device that distributes incoming network traffic across multiple servers to ensure that no single server becomes overloaded. The load balancer acts as a traffic cop, routing client requests to the appropriate server based on a variety of factors such as server availability, server response time, and server utilization.

Load balancers can improve the availability, scalability, and performance of web applications and services by distributing traffic evenly across multiple servers. They can also provide features such as health checks, session persistence, SSL termination, and content caching to further optimize the performance and reliability of the application.

When using Amazon Elastic Compute Cloud (EC2) with a load balancer, multiple EC2 instances are launched in different availability zones (AZs) and registered with the load balancer. The load balancer distributes incoming traffic across the registered EC2 instances, based on the rules and algorithms defined in the load balancer configuration.

Overall, using EC2 with a load balancer provides a reliable, scalable, and cost-effective infrastructure for running web applications or services in the cloud.

Technologies Used:

We have used various technologies basing on the requirement.

GIT: Git is a distributed version control system used for tracking changes in source code during software development. It was created by Linus Torvalds in 2005 and is now widely used by developers for managing source code and collaborating on software development projects. With Git, developers can create repositories to store their code, make changes to it, and keep track of changes made by themselves and other contributors. Git also allows for branching and merging, which enables multiple developers to work on different versions of the code simultaneously and then merge their changes back into a single codebase.

AWS: AWS stands for Amazon Web Services, which is a cloud computing platform offered by Amazon. AWS provides a wide range of cloud-based services, including computing, storage, databases, analytics, machine learning, networking, and more. With AWS, businesses and individuals can quickly and easily access computing resources on a pay-as-you-go basis, without having to invest in expensive infrastructure or hardware. AWS offers scalability, security, reliability, and flexibility, making it a popular choice for businesses of all sizes and industries. AWS also offers a wide range of tools and services for developers, such as AWS Lambda, which enables serverless computing, and Amazon S3, which provides scalable object storage.

DOCKER: Docker is a platform that allows developers to package, distribute, and run applications in containers. Containers are lightweight, standalone, and executable packages that contain everything needed to run an application, including code, libraries, system tools, and settings. Docker provides a standardized way to package and deploy applications, making it easier to move applications between different environments, such as development, testing, and production. Docker uses a client-server architecture and provides a command-line interface (CLI) for interacting with the Docker daemon, which runs on the host machine. Docker also integrates with other tools and technologies, such as Kubernetes, to provide a complete container-based ecosystem for developing, deploying, and managing applications.

Frontend: ReactJS

Backend: NodeJS

Database: MongoDB Atlas

ReactJS: It is an open-source JavaScript library for building user interfaces. It was developed by Facebook and is now widely used by developers to create reusable UI components and manage complex UI states in web applications.

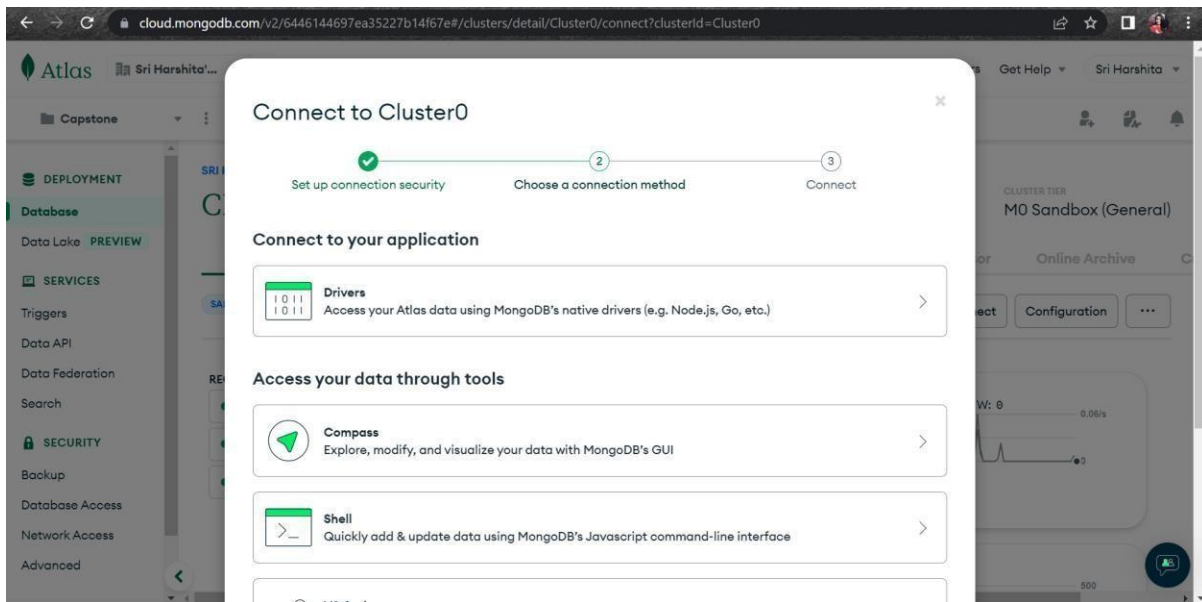
NodeJS: Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to build server-side applications using JavaScript.

MongoDB Atlas: It allows users to easily deploy, manage, and scale their MongoDB databases on major cloud platforms such as AWS, Azure, and Google Cloud.

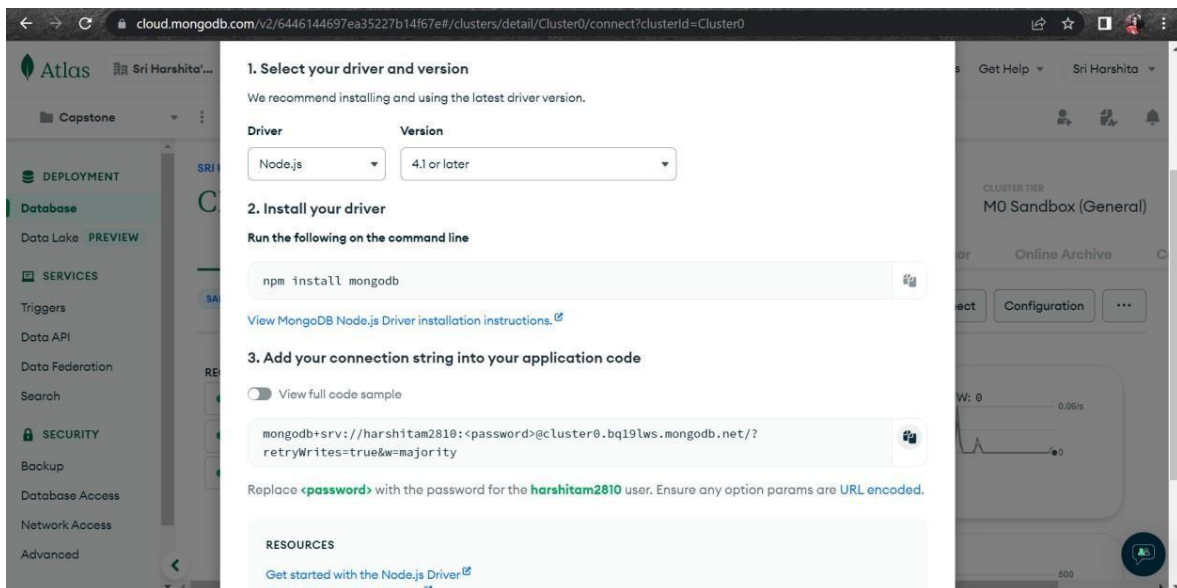
Process:

Step1: Establishing the connection with MongoDB Atlas

- We need to clone into the following repo https://github.com/snehal-herovired/DEVOPS_CAPSTONE for “Movie listing” website.
- Create a project namely (say->project1).. in MongoDB Atlas.
- Created a collection.



