

# DEVOPS CAPSTONE PROJECT

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

Group-5  
Team members:

|                  |            |
|------------------|------------|
| 1) P. D S SIVA   | 21MH5A0351 |
| 2) B. SIVA DURGA | 21P35A0371 |
| 3) K. VIVEK      | 20A91A04L8 |
| 4) S. CHITTI     | 20A91A0449 |
| 5) K. SUVARNA    | 20P31A0484 |
| 6) A. BANGARRAJU | 20A91A0466 |
| 7) D. DEEPIKA    | 20A91A0476 |
| 8) N. ANKITHA    | 20A91A0496 |

| <b>CHAPTER</b>                                 | <b>PAGE NO</b>   |
|--|------------------|
| <b><i>AIM OF THE PROJECT</i></b>               | <b><i>1</i></b>  |
| <b><i>SCOPE OF THE PROJECT</i></b>             | <b><i>1</i></b>  |
| <b><i>TECHNOLOGIES USED</i></b>                | <b><i>2</i></b>  |
| <b><i>IMPLEMENTATAION and OBSERVATIONS</i></b> | <b><i>3</i></b>  |
| <b><i>AWS DEPLOYMENT DIAGRAM</i></b>           | <b><i>13</i></b> |
| <b><i>TEAM MEMBERS CONTRIBUTION</i></b>        | <b><i>15</i></b> |
| <b><i>CHALLENGES AND RESOURCES</i></b>         | <b><i>16</i></b> |
| <b><i>CONCLUSION</i></b>                       | <b><i>17</i></b> |

## AIM OF THE PROJECT

The main Aim of the project is to deploy a website on an Amazon EC2 instance using Docker container technology. EC2 is a web service provided by AWS to easily deploy and manage virtual servers. It is highly scalable and available to use. Docker is a containerization tool which makes an application run on any platform. Mongo dB to replace the local database so that the data is stored in our cluster and images are stored in S3.

We can host a web application on EC2 which is highly reliable, can handle vast web traffic and runs smoothly. This application can be easily maintained and updated because of docker with a load balancer attached to it.

## SCOPE OF THE PROJECT

The scope of the project is to deploy a website using Amazon Elastic Compute Cloud (EC2) and Docker container technology. The project includes developing a web application and modify it according to our requirements. Configuring and setting up an EC2 instance also plays a vital role in this project. Mongo dB is used instead of the local database.

The web application is containerized using docker for easier maintenance and deployment. It involves the deployment of web application from our EC2 instance and also attaching a load balancer to reduce the web traffic load on an instance.

## TECHNOLOGIES USED



**AWS:** AWS is a cloud computing platform that provides a range of services for individuals and businesses to store, manage, and process data and applications in the cloud. With over 200 services, including computing, storage, databases, analytics, machine learning, IoT, and security, AWS is flexible, cost-effective, and reliable. 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.



**Docker:** Docker contains containers which are created when the image is starts running. Containers are self-contained, portable units of software that include all the components and dependencies an application needs to run. 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.



**GIT:** Git is a version control system that allows developers to track changes to their code and collaborate with others on a project. It is a free and open-source tool that provides features such as branching, merging, and version history. Git is a powerful tool for managing software development projects.



**Web Development:** Web development is the process of creating and maintaining websites and web applications. It involves a combination of programming, design, and content creation to create web pages that are functional and visually appealing. Web development can include both frontend development, which focuses on the user interface, and backend development, which focuses on the server-side programming that powers the website.

## TECHNOLOGIES STACK:

**Frontend:** ReactJS

**Backend:** NodeJS

**Database:** MongoDB

## IMPLEMENTATION and OBSERVATIONS

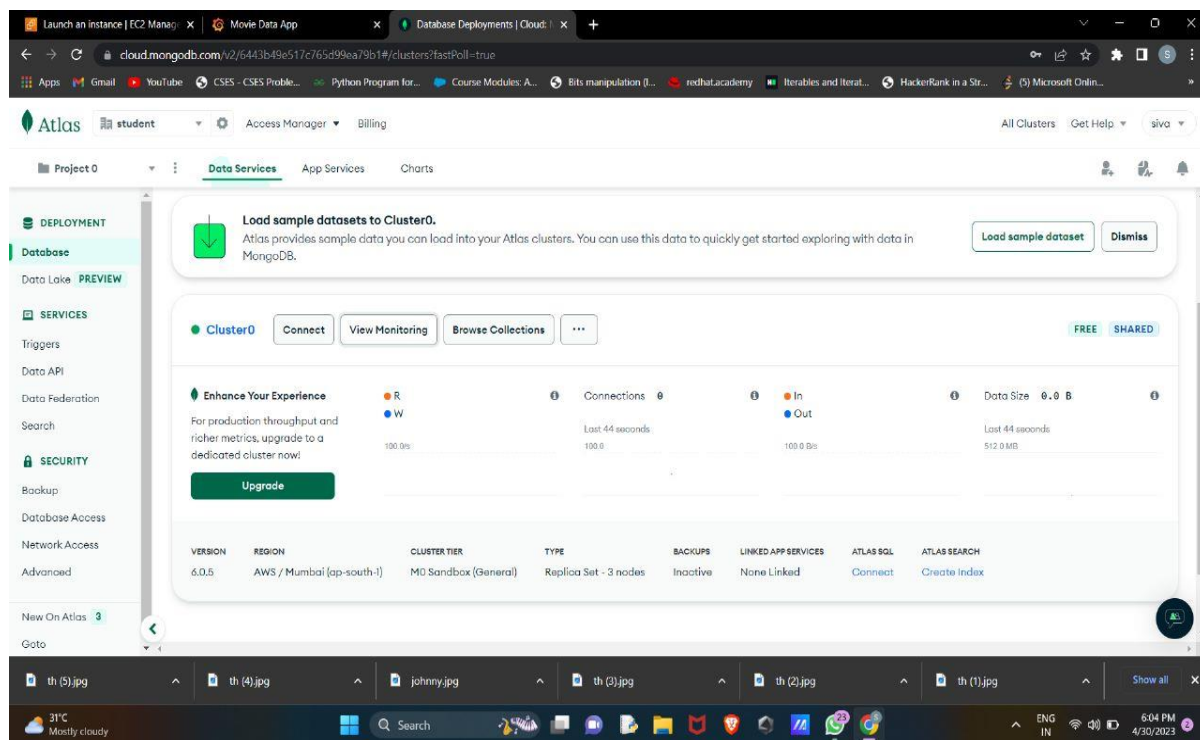
### Step 1: Connecting the Client with Atlas MongoDB

Initially, cloned the “Movie listing” website which uses ReactJS as frontend, NodeJS as backend and MongoDB as database from the repo:

