

DevOps Capstone Project

DEPLOYING A MOVIE LISTING WEBSITE USING AWS CLOUD

Team no: 5

Team Members:

- 1. B.B.V.S.S. Sushma – 20A91A0511**
- 2. Reshma – 20A91A0515**
- 3. N. Aakash - 20A91A0544**
- 4. P. Vaishnavi - 20A91A0552**
- 5. T. Rosy – 20A91A0558**
- 6. T. Kavya Sri – 20A91A0560**
- 7. Y. Pushpa Moulika – 20A91A0564**
- 8. Ch. Prudhvi - 20A91A0576**

ABOUT THE PROJECT:

The Movie Listing website is a web application that allows users to upload and view movie details. The website uses ReactJS as the frontend, NodeJS as the backend, and MongoDB as the database. The website allows users to upload movie details, including images. Initially, the images are stored in local storage, but as part of the deployment process on AWS, the images are moved to an S3 bucket for better scalability.

The deployment process involves several steps, including replacing local storage with S3, migrating the database to MongoDB Atlas, deploying the backend in an EC2 instance using Docker, modifying the frontend code to fetch data from the backend, deploying the frontend using Docker into an EC2 instance, creating a load balancer, and using DNS to point to the IP. The use of Docker ensures that the deployment process is consistent across different environments and reduces the chances of deployment-related issues.

The deployment on AWS ensures that the website is scalable and can handle increasing traffic. The use of an EC2 instance and load balancer ensures that traffic is distributed among multiple instances, and the use of MongoDB Atlas ensures that the database is scalable and highly available. The use of S3 for storing images ensures that the website can handle large amounts of data without any issues. The deployment on AWS also allows for easy monitoring and management of the website.

TOOLS AND TECHNOLOGIES USED IN THIS PROJECT:

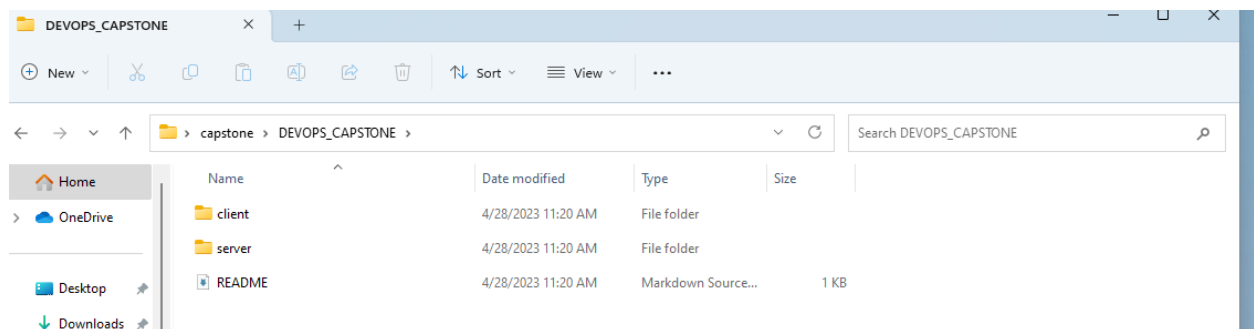




IMPLEMENTATION

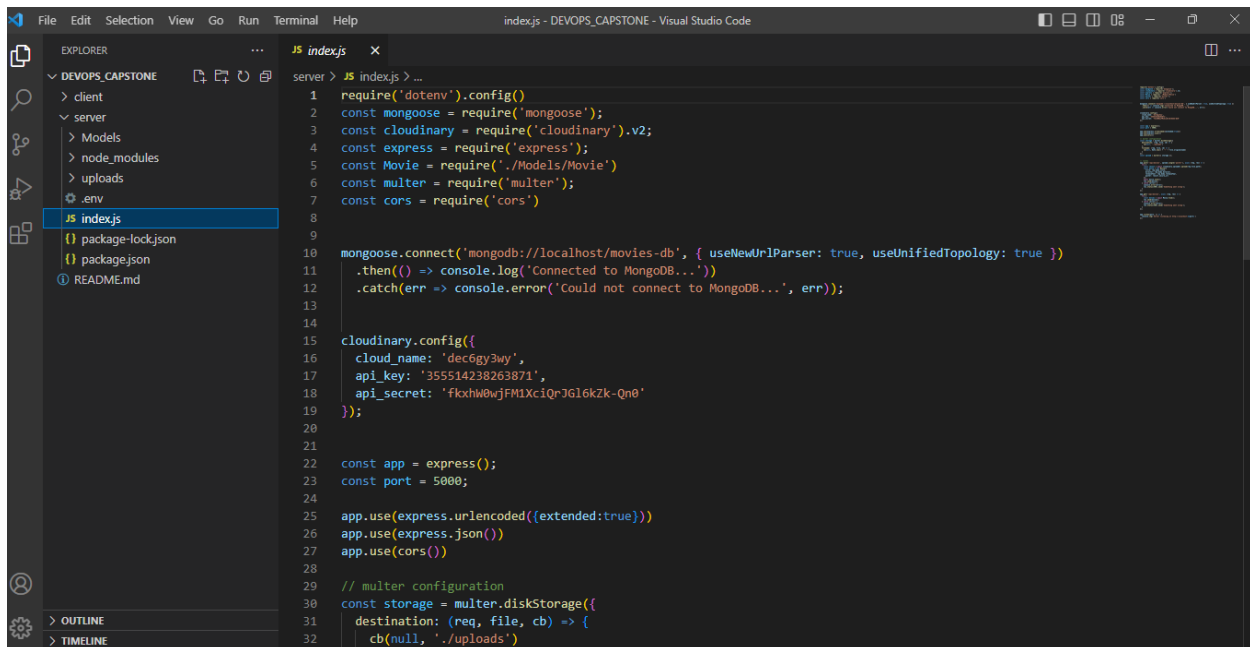
STEP 1: clone the GitHub repository provided by initiating a new repository by using **git init** and clone it by using **git clone <link>**.

```
MINGW64/c:/Users/DELL/Desktop/capstone
DELL@DESKTOP-OPHE310 MINGW64 ~/Desktop/capstone
$ git init
Initialized empty Git repository in C:/Users/DELL/Desktop/capstone/.git/
DELL@DESKTOP-OPHE310 MINGW64 ~/Desktop/capstone (master)
$ git clone https://github.com/snehal-herovired/DEVOPS_CAPSTONE
Cloning into 'DEVOPS_CAPSTONE'...
remote: Enumerating objects: 7436, done.
remote: Counting objects: 100% (99/99), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 7436 (delta 72), reused 61 (delta 61), pack-reused 7337
Receiving objects: 100% (7436/7436), 9.00 MiB | 93.00 KiB/s, done.
Resolving deltas: 100% (841/841), done.
Updating files: 100% (6965/6965), done.
DELL@DESKTOP-OPHE310 MINGW64 ~/Desktop/capstone (master)
$ |
```



STEP 2: we must replace local database with Atlas MongoDB cloud infrastructure to take the database into the cloud. For that, we create Atlas MongoDB account and create a new free-tier cluster.