- Click on connect -> then to drivers -> there will be a URL copy it and paste that in the index.js



- Now paste this URL in the index.js file
- URL: `mongodb+srv://harshitam2810:admin@cluster0.bq19lws.mongodb.net/hero`

The screenshot shows a VS Code editor with a project named 'Capstone2'. The file explorer on the left shows the project structure. The main editor displays the `index.js` file, which contains the following code:

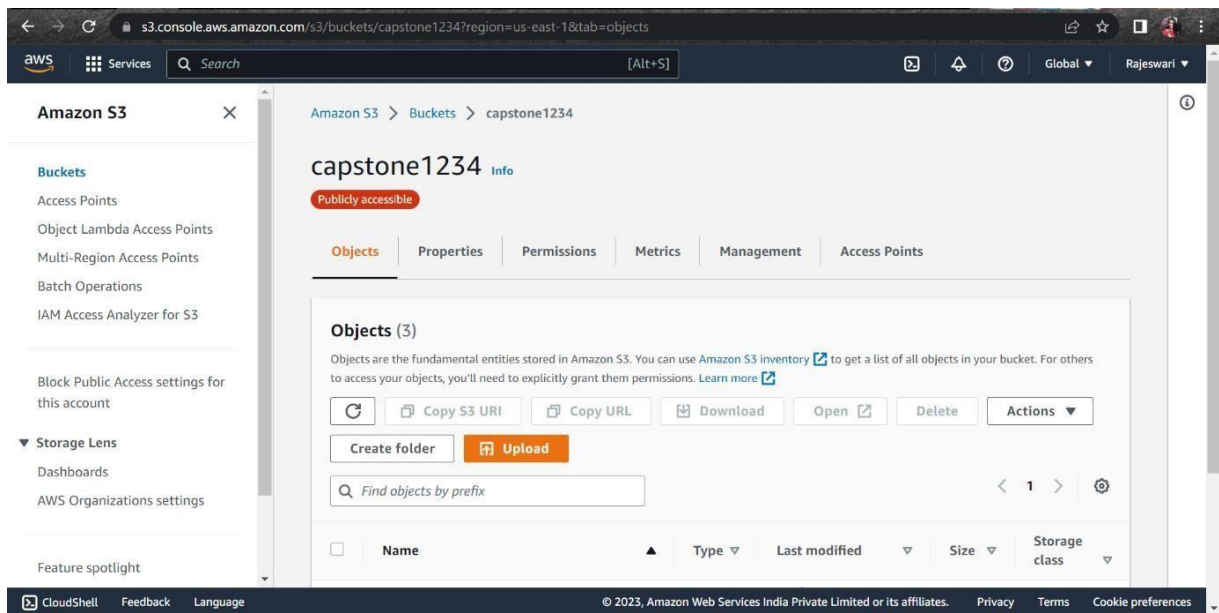
```

1 require('dotenv').config()
2 const mongoose = require('mongoose');
3 const cloudinary = require('cloudinary').v2;
4 const express = require('express');
5 const Movie = require('./Models/Movie');
6 const multer = require('multer');
7 const cors = require('cors');
8 const AWS = require('aws-sdk');
9 const fs = require('fs');
10 const s3 = new AWS.S3({
11   accessKeyId: process.env.AWS_ACCESS_KEY_ID,
12   secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
13 });
14 mongoose.connect('mongodb+srv://harshitam2810:admin@cluster0.bq19lws.mongodb.net/hero', { useNewUrlParser:
15   .then(() => console.log('Connected to MongoDB...'))
16   .catch(err => console.error('Could not connect to MongoDB...', err));
17 const app = express();
18 const port = 5000;
19 app.use(express.urlencoded({extended:true}))
20 app.use(express.json())
21 app.use(cors());

```

The terminal window at the bottom shows the command `node index.js` being executed. The output indicates that the server is listening at `http://localhost:5000` and that it has successfully connected to MongoDB.

- Creation of the S3 bucket to store images which gets injected to the website.



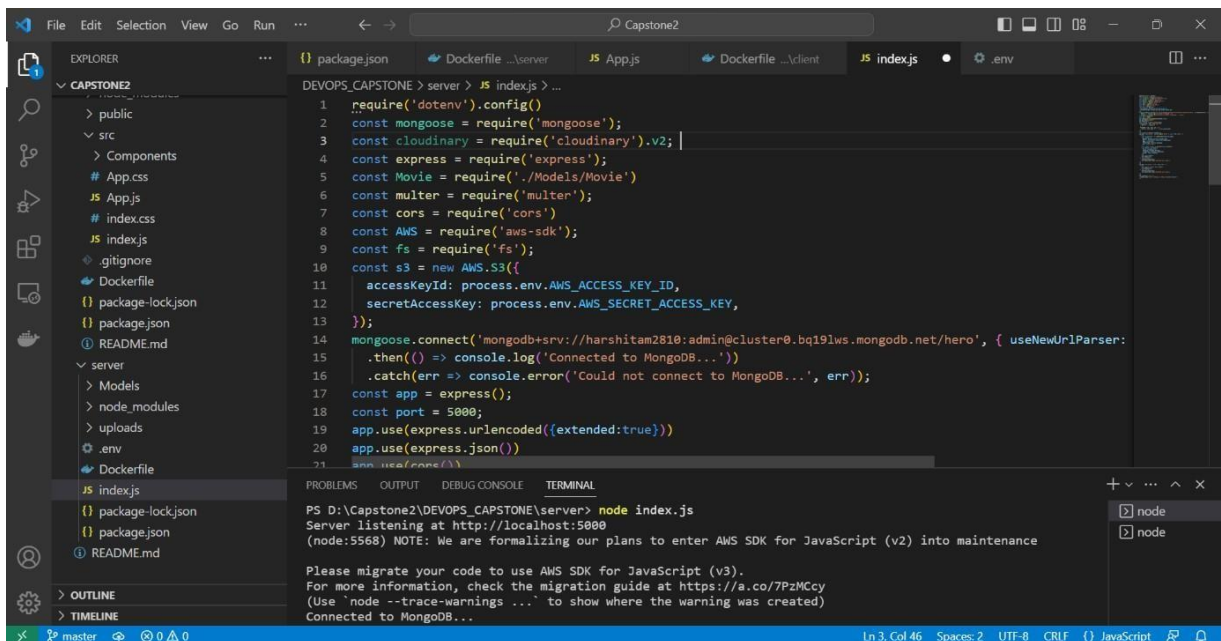
- We need to create a IAM user and then need to provide required permissions
- This must be done to upload images and access it.

STEP2: MODIFICATION OF THE CODE (BOTH FRONTEND AND BACKEND):

The AWS SDK, or AWS Software Development Kit, is a collection of libraries and tools that developers can use to interact with various AWS services programmatically. For javascript it supports three runtimes :JS for browser, Node.js for server, React Native for mobile development. So now accordingly we need to change the code.

- Paste the below code

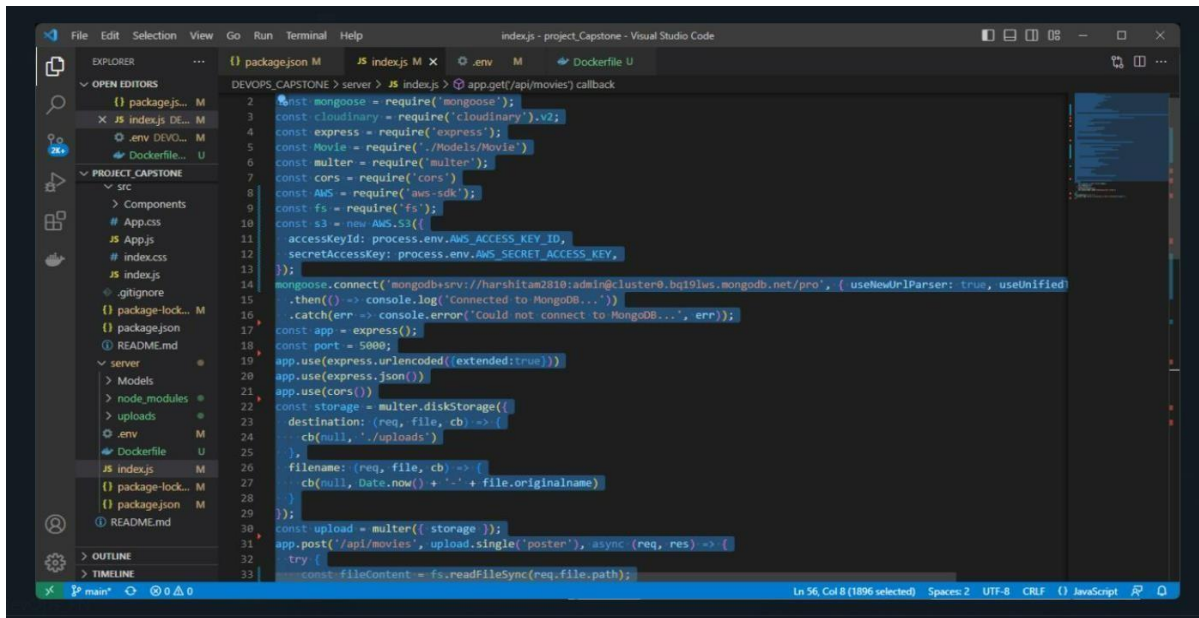
```
const AWS = require('aws-sdk');
const fs = require('fs');
const s3 = new AWS.S3({
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
});
```



```
1 require('dotenv').config()
2 const mongoose = require('mongoose');
3 const cloudinary = require('cloudinary').v2;
4 const express = require('express');
5 const Movie = require('./Models/Movie');
6 const multer = require('multer');
7 const cors = require('cors');
8 const AWS = require('aws-sdk');
9 const fs = require('fs');
10 const s3 = new AWS.S3({
11   accessKeyId: process.env.AWS_ACCESS_KEY_ID,
12   secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
13 });
14 mongoose.connect('mongodb+srv://harshitam2810:admin@cluster0.bq19lws.mongodb.net/hero', { useNewUrlParser:
15   .then(() => console.log('Connected to MongoDB...'))
16   .catch(err => console.error('Could not connect to MongoDB...', err));
17 const app = express();
18 const port = 5000;
19 app.use(express.urlencoded({extended:true}))
20 app.use(express.json())
21 app.use(cors());
```

PS D:\Capstone2\DEVOPS_CAPSTONE\server> node index.js
Server listening at http://localhost:5000
(node:5568) NOTE: We are formalizing our plans to enter AWS SDK for JavaScript (v2) into maintenance
Please migrate your code to use AWS SDK for JavaScript (v3).
For more information, check the migration guide at <https://a.co/7PzMCcy>
(Use 'node --trace-warnings ...' to show where the warning was created)
Connected to MongoDB...