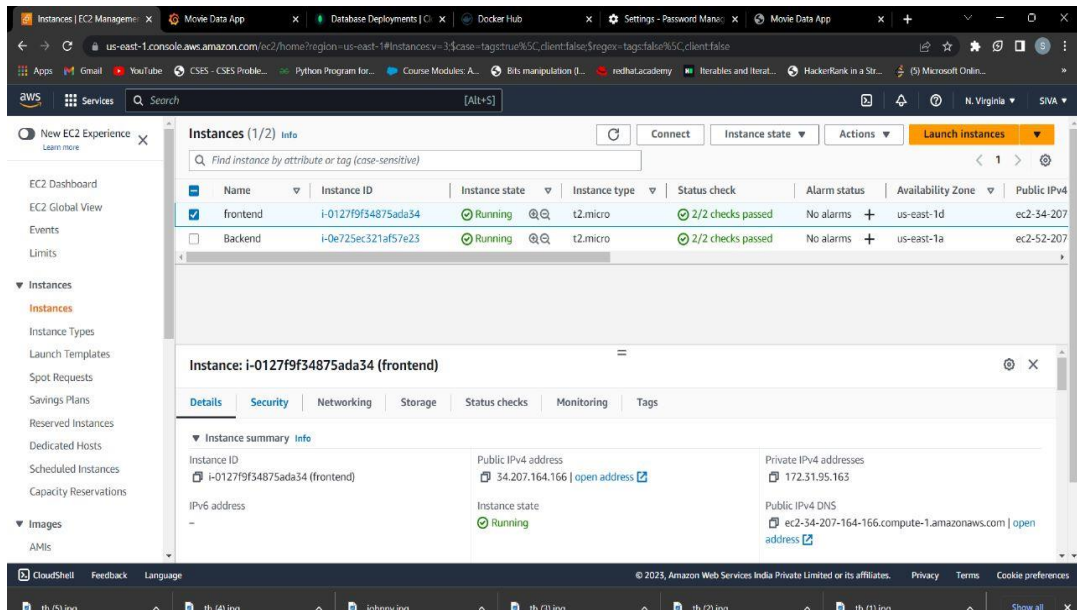
[https://github.com/snehal-heioviied/DEVOPS\\_CAPSTONE](https://github.com/snehal-heioviied/DEVOPS_CAPSTONE)

Here, we create MongoDB account and create a free tier cluster and given network access as allow access from anywhere and create the cluster.

- After creation of the cluster then connect it. Here we able to see the connection link.



## Creation of EC2 instance.



## Creating dockerfile and image in frontend(client)

```
ec2-user@ip-172-31-95-163:~/Capstone/client$ docker push siva800/client
Using default tag: latest
The push refers to repository [docker.io/siva800/client]
An image does not exist locally with the tag: siva800/client
ec2-user@ip-172-31-95-163:~/Capstone/client$ docker tag client:latest siva800/client
Error parsing reference: "latest:siva800/client" is not a valid repository/tag: invalid reference format
ec2-user@ip-172-31-95-163:~/Capstone/client$ docker tag client:latest siva800/client
Error parsing reference: "latest:siva800/client" is not a valid repository/tag: invalid reference format
ec2-user@ip-172-31-95-163:~/Capstone/client$ docker tag siva800/client
Error response from daemon: no such image: siva800/client
ec2-user@ip-172-31-95-163:~/Capstone/client$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
client               latest             d522b0c9ab75       6 minutes ago      322MB
node                 14-alpine          0da3dc2711a        4 weeks ago        119MB
ec2-user@ip-172-31-95-163:~/Capstone/client$ docker tag client:latest siva800/client
Using default tag: latest
The push refers to repository [docker.io/siva800/client]
68577fedc0da: Pushed
d522b0c9ab75: Pushed
ba3ce9d0f942: Pushed
03ac0e0f23ac: Pushed
03ac0e0f23ac: Pushed
319710dc178f: Mounted from library/node
d522b0c9ab75: Mounted from library/node
ba3ce9d0f942: Mounted from library/node
03ac0e0f23ac: Mounted from library/node
latest digest: sha256:25c2a0270ac092ac3172a2b5f643db3a51c29a231a0c3e8603bad099e size: 2206
ec2-user@ip-172-31-95-163:~/Capstone/client$ ls
Dockerfile  README.md  package-lock.json  package.json  public  src
ec2-user@ip-172-31-95-163:~/Capstone/client$ cat Dockerfile
# Base image
FROM node:14-alpine

# Set the working directory
WORKDIR /app

# Copy the package.json and package-lock.json files 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 for production
RUN npm run build

# Set the environment variable for the production build
ENV NODE_ENV=production

# Expose the port used by the application
EXPOSE 3000

# Start the application
CMD ["npm", "start"]
ec2-user@ip-172-31-95-163:~/Capstone/client$
```

```
ec2-user@ip-172-31-95-163:~/Capstone/client
---> 890539c26/49
Step 5/9 : COPY
---> 3d42fch7f551
Step 6/9 : RUN npm run build
---> Running in 251ed945ab9

> client@0.1.0 build /app
> react-scripts build

Creating an optimized production build...
Compiled with warnings.

[eslint]
src/App.js
  Line 39:13: 'res' is assigned a value but never used  no-unused-vars

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.

File sizes after gzip:
  64.43 kB  build/static/js/main.56a0fcd0.js
  782 B     build/static/css/main.1af6dd9b.css

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

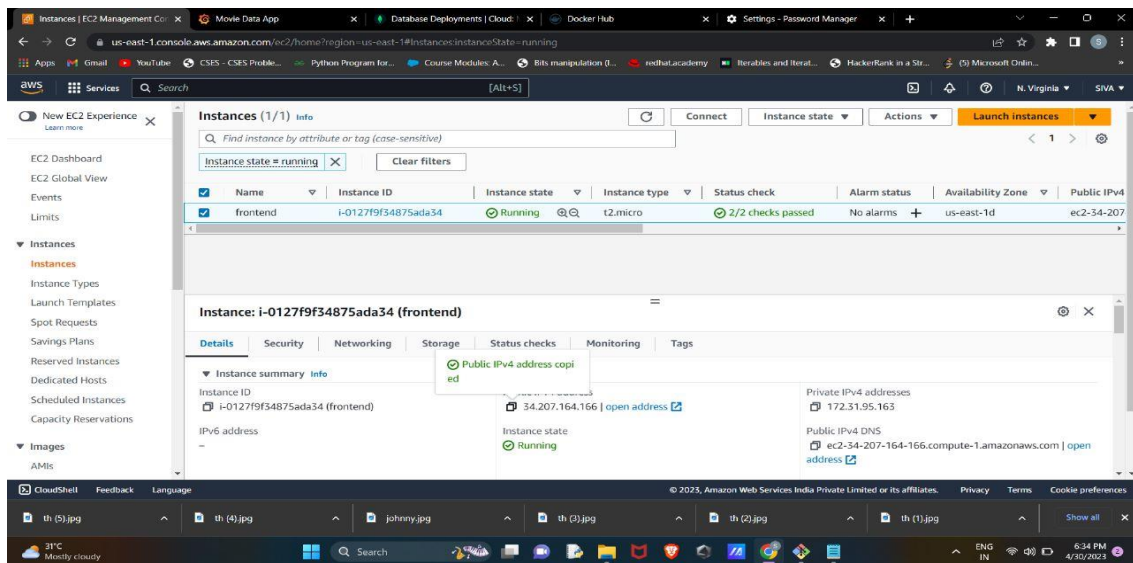
The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

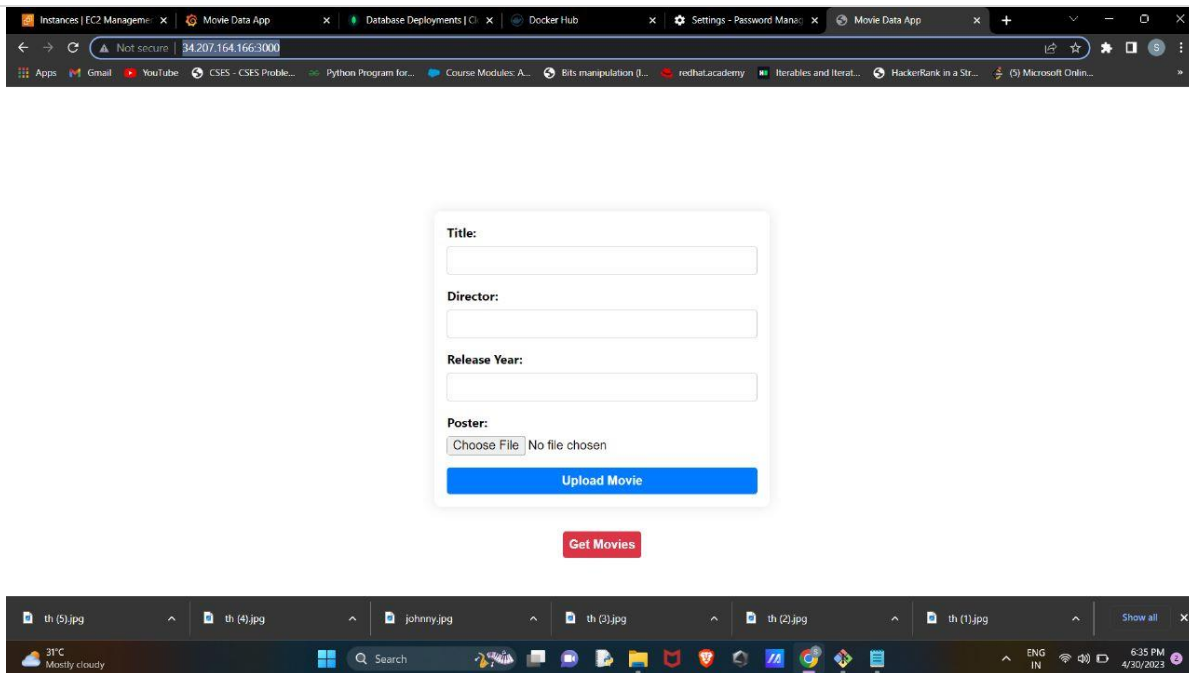
Find out more about deployment here:
https://cra.link/deployment

Removing intermediate container 251ed945ab9
---> eed9445c5708
Step 7/9 : ENV NODE_ENV=production
---> Running in 926c34ed0e
Removing intermediate container 926c34ed0e
---> 1d1abbae1ac
Step 8/9 : EXPOSE 3000
---> Running in 0ef6e4abc681
Removing intermediate container 0ef6e4abc681
---> c1d09a650703
Step 9/9 : CMD ["npm", "start"]
---> Running in cda513defd0e
Removing intermediate container cda513defd0e
---> 0242b0c8a815
Successfully tagged client:latest
[ec2-user@ip-172-31-95-163] client$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
client latest 0242b0c8a815 41 seconds ago 32.3MB
ec2-user@ip-172-31-95-163 client$
```

