

# DevOps

# CAPSTONE PROJECT

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

## Team Members:

21A950A516 – SIMHADRI HIMMASRI(**TL**)  
20A91A0525 – KALAHASTHI SAHASRA  
20p31a0592 – KANTIPUDI UMA PARVATHI DEVI  
20A91A0519 – GANTI VENKATA PATTABHI  
20A91A0517 – DEMANTH VAMSI KARTHIK REDDY CHINTAPALLI  
20A91A0508 – BHUMIKA SREE SARELLA  
20P31A0580 – YASHWANT SATYA VIGNESH DULAM

## Team No. **01**

## CONTENT

CHAPTER	PAGE NO
PURPOSE OF THE PROJECT & SCOPE OF THE PROJECT	1
TECHNOLOGIES USED IN THIS PROJECT	2
MAJOR TOOLS THAT ARE UTILIZED IN THE PROCESS OF IMPLEMENTATION	3
IMPLEMENTATION	4
REFERENCES	22
DEPLOYMENT DIAGRAM	23
CONTRIBUTION OF TEAM MEMBERS	24
CONCLUSION	25

## **PURPOSE OF THE PROJECT**

The main purpose of this project is to deploy a movie listing website that uses ReactJS as frontend, NodeJS as backend, and MongoDB as a database on an Amazon EC2 instance using Docker container technology. Docker is a software platform that allows you to build, test, and deploy applications quickly. Mongo dB to replace the local database so that the data is stored in our cluster and images are stored in S3.

Whereas EC2,aws service is easy to deploy and manage virtual servers. By the end of the project, we can host a movie listing web application on EC2 instances and balance the application load with the help of Elastic Load Balancer(ELB) and it can be maintained by Docker.

## **SCOPE OF THE PROJECT**

The scope of the project is to deploy the web application for movie listing using EC2 instance and docker containerization tool. In this project, we are using ReactJS as the frontend, NodeJS as the backend, and MongoDB as the database. And this application load will be balanced with the help of one of the AWS services called Elastic Load Balancer.

In place of a local database, we use MongoDB Atlas for our data storing and S3 for storing the uploaded or existing images.S3, MongoDB, and EC2 plays a huge role in the whole project. At last, this whole web application can be accessed by giving a DNS for it.

## TECHNOLOGIES USED



**Aws** is the world's most comprehensive and broadly adopted cloud, offering over 200 fully featured services from data centres globally. over 200 services, including computing, storage, databases, analytics, machine learning, IoT, and security. It is flexible, cost-effective, and reliable. And some popular AWS services include EC2 for computing, S3 for storage, RDS for databases, Lambda for serverless computing, DynamoDB for NoSQL databases, and ECS for container management. In this project, we used the following services:



Amazon  
**EC2**



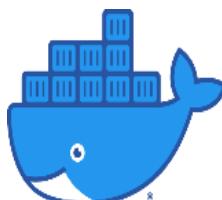
Amazon **S3**



Amazon **ELB**



**Git** is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed non-linear workflows. And we also used GitHub for cloning the repository and all. GitHub is a web-based version control and collaboration platform for software developers. Microsoft, the biggest single contributor to GitHub.



**Docker** is an open source platform that enables developers to build, deploy, run, update and manage containers—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Docker simplifies application development and deployment by enabling developers to package their applications in containers and run them on any system with Docker installed. This provides greater flexibility, portability, and scalability for applications.

## MAJOR TOOLS THAT ARE UTILIZED IN THE PROCESS OF IMPLEMENTATION

**Visual Studio Code** also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux, and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

**EC2 (Elastic Compute Cloud)** is a web service provided by Amazon Web Services (AWS) that allows users to rent virtual computers, known as instances, on which they can run their own applications. EC2 provides a flexible, scalable, and cost-effective computing environment for developers and businesses to deploy and run applications in the cloud.

**MongoDB Atlas** is an integrated suite of data services centered around a cloud database designed to accelerate and simplify how you build with data.

**Amazon S3 (Simple Storage Service)** is a cloud-based storage service that enables users to store and retrieve data from anywhere on the internet. It has various storage classes fit for multiple and diverse purposes accordingly to their requirements. It is highly reliable, secure, and scalable, and supports various types of data, including text files, images, videos, and backups.

**GitHub** is a web-based platform that provides developers with a variety of tools for software development and collaboration. It includes a version control system called Git, which allows developers to track changes to their code and collaborate with others.

**Identity and Access Management (IAM)** is a web service provided by Amazon Web Services (AWS) that enables users to securely manage access to AWS resources. IAM allows users to control who can access the AWS resources and what actions they can perform on those resources.

**Route 53** is a highly available and scalable cloud-based Domain Name System (DNS) service provided by Amazon Web Services (AWS). It helps developers and business owners by translating the domain name into an IP address. And it can be used to register and manage domain names, create and manage DNS records, and perform health checks on resources.

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

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



## TECHNOLOGIES STACK

Frontend



Database



Backend



## IMPLEMENTATION

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

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

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

A screenshot of a Firefox browser window titled "Movie Data App". The address bar shows "localhost:3000". The page displays a form for adding a movie entry. The form fields are: "Title" (input field), "Director" (input field), "Release Year" (input field), and "Poster" (input field with a "Browse..." button and placeholder "No file selected."). Below the form is a blue "Upload Movie" button. At the bottom of the page is a red "Get Movies" button. The browser's taskbar at the bottom shows various open tabs and icons.

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

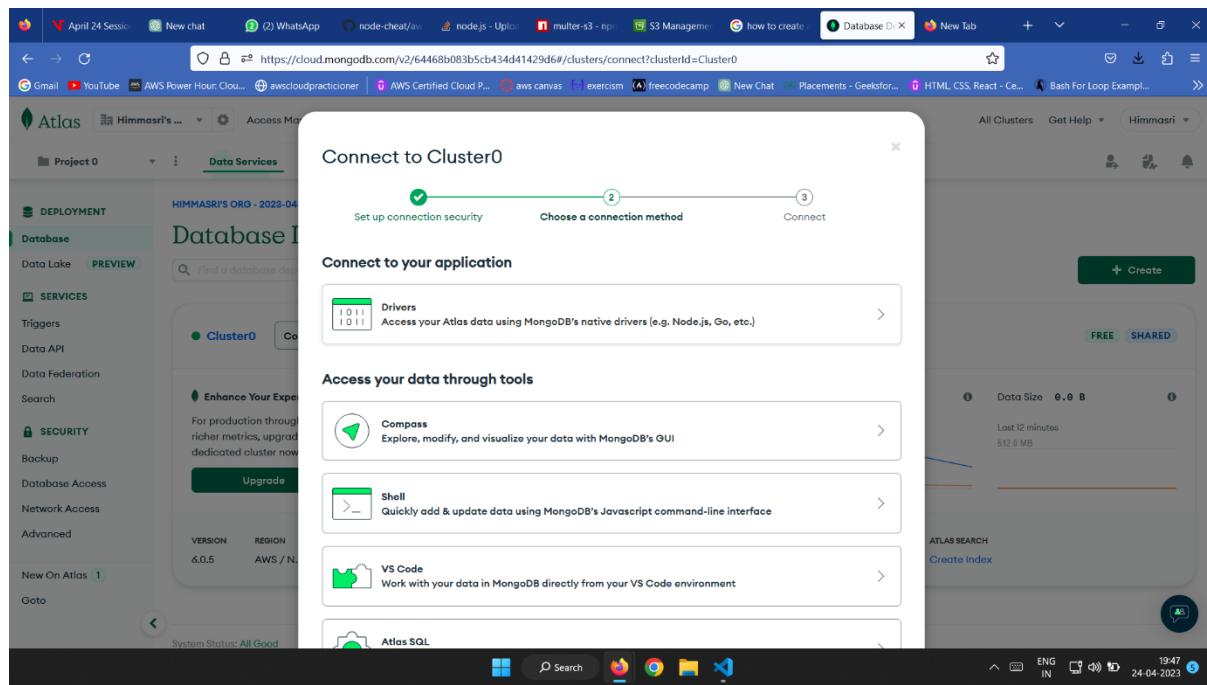
Here we are login into MongoDB Atlas account and created a cluster and given the network access asdfs allow access from anywhere