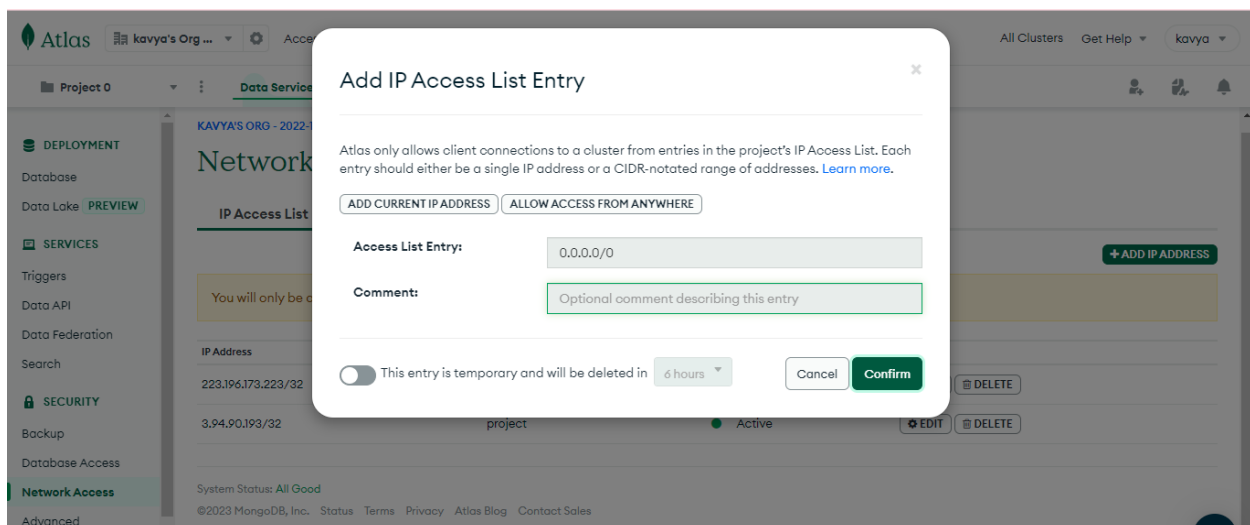
Before changing the code

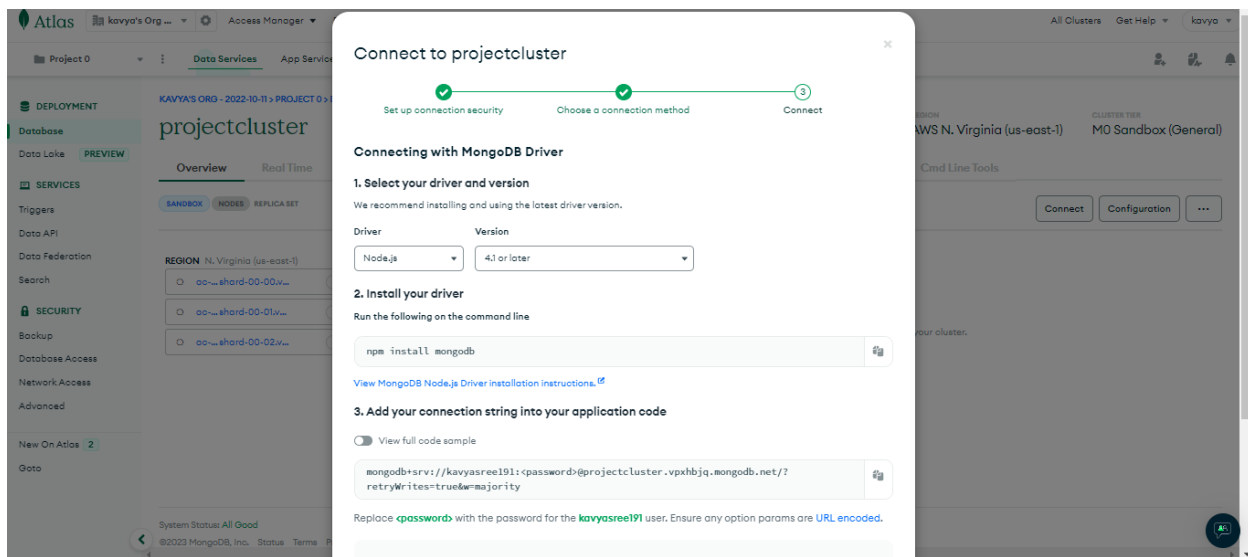


```
1 require('dotenv').config()
2 const mongoose = require('mongoose');
3 const cloudinary = require('cloudinary').v2;
4 const express = require('express');
5 const Movie = require('./Models/Movie');
6 const multer = require('multer');
7 const cors = require('cors')
8
9
10 mongoose.connect('mongodb://localhost:27020/movies-db', { useNewUrlParser: true, useUnifiedTopology: true })
11 .then(() => console.log('Connected to MongoDB...'))
12 .catch(err => console.error('Could not connect to MongoDB...', err));
13
14
15 cloudinary.config({
16   cloud_name: 'dec6gy3my',
17   api_key: '355514238263871',
18   api_secret: 'fkxhW0wjFM1XciQrJG16kZk-Qn0'
19 });
20
21
22 const app = express();
23 const port = 5000;
24
25 app.use(express.urlencoded({extended:true}))
26 app.use(express.json())
27 app.use(cors())
28
29 // multer configuration
30 const storage = multer.diskStorage({
31   destination: (req, file, cb) => {
32     cb(null, './uploads')
```

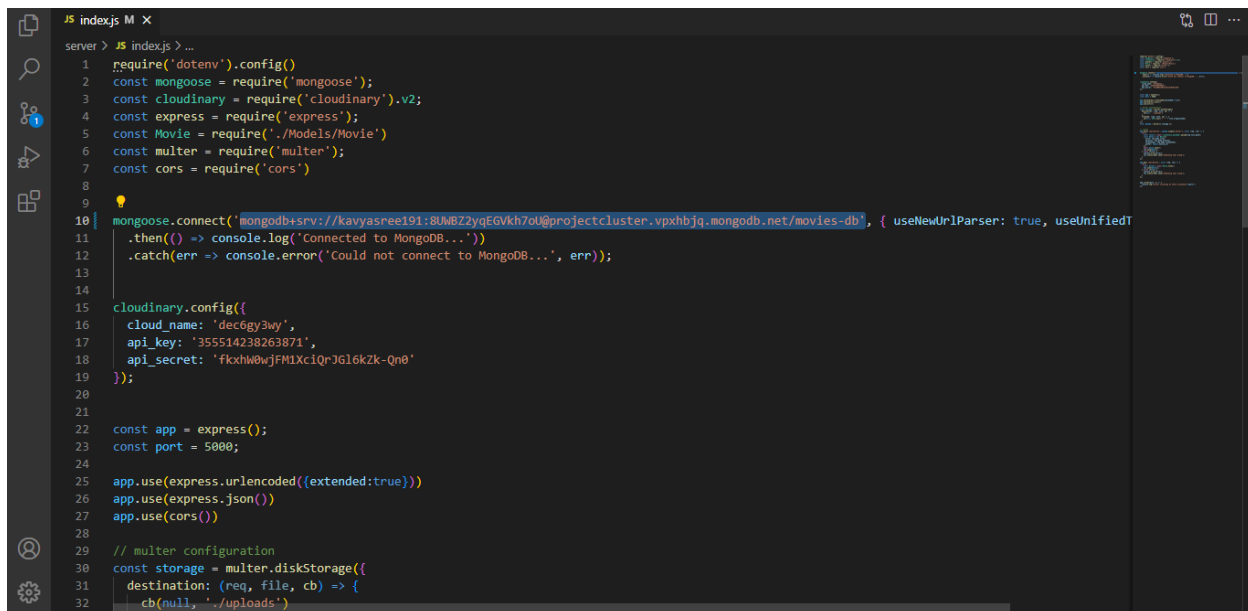
Give the network access (instance IP) to allow access and connect it by selecting your driver and version. Copy the connection string provided.

In order to connect the server with MongoDB, replace the localhost link with MongoDB connection string.

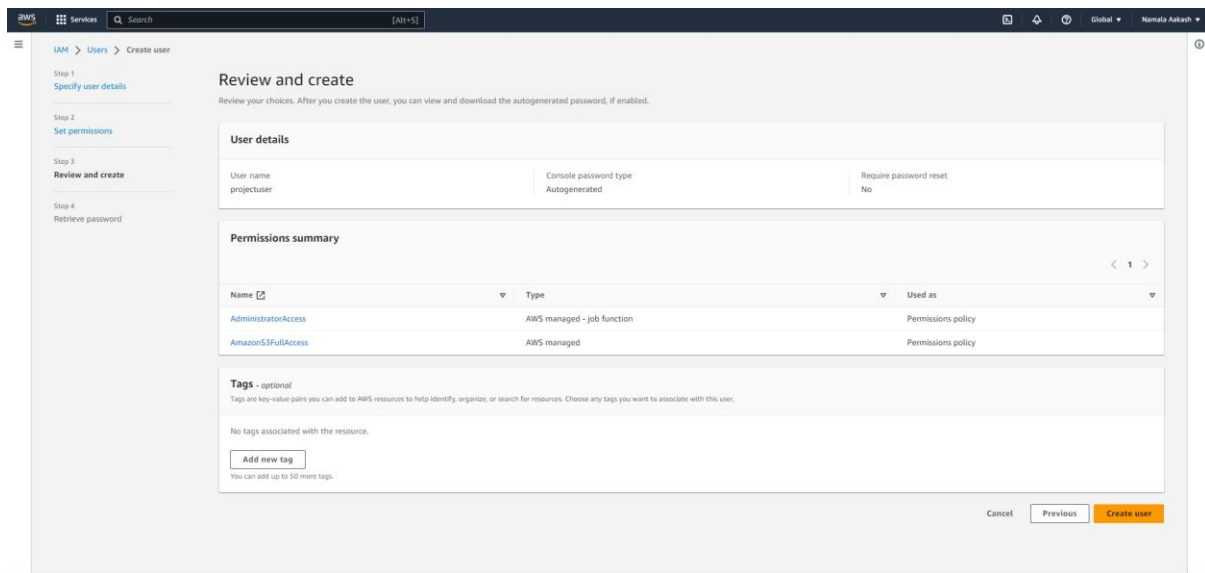




`mongodb+srv://kavyasree191:<password>@projectcluster.vpxhbjq.mongodb.net/?retryWrites=true&w=majority`



STEP 3: create a S3 bucket to store the images uploaded by the user. Create the bucket by giving necessary policy.