## Front end public Ip



Front end running successfully with 34.207.164.166:3000 PUBLIC IP.



MongoDB connection and installing mongoDB in Frontend.

```

e2-user@ip-172-31-25-146:~/Capstone/server
require('dotenv').config()
const mongoose = require('mongoose');
const cloudinary = require('cloudinary').v2;
const express = require('express');
const Movie = require('./Models/Movie');
const multer = require('multer');
const cors = require('cors');

mongoose.connect('mongodb+srv://5iva1:5iva123@cluster0.k41shx8.mongodb.net/?retryWrites=true&majority=1 ( useNewUrlParser: true, useUnifiedTopology: true )')
.then(() => console.log('Connected to MongoDB...'))
.catch(err => console.error('Could not connect to MongoDB...', err));

cloudinary.config({
  cloud_name: 'dec8gy2wy',
  api_key: '353514238263812',
  api_secret: 'fkxh4w0j9H1xc1qr3G16kzk-qn0'
});

const app = express();
const port = 5000;

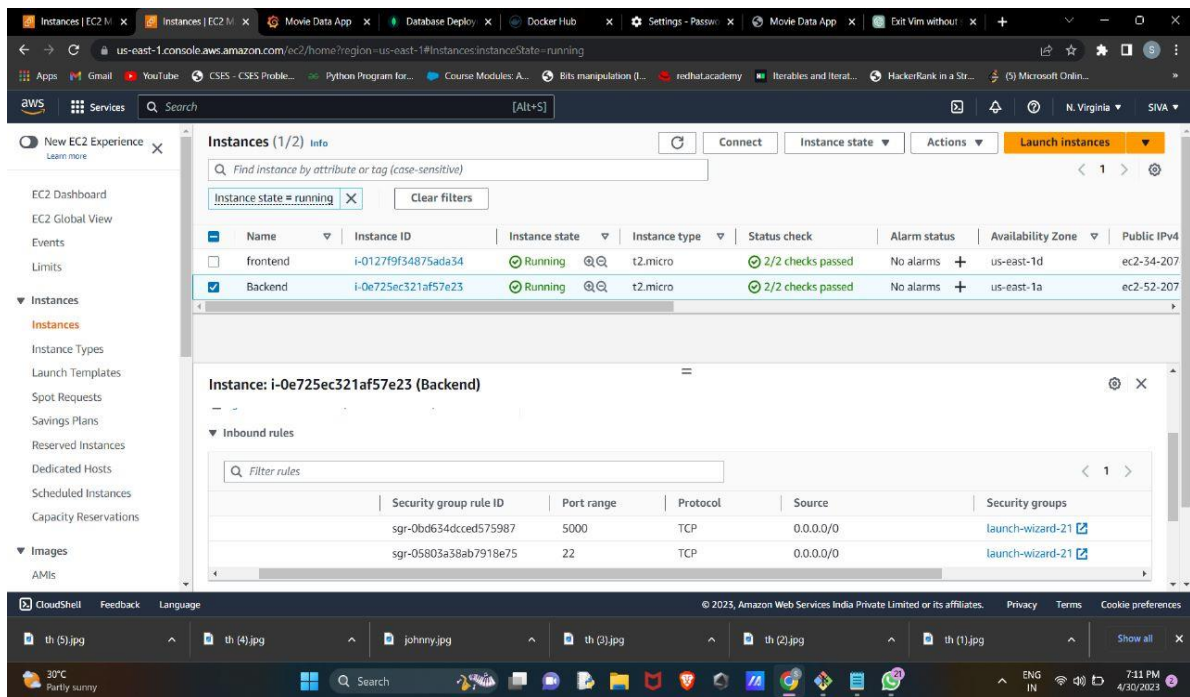
app.use(express.urlencoded({extended: true}));
app.use(express.json());
app.use(cors());

// multer configuration
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, './uploads')
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + '-' + file.originalname)
  }
});
const upload = multer({ storage });

// routes
app.post('/api/movies', upload.single('poster'), async (req, res) => {
  try {
    const result = await cloudinary.uploader.upload(req.file.path);
    const movie = new Movie({
      title: req.body.title,
      director: req.body.director,
      releaseYear: req.body.releaseYear,
      poster: result.secure_url
    });
    await movie.save();
    res.send(movie);
  } catch (error) {
    console.error(error);
    res.status(500).send('Something went wrong');
  }
});
INSERT --
  