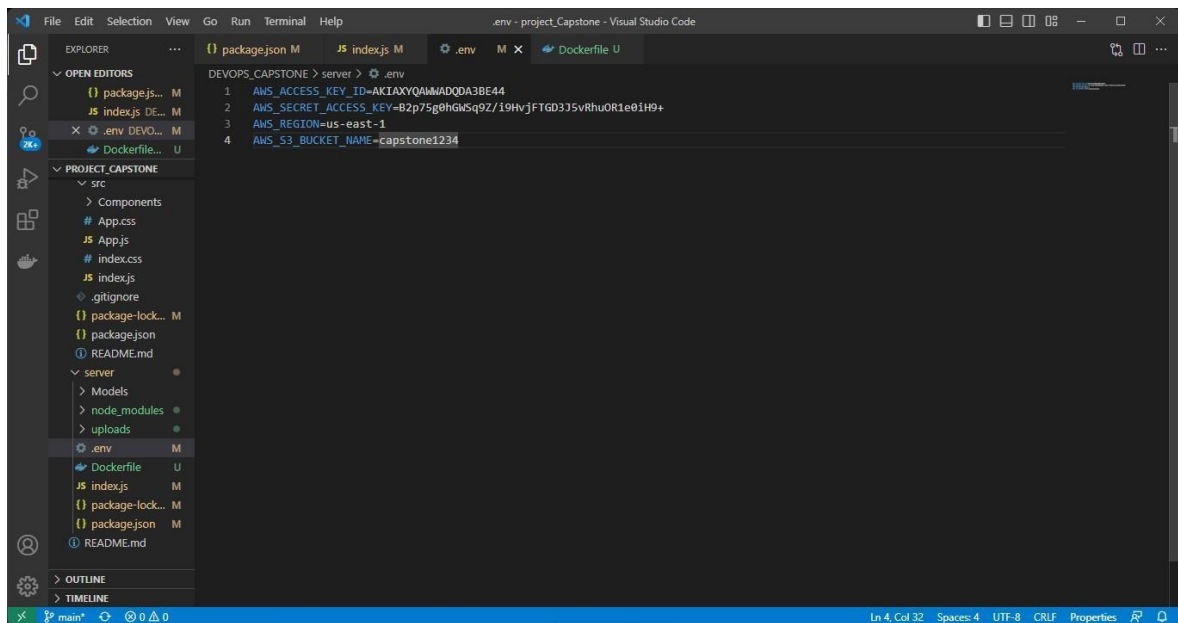
Multer: Multer is a middleware library for handling file uploads in Node.js web applications. It simplifies the process of handling file uploads by providing an easy-to-use API for handling multipart/form-data, which is the content type used for file uploads.



The screenshot shows a Visual Studio Code editor with the 'index.js' file open. The file contains the following code:

```
1 const mongoose = require('mongoose');
2 const cloudinary = require('cloudinary').v2;
3 const express = require('express');
4 const Movie = require('./Models/Movie');
5 const multer = require('multer');
6 const cors = require('cors');
7 const AWS = require('aws-sdk');
8 const fs = require('fs');
9 const s3 = new AWS.S3({
10   accessKeyId: process.env.AWS_ACCESS_KEY_ID,
11   secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
12 });
13 mongoose.connect('mongodb+srv://harshitam2810:admin@cluster0-bq191ws.mongodb.net/pro', { useNewUrlParser: true, useUnifiedTopology: true });
14 .then(() => console.log('Connected to MongoDB...'))
15 .catch(err => console.error('Could not connect to MongoDB...', err));
16 const app = express();
17 const port = 5000;
18 app.use(express.urlencoded({ extended: true }));
19 app.use(express.json());
20 app.use(cors());
21 const storage = multer.diskStorage({
22   destination: (req, file, cb) => {
23     cb(null, './uploads');
24   },
25   filename: (req, file, cb) => {
26     cb(null, Date.now() + '-' + file.originalname);
27   }
28 });
29 const upload = multer({ storage });
30 app.post('/api/movies', upload.single('poster'), async (req, res) => {
31   try {
32     const fileContent = fs.readFileSync(req.file.path);
33     const data = { title: req.body.title, poster: fileContent.toString('base64') };
34     const movie = new Movie(data);
35     movie.save().then(() => res.json(movie)).catch(err => res.status(500).json({ error: err.message }));
36   } catch (err) {
37     res.status(500).json({ error: err.message });
38   }
39 });
40 app.listen(port, () => console.log(`Server is running on port ${port}`));
```

- We need to copy the access key and ID with our credentials.



The screenshot shows a Visual Studio Code editor with the '.env' file open. The file contains the following code:

```
1 AWS_ACCESS_KEY_ID=AKIA3YQWAMADQ3BE44
2 AWS_SECRET_ACCESS_KEY=B2p75g0hGmSq9Z/19HvJFTGD335vRhuOR1e0IH9+
3 AWS_REGION=us-east-1
4 AWS_S3_BUCKET_NAME=capstone1234
```


Step3: Containerization of code:

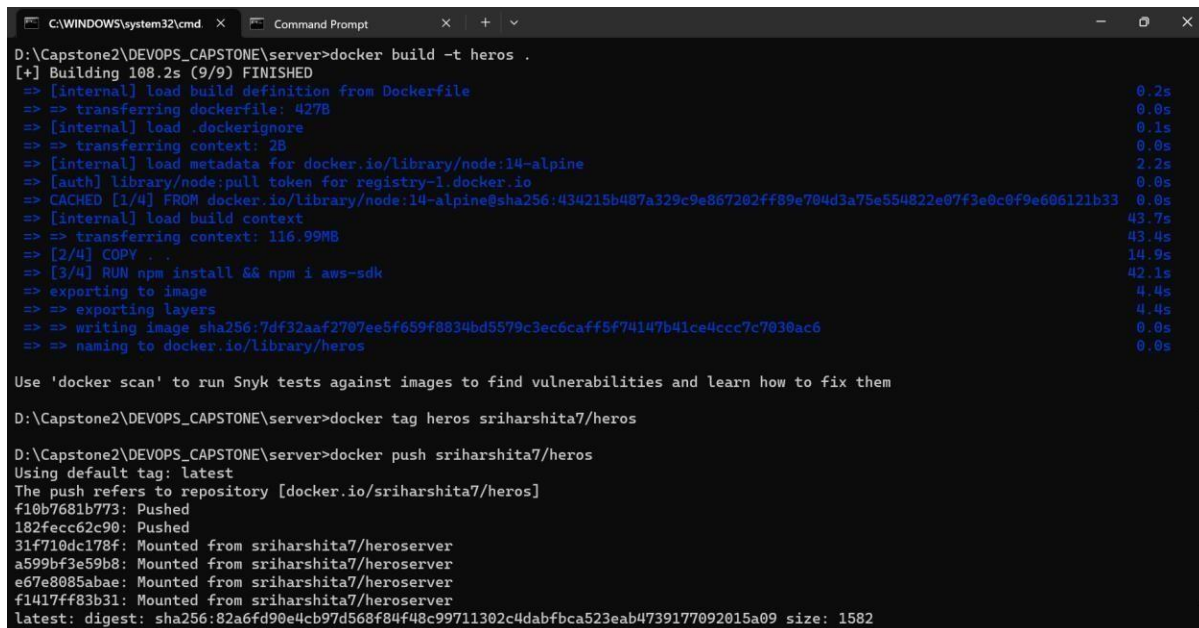
- Containerization is done so that all the files i.e code files of both front-end and back-end gets uploaded as docker images and when we try to pull these docker images using Ec2 instance then files gets stored in Ec2 instance.
- Now we need to create a docker file for our server. A Dockerfile is a text file that contains instructions for building a Docker image.
- A Dockerfile typically includes the following elements:

Base Image	Environment Variables
Commands	Exposed Ports
Container Metadata	Entrypoint etc..