The screenshot shows the MongoDB Atlas interface. At the top, there's a navigation bar with various links like 'Cloud: Mon', 'New Tab', and search fields. Below it is a toolbar with filters for 'Availability', 'Type', 'Version', and 'Configuration'. A search bar is also present. The main content area is titled 'All Clusters' and shows a table with one row. The table columns are 'Name', 'Version', 'Data Size', 'Nodes', 'Backup', 'SSL', 'Auth', and 'Alerts'. The single entry is 'Cluster0' with version 6.0.5, N/A data size, 3 nodes, OFF backup, ON SSL, and ON Auth.

This screenshot shows the 'Database Deployments' section for 'Cluster0'. It includes a sidebar with 'DEPLOYMENT' (Database Lake, PREVIEW), 'SERVICES' (Triggers, Data API, Data Federation, Search), 'SECURITY' (Backup, Database Access, Network Access, Advanced), and a 'New On Atlas' button. The main area displays metrics for R/W operations, connections, and data size over time. It also shows the cluster tier (M0 Sandbox), type (Replica Set - 3 nodes), and backup status (Inactive). Buttons for 'Connect', 'View Monitoring', and 'Browse Collections' are available.

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



Now, copied the url and connect it to the backend

The screenshot shows the MongoDB Atlas Cluster0 Overview page. It provides instructions for connecting to the cluster using Node.js:

- 1. Select your driver and version**

We recommend installing and using the latest driver version.

Driver	Version
Node.js	4.1 or later

- 2. Install your driver**

Run the following on the command line

```
npm install mongodb
```

View MongoDB Node.js Driver installation instructions.

- 3. Add your connection string into your application code**

View full code sample

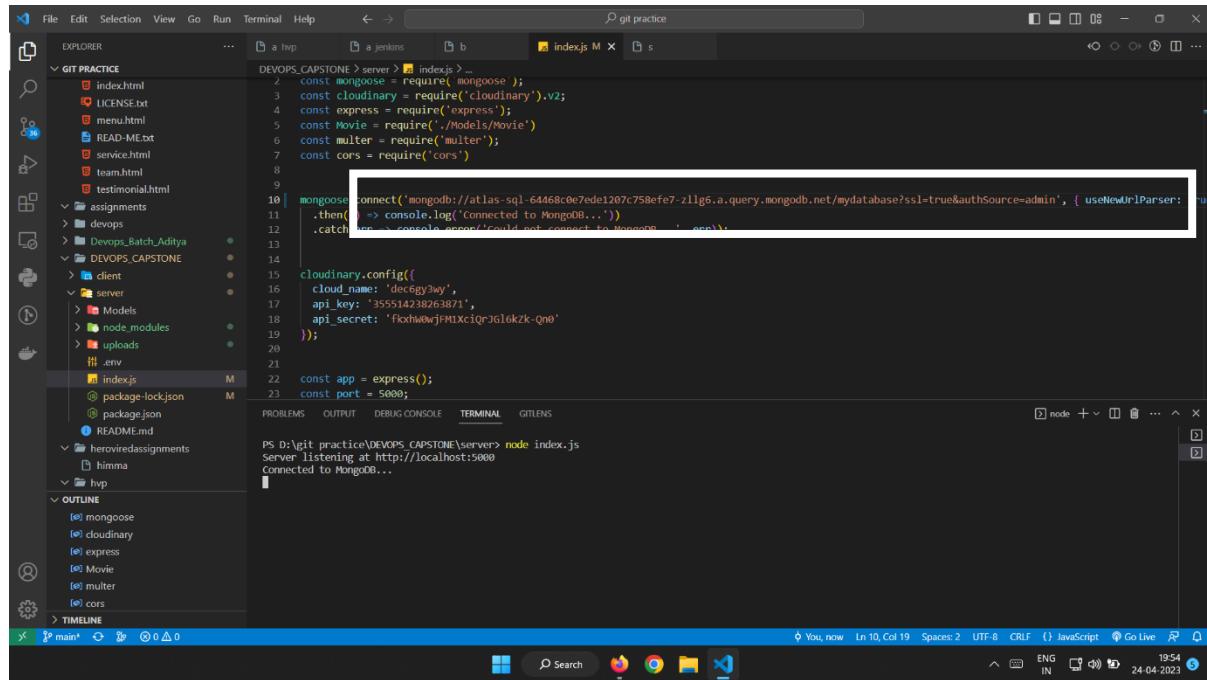
```
mongodb+srv://<himma>@cluster0.zbfvtt.mongodb.net/?retryWrites=true&w=majority
```

Replace `<password>` with the password for the `himma` user. Ensure any option params are URL encoded.

**RESOURCES**

  - Get started with the Node.js Driver
  - Node.js Starter Sample App
  - Access your Database Users
  - Troubleshoot Connections