```







```

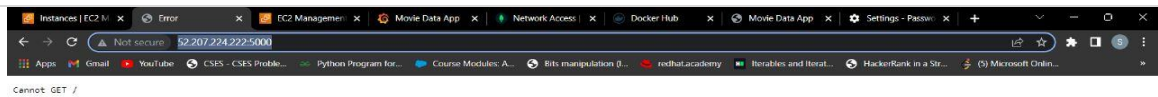
ec2-user@ip-172-31-25-146:~/Capstone/server
npm notice New major version of npm available! 8.19.2 -> 9.6.5
npm notice Changelog: https://github.com/npm/cli/releases/tag/v9.6.5
npm notice Run npm install -g npm@9.6.5 to update!
npm notice
[ec2-user@ip-172-31-25-146 server]$ client_loop: send disconnect: Connection reset by peer

$ ssh -i "NEW_PAIR.pem" ec2-user@ec2-52-207-224-222.compute-1.amazonaws.com
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

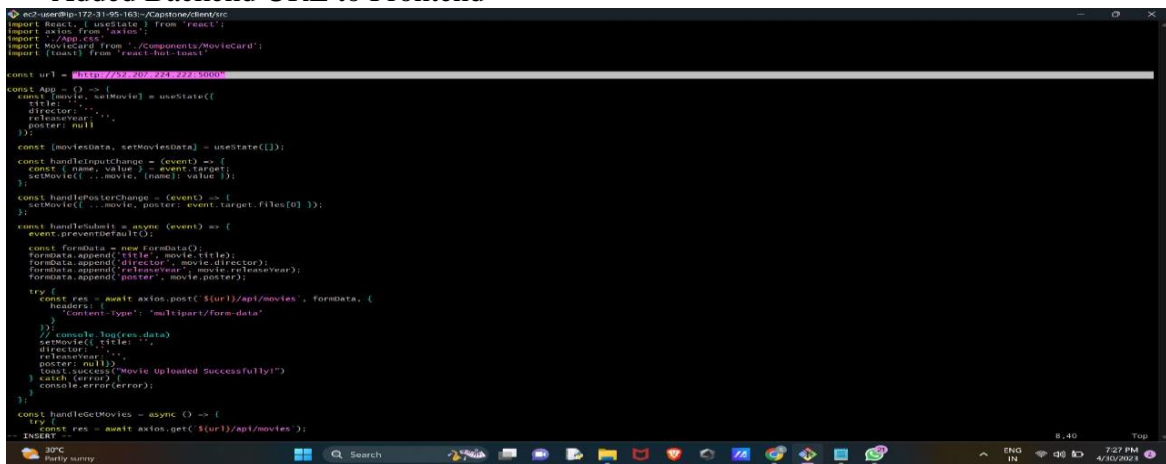
Last login: Sun Apr 30 13:17:59 2023 from 157.48.95.227
[ec2-user@ip-172-31-25-146 ~]$ ls
Capstone
[ec2-user@ip-172-31-25-146 ~]$ cd Capstone/
[ec2-user@ip-172-31-25-146 Capstone]$ ls
README.md client server
[ec2-user@ip-172-31-25-146 Capstone]$ cd server
[ec2-user@ip-172-31-25-146 server]$ ls
models index.js node_modules package-lock.json package.json uploads
[ec2-user@ip-172-31-25-146 server]$ node index.js
server listening at http://localhost:5000
Could not connect to MongoDB... MongooseServerSelectionError: Could not connect to any servers in your MongoDB Atlas cluster. One common reason is that you're trying to access the database from an IP that isn't whitelisted. Make sure your current IP address is on your Atlas cluster's IP whitelist: https://www.mongodb.com/docs/atlas/security-whitelist/
at _handleConnectionErrors (/home/ec2-user/Capstone/server/node_modules/mongoose/lib/connection.js:755:11)
at NativeConnection.openUri (/home/ec2-user/Capstone/server/node_modules/mongoose/lib/connection.js:730:11)
reason: TopologyDescription {
  type: 'ReplicaSetNoPrimary',
  servers: Map(3) {
    'ac-08udm5-shard-00-01.k41shx8.mongodb.net:27017' => [ServerDescription],
    'ac-08udm5-shard-00-02.k41shx8.mongodb.net:27017' => [ServerDescription],
    'ac-08udm5-shard-00-00.k41shx8.mongodb.net:27017' => [ServerDescription]
  },
  stale: false,
  compatible: true,
  heartbeatFrequencyMS: 10000,
  localThresholds: 15,
  setName: 'atlas-6v5b2d-shard-0',
  maxSetVersion: null,
  maxElectionId: null,
  commonWireVersion: 0,
  logicalSessionTimeoutMinutes: null
},
code: undefined
}
[ec2-user@ip-172-31-25-146 server]$ node index.js
server listening at http://localhost:5000
Connected to MongoDB...