Docker file for Server:

```
FROM node:14
WORKDIR /app
COPY. /app
RUN npm install
EXPOSE 5000
CMD ["npm", "start"]
```

- Docker build → for building images.
- Docker push → for pushing docker images into docker hub.



```
C:\WINDOWS\system32\cmd  x  Command Prompt  x  +  v
D:\Capstone2\DEVOPS_CAPSTONE\server>docker build -t heros .
[+] Building 108.2s (9/9) FINISHED
=> [internal] load build definition from Dockerfile                                0.2s
=> => transferring dockerfile: 427B                                              0.0s
=> [internal] load .dockerignore                                                  0.1s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/node:14-alpine                 2.2s
=> [auth] library/node:pull token for registry-1.docker.io                     0.0s
=> CACHED [1/4] FROM docker.io/library/node:14-alpine@sha256:434215b487a329c9e867202ff89e704d3a75e554822e07f3e0c0f9e606121b33  0.0s
=> [internal] load build context                                                  43.7s
=> => transferring context: 116.99MB                                           43.4s
=> [2/4] COPY . .                                                                14.9s
=> [3/4] RUN npm install && npm i aws-sdk                                       42.1s
=> exporting to image                                                            4.4s
=> => exporting layers                                                            4.4s
=> => writing image sha256:7df32aaf2707ee5f659f8834bd5579c3ec6cafff5f74147b41ce4ccc7c7030ac6  0.0s
=> => naming to docker.io/library/heros                                         0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

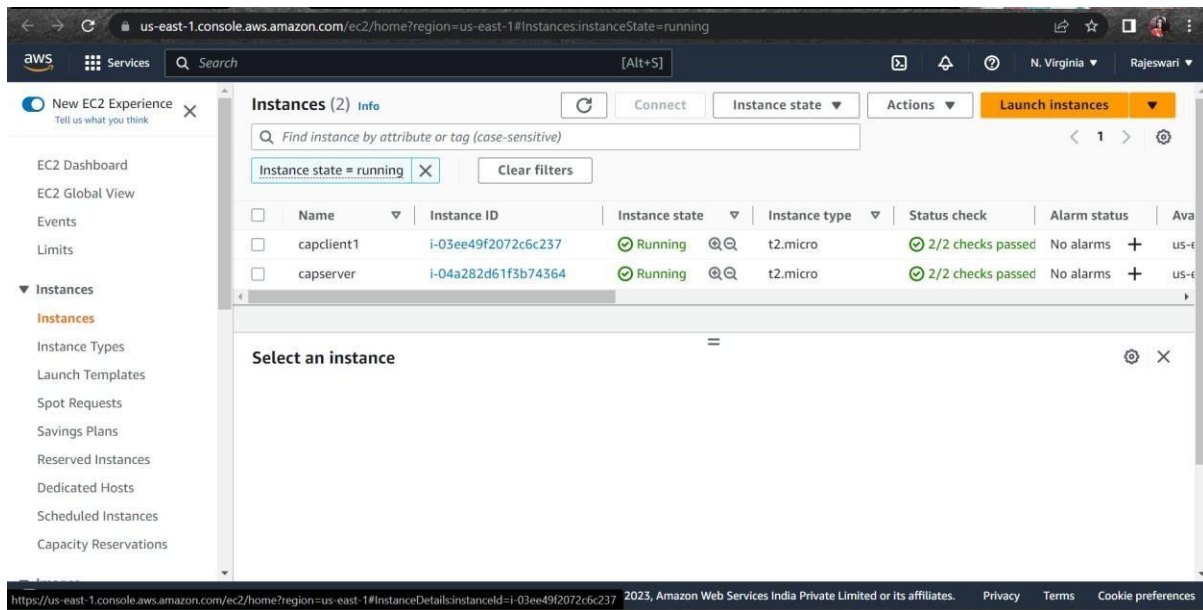
D:\Capstone2\DEVOPS_CAPSTONE\server>docker tag heros sriharshita7/heros

D:\Capstone2\DEVOPS_CAPSTONE\server>docker push sriharshita7/heros
Using default tag: latest
The push refers to repository [docker.io/sriharshita7/heros]
f10b7681b773: Pushed
182fecc62c90: Pushed
31f710dc178f: Mounted from sriharshita7/heroserver
a599bf3e59b8: Mounted from sriharshita7/heroserver
e67e8085abae: Mounted from sriharshita7/heroserver
f1417ff83b31: Mounted from sriharshita7/heroserver
latest: digest: sha256:82a6fd90e4cb97d568f84f48c99711302c4dabfbca523eab4739177092015a09 size: 1582
```

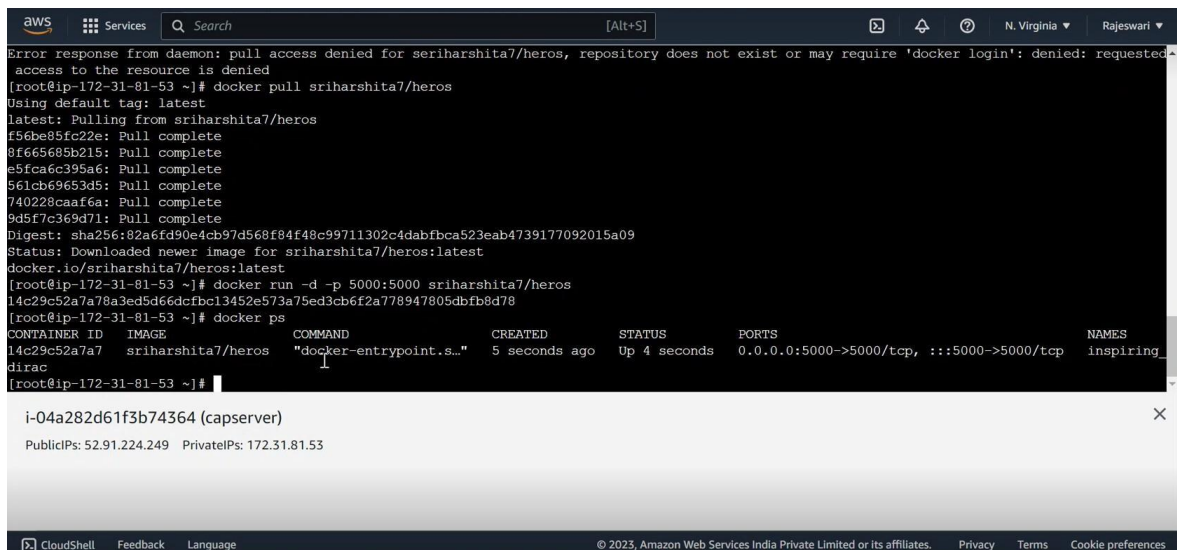
- Successfully pushed docker image (heros) into the docker hub.

Step 4: Deployment of server on Ec2 using Docker:

- Created EC2 instances for Server and Client.

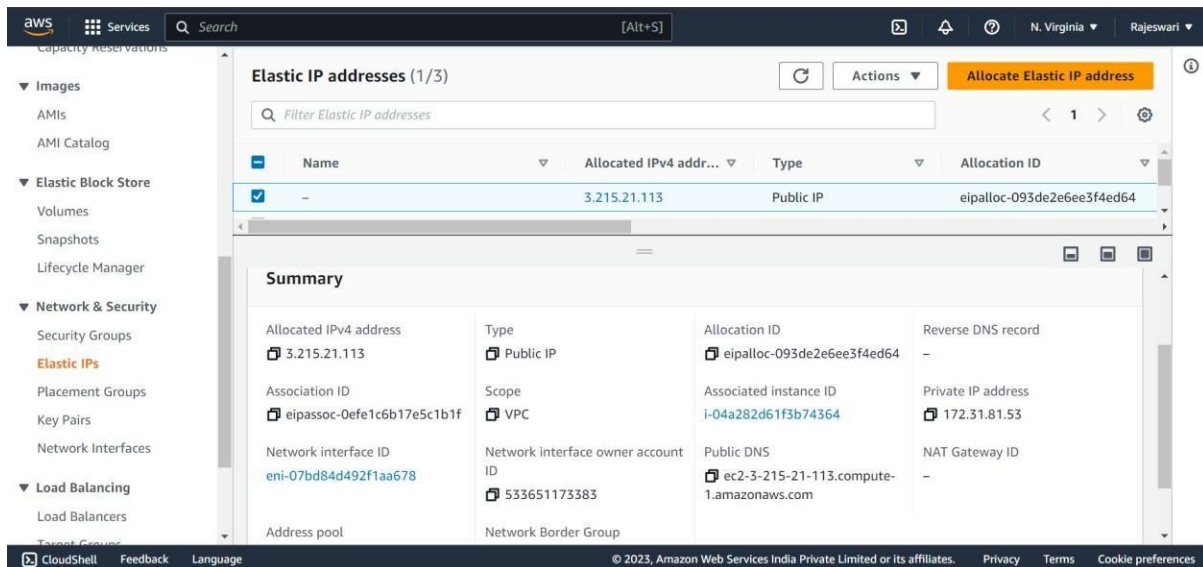


- Connect Ec2 instance
- Give command **"sudo su -"** to go to root user.
- Provide the command **"yum install docker"** to install docker in EC2 instance.
- Start the docker using **"systemctl start docker"**.
- We need to login into docker using the command **"docker login"**.
- We need to pull images using command **"docker pull <username>/imagename"**.
- Now run the docker image using command **-- docker run -d -p 5000:5000 image_name**.