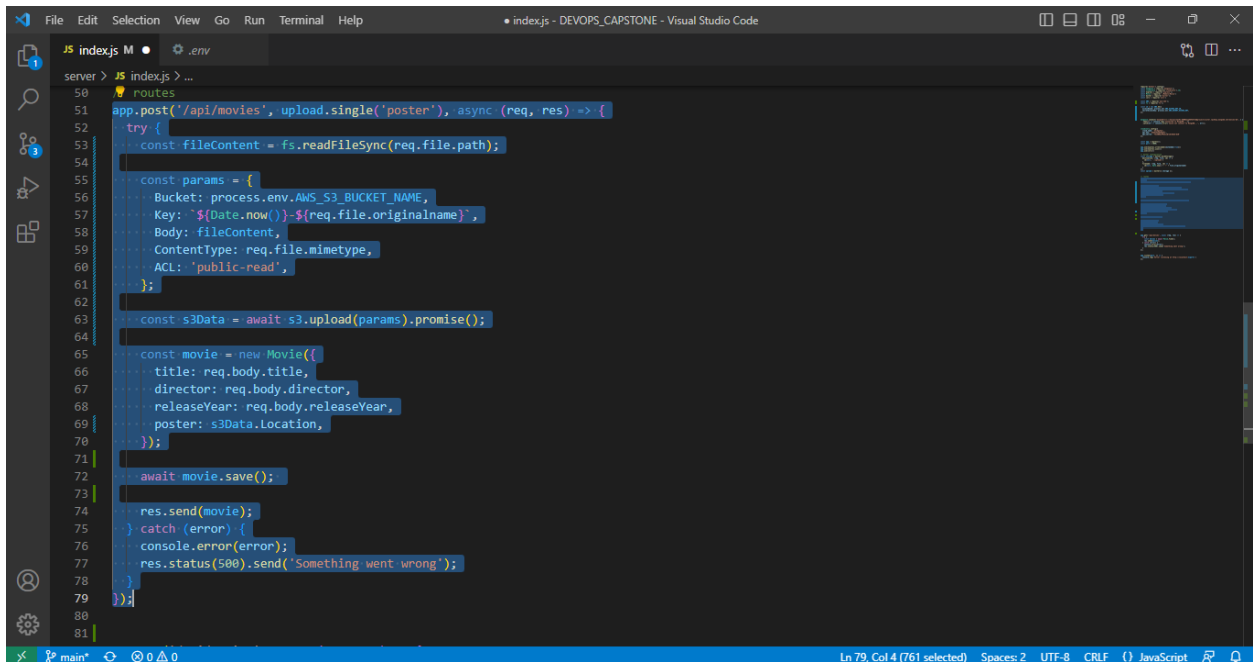
STEP 5: now we have to change the code for further implementation. Replace the existing multer code with multer-s3 with AWS access key, s3 bucket name.

Multer-S3 is a middleware that handles multi-part form data such as images in NodeJS to upload images to AWS s3 instead of local device storage.

```

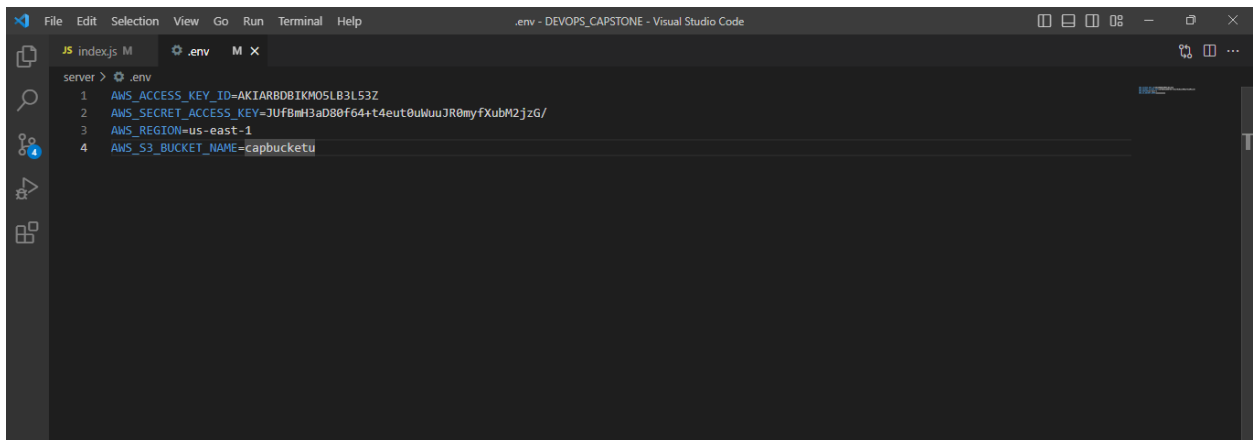
1  require('dotenv').config()
2  const mongoose = require('mongoose');
3  const cloudinary = require('cloudinary').v2;
4  const express = require('express');
5  const Movie = require('./Models/Movie')
6  const multer = require('multer');
7  const cors = require('cors')
8
9  const AWS = require('aws-sdk');
10 const fs = require('fs');
11
12 const s3 = new AWS.S3({
13   accessKeyId: process.env.AWS_ACCESS_KEY_ID,
14   secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
15 });
16
17
18
19 mongoose.connect('mongodb+srv://kavyasree191:8UMBZ2yqEGVkh7oU@projectcluster.vpxhbjq.mongodb.net/movies-db', { useNewUrlParser: true, useUnifiedTopology: true })
20   .then(() => console.log('Connected to MongoDB...'))
21   .catch(err => console.error('Could not connect to MongoDB...', err));
22
23
24 cloudinary.config({
25   cloud_name: 'dec6y3wy',
26   api_key: '355514238263871',
27   api_secret: 'fkxhW0wjFMIXciQrJG16kZk-Qn0'
28 });
29
30
31 const app = express();
32 const port = 5000;

```

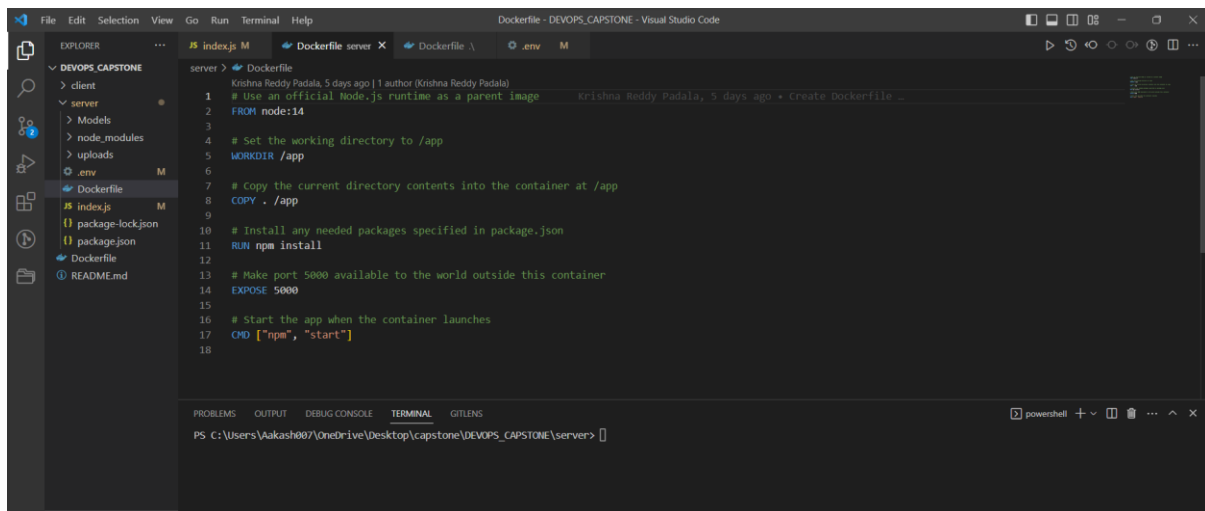
```
server > JS indexjs > ...
50 routes
51 app.post('/api/movies', upload.single('poster'), async (req, res) => {
52   try {
53     const fileContent = fs.readFileSync(req.file.path);
54
55     const params = {
56       Bucket: process.env.AWS_S3_BUCKET_NAME,
57       Key: `${Date.now()}-${req.file.originalname}`,
58       Body: fileContent,
59       ContentType: req.file.mimetype,
60       ACL: 'public-read',
61     };
62
63     const s3Data = await s3.upload(params).promise();
64
65     const movie = new Movie({
66       title: req.body.title,
67       director: req.body.director,
68       releaseYear: req.body.releaseYear,
69       poster: s3Data.Location,
70     });
71
72     await movie.save();
73
74     res.send(movie);
75   } catch (error) {
76     console.error(error);
77     res.status(500).send('Something went wrong');
78   }
79 })
80
81
```

To perform this, multer requires AWS user access information such as access key, bucket name, region. Create an .env file and upload the keys in it.

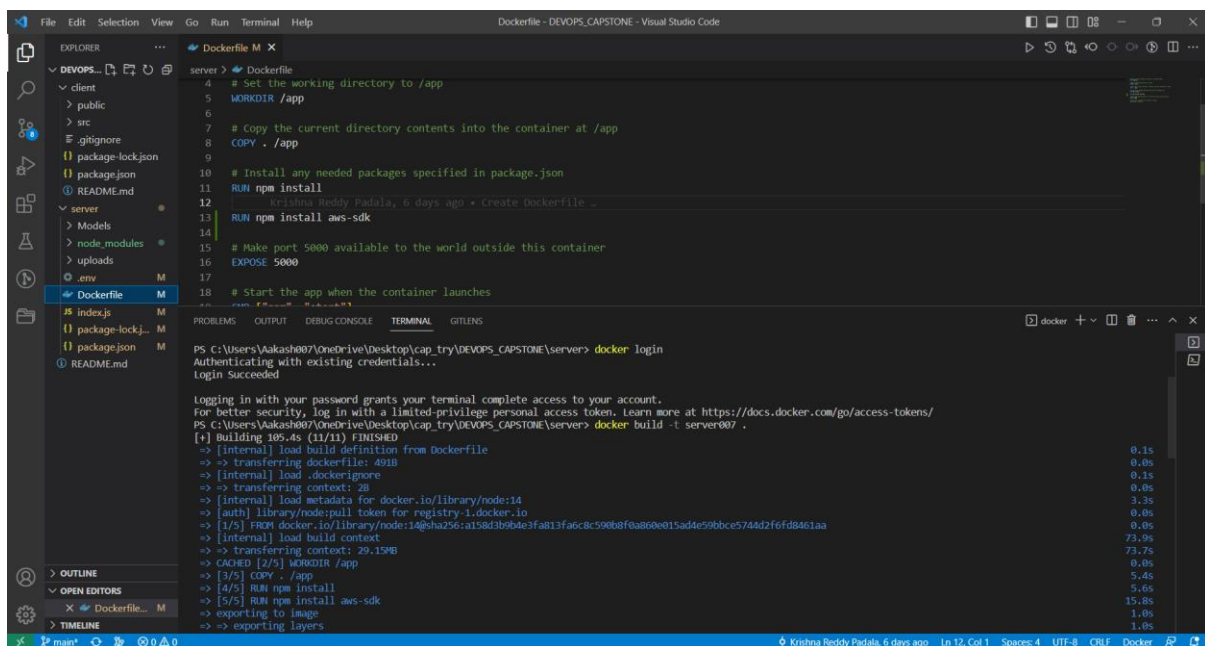


```
server > .env
1 AWS_ACCESS_KEY_ID=AKIARBD8IKM05LB3L53Z
2 AWS_SECRET_ACCESS_KEY=JUF8mH3aD80f64+t4eut0uWuuJR0myfXubM2jzG/
3 AWS_REGION=us-east-1
4 AWS_S3_BUCKET_NAME=capbucketu
```

STEP 6: create a docker file in server to create and push docker image to the Docker Hub.