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

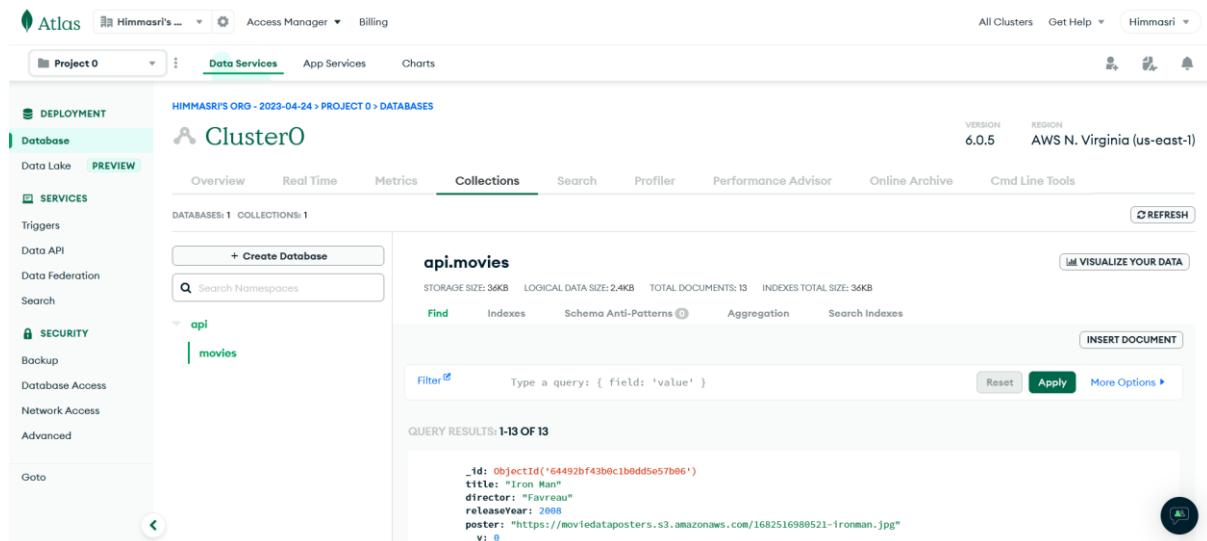


```
const mongoose = require('mongoose');
const cloudinary = require('cloudinary').v2;
const express = require('express');
const Movie = require('../Models/Movie')
const multer = require('multer');
const cors = require('cors');

mongoose.connect('mongodb://atlas-sq1-64468c0e7ede1207c758efef7-zllg6.a.query.mongodb.net/mydatabase?ssl=true&authSource=admin', { useNewUrlParser: true })
    .then(() => console.log('Connected to MongoDB...'))
    .catch(error => console.error('Could not connect to MongoDB...'));

cloudinary.config({
    cloud_name: 'decogy3wy',
    api_key: '355514238263871',
    api_secret: 'fkchWeWjFMIXciQrJGleok2k-Qn0'
});

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



The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with 'Project 0' selected under 'Data Services'. The main area shows 'Cluster0' with 'api' as the database and 'movies' as the collection. The 'Collections' tab is active. It displays 13 documents in the 'api.movies' collection. The interface includes tabs for Overview, Real Time, Metrics, Collections, Search, Profiler, Performance Advisor, Online Archive, and Cmd Line Tools. A 'REFRESH' button is visible at the top right of the collection view.

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

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

The screenshot shows the AWS Identity and Access Management (IAM) service. On the left, the navigation pane includes links for Services, EC2, VPC, and IAM. Under IAM, there are sections for Identity and Access Management (IAM), Access management (User groups, Users, Roles, Policies, Identity providers, Account settings), and Access reports (Access analyzer, Archive rules, Analyzers, Settings, Credential report). The main content area is titled 'USER' and shows a summary for a user named 'arn:aws:iam::264269359027:user/user'. The summary includes ARN, Created date (April 29, 2023, 20:09 (UTC+05:30)), Console access status (Disabled), Last console sign-in (None), and two access keys: 'Access key 1' (AKIA1TBSOM6Z2TDFOKHL - Active, Used today, Created today) and 'Access key 2' (Not enabled). Below the summary, tabs for Permissions, Groups, Tags, Security credentials, and Access Advisor are visible. The 'Permissions' tab is selected, showing 'Permissions policies (2)' attached to the user. One policy is listed: 'AmazonS3FullAccess' (AWS managed, Attached via Directly).

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

The screenshot shows the AWS Amazon S3 service. On the left, the navigation pane includes links for Services, EC2, VPC, and S3. Under S3, there are sections for Buckets (Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3), Storage Lens (Dashboards, AWS Organizations settings), and Feature spotlight. The main content area is titled 'Amazon S3 > Buckets' and shows an 'Account snapshot' with a storage lens. It lists a single bucket named 'frontendedata' with the following details: Name: frontendedata, AWS Region: US East (N. Virginia) us-east-1, Access: Public (indicated by a red warning icon), and Creation date: April 29, 2023, 20:08:36 (UTC+05:30). Action buttons for Copy ARN, Empty, Delete, and Create bucket are available at the top of the bucket list.

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

```

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 });

```

```

17 17
18 18
19 19
20 20
21 21
    const [moviesData, setMoviesData] = useState([]);
    const handleInputChange = (event) => {
        const { name, value } = event.target;

```

[main fd63d7b] completed  
4 files changed, 5 insertions(+), 5 deletions(-)  
create mode 100644 "src\public\uploads\162779561165-In Search Of Space \303\242\302\234\302\250.png"  
PS D:\git practice\DEVOPS\_CAPSTONE> git push  
Enumerating objects: 1008 (18/18), done.  
Delta compression using up to 16 threads  
Compressing objects: 1000 (10/10), done.  
Writing objects: 100% (10/10), 123.58 KB | 17.64 MB/s, done.  
Total 10 (delta 6), reused 0 (delta 0), pack-reused 0  
remote: Resolving deltas: 100% (6/6), completed with 6 local objects.  
To https://github.com/Himanshu1/DEVOPS\_CAPSTONE.git  
 0024415..fd63d7b main -> main  
PS D:\git practice\DEVOPS\_CAPSTONE> npm install aws-sdk  
[.....] \ idealtree:DEVOPS\_CAPSTONE: sill idealTree buildDeps

```

5 import {toast} from 'react-hot-toast'
6
7
8 const url = "http://3.95.106.127:8000" You, 2 hours ago + Uncommitted changes
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         const { name, value } = event.target;

```

PS D:\git practice\DEVOPS\_CAPSTONE> npm install --save multer  
added 23 packages, and audited 24 packages in 4s  
1 package is looking for funding  
 run 'npm fund' for details  
found 0 vulnerabilities  
PS D:\git practice\DEVOPS\_CAPSTONE>

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

Before containerizing we have to create instances for the server and client. Launch two instances for the server and client to deploy and connect them.

**Instances (1/2) info**

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
client	i-05e9ee037a4cde7aa	Running	t2.micro	2/2 checks passed	No alarms	+ us-east-1a	ec2-34-233-120-20
<b>server</b>	i-0f1994e979b0e7b0f	Running	t2.micro	2/2 checks passed	No alarms	+ us-east-1d	ec2-3-95-106-127.c

**Instance: i-0f1994e979b0e7b0f (server)**

Details	Security	Networking	Storage	Status checks	Monitoring	Tags
<b>Instance summary</b>						
Instance ID i-0f1994e979b0e7b0f (server)	Public IPv4 address 3.95.106.127   open address	Private IPv4 addresses 172.31.85.98				
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-3-95-106-127.compute-1.amazonaws.com   open				
Hostname type	Private IP DNS name (IPv4 only)					

## Providing Elastic IP addresses

The screenshot shows the AWS Management Console with the search bar set to "Search [Alt+S]". The left sidebar is expanded, showing sections for EC2, VPC, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, and two collapsed sections: Elastic IPs and Load Balancing. The main content area displays a success message: "Elastic IP address associated successfully. Elastic IP address 3.95.106.127 has been associated with instance i-0f1994e979b0e7b0f". Below this, a table titled "Elastic IP addresses (1/1)" lists one entry: "Name": "-", "Allocated IPv4 address": "3.95.106.127", "Type": "Public IP", "Allocation ID": "eipalloc-0f38d449d6b5f3d42", and "Reverse DNS record": "-". At the bottom, a detailed view for the IP address 3.95.106.127 is shown, with tabs for "Summary" and "Tags", and a summary section below.

The screenshot shows the VS Code interface with the following details:

- File Explorer:** On the left, it shows a project structure for "DEVOPS\_CAPSTONE". The "client" folder contains "node\_modules", "public", and "src". The "src" folder contains "Components", "App.css", "App.js", "index.css", and "index.js".
- Code Editor:** The main area displays the content of "App.js". The code uses React hooks like useState and useEffect to manage movie data and handle input changes.
- Terminal:** At the bottom, the terminal shows the output of a command: "Compiled successfully!". It also displays local and network URLs for viewing the application in a browser.
- Status Bar:** The status bar at the bottom right shows the current file is "index.js", along with other status indicators like battery level and signal strength.

## Step 5 : Containerizing of the frontend and backend code using Dockerfile

Here we are creating docker files in both frontend and backend to convert it them into docker images. Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Docker can automatically build images by reading those instructions present in the docker file.