```

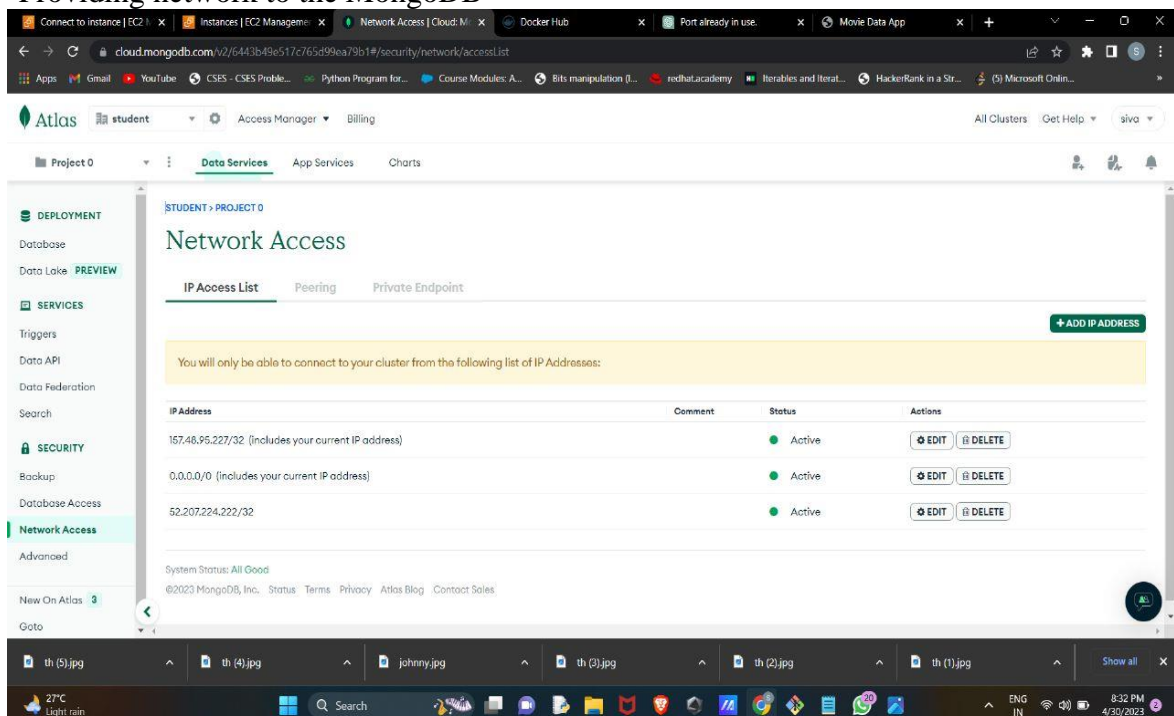
Backend accessible IP 52.207.224.222:5000



## Added Backend URL to Frontend



## Providing network to the MongoDB



Movie Uploaded Successfully!

Title:

Director:

Release Year:

Poster:

Choose File bh2.jpg

Upload Movie

Get Movies



Poster:

Choose File bh2.jpg

Upload Movie

Get Movies



Title: iron man 3

Director: Jon Favreau

Release Year: 2013



Title: Bahubali 2

Director: Rajamouli

Release Year: 2017

## Step 3: Creating S3 Bucket and uploading objects.

The screenshot shows the AWS S3 console interface. The left sidebar contains navigation options like Buckets, Access Points, and Storage Lens. The main content area displays the 'sam231' bucket. Under the 'Objects (0)' section, it states 'No objects' and 'You don't have any objects in this bucket.' There is an 'Upload' button. The top navigation bar shows 'Amazon S3 > Buckets > sam231'.

The screenshot shows the AWS IAM console 'Retrieve access keys' page. A green banner at the top says 'Access key created'. The page shows the 'Access key' table with one entry: Access key 'AKIA3BEITVNB63ILCKUR' and Secret access key '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'. Below the table, there are 'Access key best practices' and a 'Download .csv file' button.

us-east-1.console.aws.amazon.com/s3/bucket/sam231/property/policy/edit?region=us-east-1

Services iam

Amazon S3

Successfully edited bucket policy.

Bucket ARN: arn:aws:s3:::sam231

Policy

```
1 {
2   "Version": "2012-10-17",
3   "Id": "Policy1682860130899",
4   "Statement": [
5     {
6       "Sid": "Stmt1682860127551",
7       "Effect": "Allow",
8       "Principal": {
9         "AWS": "arn:aws:iam::758348229443:user/samps3"
10      },
11      "Action": "s3:*",
12      "Resource": "arn:aws:s3:::sam231"
13    }
14  ]
15 }
```

Edit statement Stmt1682860127551 Remove

1. Add actions

Choose a service

Filter services

Included

S3

Available

AMP

API Gateway

API Gateway V2

ASC

Access Analyzer

Account

Activate

Alexa for Business

Amplify

2. Add a resource Add

CloudShell Feedback Language

© 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connType=standard&iinstanceId=i-06c42ff60cd0aa4ef&osUser=ec2-user&sshPort=22#

Services Search [Alt+S]

N. Virginia gmail.com

```
AWS_ACCESS_KEY_ID=AKIA3BEITVNBVRVUMQYK5
AWS_SECRET_ACCESS_KEY=lqBEiivp721rGBApEKDiyamQn0AypPwPhTVuPxGI
AWS_REGION=us-east-1
AWS_S3_BUCKET_NAME=samp231
```

-- INSERT --

4,27 All

i-06c42ff60cd0aa4ef (backend ec2)

PublicIPs: 3.230.135.235 PrivateIPs: 172.30.5.247

CloudShell Feedback Language

© 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

```
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connType=standard&instanceId=i-06c42ff60cd0aa4ef&osUser=ec2-user&sshPort=22#/  
[ec2-user@ip-172-30-5-247 server]$ node index.js  
Server listening at http://localhost:5000  
Connected to MongoDB...  
^C  
[ec2-user@ip-172-30-5-247 server]$ ls  
Models index.js node_modules package-lock.json package.json uploads  
[ec2-user@ip-172-30-5-247 server]$ vim index.js  
[ec2-user@ip-172-30-5-247 server]$ vim .env  
[ec2-user@ip-172-30-5-247 server]$ vim .env  
[ec2-user@ip-172-30-5-247 server]$ npm install  
  
up to date, audited 189 packages in 1s  
  
13 packages are looking for funding  
  run 'npm fund' for details  
  
found 0 vulnerabilities  
[ec2-user@ip-172-30-5-247 server]$ vim .env  
[ec2-user@ip-172-30-5-247 server]$ npm install aws-sdk  
npm WARN deprecated querystring@0.2.0: The querystring API is considered Legacy. new code should use the URLSearchParams API instead.  
  
added 24 packages, and audited 213 packages in 5s  
  
22 packages are looking for funding  
  run 'npm fund' for details  
  
found 0 vulnerabilities  
[ec2-user@ip-172-30-5-247 server]$
```