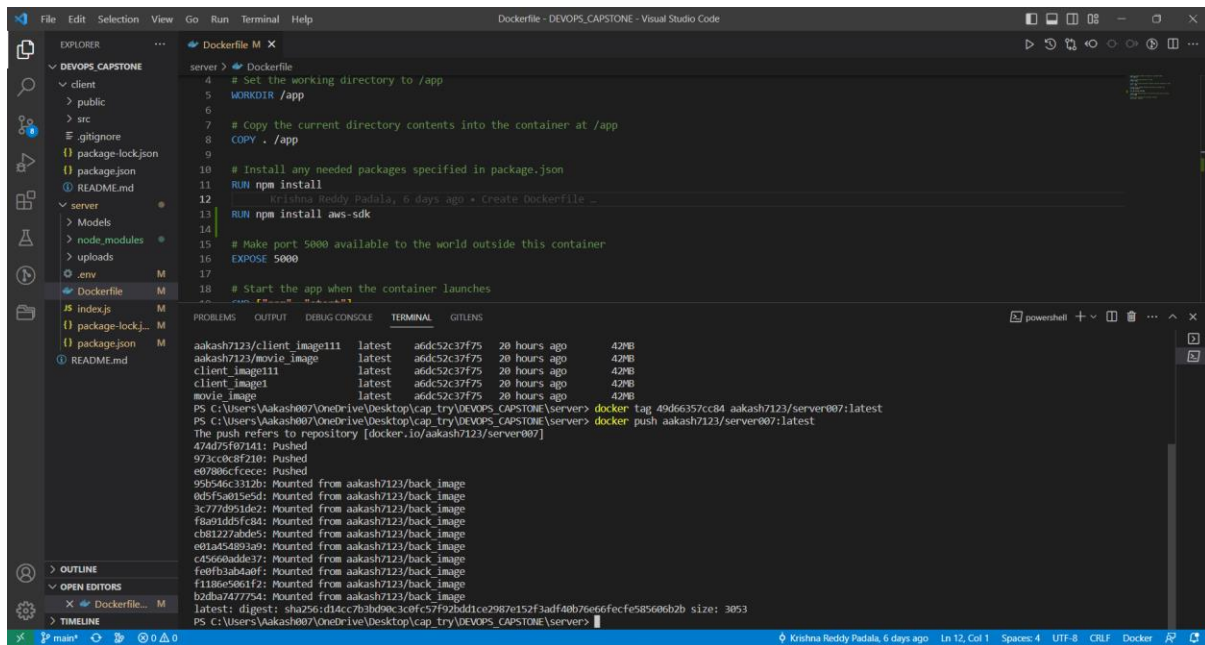
Now, we have to create a docker image to execute the code in the container. To build an image use command **docker build -t <image name>** .



Use **docker images** to view the existing images.

We use **docker tag** to maintain the build version to push the image to the docker hub

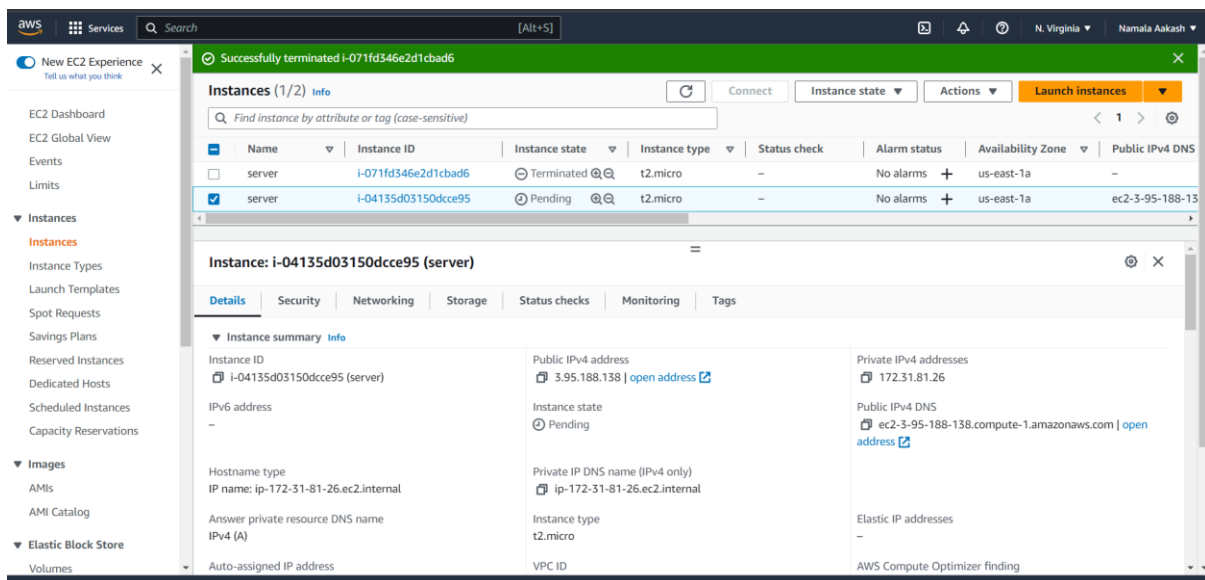
Docker tag <image_id> <username>/<image_name> and push the image to the docker hub by using command **docker push <username>/<image_name>**

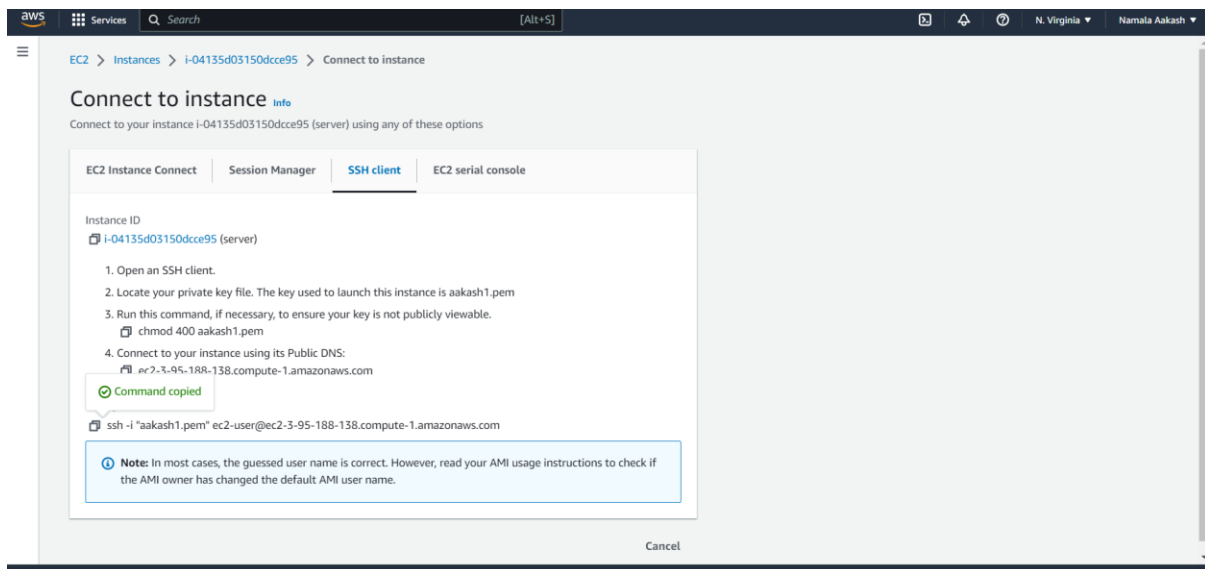


You can view the images in the Docker Hub.

Step 7: Deploying the server in the EC2 instance.

Create an EC2 instance and install Docker to pull images and start the NodeJS application which is connected to the created MongoDB database.





