

CAPSTONE PROJECT

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

GROUP-1 AND TEAM MEMBERS

- EMMANUEL DOKKADI (21P35A0347)
- POTHAMSETTI BHAVYA PRASANNA REDDY (20A91A04G7)
- NALLAMILLI PRIYANKA (20A91A04G0)
- CHALAPATHI CHAITANYA NEELIMA (21A95A0410)
- KAREDLA VIJAYA (21A95A0414)
- ANGADA VEERA VENKATA NITHYA SRI (21A95A0408)
- DAMMU HARI PRIYANKA (21A95A0411)
- NERUSUPALLI MEHER FREINEY (20P31A0439)

INDEX

- 1. OBJECTIVE OF THE PROJECT**
- 2. PURPOSE OF THE PROJECT**
- 3. SCOPE OF THE PROJECT**
- 4. TECHNOLOGIES USED IN THIS PROJECT**
- 5. TECHNOLOGIES STACK**
- 6. MAJOR TOOLS AND SERVICES UTILIZED IN THIS
PROJECT**
- 7. IMPLEMENTATION**
- 8. AWS DEPLOYMENT ARCHITECTURE**
- 9. CONCLUSION**

OBJECTIVE OF THE PROJECT

The objective is to deploy a movie listing website, built using ReactJS as frontend, NodeJS as backend, and MongoDB as the database, to the AWS cloud infrastructure with proper scaling. This involves choosing an appropriate AWS service for deployment, configuring the server, uploading the code, configuring the firewall, setting up a load balancer and automatic scaling, and monitoring the website's performance. The end result should be a highly available and scalable website that can handle high traffic volumes.

PURPOSE OF THE PROJECT

The purpose of the project is to create a movie listing website that provides users with a platform to upload and view movie details. Users will be able to upload movie details, including the title, description, and images, which will be stored in local storage. The website will be deployed to the AWS cloud infrastructure with proper scaling to ensure high availability and scalability to handle high traffic volumes. The project aims to showcase the student's skills in web development, cloud infrastructure deployment, and scaling using AWS services.

By completing this project, we will gain valuable experience in building full-stack web applications using the latest technologies and deploying them to the cloud.

SCOPE OF THE PROJECT

The scope of the project is to create a movie listing website that enables users to upload and view movie details, including the title, description, and images. The website will be built using ReactJS as the frontend technology, NodeJS as the backend technology, and MongoDB as the database technology. The scope of the project includes user authentication and authorization, data validation and error handling, search functionality, and deployment of the website to the AWS cloud infrastructure with proper scaling. The project's focus is on building a functional and secure website that can handle high traffic volumes and is highly available and scalable. The scope does not include advanced features such as movie reviews, ratings, or recommendations.

TECHNOLOGIES USED IN THIS PROJECT



- **AWS:** Amazon Web Services (AWS) is a cloud computing platform offered by Amazon that provides a wide range of cloud-based services, including compute, storage, databases, analytics, machine learning, and more. AWS allows businesses to scale and grow by providing on-demand access to a range of cloud-based computing resources that are flexible, scalable, and cost-effective. AWS offers a variety of services that can be used to build, deploy, and manage web applications, mobile apps, and other services, making it a popular choice for businesses of all sizes.
- **DOCKER:** Docker is an open-source platform that allows developers to build, package, and deploy applications as containers. Containers are lightweight, portable, and self-contained environments that contain an application and its dependencies. Docker provides a way to package and distribute applications as containers, making it easier to deploy and manage applications at scale. With Docker, developers can easily create, test, and deploy applications, which can be scaled up or down quickly to meet changing demands.
- **GIT:** Git is a distributed version control system that allows developers to manage and track changes to source code over time. With Git, developers can work collaboratively on the same codebase without worrying about overwriting each other's work. Git provides a way to create and manage different versions of a codebase, making it easier to track changes, revert to earlier versions, and collaborate on development. Git also provides a way to manage code branches, allowing developers to work on different features or versions of the codebase independently. Git has become the de facto standard for version control in software development and is widely used by developers worldwide.

TECHNOLOGIES STACK

- **Frontend:** ReactJS
- **Backend:** NodeJS
- **Database:** MongoDB

For uploading files, we are using multer in the backend.

MAJOR TOOLS AND SERVICES UTILIZED IN THIS PROJECT

These are the which are being used for completing this project:

- **MongoDB Atlas:** MongoDB Atlas is a fully-managed cloud database service that provides a secure and scalable platform for running MongoDB. MongoDB is a NoSQL database that is designed for handling large and complex data sets, and MongoDB Atlas makes it easy to deploy, manage, and scale MongoDB databases in the cloud.
- **EC2:** Amazon Elastic Compute Cloud (EC2) is a cloud-based computing service that provides resizable compute capacity in the cloud. EC2 allows users to create virtual machines, known as instances, and run applications and workloads on them. EC2 is a core component of Amazon Web Services (AWS) and is widely used for hosting web applications, running big data analytics, and performing other compute-intensive tasks in the cloud.
- **S3:** Amazon Simple Storage Service (S3) is a cloud-based storage service that provides scalable, secure, and highly available object storage in the cloud. S3 is a core component of Amazon Web Services (AWS) and is widely used for storing and retrieving data, hosting static websites, and managing data backups and archives.
- **IAM:** Amazon Identity and Access Management (IAM) is a web service that provides secure access control and user management for AWS resources. IAM allows users to control who can access their AWS resources and what actions they can perform on those resources.
- **ROUTE 53:** Amazon Route 53 is a cloud-based Domain Name System (DNS) web service that provides highly available and scalable domain name registration, routing and management. It enables users to register domain names, and route internet traffic to the AWS resources such as EC2 instances, S3 buckets, and load balancers.
- **Git hub:** GitHub is a web-based hosting service that provides version control using Git. It is primarily used for source code management and software development, but it can also be used for managing and collaborating on any kind of digital content.
- **Visual Studio Code (optional):** Visual Studio Code (VS Code) is a popular, cross-platform code editor developed by Microsoft. It provides a rich set of features for coding, debugging, and building applications, including syntax highlighting, code completion, debugging, and version control integration. VS Code supports a wide range of programming languages and frameworks, and it has a large and active community of developers who contribute to its plugins and extensions.

