Project Title

Movie Listing Website

Group - 2

Team Members :

M Sandhya : 20P31A0537

N Suryakala : 20P31A0538

P Gowri Alekya : 20P31A0546

J Sanjay : 20P31A0589

K Lokesh : 20P31A0598

P Satya Sri : 20P31A1243

V N Sai Srinitha : 20A91A0562

Introduction:

Our Movie listing website had been developed using ReactJS as frontend , NodeJS as backend, and MongoDB as database, which was deployed utilising Docker containers. On our website, we can upload our own movie data in addition to getting the most recent movie releases and showtimes.

As a user of our website, we have the option to add movie information, such as movie stills and posters, to our database. We can share your favourite films with other people using our simple upload function and contribute to the growth of our community of movie lovers.

We have decided to use Docker containers to deploy our website. To further provide quick and safe access to all of our movie posters and stills, including those uploaded by our users, we also store all of our photos on Amazon S3.

Scope of the Project:

The scope of the project for a movie listing website developed with ReactJS, NodeJS, MongoDB, and deployed using Docker containers and Amazon S3 storage includes:

1. **Easy Access**: Userd doesn't require any authentication and can easily access the list of available movie data.

2. **Movie Data**: Users should be able to upload movie data, including movie titles, director names, year and posters. All of this data should be stored in MongoDB Atlas for easy access and retrieval.

3. **Docker**: The application has been containerized using docker and pushed into Docker Hub.

4. **Elastic IP and Load Balancing**: The website should be deployed using EC2 instances with an Elastic IP and load balanced with a load balancer to ensure high availability and scalability.

5. **Amazon S3 Storage**: All movie images, including posters, should be stored in Amazon S3 for fast and secure access.

6. **Responsive Design**: The website should be designed to be responsive and work seamlessly on desktop, tablet, and mobile devices.

7. **Performance Optimization**: The website should be optimized for performance, with fast load times and minimal server response time.

Overall, the project should result in a fully functional and user-friendly movie listing website that allows users to easily find and discover new movies, and save their favorite movies for future reference. The use of modern technologies such as ReactJS, NodeJS, MongoDB, and Docker containers, as well as the deployment on EC2 instances with an Elastic IP and load balancing, will ensure a fast, secure, and highly available website.

Technologies Used:

ReactJS: The ReactJS is an open-source JavaScript framework and library developed by Facebook. It's used for building interactive user interfaces and web applications quickly and efficiently with significantly less code than you would with vanilla JavaScript.

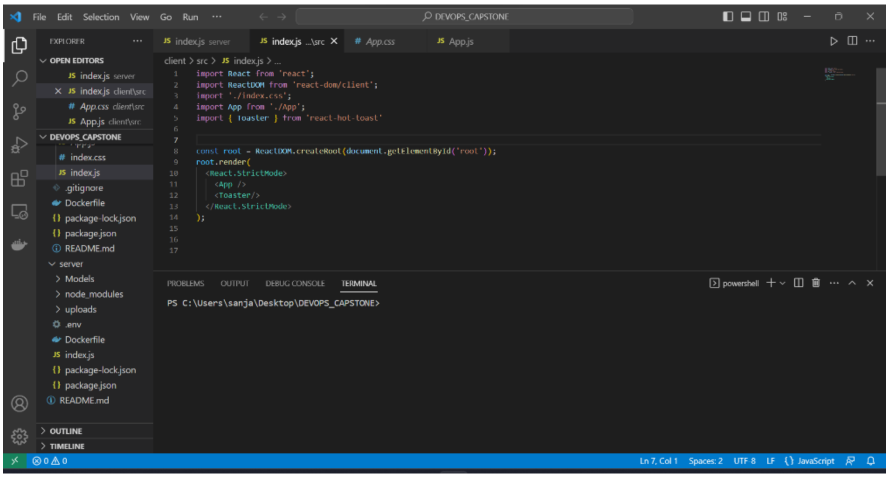
NodeJS: NodeJS is an open-source, cross-platform JavaScript runtime environment and library for running web applications outside the client's browser.

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.

Docker: Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime.