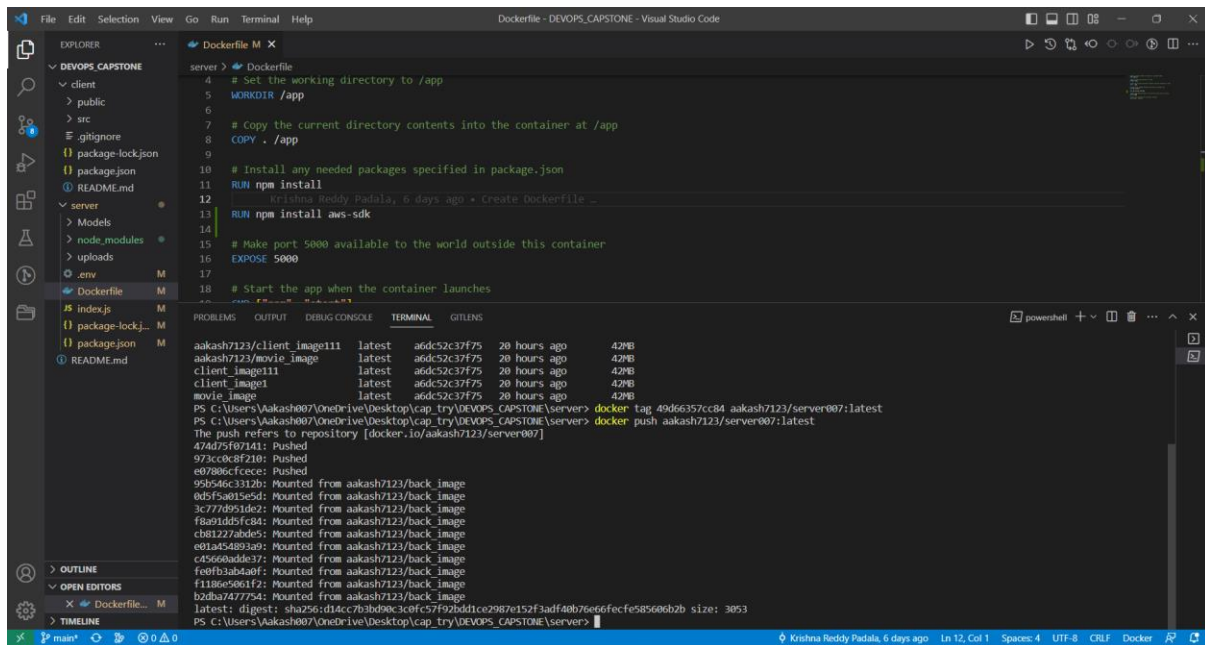
Install and start the docker

```
[ec2-user@ip-172-31-81-26 ~]$
--|  --|  )
--| (  /  Amazon Linux 2 AMI
---\  ---\
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-81-26 ~]$ sudo su
[root@ip-172-31-81-26 ec2-user]# yum update
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
No packages marked for update
[root@ip-172-31-81-26 ec2-user]# yum install docker

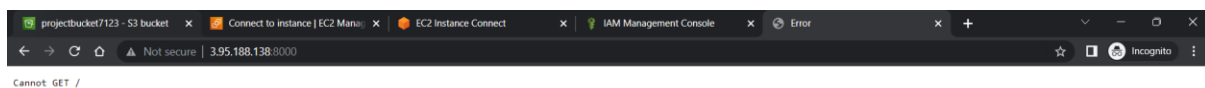
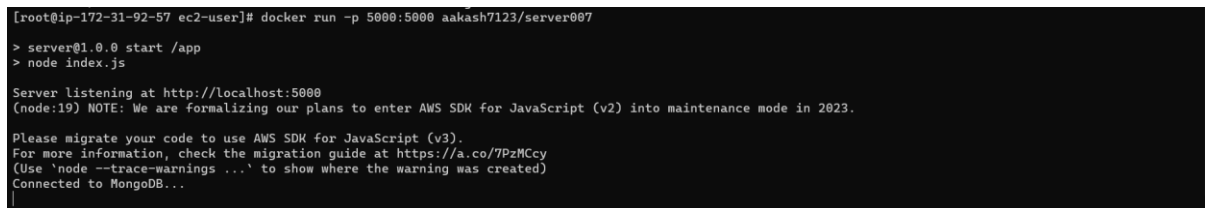
Complete!
[root@ip-172-31-81-26 ec2-user]# systemctl start docker
[root@ip-172-31-81-26 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: active (running) since Sat 2023-04-29 14:46:38 UTC; 7s ago
     Docs: https://docs.docker.com
   Process: 3463 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
   Process: 3462 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
   Main PID: 3466 (dockerd)
     Tasks: 7
    Memory: 20.7M
   CGroup: /system.slice/docker.service
           └─3466 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536

Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.278012551Z" level=info msg="ClientConn switching balancer to...le=grpc
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.320249232Z" level=warning msg="Your kernel does not support ...weight"
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.320672857Z" level=warning msg="Your kernel does not support ...device"
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.321139174Z" level=info msg="Loading containers: start."
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.531901068Z" level=info msg="Default bridge (docker0) is assi...ddress"
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.582762052Z" level=info msg="Loading containers: done."
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.598482031Z" level=info msg="Docker daemon" commit=6051f14 gr...0.10.23
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.598948850Z" level=info msg="Daemon has completed initialization"
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal systemd[1]: Started Docker Application Container Engine.
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.622214739Z" level=info msg="API listen on /run/docker.sock"
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-172-31-81-26 ec2-user]#
```

Login to the Docker Hub by specifying the username and password. Pull the image created by using **docker pull <username>/<image_name>**



Now, run the image with port 5000 using **docker run -p 5000:5000**
<username>/<image_name>



Step 8: modifying frontend code.

To fetch data from backend, update the code by replacing the url with the backend's port to parse the components from the backend.

The screenshot shows a Visual Studio Code editor with a project named 'DEVOPS_CAPSTONE'. The Explorer sidebar on the left shows the file structure: 'client' (containing 'node_modules', 'public', and 'src'), 'App.css', 'index.css', 'App.js', 'index.js', '.gitignore', 'Dockerfile', 'package-lock.json', 'package.json', 'README.md', and 'server'. The main editor displays the 'index.js' file, which contains a React application using 'useState' and 'useEffect' hooks. The application has a 'url' state and a 'handleInputChange' function. The terminal at the bottom shows the command 'docker build -t client .' being executed, with the output indicating the build is finished and the image is being pushed to Docker Hub.

```
client > src > JS App.js > 100 url
You, 24 minutes ago [2 authors (Mobasshir Hasan and others)]
1 import React, { useState } from 'react';
2 import axios from 'axios';
3 import './App.css';
4 import MovieCard from './Components/MovieCard';
5 import {toast} from 'react-hot-toast'
6
7
8 const url = "http://3.84.89.25:5000"
9
10 const App = () => {
11   const [movie, setMovie] = useState({
12     title: '',
13     director: '',
14     releaseYear: '',
15     poster: null
16   });
17
18   const [moviesData, setMoviesData] = useState([]);
19
20   const handleInputChange = (event) => {
21
22   }
23 }
24
25 export default App;
```

```
PS C:\Users\Aakash007\OneDrive\Desktop\cap_try\DEVOPS_CAPSTONE\client> docker build -t client .
[+] Building 137.6s (15/15) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/nginx:alpine
=> [internal] load metadata for docker.io/library/node:14-alpine
=> [auth] library/node:pull token for registry-1.docker.io
=> [auth] library/nginx:pull token for registry-1.docker.io
=> CACHED [stage-1 1/2] FROM docker.io/library/nginx:alpine@sha256:dd2a9179765849767b10e2adde7e10c4ad6b7e4d4840eeb77ec93f880cd2db27
=> [internal] load build context
=> => transferring context: 1.40MB
```

Create a docker file in the client to run the commands and build a docker image.