- **DOCKER HUB:** Docker Hub is a cloud-based registry service for storing and distributing Docker container images. It is the default registry for storing Docker images and is integrated with the Docker engine and Docker CLI, making it easy to store, share, and manage Docker images.

Docker Hub allows users to upload and download images, create and manage repositories, and set up automated builds using Docker files. It also provides access to public images, making it easy to find and use pre-built images for popular software packages and tools.

- **LOAD BALANCER:** A load balancer typically works by monitoring the incoming traffic and distributing it across multiple servers using a variety of algorithms, such as round-robin, least connections, IP hash, and others. The load balancer can also perform health checks on the servers to ensure that they are available and can handle incoming requests.

Load balancing provides several benefits, including improved performance, scalability, and availability. By distributing traffic across multiple servers, load balancing can handle more requests and provide faster response times than a single server.

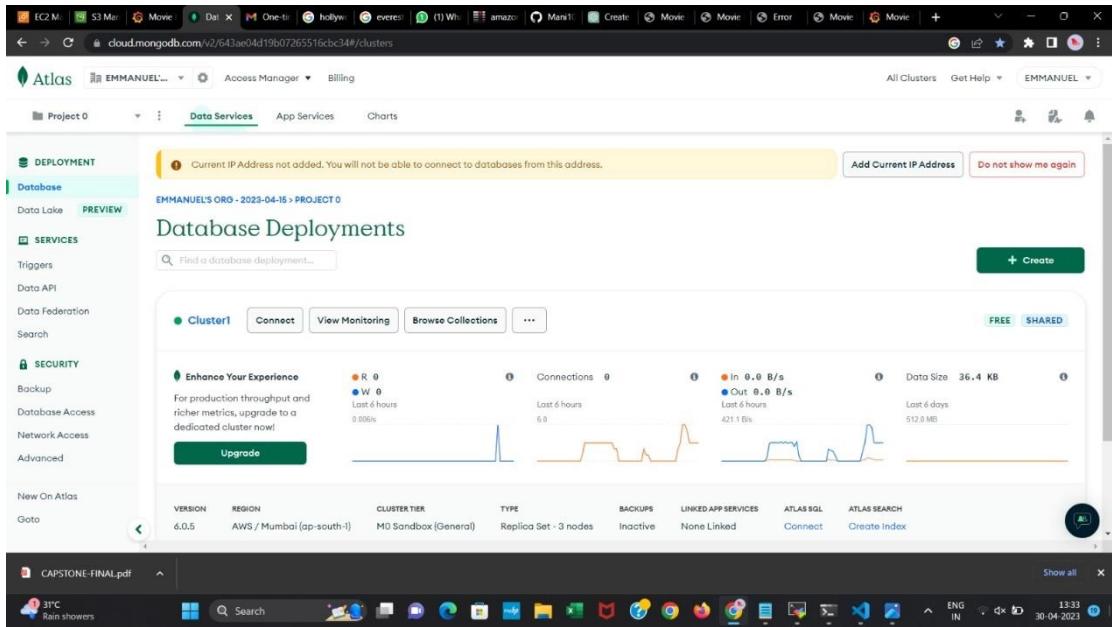
IMPLEMENTATION

STEP 1: Connecting server with MongoDB

Initially, clone the repository given below into your local desktop:

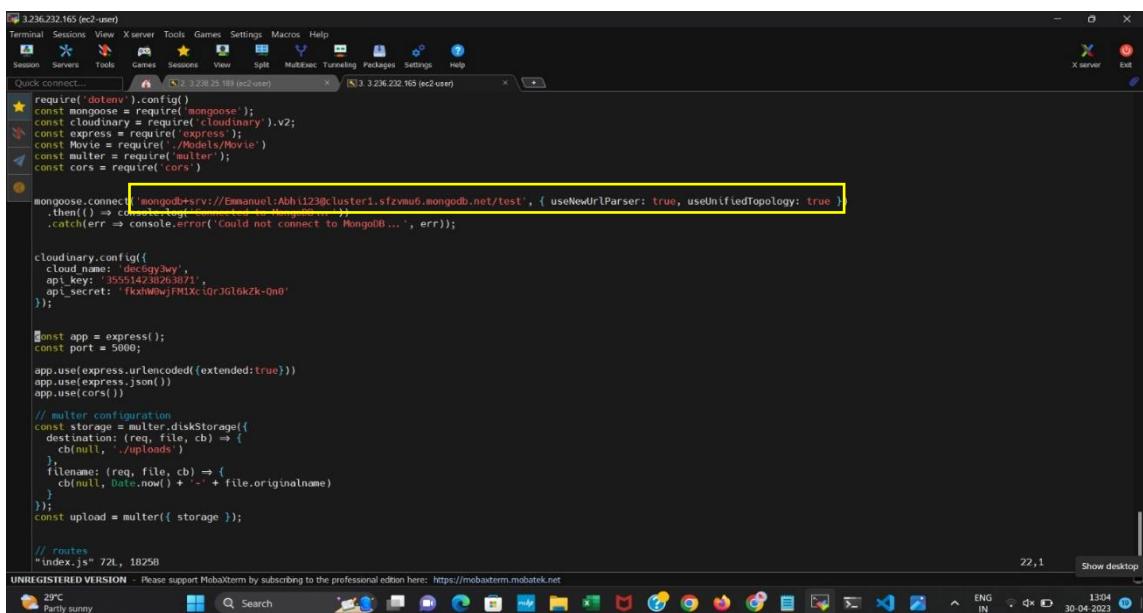
https://github.com/snehal-herovired/DEVOPS_CAPSTONE

- Create an instance, add the public IP of that instance during the MongoDB creation.