i-06c42ff60cd0aa4ef (backend ec2)  
PublicIPs: 3.230.135.235 PrivateIPs: 172.30.5.247

```
CloudShell Feedback Language © 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences  
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connType=standard&instanceId=i-06c42ff60cd0aa4ef&osUser=ec2-user&sshPort=22#/  
[ec2-user@ip-172-30-5-247 server]$ npm WARN deprecated querystring@0.2.0: The querystring API is considered Legacy. new code should use the URLSearchParams API instead.  
  
added 24 packages, and audited 213 packages in 5s  
  
22 packages are looking for funding  
  run 'npm fund' for details  
  
found 0 vulnerabilities  
[ec2-user@ip-172-30-5-247 server]$ npm install multer multer s3  
npm WARN deprecated natives@1.1.6: This module relies on Node.js's internals and will break at some point. Do not use it, and update to graceful-fs@4.x.  
  
added 13 packages, and audited 226 packages in 3s  
  
22 packages are looking for funding  
  run 'npm fund' for details  
  
4 vulnerabilities (2 moderate, 2 high)  
  
To address all issues, run:  
  npm audit fix  
  
Run 'npm audit' for details.  
[ec2-user@ip-172-30-5-247 server]$ node index.js  
Server listening at http://localhost:5000  
(node:4421) NOTE: We are formalizing our plans to enter AWS SDK for JavaScript (v2) into maintenance mode in 2023.  
  
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...
```

i-06c42ff60cd0aa4ef (backend ec2)  
PublicIPs: 3.230.135.235 PrivateIPs: 172.30.5.247



PrzmeReReCoWrtloWCoShDeMiBaAdHcAEDeNeDcnoimx

s3.console.aws.amazon.com/s3/buckets/sam231?region=us-east-1&tab=objects

awsServicesSearch[Alt+S]

Globalgmail.com

Amazon S3 > Buckets > sam231

Info

ObjectsPropertiesPermissionsMetricsManagementAccess Points



Objects (2)

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

RefreshCopy S3 URICopy URLDownloadOpenDeleteActionsCreate folderUpload

Find objects by prefix

< 1 > ⚙

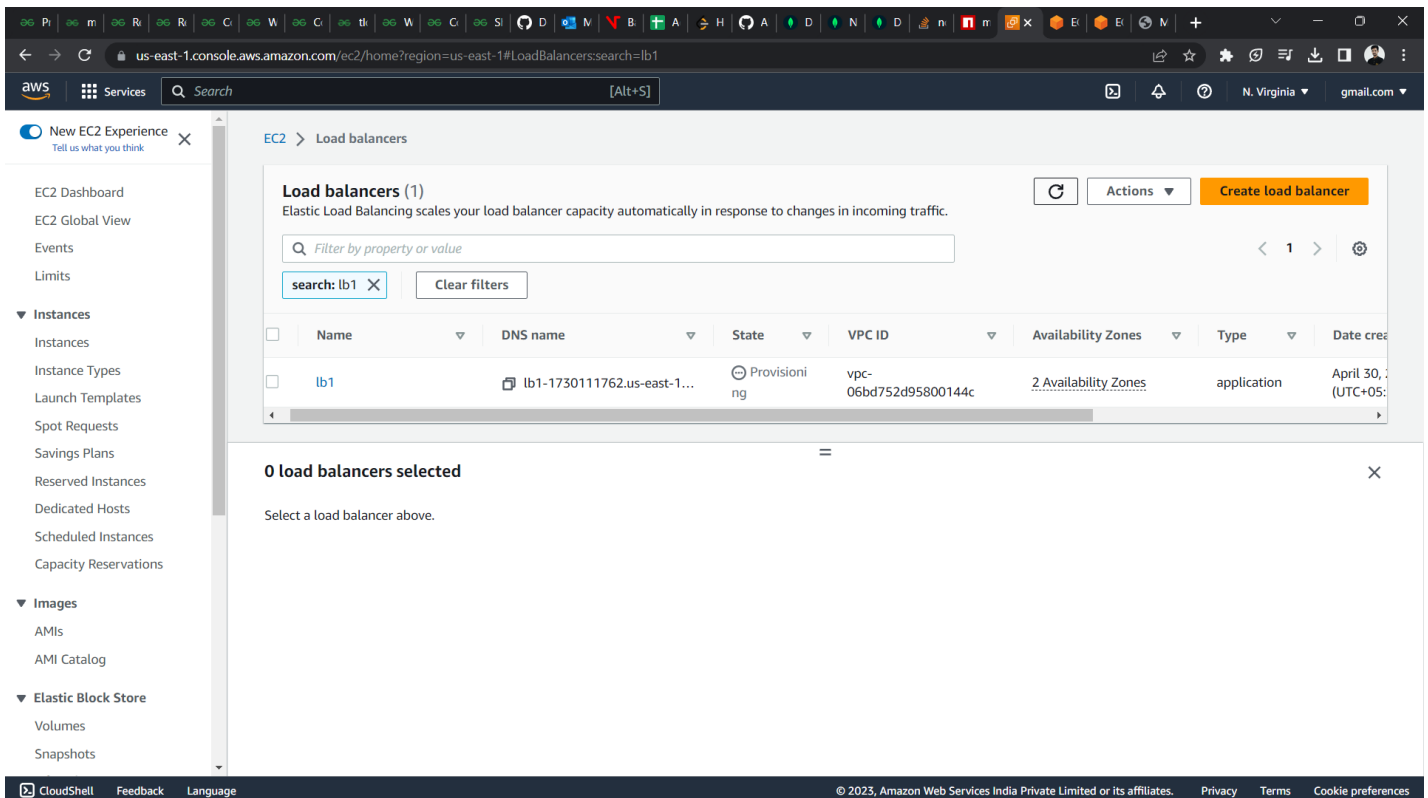
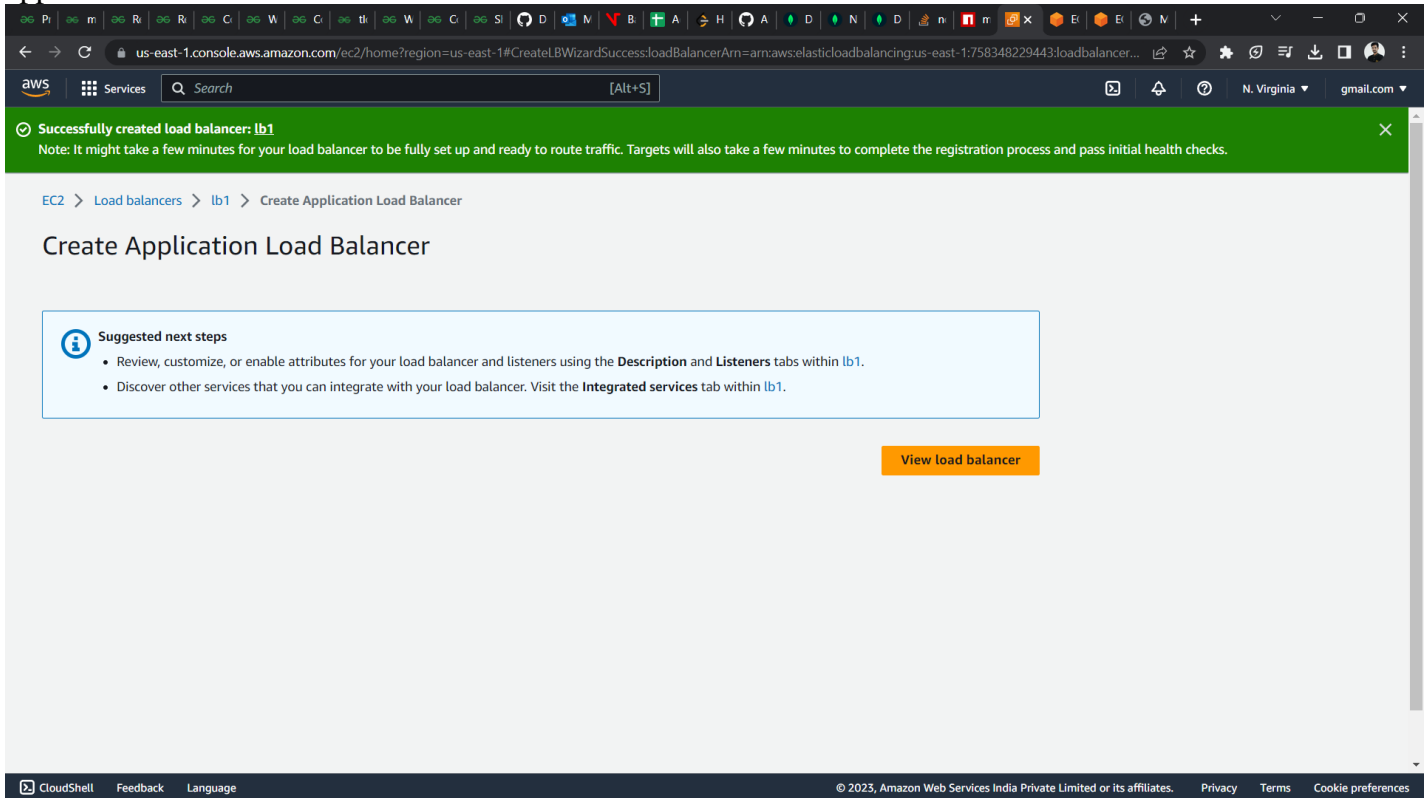
| <input type="checkbox"/> | Name  | Type | Last modified                        | Size     | Storage class |
|--------------------------|---|------|--------------------------------------|----------|---------------|
| <input type="checkbox"/> |  <a href="#">1652423637278.jpg</a> | jpg  | April 30, 2023, 19:39:57 (UTC+05:30) | 218.6 KB | Standard      |
| <input type="checkbox"/> |  <a href="#">1652835462423.jpg</a> | jpg  | April 30, 2023, 19:39:58 (UTC+05:30) | 188.8 KB | Standard      |