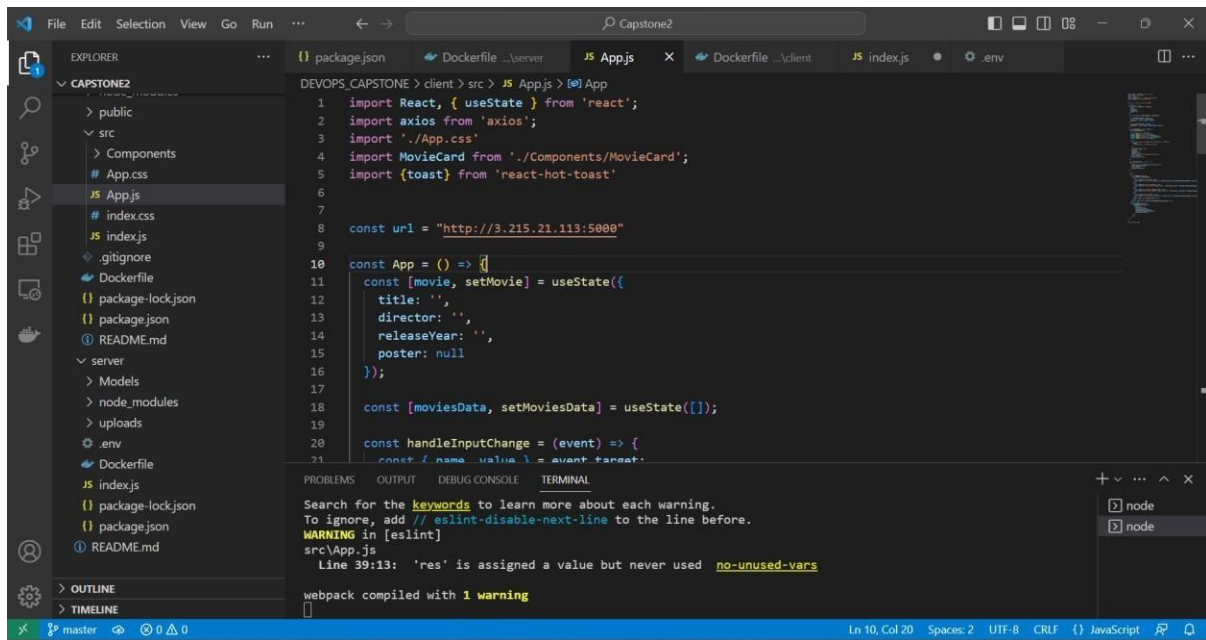


Step5:Creating docker-image & Deployment of client on Ec2 using docker

- Associated elastic IP to the server EC2 machine.



- Modified local host with elastic IP of the server instance.



- Adding of dockerfile that can be used to call all commands on CLI

Client's 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 ./

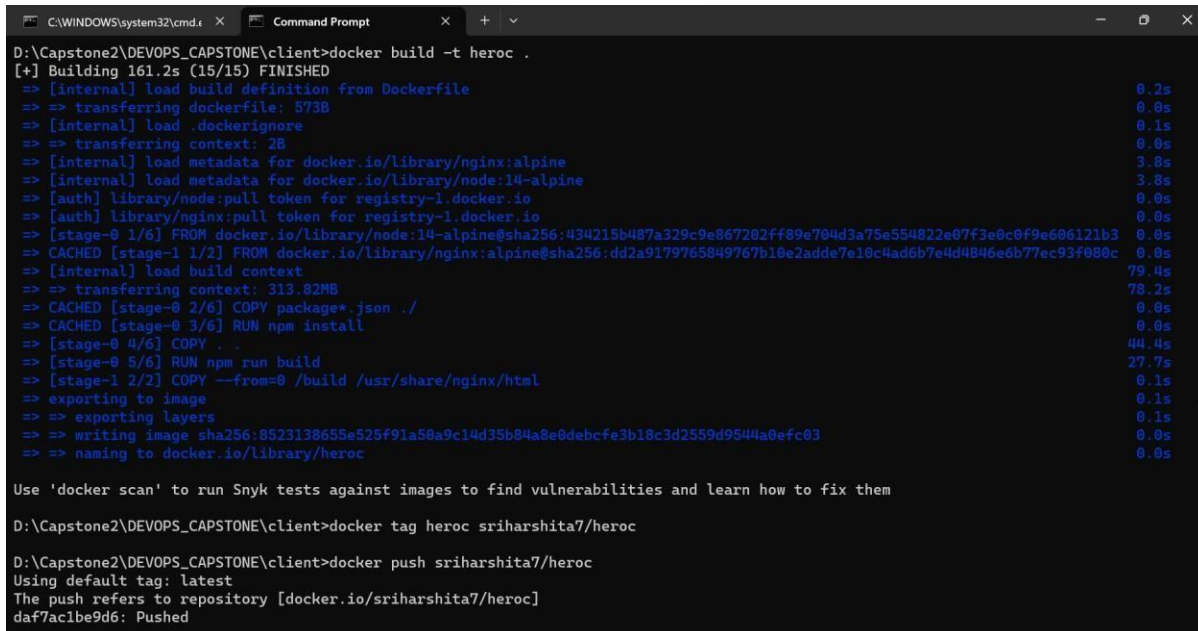
# Install dependencies
RUN npm install

# 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 5000
CMD ["nginx", "-g", "daemon off;"]
```

- Docker build → for building images.
- Docker push → for pushing docker images into docker hub.



```
C:\WINDOWS\system32\cmd.exe x Command Prompt
D:\Capstone2\DEVOPS_CAPSTONE\client>docker build -t heroc .
[+] Building 161.2s (15/15) FINISHED
=> [internal] load build definition from Dockerfile 0.2s
=> => transferring dockerfile: 573B 0.0s
=> [internal] load .dockerignore 0.1s
=> => transferring context: 28 0.0s
=> [internal] load metadata for docker.io/library/nginx:alpine 3.8s
=> [internal] load metadata for docker.io/library/node:14-alpine 3.8s
=> [auth] library/node:pull token for registry-1.docker.io 0.0s
=> [auth] library/nginx:pull token for registry-1.docker.io 0.0s
=> [stage-0 1/6] FROM docker.io/library/node:14-alpine@sha256:434215b487a329c9e867202ff89e704d3a75e554822e07f3e0c0f9e606121b3 0.0s
=> CACHED [stage-1 1/2] FROM docker.io/library/nginx:alpine@sha256:dd2a9179765849767b10e2adde7e10c4ad6b7e4d4846e6b77ec93f080c 0.0s
=> [internal] load build context 79.4s
=> => transferring context: 313.82MB 78.2s
=> CACHED [stage-0 2/6] COPY package*.json ./ 0.0s
=> CACHED [stage-0 3/6] RUN npm install 0.0s
=> [stage-0 4/6] COPY . . 44.4s
=> [stage-0 5/6] RUN npm run build 27.7s
=> [stage-1 2/2] COPY --from=0 /build /usr/share/nginx/html 0.1s
=> exporting to image 0.1s
=> exporting layers 0.1s
=> writing image sha256:8523138655e525f91a50a9c14d35b84a8e0debcfe3b18c3d2559d9544a0efc03 0.0s
=> naming to docker.io/library/heroc 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

D:\Capstone2\DEVOPS_CAPSTONE\client>docker tag heroc sriharshita7/heroc

D:\Capstone2\DEVOPS_CAPSTONE\client>docker push sriharshita7/heroc
Using default tag: latest
The push refers to repository [docker.io/sriharshita7/heroc]
daf7ac1be9d6: Pushed
```

```

C:\WINDOWS\system32\cmd.exe x Command Prompt x + v
=> CACHED [stage-1 1/2] FROM docker.io/library/nginx:alpine@sha256:dd2a9179765849767b10e2adde7e10c4ad6b7e4d4846e6b77ec93f080c 0.0s
=> [internal] load build context
=> => transferring context: 313.82MB 79.4s
=> CACHED [stage-0 2/6] COPY package*.json ./ 78.2s
=> CACHED [stage-0 3/6] RUN npm install 0.0s
=> [stage-0 4/6] COPY . 44.4s
=> [stage-0 5/6] RUN npm run build 27.7s
=> [stage-1 2/2] COPY --from=0 /build /usr/share/nginx/html 0.1s
=> exporting to image 0.1s
=> exporting layers 0.1s
=> writing image sha256:8523138655e525f91a50a9c14d35b84a8e0debcfe3b18c3d2559d9544a0efc03 0.0s
=> naming to docker.io/library/heroc 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

D:\Capstone2\DEVOPS_CAPSTONE\client>docker tag heroc sriharshita7/heroc

D:\Capstone2\DEVOPS_CAPSTONE\client>docker push sriharshita7/heroc
Using default tag: latest
The push refers to repository [docker.io/sriharshita7/heroc]
daf7ac1be9d6: Pushed
31531248c7cb: Mounted from sriharshita7/client3
f9cb3f1fd3d: Mounted from sriharshita7/client3
f0fb842dea41: Mounted from sriharshita7/client3
c1cd5c8c68ef: Mounted from sriharshita7/client3
1d54586a1706: Mounted from sriharshita7/client3
1003ff723696: Mounted from sriharshita7/client3
f1417ff83b31: Mounted from sriharshita7/heros
latest: digest: sha256:84483dd35105b1a2950ca0282ba120b00ffe3850a87e5bcc844d1d29a9aeb04f size: 1991