The screenshot shows the MongoDB Atlas interface. On the left sidebar, there are sections for Deployment, Database, Services, Security, and more. Under Deployment, it says 'EMMANUEL'S ORO - 2023-04-15 > PROJECT 0'. The main area is titled 'Database Deployments' and shows a single cluster named 'Cluster1'. It displays various metrics like R: 0, W: 0, Connections: 0, and Data Size: 36.4 KB. A yellow banner at the top states 'Current IP Address not added. You will not be able to connect to databases from this address.' There are buttons to 'Add Current IP Address' or 'Do not show me again'. At the bottom, there are tabs for VERSION (6.0.5), REGION (AWS / Mumbai (ap-south-1)), CLUSTER TIER (M0 Sandbox (General)), TYPE (Replica Set - 3 nodes), BACKUPS (Inactive), LINKED APP SERVICES (None Linked), ATLAS SQL (Connect), and ATLAS SEARCH (Create Index). The status bar at the bottom right shows the date and time as 30-04-2023 13:33.

- To connect your server with your MongoDB, replace the local host link with created MongoDB cluster link.



The screenshot shows a terminal window titled '3.236.232.165 (ec2-user)'. The code in the terminal is as follows:

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

mongoose.connect('mongodb+srv://Emmanuel:Abhi123@cluster1.sfrvwu6.mongodb.net/test', { useNewUrlParser: true, useUnifiedTopology: true })
.then(() => console.log('Connected to MongoDB'))
.catch(err => console.error('Could not connect to MongoDB...', err));

cloudinary.config({
  cloud_name: 'decogzy3wy',
  api_key: '35514238263073',
  api_secret: 'fkxhW0wJFM1XcI0rJGt6KZk-Qn0'
});

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

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

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

// routes
// index.js 72L, 1825B
```

The status bar at the bottom right shows the date and time as 30-04-2023 13:34.

- Commands used to start the integrated terminal:

1. npm install (for both client and server)
2. npm start (for both client and server)
3. node index.js (for both client and server)

STEP 2: Instance creation and security group configuration for front and backend.

Frontend:

- Security groups port range should be 3000, anywhere (IPv4).

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with navigation links like EC2 Dashboard, EC2 Global View, Events, Limits, Instances, Images, and Elastic Block Store. The main area displays a table of instances. Three instances are listed: 'Backend' (terminated), 'Frontend' (terminated), and 'frontend' (running). The 'frontend' instance is selected. Below the table, a detailed view for the 'frontend' instance is shown, including its Instance ID (i-049b2c3d2518def65), Public IP address (3.238.25.189), Instance state (Running), Instance type (t2.micro), and VPC ID (vpc-0ba19804339fa65c (vpc-1)).

The screenshot shows the AWS Security Groups page. In the top navigation bar, it says 'EC2 > Security Groups > sg-0209e455c8bf384e6 - launch-wizard-1 > Edit inbound rules'. The main content area is titled 'Edit inbound rules'. It shows a table of inbound rules. There are two rules listed: one for 'Custom TCP' (port 3000) and one for 'SSH' (port 22). Both rules have 'Custom' selected for the source and '0.0.0.0/0' selected for the destination. At the bottom of the table, there are buttons for 'Add rule', 'Cancel', 'Preview changes', and 'Save rules'.

BACKEND:

- Security groups port range should be 5000, anywhere (IPv4).

The screenshot shows the AWS EC2 Instances page. The left sidebar has sections for EC2 Dashboard, EC2 Global View, Events, Limits, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Scheduled Instances, Capacity Reservations, Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes). The main content area shows 'Instances (1/4) Info'. A table lists one instance: 'Backend' (i-0394cf72dd29b0992), which is 'Running' on a 't2.micro' instance type. It has 2/2 checks passed and is in the 'us-east-1f' availability zone. Below the table is a detailed view for 'Instance: i-0394cf72dd29b0992 (Backend)'. The 'Details' tab is selected, showing fields like Instance ID (i-0394cf72dd29b0992), Public IPv4 address (3.236.232.165), Instance state (Running), and Instance type (t2.micro).

The screenshot shows the AWS Security Groups page. The URL is 'us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ModifyInboundSecurityGroupRules:securityGroupId=sg-0594967afed7762dc'. The left sidebar shows 'EC2 > Security Groups > sg-0594967afed7762dc - launch-wizard-2 > Edit inbound rules'. The main content area shows the 'Edit inbound rules' wizard. It lists two rules: one for 'Custom TCP' on port 5000 with source '0.0.0.0/0' and another for 'SSH' on port 22 with source '0.0.0.0/0'. There is a 'Add rule' button at the bottom left and 'Cancel', 'Preview changes', and 'Save rules' buttons at the bottom right.

STEP 3: Create S3 bucket for storing the uploaded images

- Create S3 bucket for storing the images which are being uploaded in the website.

Amazon S3 > Buckets

Buckets (2) Info

Buckets are containers for data stored in S3. Learn more ?

Name AWS Region Access Creation date

elasticbeanstalk-us-east-1-840387266891 US East (N. Virginia) us-east-1 Objects can be public March 27, 2023, 14:27:59 (UTC+05:30)

projectbuck-1 US East (N. Virginia) us-east-1 Objects can be public April 29, 2023, 23:59:22 (UTC+05:30)

Find buckets by name

View Storage Lens dashboard | Create bucket

- Create a bucket policy.

Block all public access Off

Individual Block Public Access settings for this bucket

Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more ?