CloudShellFeedbackLanguage© 2023, Amazon Web Services India Private Limited or its affiliates. PrivacyTermsCookie preferences



## Step 4: Attaching load balancer

Create the load balancer which is used to increase capacity (concurrent users) and reliability of applications.



us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:

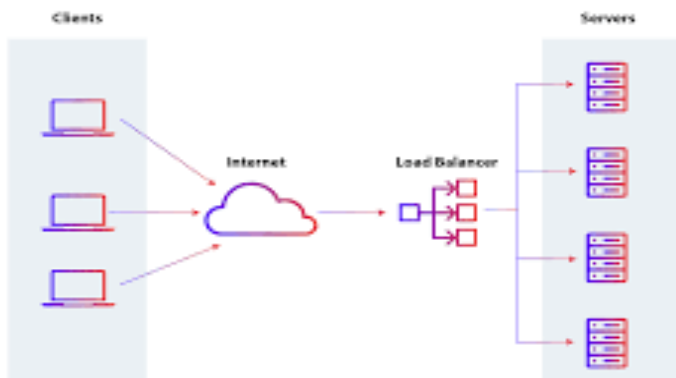
Instances (4) Info

Find instance by attribute or tag (case-sensitive)

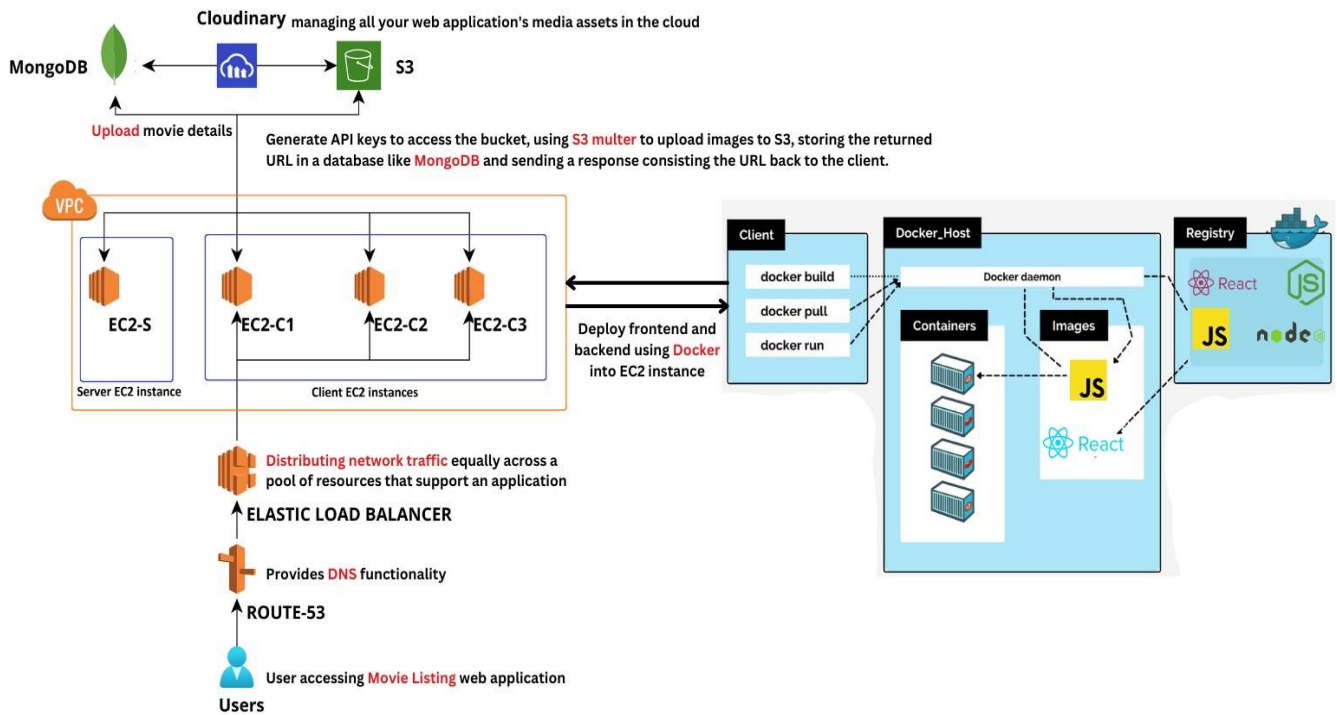
|                          | Name         | Instance ID         | Instance state | Instance type | Status check      | Alarm status | Availability Zone | Public IPv4 DNS    |
|--------------------------|--------------|---------------------|----------------|---------------|-------------------|--------------|-------------------|--------------------|
| <input type="checkbox"/> | backend ec2  | i-06c42ff60cd0aa4ef | Running        | t2.micro      | 2/2 checks passed | No alarms    | us-east-1f        | ec2-3-230-135-235  |
| <input type="checkbox"/> | frontend ec2 | i-0c1179a5eba1e1bba | Running        | t2.micro      | 2/2 checks passed | No alarms    | us-east-1f        | ec2-34-239-166-14  |
| <input type="checkbox"/> | ba1          | i-0b7bf4af06b5ab69e | Running        | t2.micro      | Initializing      | No alarms    | us-east-1f        | ec2-3-235-188-244  |
| <input type="checkbox"/> | fr1          | i-047ea6503bffd8675 | Running        | t2.micro      | Initializing      | No alarms    | us-east-1f        | ec2-44-210-19-86.c |