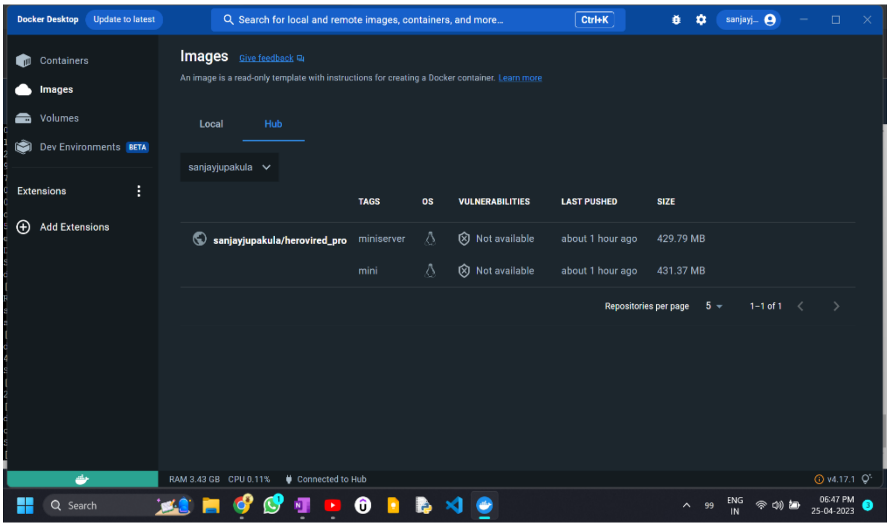
AWS Cloud: AWS is designed to allow application providers, ISVs, and vendors to quickly and securely host your applications – whether an existing application or a new SaaS-based application. You can use the AWS Management Console or well-documented web services APIs to access AWS's application hosting platform.

Proof of Project:



```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { Toaster } from 'react-hot-toast'


const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
    <Toaster/>
  </React.StrictMode>
);
```

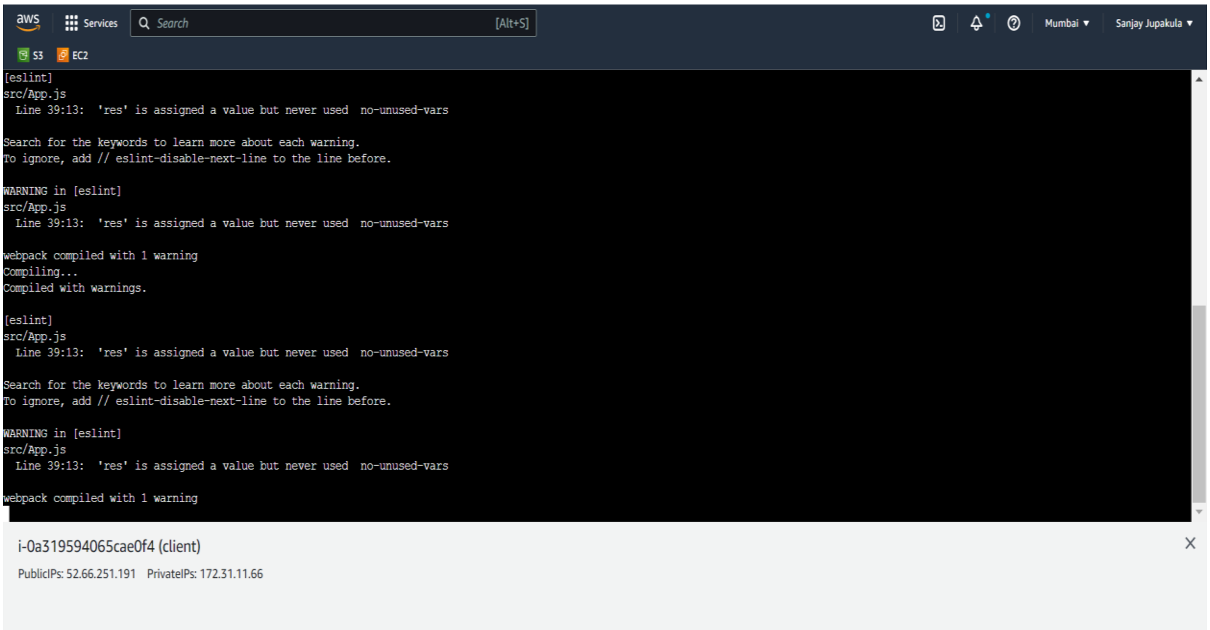
```
PS C:\Users\sanja\Desktop\DEVOPS_CAPSTONE>
```



Images Give feedback

An image is a read-only template with instructions for creating a Docker container. Learn more