```
{ "Version": "2012-10-17", "Id": "Policy1682793116413", "Statement": [ { "Sid": "Stmt1682793107554", "Effect": "Allow", "Principal": { "AWS": "arn:aws:iam::840387266891:user/abhi" }, "Action": "s3:", "Resource": "arn:aws:s3:::projectbuck-1" } ] }
```

Edit Delete Copy

- Create IAM user to give access in order to upload images into bucket.

The screenshot shows the AWS IAM console with the URL <https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-1#users>. The left sidebar shows navigation options like Dashboard, Access management, and Access reports. The main area displays a table of users with columns: User name, Groups, Last activity, MFA, Password age, and Active key age. The user 'abhi' is selected.

User name	Groups	Last activity	MFA	Password age	Active key age
abhi	None	41 minutes ago	None	None	13 hours ago
emmanuel_dokkadi	None	Never	Virtual	61 days ago	-
new_user	None	9 days ago	Virtual	61 days ago	33 days ago

STEP 4: Modifying frontend and backend codes

- Multer S3:** Multer S3 is a middleware for handling file uploads in Node.js applications using Amazon S3 storage. Multer is a popular middleware for handling file uploads in Node.js, and Multer S3 extends its functionality by enabling files to be uploaded directly to an S3 bucket.
- Now make changes in the code for multer configuration.

The screenshot shows a VS Code editor with the file `index.js` open. The code is for a Node.js application that connects to MongoDB and uses Cloudinary and AWS S3 for file storage. A yellow box highlights the AWS S3 configuration code:

```

const s3 = new aws.S3({
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  region: process.env.AWS_REGION,
});

mongoose.connect('mongodb+srv://bhavya:bhavya@cluster01.neysay.mongodb.net/test', { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('Connected to MongoDB...'))
  .catch(err => console.error('Could not connect to MongoDB...', err));

```

The terminal below shows the command `node index.js` running successfully and the Dockerfile being built with `docker build -t server .`

- Similarly make necessary changes in env. File.
 - Add the required details such as Access Key, bucket name, ...etc from created IAM User.

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

- File Explorer:** Shows files like `index.js`, `.env`, `Dockerfile`, `index.json`, `package-lock.json`, `package.json`, and `README.md`.
- Editor:** The `.env` file is open, displaying environment variables:

```
AWS_ACCESS_KEY_ID=  
AWS_SECRET_ACCESS_KEY=  
AWS_REGION=us-east-1  
AWS_S3_BUCKET_NAME=movieclistbuck
```
- Terminal:** Shows the command `node index.js` running and the server listening at `http://localhost:5000`. It also shows the command `docker build -t server .` being run.
- Status Bar:** Shows the path `C:\Users\bhavy\OneDrive\Desktop\capstone project\DEVOPS_CAPSTONE\server`, line count (Ln 4, Col 33), and file properties.

STEP 5: Containerization

Docker file for server

```
# Use an official Node.js runtime as a parent image
FROM node:14

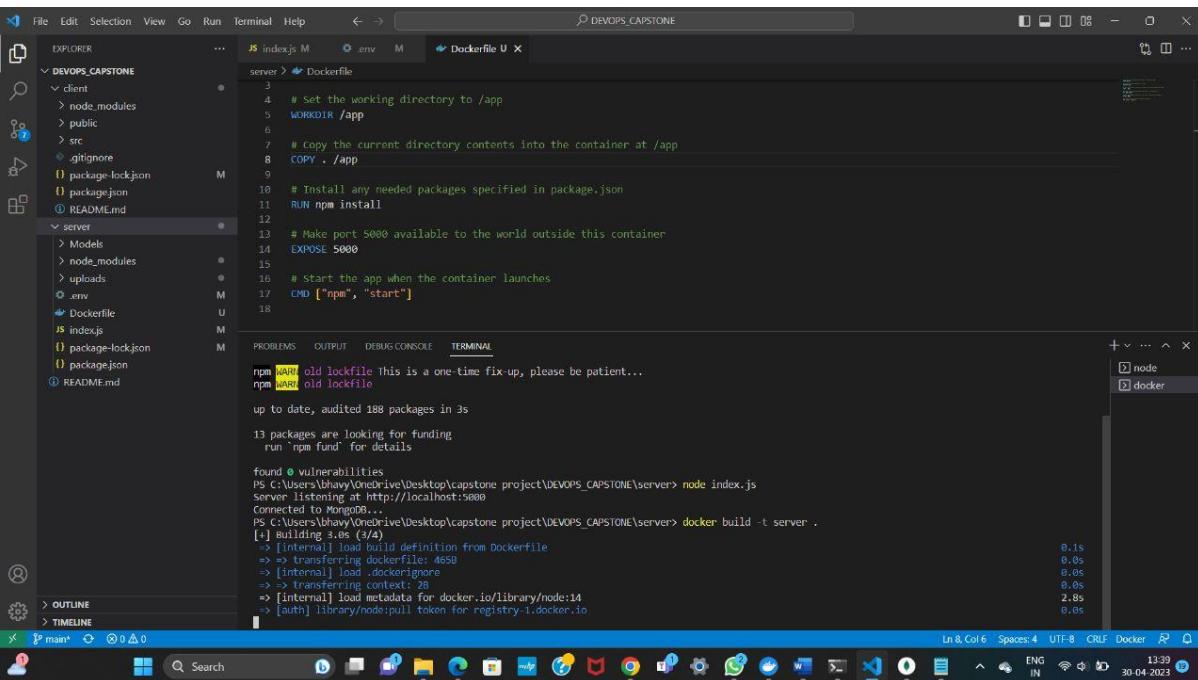
# Set the working directory to /app
WORKDIR /app

# 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"]
```



The screenshot shows the Visual Studio Code interface with the Docker extension installed. The left sidebar displays a project structure for 'DEVOPTS_CAPSTONE' containing 'client', 'server', and 'README.md' folders. The 'server' folder is expanded, showing 'Models', 'node_modules', 'uploads', '.env', 'Dockerfile', 'index.js', 'package-lock.json', 'package.json', and 'README.md'. The 'Dockerfile' tab in the center has the Dockerfile content pasted from above. The 'TERMINAL' tab at the bottom shows the command line output:

```
npm i
npm WARN old lockfile This is a one-time fix-up, please be patient...
npm WARN old lockfile
up to date, audited 188 packages in 3s
13 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\bhavya\OneDrive\Desktop\capstone project\DEVOPTS_CAPSTONE\server> node index.js
Server listening at http://localhost:5000
Connected to MongoDB.
PS C:\Users\bhavya\OneDrive\Desktop\capstone project\DEVOPTS_CAPSTONE\server> docker build -t server .
[+] Building 3.0s (3/4)
> [internal] load build definition from Dockerfile
>> transferring dockerfile: 465B
> [internal] load .dockerignore
>> transferring context: 2B
> [internal] load metadata for docker.io/library/node:14
> [auth] library/node:pull token for registry-1.docker.io
0.1s
0.0s
0.0s
0.0s
2.8s
0.0s
```

Follow these commands to build the docker image:

- docker build -t server .
- docker tag server:latest bhavyaprasannareddy/server

- docker images
 - docker push bhavyaprasannareddy/server

The screenshot shows a Windows desktop environment with the Visual Studio Code application open. The title bar reads "DEVOPS_CAPSTONE". The left sidebar displays a file tree with a "server" folder containing "Models", "node_modules", "uploads", ".env", "Dockerfile", "index.js", "package-lock.json", "package.json", and "README.md". The "Dockerfile" tab is active, showing the following content:

```
server > Dockerfile
3
4  # Set the working directory to /app
5 WORKDIR /app
6
7 # Copy the current directory contents into the container at /app
8 COPY . /app
9
10 # Install any needed packages specified in package.json
11 RUN npm install
12
13 # Make port 5000 available to the world outside this container
14 EXPOSE 5000
15
16 # Start the app when the container launches
17 CMD ["npm", "start"]
18
```

The "TERMINAL" tab is active, showing the following command-line session:

```
npm WARN old lockfile This is a one-time fix-up, please be patient...
npm WARN old lockfile
up to date, audited 188 packages in 3s
13 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
PS C:\Users\bhavya\OneDrive\Desktop\capstone project\DEVOPS_CAPSTONE\server> node index.js
server listening at http://localhost:5000
Connected to MongoDB.
PS C:\Users\bhavya\OneDrive\Desktop\capstone project\DEVOPS_CAPSTONE\server> docker build -t server .
[1] Building 3.0s (3/4)
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 465B
-> [internal] load .dockerignore
-> => transferring context: 2B
-> [internal] load metadata for docker.io/library/node:14
-> [auth] library/node:pull token for registry-1.docker.io
```

The bottom status bar shows "Ln & Col 6" and "Spaces: 4" and "UTF-8" and "CR/LF" and "Docker".

```

DEVOPS_CAPSTONE
|- client
|   |- node_modules
|   |- public
|   |- src
|   |- .gitignore
|   |- package-lock.json
|   |- package.json
|   |- README.md
|- server
  |- Models
  |- node_modules
  |- uploads
  |- .env
  |- Dockerfile
  |- index.js
  |- package-lock.json
  |- package.json
  |- README.md