The screenshot shows the 'Dockerfile' file in the 'client' directory of the 'DEVOPS_CAPSTONE' project. The Dockerfile contains instructions to build a Docker image for the application. It starts with 'WORKDIR /', followed by copying 'package.json' and 'package-lock.json' to the container, installing dependencies with 'npm install', copying the rest of the application code, building the application with 'npm run build', and finally serving the application with a lightweight HTTP server using 'nginx'.

```
client > Dockerfile
1 WORKDIR /
2
3 # Copy package.json and package-lock.json to the container
4 COPY package*.json ./
5
6 # Install dependencies
7 RUN npm install
8
9 # Copy the rest of the application code to the container
10 COPY . .
11
12 # Build the application
13 RUN npm run build
14
15 # Serve the application with a lightweight HTTP server
16 FROM nginx:alpine
17 COPY --from=0 /build /usr/share/nginx/html
18 EXPOSE 80
19 CMD ["nginx", "-g", "daemon off;"]
```

```
client > Dockerfile
5 WORKDIR /
6
7 # Copy package.json and package-lock.json to the container
8 COPY package*.json ./
9
10 # Install dependencies
11 RUN npm install
12
13 # Copy the rest of the application code to the container
14 COPY . .
15
16 # Build the application
17 RUN npm run build
18
19 # Serve the application with a lightweight HTTP server
20 FROM nginx:alpine
21 COPY --from=0 /build /usr/share/nginx/html
22 EXPOSE 80
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GIT LENS

To address issues that do not require attention, run:
npm audit fix

To address all issues (including breaking changes), run:
npm audit fix --force

Run 'npm audit' for details.

PS C:\Users\Vaakash007\OneDrive\Desktop\capstone\DEVOPS_CAPSTONE\client> cd ..
PS C:\Users\Vaakash007\OneDrive\Desktop\capstone\DEVOPS_CAPSTONE\client> docker build -t client_image .
[+] Building 10.6s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 452B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:14-alpine
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load build context
=> => transferring context: 2B
=> [1/7] FROM docker.io/library/node:14-alpine@sha256:434215b487a329c9e867202ff89e704d3a75e554822e07f3e0c0f9e606121b33
=> [6/7] COPY client/public /app/public:

Tag and push the docker image to the Docker Hub.

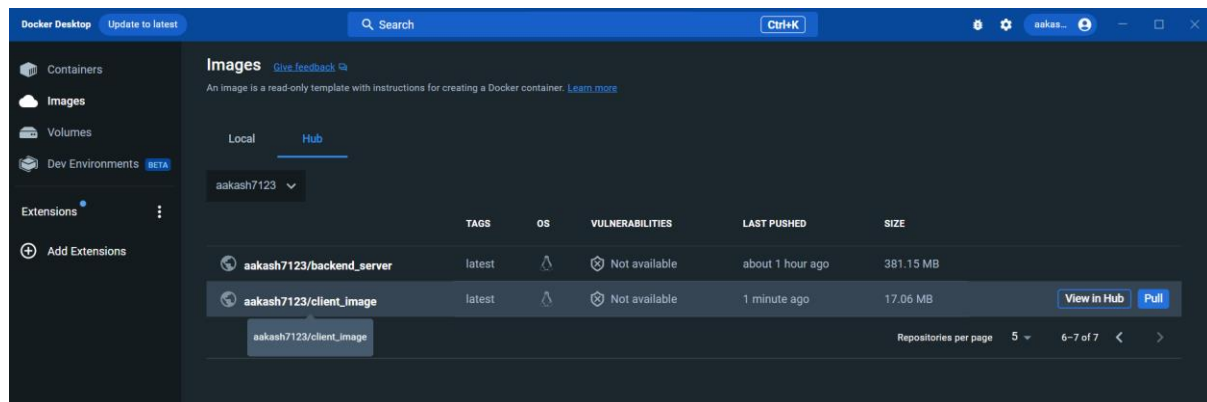
```
client > src > App.js > url
1 import React, { useState } from 'react';
2 import axios from 'axios';
3 import './App.css';
4 import MovieCard from './Components/MovieCard';
5 import { toast } from 'react-hot-toast';
6
7
8 const url = "http://3.84.89.25:5000"
9
10 const App = () => {
11   const [movie, setMovie] = useState({
12     title: '',
13     director: '',
14     releaseYear: '',
15     poster: null
16   });
17
18   const [moviesData, setMoviesData] = useState([]);
19
20   const handleInputChange = (event) => {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GIT LENS

PS C:\Users\Vaakash007\OneDrive\Desktop\cap_try\DEVOPS_CAPSTONE\client> docker images

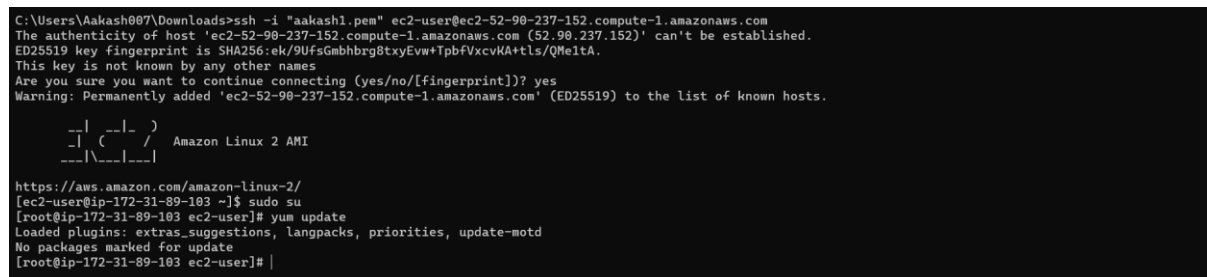
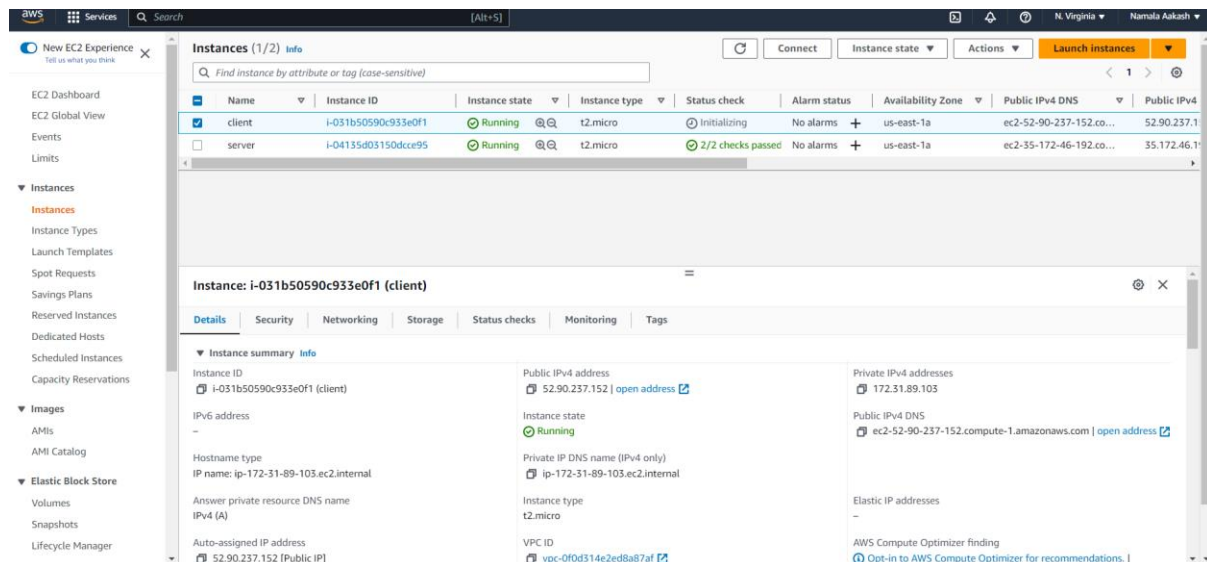
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
client	latest	ef786225f96f	10 seconds ago	42MB
<none>	<none>	f086174b96d8	15 minutes ago	42MB
aakash7123/server	latest	86892d691c1c	22 minutes ago	1.04GB
server	latest	86892d691c1c	22 minutes ago	1.04GB

PS C:\Users\Vaakash007\OneDrive\Desktop\cap_try\DEVOPS_CAPSTONE\client> docker tag client aakash7123/client
PS C:\Users\Vaakash007\OneDrive\Desktop\cap_try\DEVOPS_CAPSTONE\client> docker push aakash7123/client
Using default tag: latest
The push refers to repository [docker.io/aakash7123/client]
ff7664816e39: Pushed
31511248c7cb: Layer already exists
f0c3ff1fd4dc: Layer already exists
f0f8842de4d1: Layer already exists
c1d58c68ef: Layer already exists
1d54586a1786: Layer already exists
1003ff723696: Layer already exists



Step 9: Deploying the client in the EC2 instance.

Create an EC2 instance and connect through SSH.



Install and start docker. Later, login to the Docker Hub by specifying username and password to pull the created docker image.


```

[ec2-user@ip-172-31-89-103 ~]$ sudo su
[root@ip-172-31-89-103 ec2-user]# yum update
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
No packages marked for update
[root@ip-172-31-89-103 ec2-user]# yum install docker
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
--> Package docker.x86_64 0:20.10.23-1.amzn2.0.1 will be installed
--> Processing Dependency: runc >= 1.0.0 for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Processing Dependency: libcgrouper >= 0.40.rc1-5.15 for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Processing Dependency: containerd >= 1.3.2 for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Processing Dependency: pigz for package: docker-20.10.23-1.amzn2.0.1.x86_64
--> Running transaction check
--> Package containerd.x86_64 0:1.6.19-1.amzn2.0.1 will be installed
--> Package libcgrouper.x86_64 0:0.41-21.amzn2 will be installed
--> Package pigz.x86_64 0:2.3.4-1.amzn2.0.1 will be installed
--> Package runc.x86_64 0:1.1.4-1.amzn2.0.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

```

```

[root@ip-172-31-89-103 ec2-user]# systemctl start docker
[root@ip-172-31-89-103 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: active (running) since Sat 2023-04-29 17:47:41 UTC; 8s ago
     Docs: https://docs.docker.com
   Process: 3465 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
   Process: 3464 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
  Main PID: 3468 (dockerd)
    Tasks: 7
   Memory: 20.8M
   CGroup: /system.slice/docker.service
           └─3468 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nfile=32768:65536

Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.351863280Z" level=info msg="ClientConn switching balancer t...le=grpc
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.388231397Z" level=warning msg="Your kernel does not support...weight"
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.388767025Z" level=warning msg="Your kernel does not support...device"
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.389348207Z" level=info msg="Loading containers: start."
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.661984254Z" level=info msg="Default bridge (docker0) is ass...ddress"
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.716713537Z" level=info msg="Loading containers: done."
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.735015291Z" level=info msg="Docker daemon" commit=6051f14 g...0.10.23
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.735601013Z" level=info msg="Daemon has completed initialization"
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal systemd[1]: Started Docker Application Container Engine.
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.769964940Z" level=info msg="API listen on /run/docker.sock"
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-172-31-89-103 ec2-user]#

```

Pull the image by using **docker pull <username>/<image_name>** and the image with port 80 by using **docker run -p 80:80 <username>/<image_name>** to connect frontend to the database.