D:\Capstone2\DEVOPS_CAPSTONE\client>

```

Step 6: Deployment of Ec2 instance for client

- Give command “**sudo su –**” to go to root user.
- Provide the command “**yum install docker**” to install docker in EC2 instance.
- Start the docker using “**systemctl start docker**”.
- We need to login into docker using the command “**docker login**”.
- We need to pull images using command –“**docker pull <username>/imagename**”.
- Now run the docker image using command -- **docker run -d -p 3001:80 image_name**
-

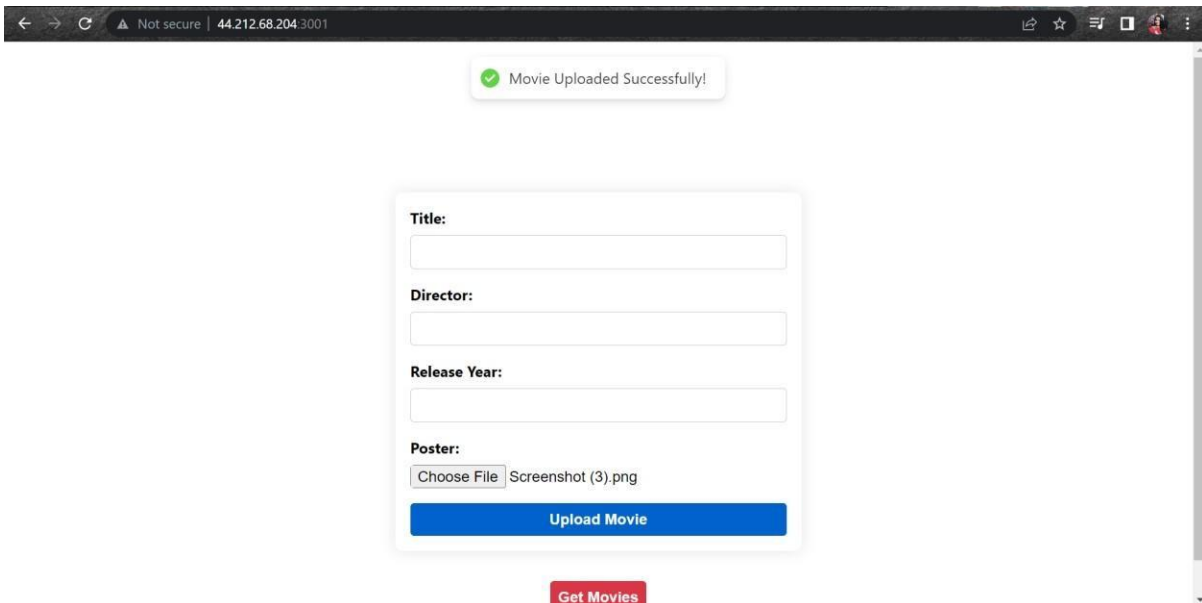
```

aws Services Q Search [Alt+S] N. Virginia Rajeswari
[root@ip-172-31-91-8 ~]# systemctl start docker
[root@ip-172-31-91-8 ~]# docker pull sriharshita7/heroc
Using default tag: latest
latest: Pulling from sriharshita7/heroc
f56be85fc22e: Pull complete
2ce963c369bc: Pull complete
59b9d2200e63: Pull complete
3e1e579c95fe: Pull complete
547a97583f72: Pull complete
1f21f983520d: Pull complete
c23b4f8cf279: Pull complete
0db1c6c9c3c9: Pull complete
Digest: sha256:84483dd35105b1a2950ca0282ba120b00ffe3850a87e5bcc844d1d29a9aeb04f
Status: Downloaded newer image for sriharshita7/heroc:latest
docker.io/sriharshita7/heroc:latest
[root@ip-172-31-91-8 ~]# docker run -d -p 3001:80 sriharshita7/heroc
53d33d17915313db940ddb38289aldbf9367257bdf31d2f468bd811bde80bec
[root@ip-172-31-91-8 ~]# docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED        STATUS        PORTS
53d33d179153   sriharshita7/heroc "/docker-entrypoint..." 5 seconds ago  Up 4 seconds  0.0.0.0:3001->80/tcp, :::3001->80/tcp
[root@ip-172-31-91-8 ~]#