Select an instance

CloudShell Feedback Language © 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences



## AWS DEPLOYMENT DIAGRAM:



This project includes the utilization of various tools. Multiple observations are noted based on the tools. Some key observations are:

**Visual Studio Code:** Visual Studio Code is a source-code editor that can be used with a variety of programming languages. The code has been according to the requirements using VS Code.

**MongoDB Atlas:** MongoDB Atlas is a fully-managed cloud database that handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS, Azure, and GCP). The local database has been replaced by this database and the data uploaded is stored here.

**EC2:** 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. The containerized frontend and backend have been deployed using EC2. The client instances are also attached to the load balancer to manage the traffic.

**S3:** Amazon S3 (Simple Storage Service) is a cloud-based storage service that enables users to store and retrieve data from anywhere on the internet. The images we have uploaded in the frontend have been stored in S3 buckets which was created.

**IAM:** AWS 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. A user has been created for the access controls and full administrator access was given.

**Route 53:** Amazon Route 53 is a highly available and scalable cloud-based Domain Name System (DNS) service provided by Amazon Web Services (AWS). It helps the developers and business owners by translating the domain name into IP address. DNS has been assigned which can be accessed by anyone to view the website.

**GitHub:** GitHub is a web-based platform that provides developers with a variety of tools for software development and collaboration. The project has been uploaded in GitHub which would be accessible for future changes.

A project requires clear objectives and ground rules to work on to achieve the goal of any aspect.

Jira has been used to plan the sprints, track the changes and complete the project efficiently.

Slack a great communication tool where we can interact efficiently and to collaborate.

Zoom is a collaboration app that enabled conversations.

## TASKS ASSIGNED FOR TEAM MEMBERS:

During the project, we distributed the workload among team members based on their individual strengths and interests.

We communicated regularly and updates on our project management tool through WhatsApp group.

In the "Tasks Assigned" column, we listed each task that needed to be completed for the project, along with a brief description of the task. In the "Team Members" column, we listed the name of the team member who was responsible for completing that task.

By using this spreadsheet, we were able to keep track of who was responsible for each task and ensure that everyone was contributing equally to the project. We updated the spreadsheet regularly to reflect any changes to the task assignments, and we used it as a reference during our team meetings to discuss progress and identify any issues.

| S No. | TASKS ASSIGNED                                   | TEAM MEMBERS           |
|-------|--|------------------------|
| 1     | Connecting the Server with Atlas MongoDB.        | Siva durga, Siva       |
| 2     | Creating IAM user and Configuring S3-bucket.     | Vivek, Chitti, Suvarna |
| 3     | Configuring the Application Code with S3-multer. | BangarRaju, Siva durga |
| 4     | Containerization of the code using Dockerfile.   | Siva, Ankitha          |
| 5     | Deploying server on EC2 using Docker.            | Suvarna, Siva durga    |
| 6     | Deploying client on EC2 using Docker.            | Ankitha, Deepika       |
| 7     | Creation a target group and Load Balancer.       | Siva, Vivek, Chitti    |
| 8     | Creating AWS Deployment Diagram.                 | Deepika, Bangaraju     |
| 9     | Preparing Project Documentation.                 | Chitti , Vivek         |

Overall, we believe that our team was successful in distributing the workload effectively and working together to achieve our goals. By outlining our roles and responsibilities, maintaining open communication, and evaluating our performance, we were able to complete the project on time and to a high standard.

## CHALLENGES FACED

- ☐ Due to lack of node of react knowledge, it took lot of time to analyses the code.
- ☐ We face most of the issues while modifying the server code for configuring with S3-multer.
- ☐ Due to lack of good internet, Docker image creation took a lot of time.
- ☐ When something went wrong, we had to reframe the frontend (connection) and dockarize the frontend code again.
- ☐ Due to our lack of knowledge on how to create an AWS architecture diagram, we had to learn how to work with different editing tools like Canva and figma...etc.

## RESOURCES

**MongoDB** : <https://www.mongodb.com/docs/>

**AWS** : <https://docs.aws.amazon.com/>

**S3-multer** : <https://www.npmjs.com/package/multer-s3>

**Docker** : <https://docs.docker.com/>

**Git** : <https://git-scm.com/docs>

**AWS Architecture Diagram Developing tools:** <https://aws.amazon.com/architecture/icons/>

## CONCLUSION

At the end of this project, we successfully deployed frontend using docker into EC2 instance. The given capstone project involves using ReactJS as Frontend, NodeJS as Backend, MongoDB as Database, multer in Backend for file upload and cloud infrastructure (AWS) to deploy. The deployed frontend application gives us the required “Movie Listing” web site in which users can upload movie details.

For achieving this, we used multer-s3 library for storing images, replaced local database with Atlas MongoDB cloud infrastructure to take the database into the cloud, deployed backend and frontend in EC2 instances using Docker and attached Elastic IP to this instance and modified the Frontend Code to be able to fetch data from Backend.

The frontend user interface of the application having 4 fields like **Title, Director, Release year** and **Poster**. And having 2 functions **upload Movie** and **Get Movies**. When the user enters the details of the movie the data will be stored in mongoDB database. The poster field having images these images are stored in S3 bucket using S3-multer and by using clouinary which provides a URL of that image which is stored in mongoDB database. 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.