The screenshot shows a Visual Studio Code editor with a React application project named 'DEVOPS_CAPSTONE'. The file explorer on the left shows the project structure, including 'client', 'node_modules', 'public', 'src', 'Components', 'App.css', 'App.js', 'index.css', 'index.js', 'gitignore', 'Dockerfile', 'package-lock.json', 'package.json', 'README.md', 'server', and 'server.js'. The main editor displays the 'App.js' file, which contains a React component for a movie application. The terminal window at the bottom shows the following commands and output:

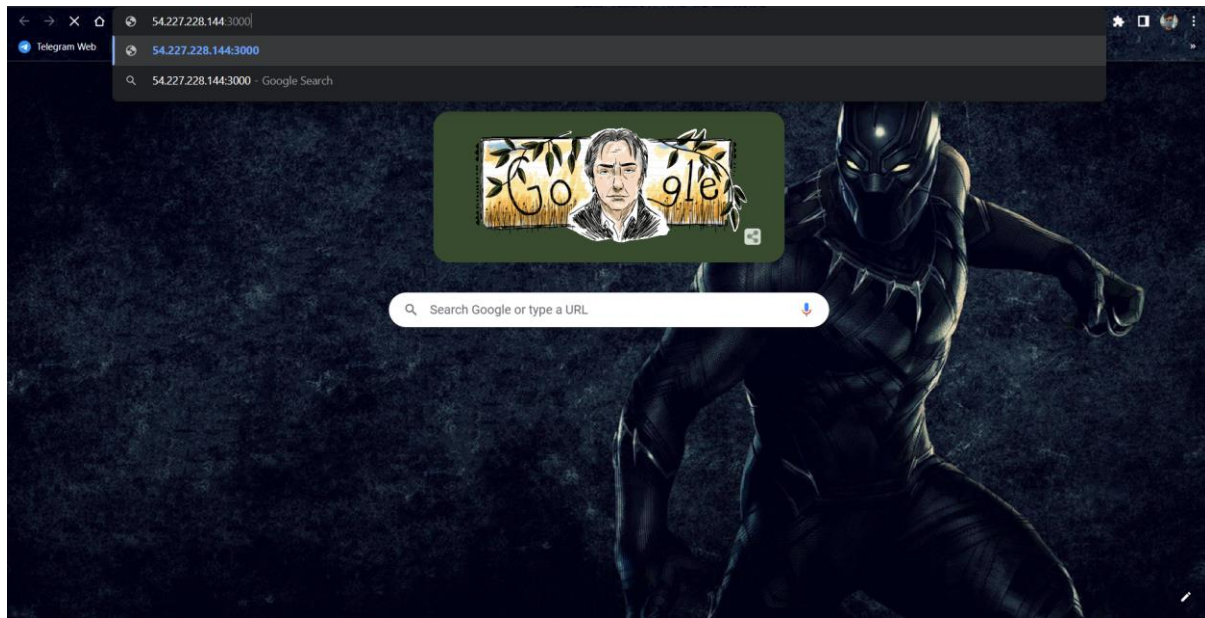
```

PS C:\Users\Vaakash007\OneDrive\Desktop\cap_try\DEVOPS_CAPSTONE\client> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
client        latest    ef786225f9ef   10 seconds ago  429B
aakash7123/server  latest    f006174b96d8   15 minutes ago  429B
aakash7123/client  latest    86892d691c1c   22 minutes ago  1.04GB
server         latest    86892d691c1c   22 minutes ago  1.04GB
PS C:\Users\Vaakash007\OneDrive\Desktop\cap_try\DEVOPS_CAPSTONE\client> docker tag client aakash7123/client
PS C:\Users\Vaakash007\OneDrive\Desktop\cap_try\DEVOPS_CAPSTONE\client> docker push aakash7123/client
Using default tag: latest
The push refers to repository [docker.io/aakash7123/client]
ff76d416e39: Pushed
3151248c7cb: Layer already exists
f9cb3f1fd3d: Layer already exists
f0f8842de4d1: Layer already exists
c1dc5c8c8ef: Layer already exists
1d54586a1706: Layer already exists
1083ff72206e: Layer already exists

```

To view the website, use the public IP address of client instance with the port.

<http://54.227.228.144:3000>



you can view the website. Enter the details and images and submit it. The details are uploaded to the database server and images are uploaded to S3 bucket specified.



Title:

Director:

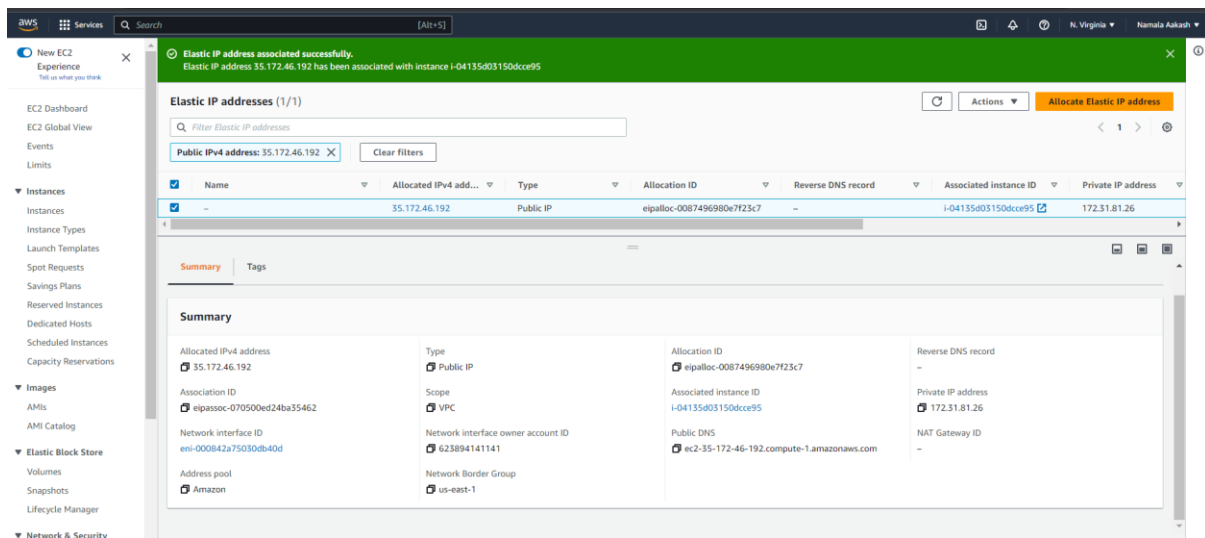
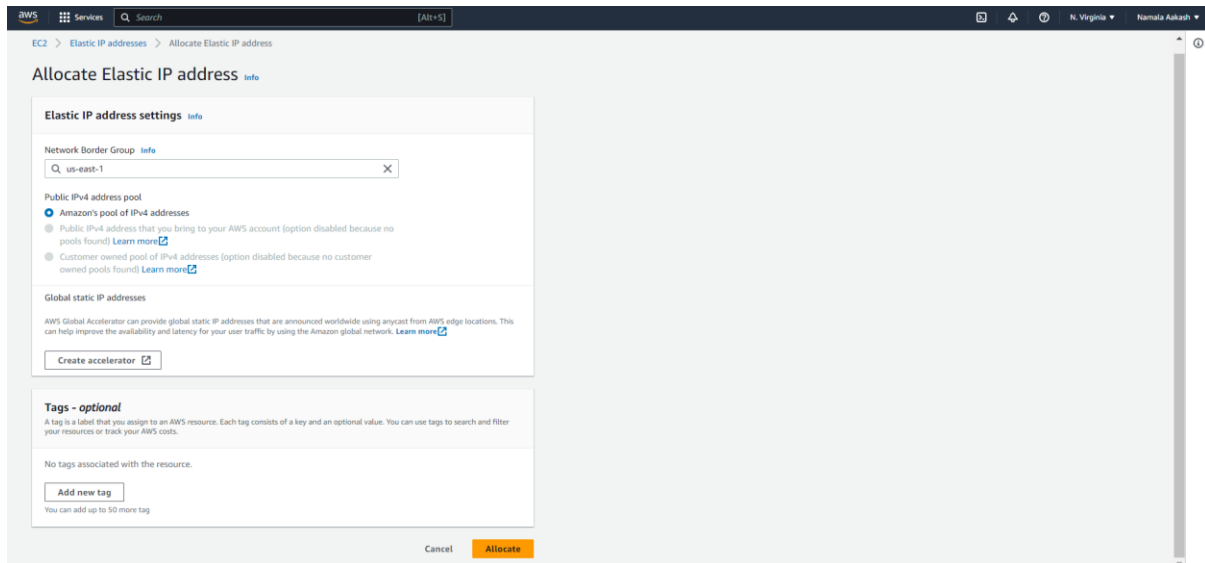
Release Year:

Poster:

charlie.jpg

Step 10: Allocating Elastic IP to instance.

By using elastic IP we can move all the attributes of the network interface to another instance in a single step. We can allocate by specifying the region, resource type and IP address.



Step 11: creating load balancer and assigning a DNS name

To create a load balancer to redirect the traffic, create target groups to the respective instance with protocol as TCP 80.

aws Services loadb X

EC2 > Target groups > Create target group

Step 1
Specify group details

Step 2
Register targets

Specify group details

Your load balancer routes requests to the targets in a target group and performs health checks on the targets.

Basic configuration

Settings in this section can't be changed after the target group is created.

Choose a target type

☒ **Instances**

- Supports load balancing to instances within a specific VPC.
- Facilitates the use of [Amazon EC2 Auto Scaling](#) to manage and scale your EC2 capacity.

☐ **IP addresses**

- Supports load balancing to VPC and on-premises resources.
- Facilitates routing to multiple IP addresses and network interfaces on the same instance.
- Offers flexibility with microservice based architectures, simplifying inter-application communication.
- Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.