Dockerfile
3
4 # Set the working directory to /app
5 WORKDIR /app
6
7 # Copy the current directory contents into the container at /app
8 COPY . /app
9
10 # Install any needed packages specified in package.json
11 RUN npm install
12
13 # Make port 5000 available to the world outside this container
14 EXPOSE 5000
15
16 # Start the app when the container launches
17 CMD [ "npm", "start" ]
18

```

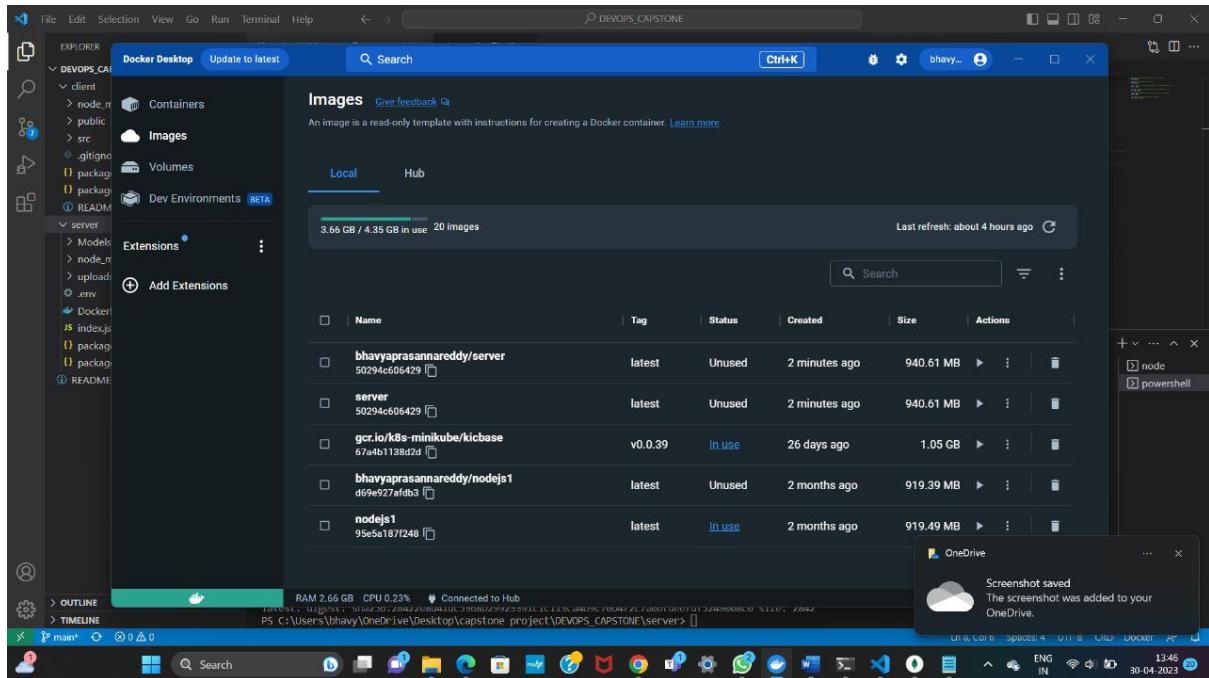
The screenshot shows the VS Code interface with the 'Dockerfile' tab selected in the top bar. The code editor displays a Dockerfile with several commands: setting the working directory to /app, copying the current directory contents into the container, installing npm packages, exposing port 5000, and starting the application with 'npm start'. The Explorer sidebar on the left shows the project structure with 'client' and 'server' folders expanded, revealing their contents like 'node_modules', 'public', 'src', and various configuration files. The bottom status bar shows the command 'PS C:\Users\bhavy\OneDrive\Desktop\capstone project\DEVOPS_CAPSTONE\server> docker push bhavyaprasannareddy/server'.

IN DOCKER HUB:

TAGS	OS	VULNERABILITIES	LAST PUSHED	SIZE
latest		Not available	2 months ago	352.74 MB
latest		Not available	1 minute ago	358.75 MB

The screenshot shows the Docker Desktop interface with the 'Images' tab selected. It displays two repositories from the Docker Hub: 'bhavyaprasannareddy/nodejs1' and 'bhavyaprasannareddy/server'. Both images are tagged 'latest' and have not been pushed recently. The Docker Desktop sidebar shows the project structure with 'client' and 'server' folders expanded. The bottom status bar shows the command 'PS C:\Users\bhavy\OneDrive\Desktop\capstone project\DEVOPS_CAPSTONE\server> v4.16.3'.

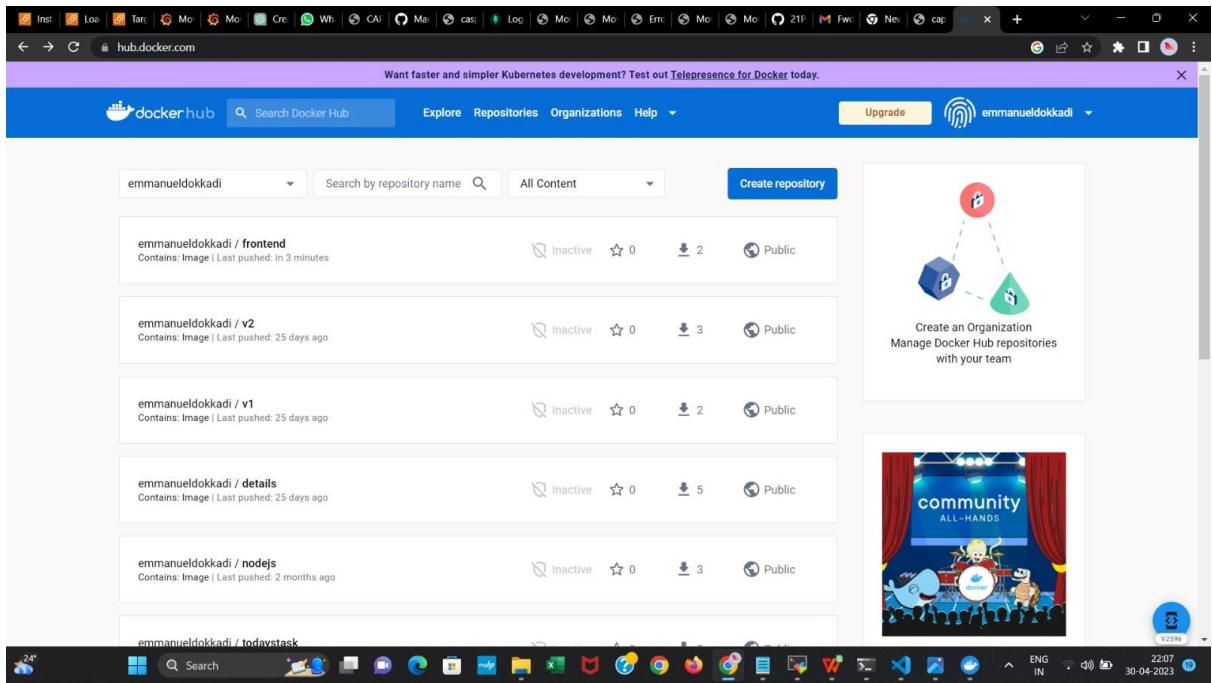
IN LOCAL:



```
3.238.25.189 (ec2-user)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiTerm Tunneling Packages Settings Help
Quick connect... 3.238.25.189 (ec2-user) 3.230.176.161 (ec2-user) ...
Successfully tagged frontend:latest
[ec2-user@ip-10-0-0-49 client]$ docker run -d -p frontend
"docker run" requires at least 1 argument.
See 'docker run --help'.
Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG ...]