i-03ee49f2072c6c237 (capclient1)
PublicIPs: 44.212.68.204 PrivateIPs: 172.31.91.8
CloudShell Feedback Language © 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

```


- Once after the deployment is successful check for website using public IP address of Client instance and at the same time listening to port number (44.212.68.204:3001) .
- The web page gets opened. Then give the required details.



← → ↻ Not secure | 44.212.68.204:3001

✓ Movie Uploaded Successfully!

Title:

Director:

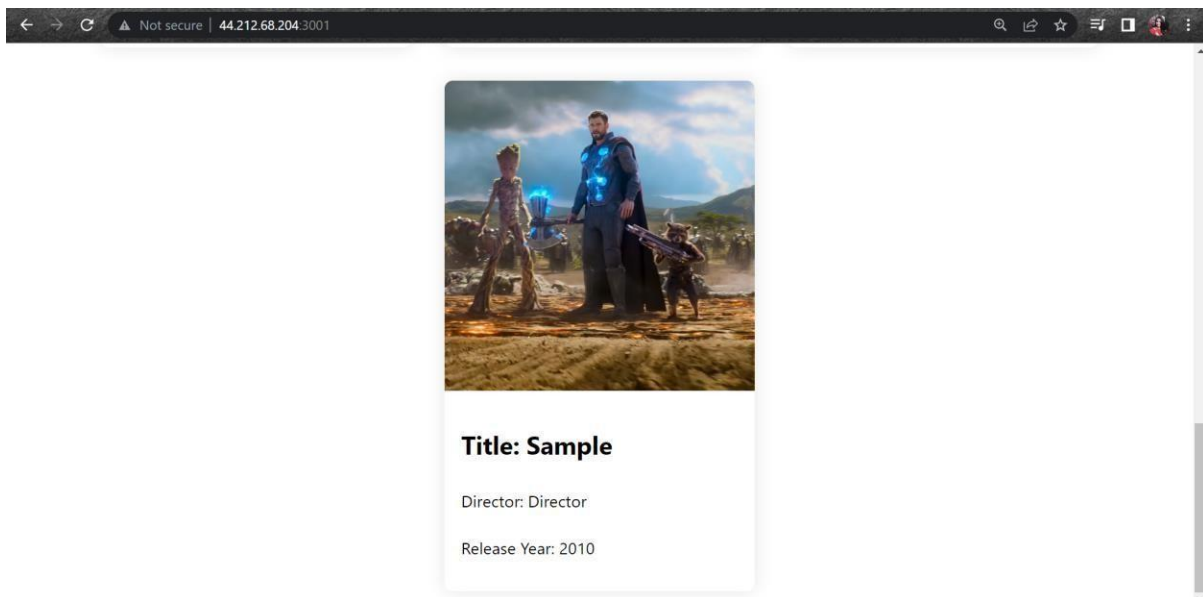
Release Year:

Poster:

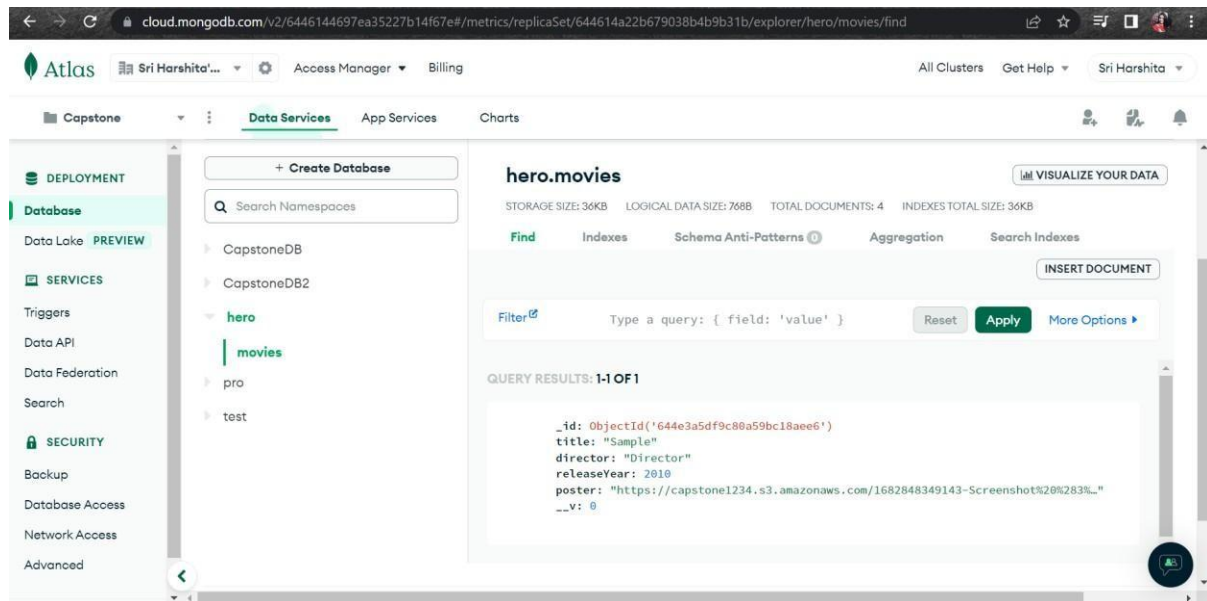
Choose File Screenshot (3).png

Upload Movie

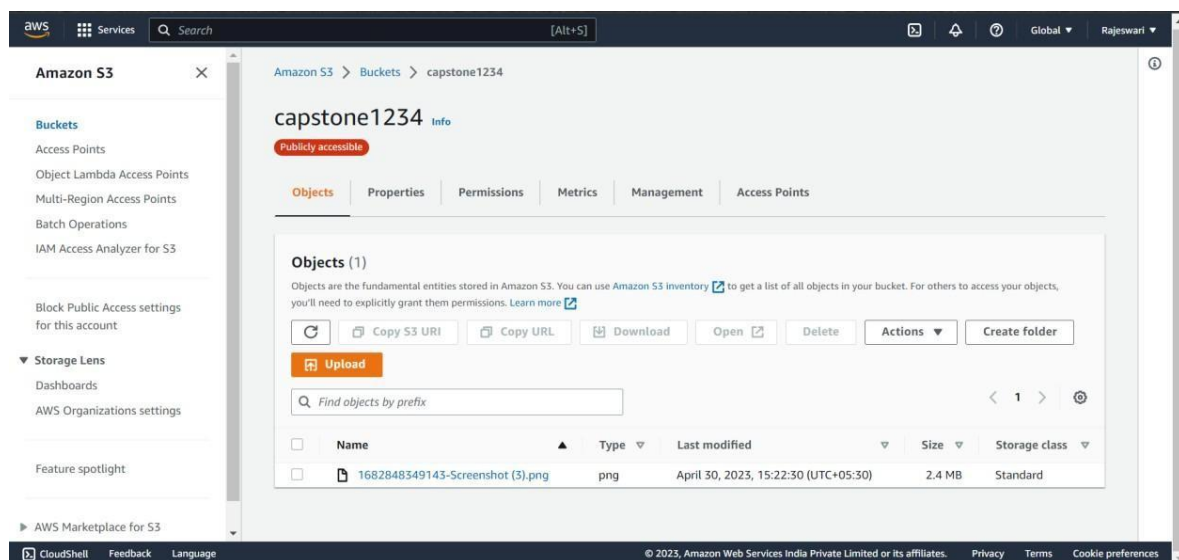
Get Movies



- The details are uploaded to the -MongoDb server



- Photos are uploaded to -Created S3 bucket.



Step7: Creation of target group and load balancer

- Created a target group and included client EC2 machine with TCP port 3001.

The screenshot shows the AWS Management Console interface for a target group named 'her0tg'. The left sidebar contains navigation links for EC2, Elastic Block Store, and Network & Security. The main content area displays the 'Details' tab for the target group 'arn:aws:elasticloadbalancing:us-east-1:533651173383:targetgroup/her0tg/af5552b62700c73'. The details include:

- Target type: Instance
- Protocol: Port TCP: 3001
- VPC: vpc-03b57040e0ba4d9ea
- IP address type: IPv4
- Load balancer: her0lb
- Total targets: 1
- Healthy: 1
- Unhealthy: 0
- Unused: 0
- Initial: 0
- Draining: 0

Below the details, there is a section for 'Registered targets (1)' with a table showing the target details:

Instance ID	Name	Port	Zone	Health status	Health status details
i-03ee49f2072e6c237	capclient1	3001	us-east-1c	Healthy	

- Created a network loadbalancer and attached the above target group.

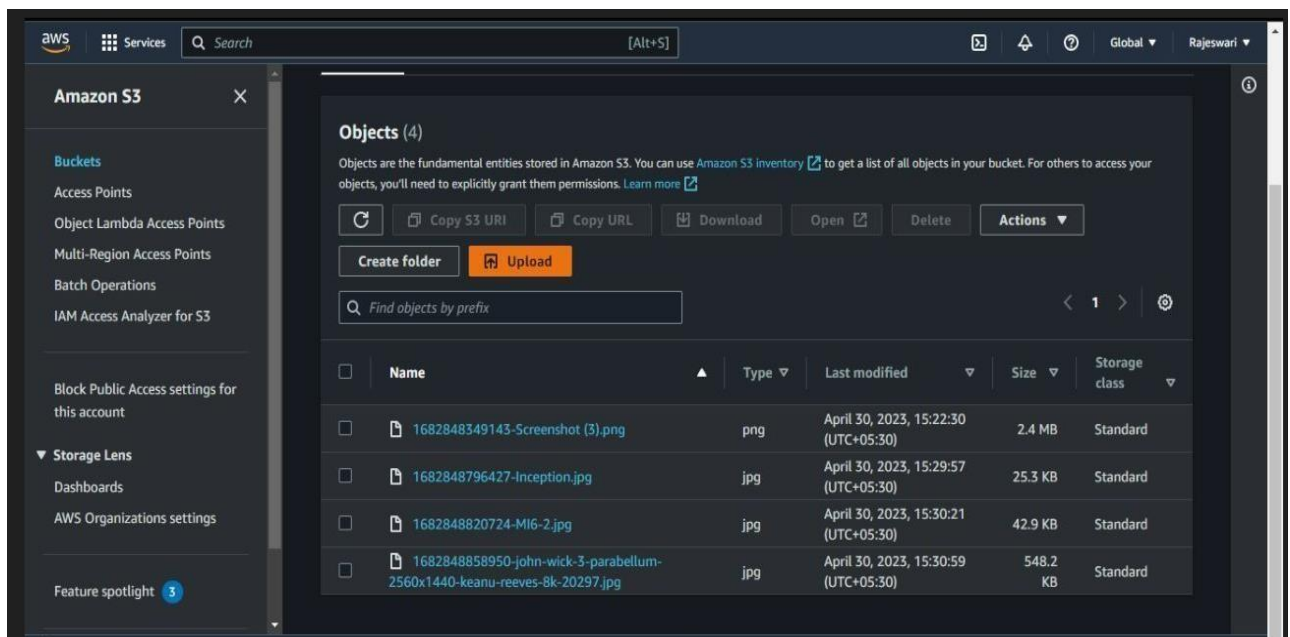
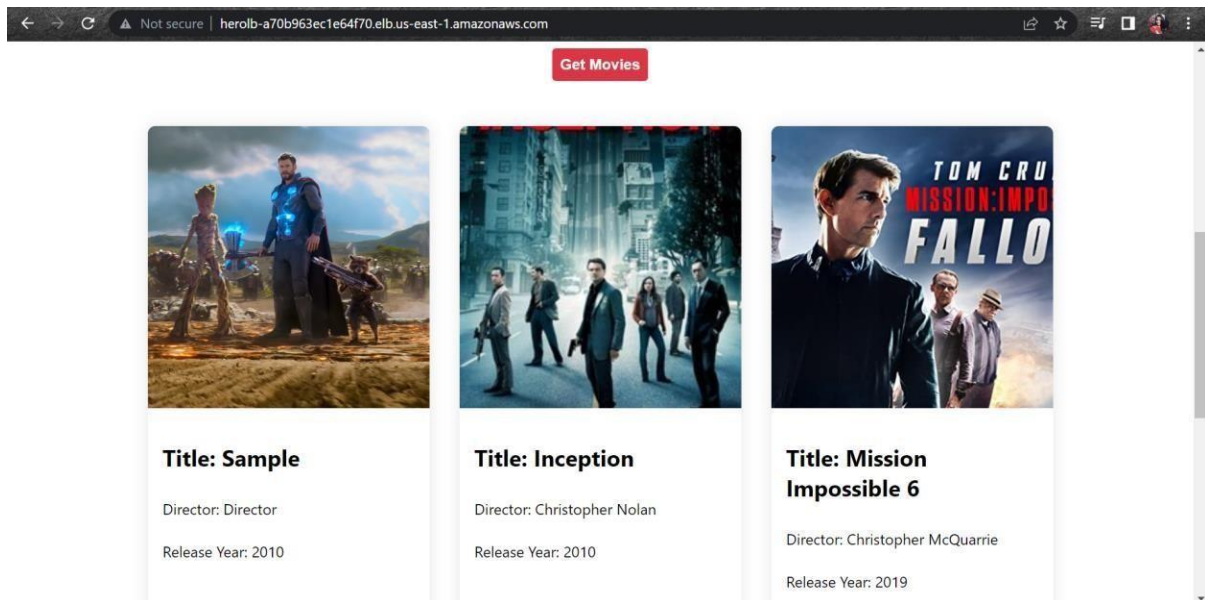
The screenshot shows the AWS Management Console interface for a network load balancer named 'her0lb'. The left sidebar contains navigation links for Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Scheduled Instances, Capacity Reservations, Images, Elastic Block Store, and Network & Security. The main content area displays the 'Details' tab for the load balancer 'arn:aws:elasticloadbalancing:us-east-1:533651173383:loadbalancer/net/her0lb/a70b963ec1e64f70'. The details include:

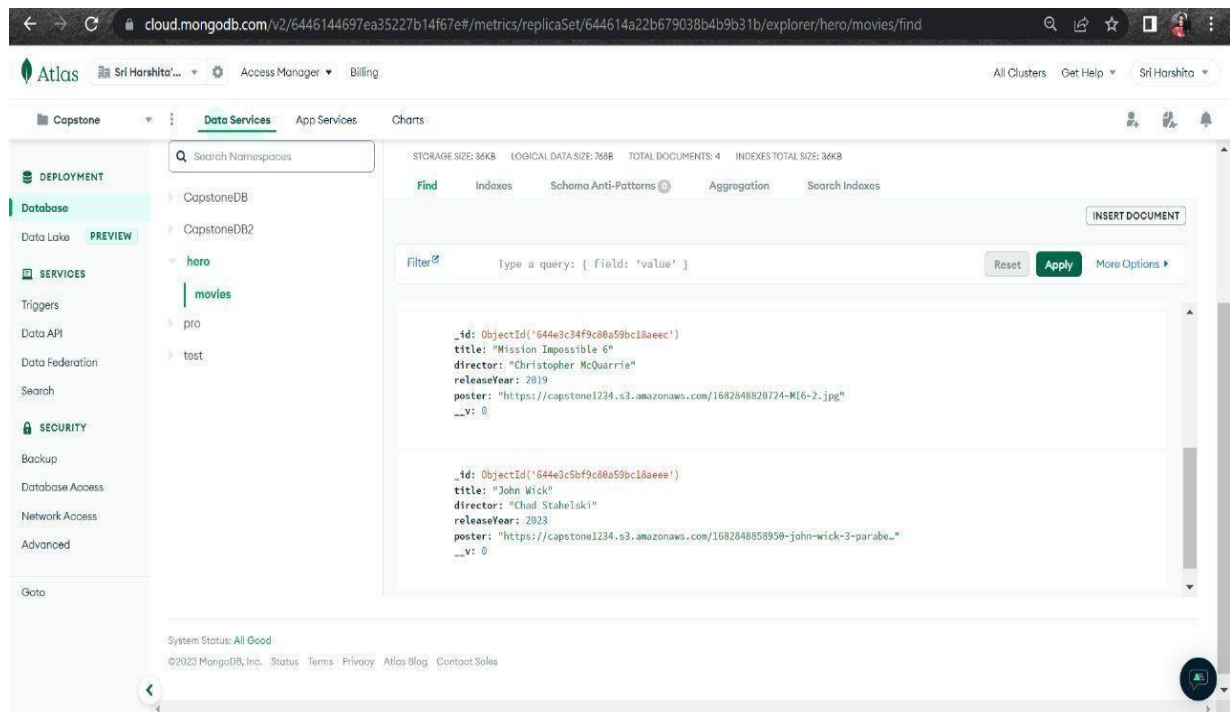
- Load balancer type: Network
- DNS name: her0lb-a70b963ec1e64f70.elb.us-east-1.amazonaws.com (A Record)