```

    DEVOPS_CAPSTONE
      ↘ client
        > build
        > node_modules
        > public
        > src
        ◆ .gitignore
        ⚡ Dockerfile
        { package-lock.json
        { package.json
        README.md
      ↘ server
        > Models
        > node_modules
        > uploads
        .env
        ⚡ Dockerfile
        JS index.js
        { package-lock.json
        { package.json

```

```

5   const Movie = require('./Model');
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
15  mongoose.connect('mongodb+srv://');
16  .then(() => console.log('Connected to MongoDB'))
17  .catch(err => console.error(`Error connecting to MongoDB: ${err}`));
18
19
20
21  const app = express();
22  const port = 5000;
23
24  app.use(express.urlencoded({extended: true}));

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

### Server side DockerFile:

```

# Use an official Node.js runtime as a parent image
FROM node:14
# Set the working directory to /app
WORKDIR /app
# Copy the current directory contents into the container at /app
COPY . /app
# Install any needed packages specified in package.json
RUN npm install
# Make port 5000 available to the world outside this container
EXPOSE 5000
# Start the app when the container launches
CMD ["npm", "start"]

```

### Client side DockerFile:

```

# Use an official Node.js runtime as a parent image
FROM node:14-alpine
# Set the working directory to /app
WORKDIR /
# Copy package.json and package-lock.json to the container
COPY package*.json /
# Install dependencies
RUN npm install
# Copy the rest of the application code to the container
COPY ..
# Build the application
RUN npm run build
# Serve the application with a lightweight HTTP server

```

```

FROM nginx:alpine
COPY --from=0 /build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

Now we have to build images for both server and client using the “**docker build**” command

- **Docker build:** It builds Docker images from a Dockerfile and a “context”. A build's context is the set of files located in the specified PATH or URL.

Building an images for backend and frontend

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.1574]
(c) Microsoft Corporation. All rights reserved.

D:\git practice\DEVOPS_CAPSTONE>docker login
Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/

D:\git practice\DEVOPS_CAPSTONE>cd server

D:\git practice\DEVOPS_CAPSTONE>server>docker build -t server .
[+] Building 55.0s (10/10) FINISHED
=> [internal] load build definition from Dockerfile          0.1s
=> => transferring dockerfile: 32B                           0.0s
=> [internal] load .dockignore                            0.0s
=> => transferring context: 2B                           0.0s
=> [internal] load metadata for docker.io/library/node:14  2.5s
=> [auth] library/node:pull token for registry-1.docker.io  0.0s
=> [1/4] FROM docker.io/library/node:14@sha256:a158d3b904e3fa813fa6c8c590b8f0a860e015ad4e59bbcc5  0.3s
=> [internal] load build context                         3.3s
=> => transferring context: 1.12MB                      3.1s
=> CACHED [2/4] WORKDIR /app                           0.0s
=> [3/4] COPY ./app                                     5.9s
=> [4/4] RUN npm install                               41.8s
=> exporting to image                                  1.4s
=> => exporting layers                                1.4s
=> => writing image sha256:3b2f5108a2440fa74d0ed8ch1826c4e11673b2c9144534e079e2d87a13ab9a11  0.0s
=> => naming to docker.io/library/server               0.0s

```

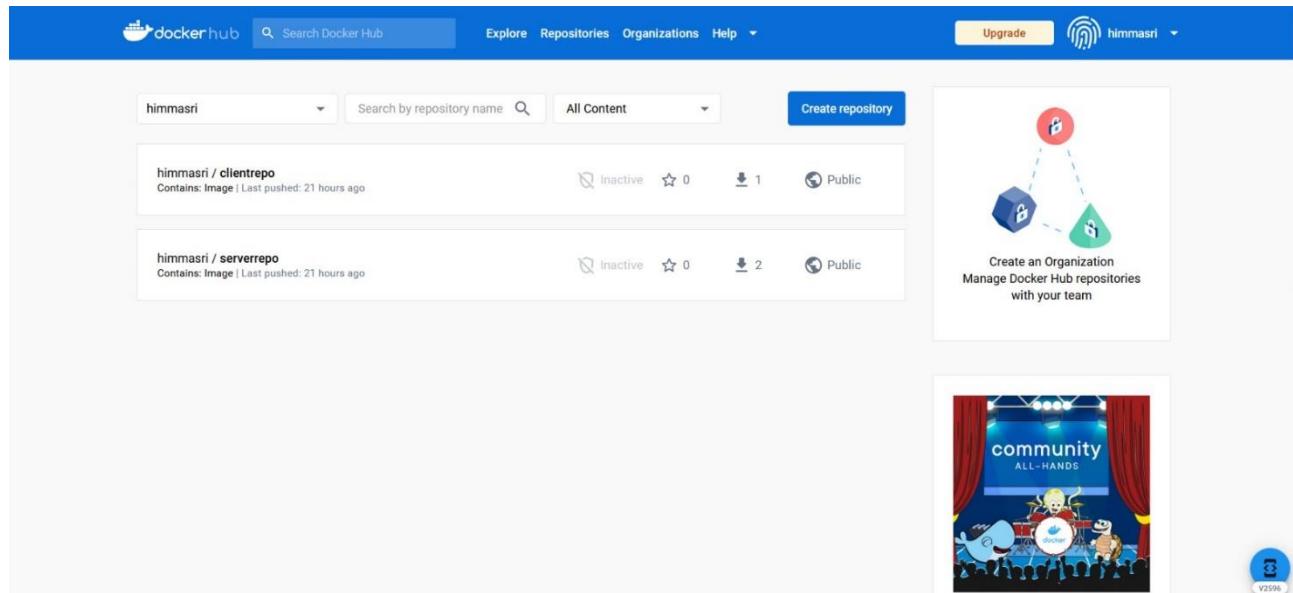
```

C:\Windows\System32\cmd.exe
D:\git practice\DEVOPS_CAPSTONE>cd client

D:\git practice\DEVOPS_CAPSTONE>client>docker build -t client .
[+] Building 58.8s (15/15) FINISHED
=> [internal] load build definition from Dockerfile          0.1s
=> => transferring dockerfile: 573B                           0.0s
=> [internal] load .dockignore                            0.0s
=> => transferring context: 2B                           0.0s
=> [internal] load metadata for docker.io/library/node:14-alpine  2.3s
=> [internal] load metadata for docker.io/library/nginx:alpine  0.0s
=> [auth] library/nginx:pull token for registry-1.docker.io  0.0s
=> [auth] library/node:pull token for registry-1.docker.io  0.0s
=> [internal] load build context                         234.35MB
=> => transferring context: 234.35MB                     234.35MB
=> CACHED [stage-1 1/6] FROM docker.io/library/nginx:alpine@sha256:dd2a9179765849767b10e2adde7e10c4ad6b7e4d4846e6b77ec93f080cd2db27
=> [stage-0 1/6] FROM docker.io/library/node:14-alpine@sha256:434215b487a329c9e86720ff89e704d3a75e554822e07f3e0c0f9e606121b33
=> 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 . .                             0.0s
=> [stage-0 5/6] RUN npm run build                   0.0s
=> [stage-1 2/2] COPY --from=0 /build /usr/share/nginx/html  0.0s
=> exporting to image                                  0.0s
=> => exporting layers                                0.0s
=> => writing image sha256:35324937f2e1cb6448cdbbe29dc05dbe56f2cec4896ca1dae62d1698650d9baa  0.0s
=> => naming to docker.io/library/client              0.0s

```

Creating a repository to store the images.