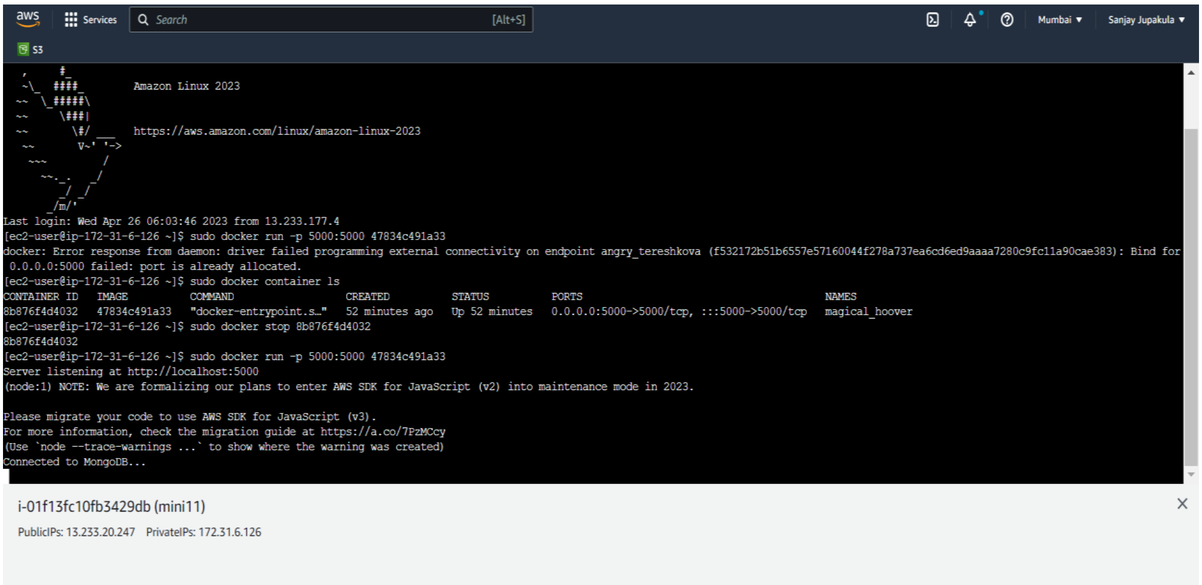
	TAGS	OS	VULNERABILITIES	LAST PUSHED	SIZE
sanjayjupakula/herovired_pro	miniserver		Not available	about 1 hour ago	429.79 MB
	mini		Not available	about 1 hour ago	431.37 MB



```
[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.

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

webpack compiled with 1 warning
Compiling...
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.

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

webpack compiled with 1 warning
```

i-0a319594065cae0f4 (client)