Run a command in a new container
[ec2-user@ip-10-0-0-49 client]$ docker run -d -p 3000:3000 frontend
[ec2-user@ip-10-0-0-49 client]$ ls
Dockerfile README.md package-lock.json package.json public src
[ec2-user@ip-10-0-0-49 client]$ cd src
[ec2-user@ip-10-0-0-49 src]$ ls
App.css App.js Components index.css index.js
[ec2-user@ip-10-0-0-49 src]$ vim App.js
[ec2-user@ip-10-0-0-49 src]$ 
[ec2-user@ip-10-0-0-49 client]$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
0b22c7305ed9 frontend "docker-entrypoint.s..." 20 minutes ago Up 20 minutes 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp brave_jones
0b22c7305ed9
[ec2-user@ip-10-0-0-49 client]$ docker run -d -p 3000:3000 frontend
cbc5966b05df6747ca16c7772c8dc9eef054e8260795425b501c2b441e75b1d4
[ec2-user@ip-10-0-0-49 client]$ cd src
[ec2-user@ip-10-0-0-49 src]$ vim App.js
[ec2-user@ip-10-0-0-49 src]$ 
[ec2-user@ip-10-0-0-49 client]$ docker tag frontend:latest emmanueldokkadi/frontend
[ec2-user@ip-10-0-0-49 client]$ docker push emmanueldokkadi/frontend
Using default tag: latest
The push refers to repository [docker.io/emmanueldokkadi/frontend]
245e9d7d2106: Pushed
c301ad95458b: Pushed
6ac6e7415f1f: Pushed
757341575101: Pushed
96095a1f1f1f: Pushed
31f710a1c178: Mounted from library/node
a599bf3e50b0: Mounted from library/node
e67e8805abae: Mounted from library/node
f1417ff83331: Layer already exists
latest: digest: sha256:3129ecad03896f119bf3277be5e5e90a601d2bf773f58f992597b086028c3f77 size: 2286
[ec2-user@ip-10-0-0-49 client]$ 
```

UNREGISTERED VERSION - Please support MoboXterm by subscribing to the professional edition here: <https://moboxterm.mobatek.net>



- Similarly continue the process for client.

Docker file for client

```
# base image
FROM node:14-alpine

# set working directory
WORKDIR /app

# add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH

# install and cache app dependencies
COPY client/package.json /app/package.json

RUN npm install --silent
RUN npm install react-scripts@4.0.3 -g --silent

# start app
COPY client/public /app/public
COPY client/src /app/src
CMD ["npm", "start"]
```

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a project named 'DEVOPS_CAPSTONE' with subfolders 'client' and 'server'. Inside 'client', there are files like 'Dockerfile', 'package-lock.json', 'package.json', 'README.md', and 'index.js'. The 'Dockerfile' tab is selected, displaying the following content:

```
client > Dockerfile
1 # base image
2 FROM node:14-alpine
3
4 # set working directory
5 WORKDIR /app
6
7 # add '/app/node_modules/.bin' to $PATH
8 ENV PATH /app/node_modules/.bin:$PATH
9
10 # install and cache app dependencies
11 COPY client/package.json /app/package.json
12 RUN npm install --silent
13 RUN npm install react-scripts@4.0.3 -g --silent
14
15 # start app
16 COPY client/public /app/public
17 COPY client/src /app/src
18 CMD ["npm", "start"]
```

The Terminal tab shows ESLint errors in 'src\app.js':

```
[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
```

The status bar at the bottom indicates the command is 'PS C:\Users\bhavya\OneDrive\Desktop\capstone project\DEV\myhp\STONE\client> docker build -t bhavyaprashnareddy/client .' and the date is '30-04-2023'.

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a project named 'Capstone_PROJECT' with subfolders 'Capstone', 'client', 'public', and 'src'. Inside 'src', there are files like 'Components', '# App.css', 'App.js', 'index.css', 'index.js', '.gitignore', 'Dockerfile', 'package-lock.json', 'package.json', 'README.md', 'server', 'Models', 'node_modules', 'uploads', '.env', and 'index.js'. The 'App.js' tab is selected, displaying the following code:

```
const url = "http://3.230.176.161:5000"

const App = () => {
  const [movie, setMovie] = useState({
    title: '',
    director: '',
    releaseYear: '',
    poster: null
  });

  const [moviesData, setMoviesData] = useState([]);

  const handleInputChange = (event) => {
    const { name, value } = event.target;
    setMovie({ ...movie, [name]: value });
  };

  const handlePosterChange = (event) => {
    setMovie({ ...movie, poster: event.target.files[0] });
  };

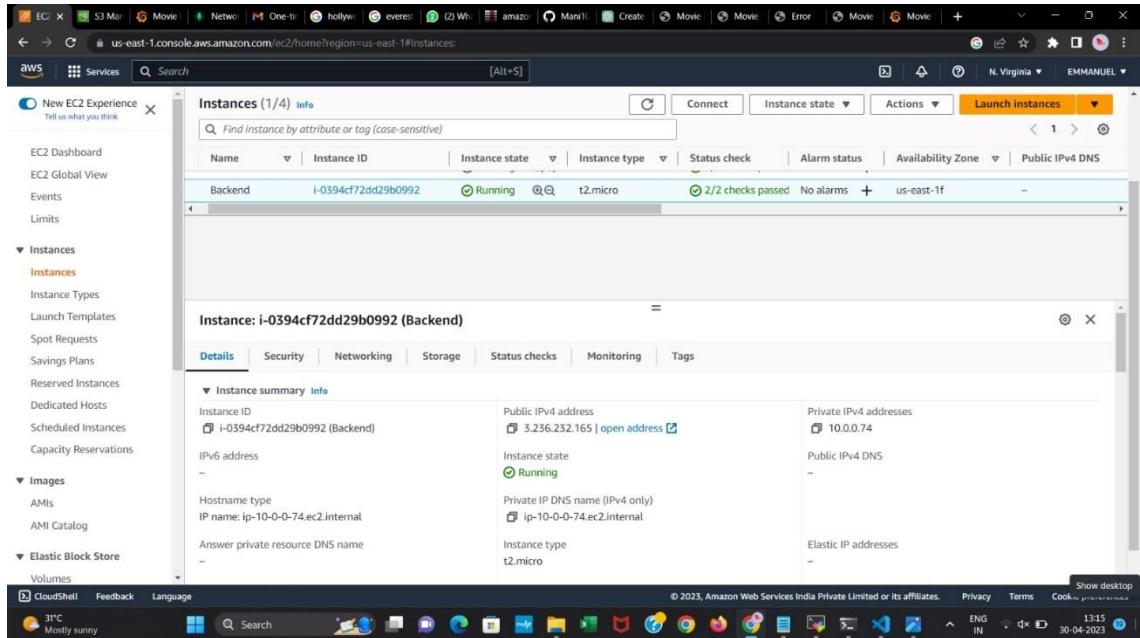
  const handleSubmit = async (event) =>
```

The status bar at the bottom indicates the line is 'Ln 8, Col 34' and the date is '30-04-2023'.

- Replace the URL with instance IP address and port in the file App.js.

STEP 6: Deploying server on EC2 using docker

- Creating EC2 instance to deploy the backend with the help of docker