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

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

```
D:\git practice\DEVOPS_CAPSTONE\server>docker tag server:latest himmasri/serverrepo:latest
D:\git practice\DEVOPS_CAPSTONE\server>docker push himmasri/serverrepo:latest
The push refers to repository [docker.io/himmasri/serverrepo]
85319e113b20: Pushed
1d409ea365aa: Pushed
50b414c052eb: Pushed
0d5f5a015e5d: Pushed
3c777d951de2: Mounted from library/node
f8a91dd5fc84: Mounted from library/node
cb81227abde5: Mounted from library/node
e01a454893a9: Mounted from library/node
c45660adde37: Pushed
fe0fb3ab4a0f: Pushed
f1186e50c1f2: Pushed
b2dbb747754: Pushed
latest: digest: sha256:1a20ef5c0a89d9180c7e440ceb63783338d4ad2db649983cf7d8196d0504014b size: 2845
```

```
D:\git practice\DEVOPS_CAPSTONE\client>docker tag client:latest himmasri/clientrepo:latest
D:\git practice\DEVOPS_CAPSTONE\client>docker push himmasri/clientrepo:latest
The push refers to repository [docker.io/himmasri/clientrepo]
15f8768e751c: Pushed
31531248c7cb: Mounted from library/nginx
f9cb3f1f1d3d: Pushed
f0fb842dea41: Pushed
c1cd5c8c68ef: Pushed
1d54586a1706: Pushed
1003ff723696: Mounted from library/nginx
f1417ff83b31: Mounted from library/nginx
latest: digest: sha256:b9fe3baa3318cac04d70f735f2548f15c87dfac2e50de04b128279585a974fa7 size: 1991

D:\git practice\DEVOPS_CAPSTONE\client>docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
himmasri/clientrepo  latest   35324937f2e1  2 hours ago  41.8MB
client              latest   35324937f2e1  2 hours ago  41.8MB
himmasri/serverrepo  latest   3b2f5108a244  2 hours ago  1.14GB
server              latest   3b2f5108a244  2 hours ago  1.14GB
hello-world         latest   feb5d9fea6a5   19 months ago 13.3kB
```

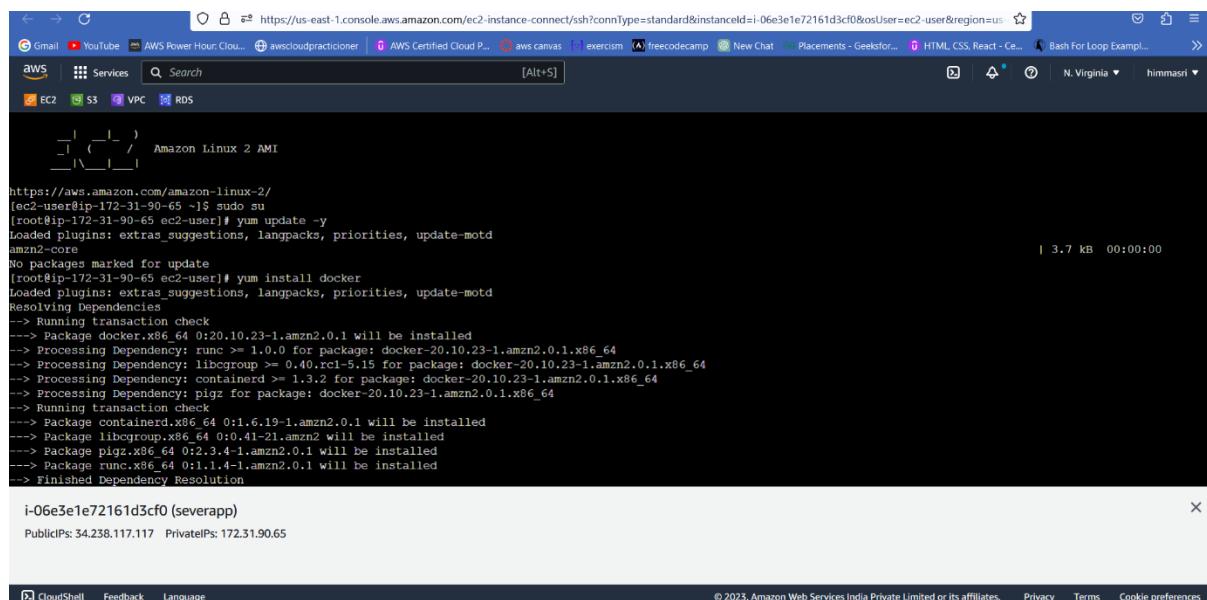
Now we will list all the images that we have still now using the command **docker ps**

```
D:\git practice\DEVOPS_CAPSTONE\client>docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
himmasri/clientrepo latest   35324937f2e1  2 hours ago   41.8MB
client              latest   35324937f2e1  2 hours ago   41.8MB
himmasri/serverrepo latest   3b2f5108a244  2 hours ago   1.14GB
server              latest   3b2f5108a244  2 hours ago   1.14GB
hello-world         latest   feb5d9fea6a5  19 months ago  13.3kB

D:\git practice\DEVOPS_CAPSTONE\client>
```

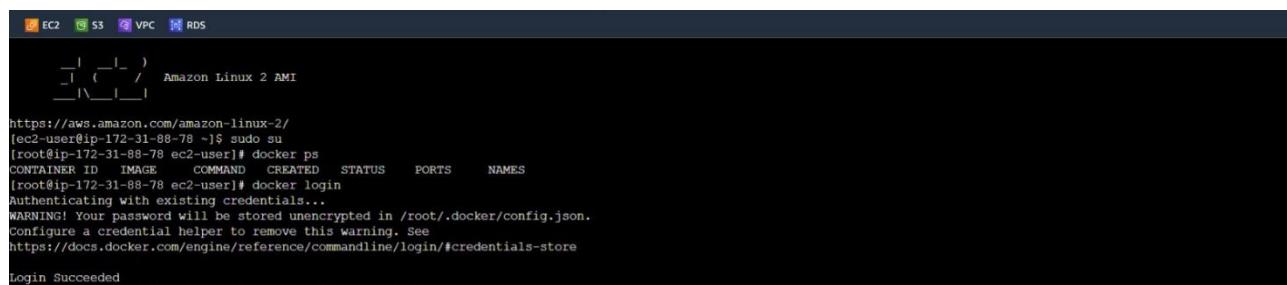
## Step 6 : Deploying client and server on EC2 instances

We have to pull images from the corresponding repositories. So before pulling the images, we have to update the system and install git and docker.



```
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-90-65 ~]$ sudo su
[root@ip-172-31-90-65 ec2-user]# yum update -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
No packages marked for update
[root@ip-172-31-90-65 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: libcgroup >= 0.40.rcl-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 libcgroup.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
i-06e3e1e72161d3cf0 (severapp)
Public IPs: 34.238.117.117 Private IPs: 172.31.90.65
```

Now, login into docker through the username and password using the docker login command and then pull the images into the instances using the docker pull



```
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-88-78 ~]$ sudo su
[root@ip-172-31-88-78 ec2-user]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS      NAMES
[root@ip-172-31-88-78 ec2-user]# docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
```

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

```

[EC2 S3 VPC RDS]
Amazon Linux 2 AMI

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-88-78 ~]$ sudo su
[root@ip-172-31-88-78 ec2-user]# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[root@ip-172-31-88-78 ec2-user]# docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@ip-172-31-88-78 ec2-user]# docker pull himmasri/client
Using default tag: latest
Error response from daemon: manifest for himmasri/client:latest not found: manifest unknown: manifest unknown
[root@ip-172-31-88-78 ec2-user]# docker pull himmasri/clientrepo
Using default tag: latest
latest: Pulling from himmasri/clientrepo
f56be85fc22e: Pull complete
2ce963c369bc: Pull complete
59b9d2200e63: Pull complete
3e1e579c95fe: Pull complete

[root@ip-172-31-22-96 ec2-user]# docker pull himmasri/serverrepo
Using default tag: latest
latest: Pulling from himmasri/serverrepo
2ff1d7c41c74: Already exists
b253aaefaa7: Already exists
3d2201bd995c: Already exists
1de76e268b10: Already exists
d9a0df589451: Already exists
6f51ee005dea: Already exists
5f32ed3c3f27: Already exists
0e8cc2f24a4d: Already exists
0d27a8e86132: Already exists
ded92201424e: Pull complete
0e764d11e393: Pull complete
9dbe5c31b7e8: Pull complete
Digest: sha256:la20e5c0a89d9180c7e440ceb63783338d4ad2db649983cf7d8196d0504014b
Status: Downloaded newer image for himmasri/serverrepo:latest
docker.io/himmasri/serverrepo:latest
[root@ip-172-31-22-96 ec2-user]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
himmasri/serverrepo latest 3b2f5108a244 16 hours ago 1.14GB
umakantipudi/capserver-repo latest f4a7228e5e18 25 hours ago 1.14GB
[root@ip-172-31-22-96 ec2-user]# docker run -d -p 8000:5000 himmasri/serverrepo
7897d75402ac286ff9d833382f13bdec693787d13f742bf1b17b8da60cd1103af
[root@ip-172-31-22-96 ec2-user]#

```

i-0beb909a16f42fd83 (server)

## Step 7: Checking whether movies are uploading or not and also getting movies when we click on get movies button or not

After deploying to check the website we use the public IP address of the Client Instance along with the listening port number.