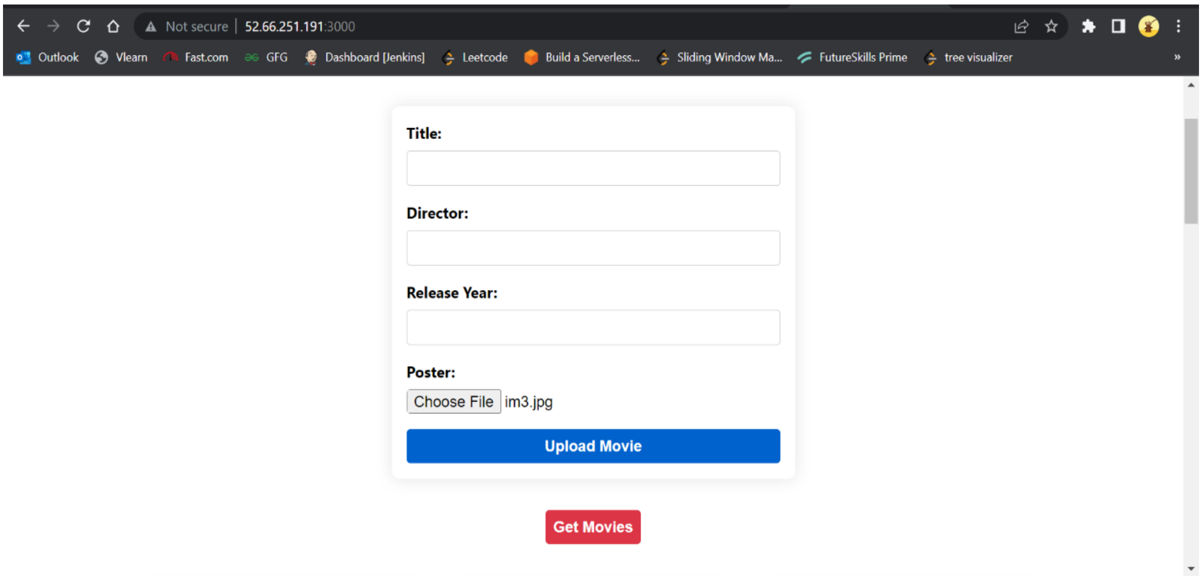
PublicIPs: 52.66.251.191   PrivateIPs: 172.31.11.66



```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Wed Apr 26 06:03:46 2023 from 13.233.177.4
[ec2-user@ip-172-31-6-126 ~]$ sudo docker run -p 5000:5000 47834c491a33
docker: Error response from daemon: driver failed programming external connectivity on endpoint angry_tereshkova (f532172b51b6557e57160044f278a737ea6cd6ed9aaaa7280c9fc11a90cae383): Bind for
 0.0.0.0:5000 failed: port is already allocated.
[ec2-user@ip-172-31-6-126 ~]$ sudo docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                       NAMES
8b876f4d4032   47834c491a33   "docker-entrypoint.s…"   52 minutes ago   Up 52 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   magical_hoover
[ec2-user@ip-172-31-6-126 ~]$ sudo docker stop 8b876f4d4032
8b876f4d4032
[ec2-user@ip-172-31-6-126 ~]$ sudo docker run -p 5000:5000 47834c491a33
Server listening at http://localhost:5000
(node:1) 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-01f13fc10fb3429db (mini11)

PublicIPs: 13.233.20.247   PrivateIPs: 172.31.6.126



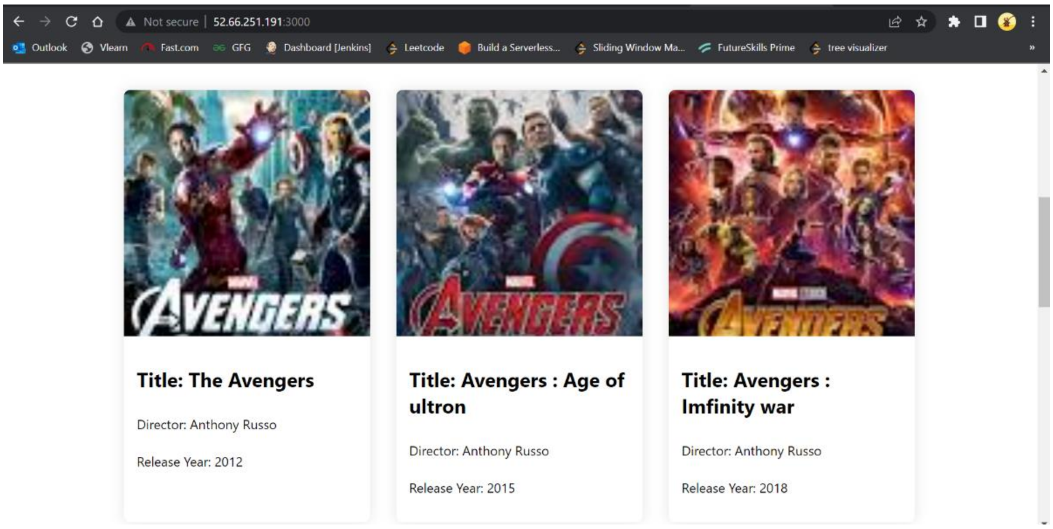
Title:

Director:

Release Year:

Poster:

Choose File im3.jpg