- Status: Active
- VPC: vpc-03b57040e0ba4d9ea
- IP address type: IPv4
- Scheme: Internet-facing
- Availability Zones: us-east-1b (use1-az1), us-east-1e (use1-az3), us-east-1f (use1-az5), us-east-1a (use1-az6), us-east-1c (use1-az2), us-east-1d (use1-az4)
- Hosted zone: Z26RNL4JYFTOTI

Below the details, there is a section for 'Listeners (1)' with a table showing the listener details:

Protocol:Port	Default action	ARN	Security policy	Default SSL cert	ALPN policy
TCP:80	Forward to target group her0tg	ARN	Not applicable	Not applicable	None

- Now copy the DNS of load-balancer and paste it in web-browser.



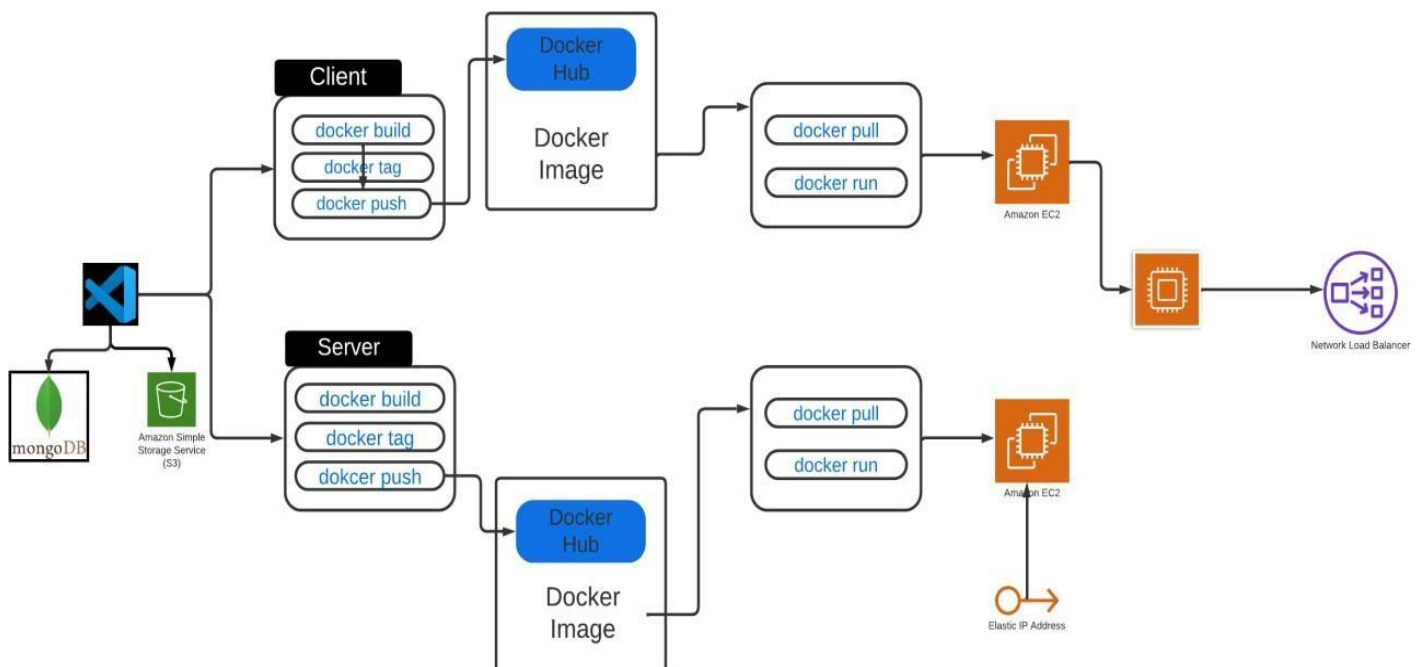


DNS NAME: herolb-a70b963ec1e64f70.elb.us-east-1.amazonaws.com

Project Git-Hub Repository link:

https://github.com/Harshitamattaparti/Capstone_project2

AWS DEPLOYMENT DIAGRAM:



CONTRIBUTATION OF TEAM MEMBERS:

Various tasks which contributed towards the growth of the project were done by our team mates according to their strengths and interests.

We made use of various resources like Teams and Whatsapp in order to communicate regularly to discuss about the daily updates of the project and also to resolve any of the issues faced by the fellow team mates.

To track the contributions of each team member, we created an Excel spreadsheet with "**Task Contributed**" and "**Members**" as columns. In the "Task Contributed" column, we listed each task that needed to be completed for the project, along with a brief description of the task. In the "Members" column, we listed the name of the team member who was responsible for completing that task.

We also made sure that we create and maintain a spreadsheet with columns "Tasks" and "Members" in order to keep a track of the progress as well as to know who is responsible for which task. In this way, we could track the contributions of every team member.

S No.	Process	MEMBERS
1	Connecting the Server with Atlas MongoDB.	Annapurna, Harshita
2	Creating IAM user and Configuring S3-bucket.	Ramalakshmi, Rajeswari
3	Configuring the Application Code with S3-multer.	Sridevi, Rajeswari
4	Containerization of the code using Dockerfile.	Rajeswari, Priyanka
5	Deploying the server in EC2 using Docker.	Harshita, Ramalakshmi, Annapurna
6	Deploying the client in EC2 using Docker.	Sridevi, Harshita
7	Creating a target group and a Load Balancer.	Sridevi, Priyanka
8	Creating an AWS Deployment Diagram.	Annapurna, Harshitha
9	Preparing Project Documentation.	Ramalakshmi, Priyanka