Instances (1/6) <a href="#">Info</a>							
<a href="#">Find instance by attribute or tag (case-sensitive)</a> <span style="float: right;">C Connect Instance state Actions Launch instances</span>							
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
<input checked="" type="checkbox"/> client	i-05e9ee037a4cde7aa	<span>Running</span>	t2.micro	<span>2/2 checks passed</span>	<span>✓</span>	us-east-1a	ec2-34-233-120-20
<input type="checkbox"/> clientinstance2	i-01732faaeb7286b2	<span>Terminated</span>	t2.micro	<span>-</span>	<span>✗</span>	us-east-1a	-
<input type="checkbox"/> clientinstance1	i-0b6d366e548c72d3a	<span>Terminated</span>	t2.micro	<span>-</span>	<span>✗</span>	us-east-1a	-
<input type="checkbox"/> instance2	i-0941c5b2a0620478d	<span>Running</span>	t2.micro	<span>2/2 checks passed</span>	<span>✓</span>	us-east-1a	ec2-54-175-128-19
<input type="checkbox"/> instance1	i-021331b3ba599b3be	<span>Running</span>	t2.micro	<span>2/2 checks passed</span>	<span>✓</span>	us-east-1d	ec2-18-212-26-221
<input type="checkbox"/> server	i-0f1994e979b0e7b0f	<span>Running</span>	t2.micro	<span>2/2 checks passed</span>	<span>✓</span>	us-east-1d	ec2-3-95-106-127.c

**Instance: i-05e9ee037a4cde7aa (client)**

Details		Security		Networking		Storage		Status checks		Monitoring		Tags	
<span>Instance summary</span> <a href="#">Info</a>								<span>Public IPv4 address copied</span>					
Instance ID		i-05e9ee037a4cde7aa (client)		Instance state								Private IPv4 addresses	
IPv6 address		-		Running								Public IPv4 DNS	
Hostname type												34.233.120.207   <a href="#">open address</a>	
												172.31.28.206	
												ec2-34-233-120-207.compute-1.amazonaws.com   <a href="#">open address</a>	

The screenshot shows a Firefox browser window with the title "Movie Data App". The address bar displays the URL "34.233.120.207:8000". The main content area contains a form for entering movie details:

- Title:**
- Director:**
- Release Year:**
- Poster:**  No file selected.

Below the form are two buttons: a blue "Upload Movie" button and a red "Get Movies" button.

---

Now we will enter the details in the displayed web page and submit the details. The details are uploaded to the MongoDB server and photos are uploaded to the attached S3 bucket.

The screenshot shows the same Firefox browser window as the previous one, but with different input values in the form fields:

- Title:** Iron Man
- Director:** Favreau
- Release Year:** 2013
- Poster:**  ironman.jpg

The "Upload Movie" button is highlighted in blue, indicating it is the next action to be taken.

Movie Data App

34.233.120.207:8000

Movie Uploaded Successfully!

Title:

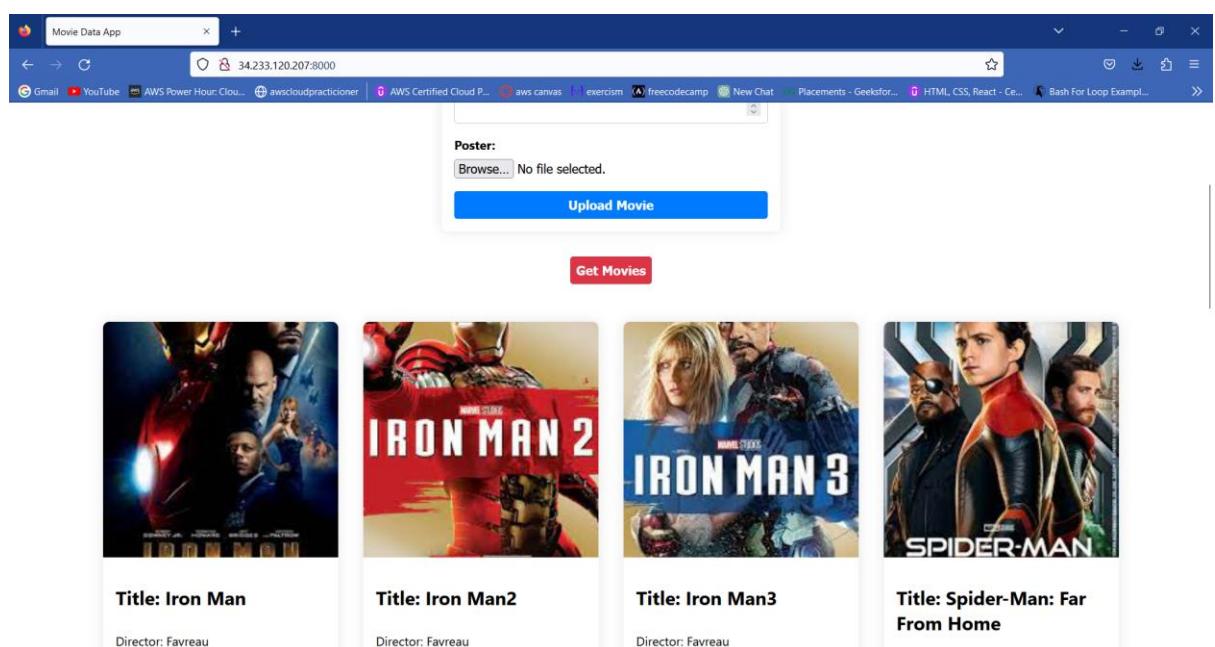
Director:

Release Year:

Poster:  ironman.jpg

**Upload Movie**

**Get Movies**



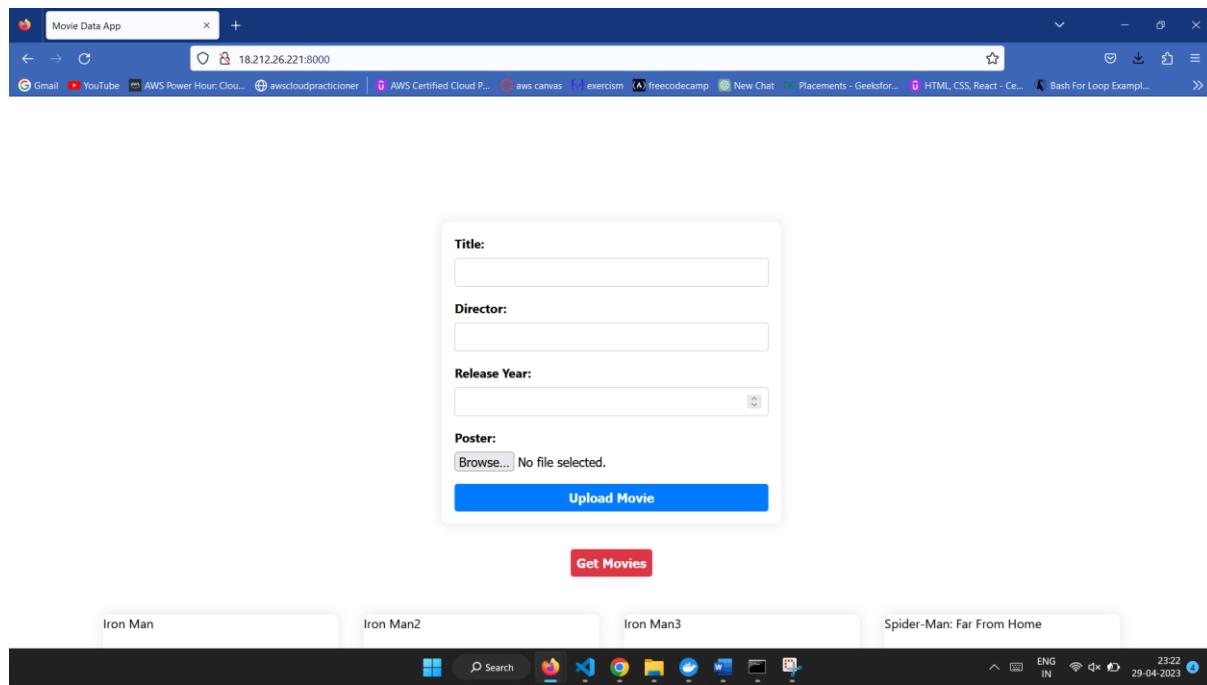
Here the uploaded images are reflected in the S3 bucket and also in the MongoDB Atlas

The screenshot shows the MongoDB Atlas Data Services interface. On the left, there's a sidebar with 'Project 0' selected, showing options like Deployment, Database (movies), Services, Triggers, Data API, Data Federation, Search, Security, Backup, Database Access, Network Access, and Advanced. The main area displays the 'api' endpoint for the 'movies' database. It includes a search bar ('Search Namespaces'), a document viewer with two movie documents (Iron Man 3 and Spider-Man: Far From Home), and a 'Find' section. At the bottom, there are links for System Status, Status, Terms, Privacy, Atlas Blog, and Contact Sales.

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

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

The screenshot shows the AWS EC2 Target Groups page. The left sidebar lists services like AMI Catalog, Elastic Block Store, Network & Security, Load Balancing, Target Groups (selected), and Auto Scaling. The main area shows a table for 'Target groups (1/1)'. One target group, 'tg1', is listed with ARN, Port (8000), Protocol (TCP), Target type (Instance), and Load balancer (elb). Below this, a detailed view for 'Target group: tg1' shows 'Registered targets (3)' with three entries: 'instance1' (healthy), 'instance2' (healthy), and 'client' (healthy). The bottom of the page includes CloudShell, Feedback, Language, and cookie preferences links.



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

Protocol:Port	Default action	ARN	Security policy	Default SSL cert	ALPN policy
TCP:8000	Forward to target group • tg1	ARN	Not applicable	Not applicable	None

Movie Data App

elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000

Title:

Director:

Release Year:

Poster:  Browse... No file selected.

**Upload Movie**

**Get Movies**

Movie Data App

elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000

Title:

Director:

Release Year:

Poster:  Browse... avengers end game.jpg

**Upload Movie**

**Get Movies**

Movie Data App

elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000

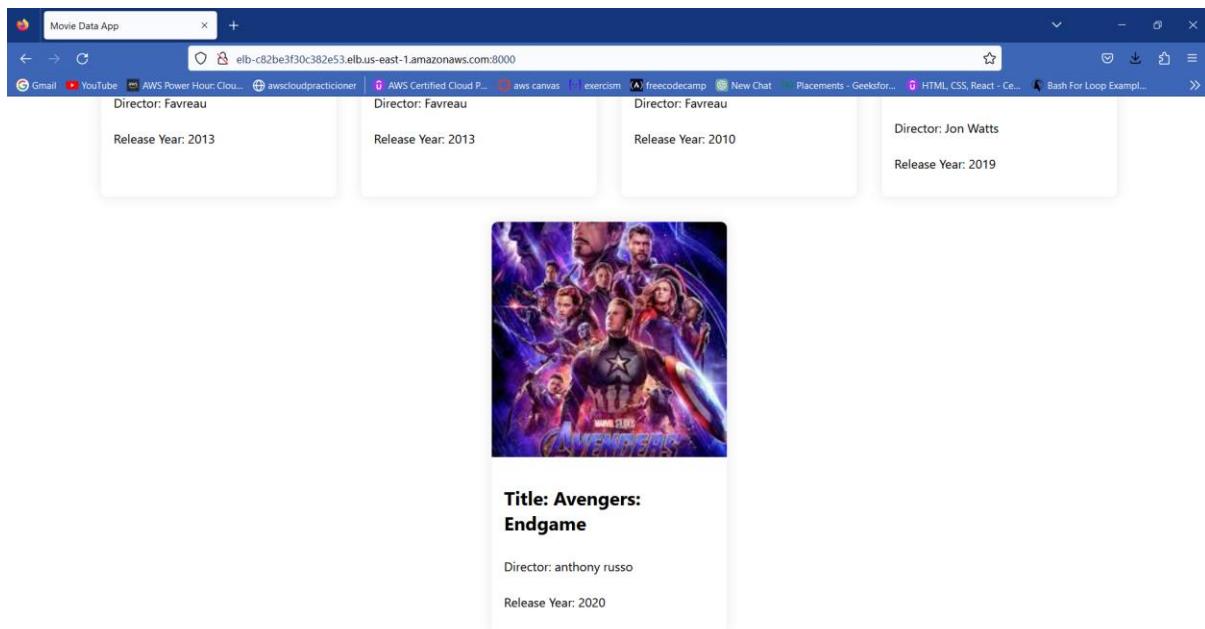
Movie Uploaded Successfully!

Title:

Director:

Release Year:

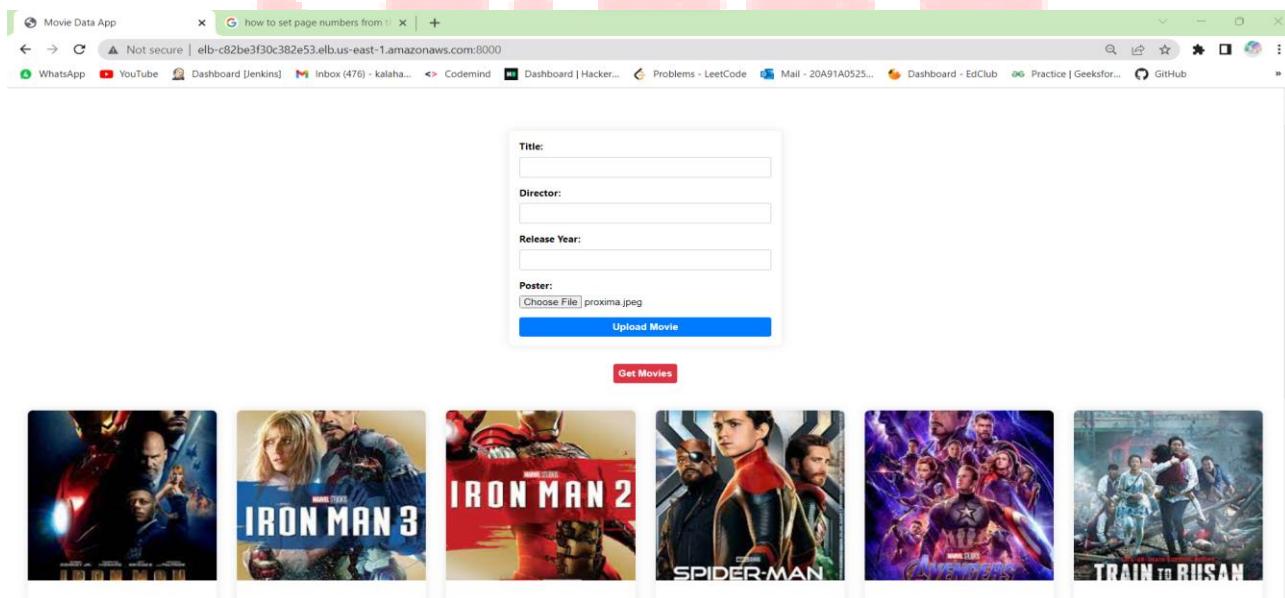
Poster:



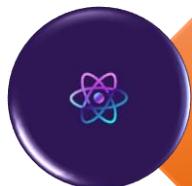
## Step 9 : Providing DNS

After the creation of the load balancer, we are able to get a DNS name which is used to turn domain names into IP addresses, which allow browsers to get to websites and other internet resources.

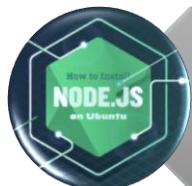
["http://elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000/"](http://elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000/)



## REFERENCE LINK



**34.233.120.207:8000**



**3.95.106.127:8000**



**[https://github.com/Himmasri/DEVOPS\\_CAPSTONE](https://github.com/Himmasri/DEVOPS_CAPSTONE)**



**<http://elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000/>**

**Frontend**

**[34.233.120.207:8000](http://34.233.120.207:8000)**

**Backend**

**[3.95.106.127:8000](http://3.95.106.127:8000)**

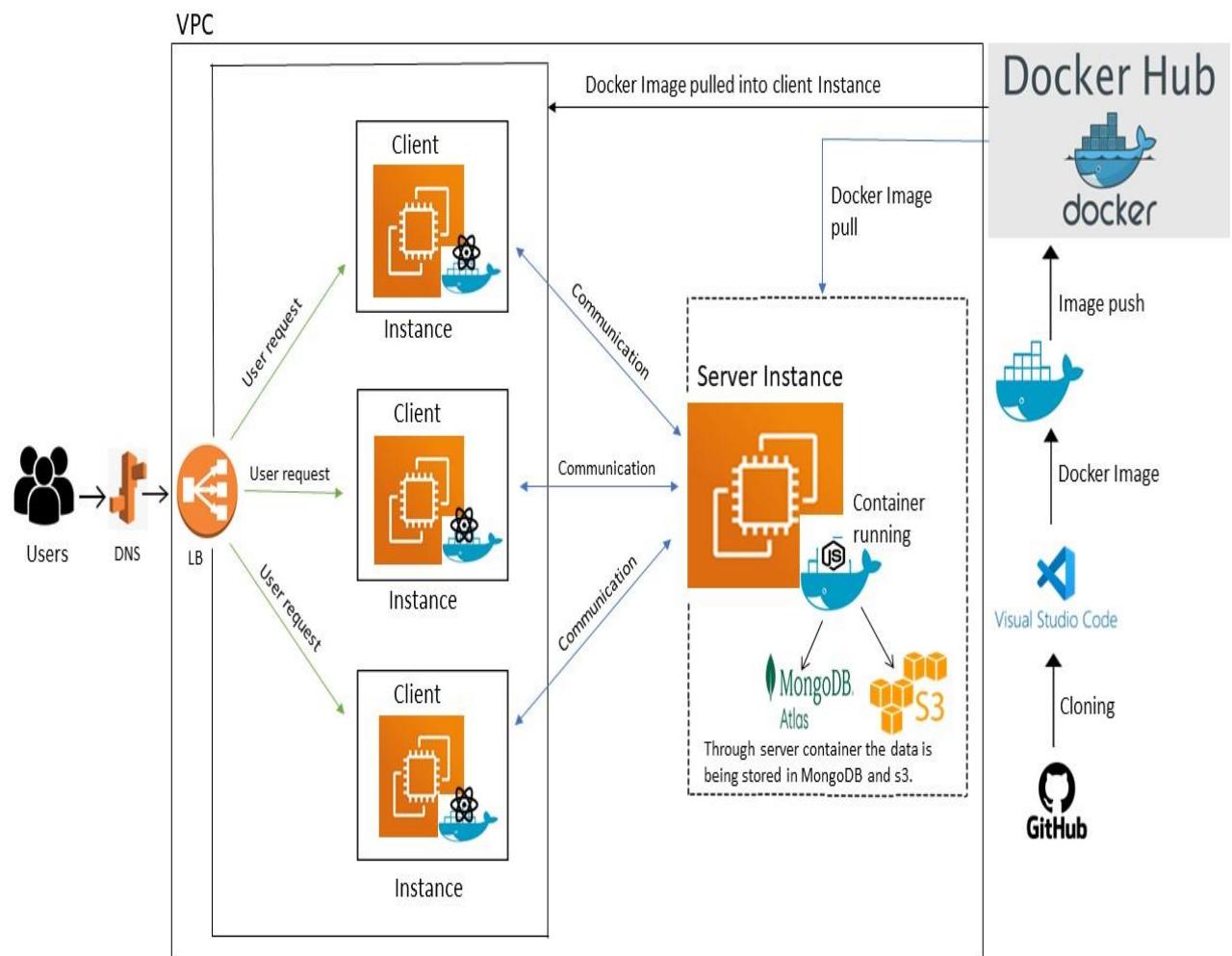
**DNS**

**[elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000/](http://elb-c82be3f30c382e53.elb.us-east-1.amazonaws.com:8000/)**

**Git-Repository Link**

**[https://github.com/Himmasri/DEVOPS\\_CAPSTONE](https://github.com/Himmasri/DEVOPS_CAPSTONE)**

## DEPLOYMENT DIAGRAM



## CONTRIBUTION OF TEAM MEMBERS

Himma

- Creation of IAM role and configuring the S3 bucket in Aws, Database creation in MongoDB Atlas

Venkat &  
Karthik

- Building the multer S3 code snippet to store poster into bucket and connecting the server to the MongoDB cluster to store data

Sahasra

- Docker file creation and Image build

Uma

- Launching of Ec2 Instances and deployed front and back end containers

Yeswanth

- Providing proper load balance to the application

Uma &  
Himma

- Deployment of "Movie Listing" application Architecture

Bhumika &  
Sahasra

- Project Documentation

## CONCLUSION

As a conclusion of this project, we successfully deployed the Movie Listing web application using docker and EC2 instances. This web application includes ReactJS as a frontend and NodeJS as a backend and used multer for uploading files into S3 bucket and used AWS services to deploy the application. This application gives the movie details that are present in the database and also we can upload movies into it.

For uploading movies we used MongoDB Atlas as a database and to store the posters we used S3 multer to upload it to the S3 buckets. Deployed frontend and backend using EC2 instances using Docker and attached an Elastic IP address to that instances and modified the code in the frontend to fetch the data from the backend. In the frontend, we have title, director, movie released year and poster fields. And it has two functions upload movies and get movies. Using upload movies, user can able to upload movies into it. And when the user clicks on Get Movies button the user will get details of all the movies which are stored in mongoDB database and images are fetched from S3 bucket.

Those uploaded poster and movie details will be reflected in S3 bucket and also MongoDB Atlas database. All the details that are present in MongoDB and S3 bucket will be reflected on to the frontend pages as