Title: The Avengers

Director: Anthony Russo

Release Year: 2012

Title: Avengers : Age of ultron

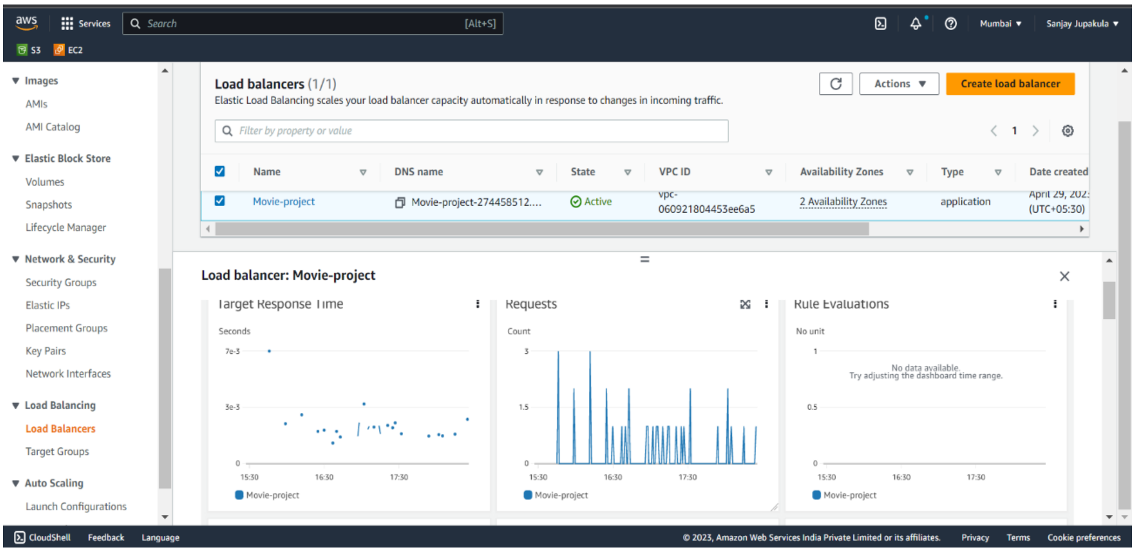
Director: Anthony Russo

Release Year: 2015

Title: Avengers : Imfinity war

Director: Anthony Russo

Release Year: 2018



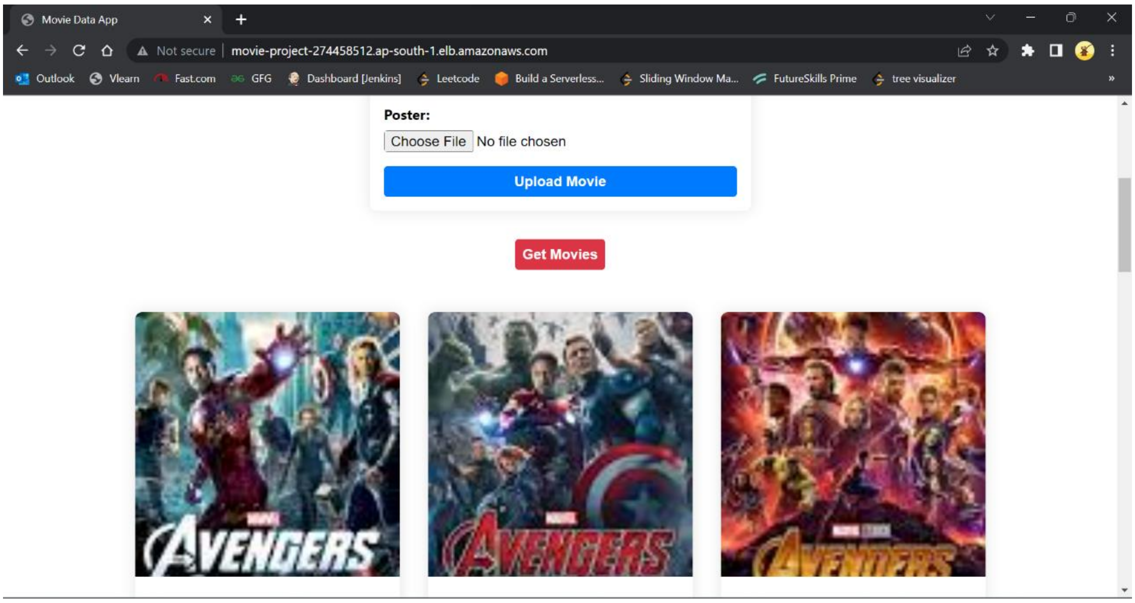
Load balancers (1/1)

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

☑	Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
☑	Movie-project	Movie-project-274458512....	✓ Active	vpc-060921804453ee6a5	2 Availability Zones	application	April 29, 202: (UTC+05:30)

Load balancer: Movie-project

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



Poster:

Choose File No file chosen

Upload Movie

Get Movies

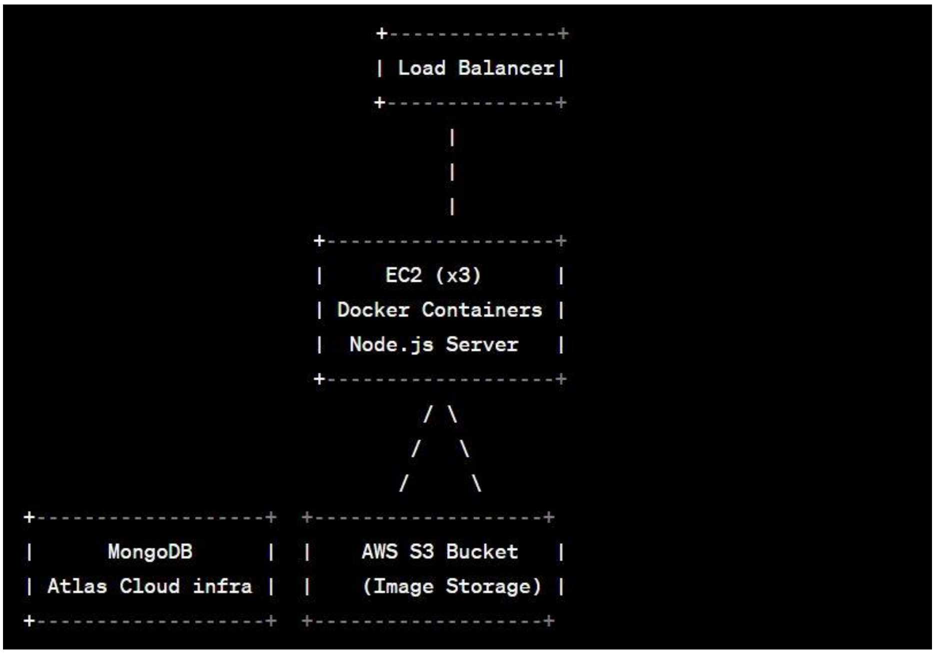
Challenges faced:

1. Converting the local database to the cloud.
   It has been done using MongoDB Atlas.
2. Connecting backend EC2 instance to the frontend EC2 instance.
   It has been done using an Elastic IP address and using port 5000.
3. Storing the poster images.
   They have been stored using an Amazon S3 bucket using multer.

Project URL:

http://movie-project-274458512.ap-south-1.elb.amazonaws.com/

Deployment Diagram:



```
                             +-------------+
                             | Load Balancer|
                             +-------------+
                                   |
                                   |
                                   |
                        +-------------------+
                        |     EC2 (x3)      |
                        | Docker Containers |
                        |  Node.js Server   |
                        +-------------------+
                                 / \
                                /   \
                               /     \
+-------------------+  +-------------------+
|      MongoDB      |  |   AWS S3 Bucket   |
| Atlas Cloud infra |  |  (Image Storage)  |
+-------------------+  +-------------------+
```

Improvements:

The above architecture can be further improved for better performance, scalability, and security. Some suggestions are:

Use AWS Fargate or Amazon Elastic Kubernetes Service (EKS) to manage and orchestrate the backend and frontend containers, which can provide automatic scaling, load balancing, and self-healing capabilities.

Use Amazon CloudFront to distribute static and dynamic content, reduce latency, and provide security and compliance controls.

Use Amazon Aurora or Amazon DynamoDB as the database to improve scalability, reliability, and performance.

Use AWS WAF (Web Application Firewall) to protect the application from common web attacks, such as SQL injection and cross-site scripting (XSS).

Use AWS Certificate Manager to manage SSL/TLS certificates and encrypt traffic between the user and the load balancer.

Use AWS CloudTrail and Amazon CloudWatch to monitor and audit the activity and performance of the infrastructure and applications.

Conclusion:

In conclusion, our movie listing website provides movie fans with a cutting-edge and flawless user experience. It was developed using ReactJS, NodeJS, and MongoDB and deployed using Docker containers. Our website provides a large selection of movie options for visitors and allows users to contribute their own movie material.

We have kept all of our movie posters in Amazon S3 to ensure quick and secure access and the data in the MongoDB Atlas. In order to guarantee dependability and scalability for our consumers, our website is also deployed in EC2 instances utilizing an elastic IP and load balanced using a load balancer.

Overall, we work hard to give our users the finest movie experience possible, with a cutting-edge and user-friendly interface, current movie lists, and dependable and efficient technology. For many years to come, we intend to keep enhancing our website and giving movie fans a first-rate experience.