☐ **Lambda function**

- Facilitates routing to a single Lambda function.
- Accessible to Application Load Balancers only.

☐ **Application Load Balancer**

aws Services loadb X

Include as pending below

4 selections are now pending below. Include more or register targets when ready.

Review targets

Targets (4) Remove all pending

All

Remove	Health status	Instance ID	Name	Port	State	Security groups	Zone	Subnet ID
X	Pending	i-0bf910d775694f8d4	client	80	Running	launch-wizard-47	us-east-1a	subnet-023ae111e0ccdaf70
X	Pending	i-001ec686a5b1a3b0	client4	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb
X	Pending	i-0d5830913145d40f9	client3	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb
X	Pending	i-05f3cc99c9c9c9c9c	client2	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb

4 pending Cancel Previous Create target group

aws Services loadb X

Include as pending below

4 selections are now pending below. Include more or register targets when ready.

Review targets

Targets (4) Remove all pending

All

Remove	Health status	Instance ID	Name	Port	State	Security groups	Zone	Subnet ID
X	Pending	i-0bf910d775694f8d4	client	80	Running	launch-wizard-47	us-east-1a	subnet-023ae111e0ccdaf70
X	Pending	i-001ec686a5b1a3b0	client4	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb
X	Pending	i-0d5830913145d40f9	client3	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb
X	Pending	i-05f3cc99c9c9c9c9c	client2	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb

4 pending Cancel Previous Create target group

Now create load balancer by giving basic configuration (vpc, region, listener and routing) and mapping listeners to get a DNS name.

Include as pending below

4 selections are now pending below. Include more or register targets when ready.

Review targets

Targets (4)

All

Remove	Health status	Instance ID	Name	Port	State	Security groups	Zone	Subnet ID
X	Pending	i-0bf910d775694f8d4	client	80	Running	launch-wizard-47	us-east-1a	subnet-023ae111e0ccdaf70
X	Pending	i-001ec686aa5b1a3b0	client4	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb
X	Pending	i-0d5830913145d40f9	client3	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb
X	Pending	i-05f3cc99c9dbdd44a2	client2	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb

4 pending

Cancel Previous **Create target group**

Include as pending below

4 selections are now pending below. Include more or register targets when ready.

Review targets

Targets (4)

All

Remove	Health status	Instance ID	Name	Port	State	Security groups	Zone	Subnet ID
X	Pending	i-0bf910d775694f8d4	client	80	Running	launch-wizard-47	us-east-1a	subnet-023ae111e0ccdaf70
X	Pending	i-001ec686aa5b1a3b0	client4	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb
X	Pending	i-0d5830913145d40f9	client3	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb
X	Pending	i-05f3cc99c9dbdd44a2	client2	80	Running	launch-wizard-48	us-east-1b	subnet-0d3b8137d8c05badb

4 pending

Cancel Previous **Create target group**

Update the existing one to point to the load balancer.

DNS name: client-lb-761177214.us-east-1.elb.amazonaws.com Now access to the DNS application.

AWS

Services

loadb

N. VirginiaNamata Aakash

Capacity Reservations

▼ Images

AMIsAMI Catalog

▼ Elastic Block Store

VolumesSnapshotsLifecycle Manager

▼ Network & Security

Security GroupsElastic IPsPlacement GroupsKey PairsNetwork Interfaces

▼ Load Balancing

Load BalancersTarget Groups

▼ Auto Scaling

Launch ConfigurationsAuto Scaling Groups

EC2 > Load balancers

Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Filter by property or value

< 1 >

☒

Name

DNS name

State

VPC ID

Availability Zones

Type

Date created

☒

client-LB

client-LB-761177214.us-e...

Active

vpc-0f0d314e2ed8a87af

6 Availability Zones

application

April 30, 2023, 23:29 (UTC+05:30)

Load balancer: client-LB

DetailsListenersNetwork mappingSecurityMonitoringIntegrationsAttributesTags

Details

arn:aws:elasticloadbalancing:us-east-1:623894141141:loadbalancer/app/client-LB/542da221d085d691

Load balancer type

Application

DNS name

client-LB-761177214.us-east-1.elb.amazonaws.com (A Record)

Status

Active

VPC

vpc-0f0d314e2ed8a87af

IP address type

IPv4

Scheme

Internet-facing

Availability Zones

us-east-1a, us-east-1b, us-east-1c, us-east-1d, us-east-1e, us-east-1f

Hosted zone

775CXPOTPD332K

CloudShellFeedbackLanguage

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

CONTRIBUTONS AND CHALLENGES FACED BY TEAM MEMBERS:

Task 1: Connecting the Server with Atlas MongoDB

The task1 is done by Sushma, Reshma, Kavya. Here they are done the process of connecting the server with Mongo db.

Task 2: Creating IAM user and s3.

The task2 is done by Rosy, Sushma, Reshma. They create s3 bucket with necessary bucket policies and created the IAM user.

Task 3: Configuring the Application Code with S3-multer with backend.

The task3 is done by Aakash, Prudhvi, kavya, Rosy

Task 4: Containerization of the code using Docker file.

The task 4 is done by Aakash, Vaishnavi, Pushpa

Task 5: Deploying server on EC2 using Docker.

The task 5 is done by Aakash, Prudhvi, Pushpa

Task 6: load balancing, Elastic IP, DNS: done by Vaishnavi, kavya, Rosy

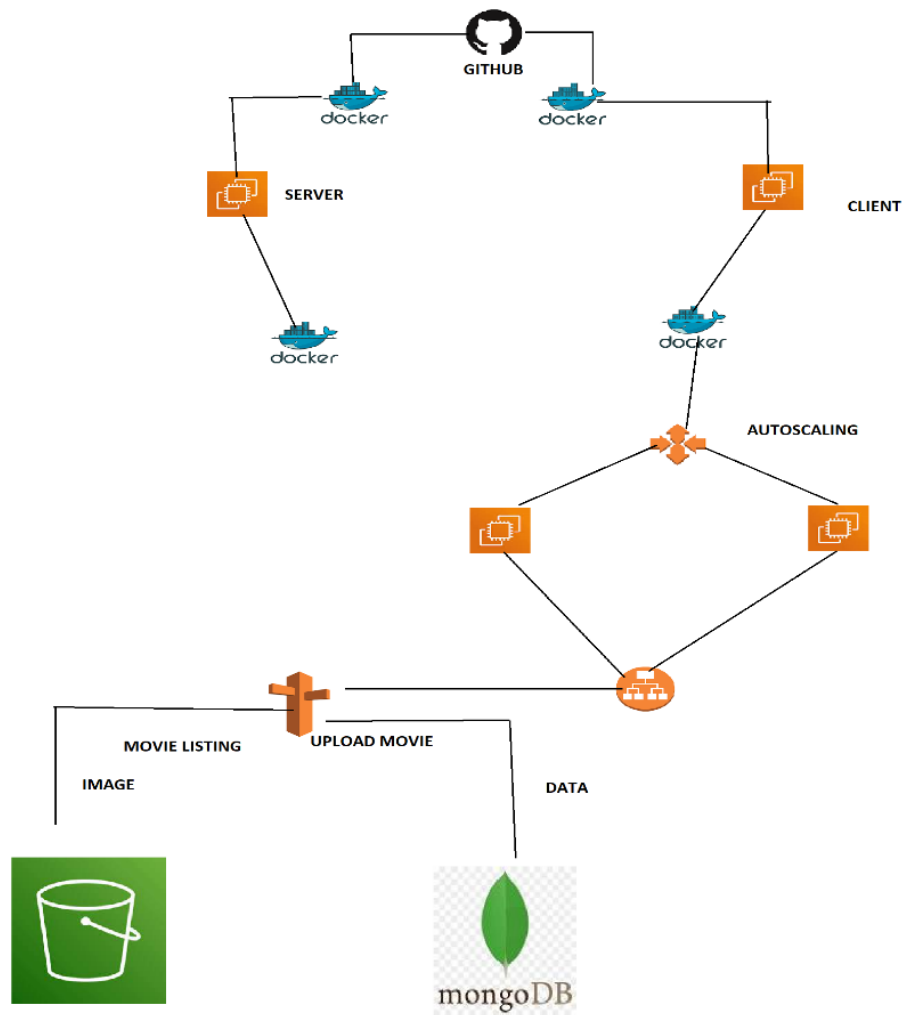
Task 7: documentation

Done by Pushpa, Vaishnavi, Prudhvi

GIT HUB REPOSITORY LINK :

https://github.com/aakash-namala/capstone_project_herovired

DEPLOYMENT ARCHITECTURE:



Conclusion:

In conclusion, deploying the "Movie listing" website to the cloud infrastructure (AWS) with proper scaling requires a series of steps, including using AWS S3 for storing images, replacing the local database with Atlas MongoDB cloud infrastructure, deploying the backend in an EC2 instance with Docker, modifying the frontend code to fetch data from the backend, creating a load balancer to properly scale the website traffic, and hosting the Docker images into AWS ECR/Docker hub. DNS configuration is also required to point to the IP.

By following these steps, the website can be deployed into the AWS cloud infrastructure, allowing for better scalability and reliability. Proper scaling is achieved by creating a load balancer that evenly distributes traffic between multiple instances of the backend running on EC2 instances. Storing images in S3 and the database in Atlas MongoDB ensures high availability and durability of data.

REFERENCE LINKS :

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/>