```
[ec2-user@ip-10-0-0-74 ~]$ sudo yum install nodejs
[sudo] password for ec2-user:
Last metadata expiration check: 0:19:18 ago on Sun Apr 30 06:48:01 2023.
Dependencies resolved.
=====
Transaction Summary
=====
Install 6 Packages

Total download size: 31 M
Installed size: 168 M
Is this ok [y/N]: y
Downloading Packages:
(1/6): npm-8.19.2-1.i686.rpm | 13 MB/s | 2.0 MB 00:00
(2/6): libuv-1.42.1-1.i686.rpm | 1.6 MB/s | 99 kB 00:00
(3/6): libbrotli-1.0.0-4.i686.rpm | 1.2 MB/s | 315 kB 00:00
(4/6): nodejs-full-18.12.1-1.i686.rpm | 18 MB/s | 8.2 MB 00:00
(5/6): nodejs-l10n-18.12.1-1.i686.rpm | 22 MB/s | 14 MB 00:00
(6/6): nodejs-docs-18.12.1-1.i686.rpm | 14 MB/s | 7.2 MB 00:00
Total
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing
Installing : nodejs-docs-18.12.1-1.i686.rpm
1/1
1/6
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
```

- Connect EC2 instance through local machine using MobaXterm and follow the below commands:

1. Sudo yum update -y
2. Sudo yum Install docker -y
3. Sudo service docker start

4. Sudo usermod -a -G docker ec2-user
5. Sudo chmod 777 /var/run/docker.sock
6. Sudo yum install nodejs
7. Npm init -y
8. Npm Install [mongodb@5.3](#)
9. Node index.js

```

3236232.165 (ec2-user)
Terminal Sessions View Xserver Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect... [ 2. 3.236.232.165 (ec2-user) ] [ 3. 3.236.232.165 (ec2-user) ]
git> git clone https://github.com/Mani10101/Capstone.git
Cloning into 'Capstone'...
remote: Enumerating objects: 7442, done.
remote: Counting objects: 100% (7442/7442), done.
remote: Compressing objects: 100% (6331/6331), done.
remote: Total 7442 (delta 836), pack-reused 0
Receiving objects: 100% (7442/7442), 9.01 MiB | 19.34 MiB/s, done.
[ec2-user@ip-10-0-0-74 ~]$ ls
Capstone
[ec2-user@ip-10-0-0-74 ~]$ cd Capstone
[ec2-user@ip-10-0-0-74 Capstone]$ cd server
[ec2-user@ip-10-0-0-74 server]$ ls
Models index.js node_modules package-lock.json package.json uploads
[ec2-user@ip-10-0-0-74 server]$ vum index.js
[ec2-user@ip-10-0-0-74 server]$ npm install nodejs
[ec2-user@ip-10-0-0-74 server]$ sudo yum install nodejs
Last metadata expiration check: 0:19:18 ago on Sun Apr 30 06:48:01 2023.
Dependencies resolved.
=====
Package           Architecture      Version          Repository      Size
=====
Installing:
nodejs            x86_64          1:18.12.1-1.amzn2023.0.3      amazonlinux   99 k
         依存関係:
libbrotli          x86_64          1:0.9-4.amzn2023.0.2      amazonlinux   315 k
nodejs-libs        x86_64          1:18.12.1-1.amzn2023.0.3      amazonlinux   14 M
Installing weak dependencies:
nodejs-docs        noarch         1:18.12.1-1.amzn2023.0.3      amazonlinux   7.2 M
nodejs-full-i18n   x86_64          1:18.12.1-1.amzn2023.0.3      amazonlinux   8.2 M
npm               x86_64          1:8.19.2-1.18.12.1.1.amzn2023.0.3      amazonlinux   2.0 M
=====
Transaction Summary
=====
Install 6 Packages
Total download size: 31 M
Show desktop
UNREGISTERED VERSION - Please support MobaTerm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
29°C Party sunny Search EN IN 13:01 30-04-2023

```

```

3236232.165 (ec2-user)
Terminal Sessions View Xserver Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect... [ 2. 3.236.232.165 (ec2-user) ] [ 3. 3.236.232.165 (ec2-user) ]
[ec2-user@ip-10-0-0-74 ~]$ cat > index.js <<EOF
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cloudinary": "^1.36.1",
    "cors": "2.8.5",
    "dotenv": "16.0.3",
    "express": "4.18.2",
    "mongoose": "7.0.4",
    "multer": "1.4.5-lts.1",
    "nodemon": "2.0.22"
  },
  "devDependencies": {},
  "description": ""
}

[ec2-user@ip-10-0-0-74 server]$ npm install mongodb@5.3
npm WARN old lockfile
npm WARN old lockfile The package-lock.json file was created with an old version of npm,
npm WARN old lockfile so supplemental metadata must be fetched from the registry.
npm WARN old lockfile This is a one-time fix-up, please be patient...
npm WARN old lockfile

added 1 package, changed 1 package, and audited 189 packages in 10s
13 packages are looking for funding
  run npmtodo for details
found 0 vulnerabilities
npm notice New major version of npm available! 8.19.2 → 9.6.5
npm notice Changelog: https://github.com/npm/cli/releases/tag/v9.6.5
npm notice Run npm@9.6.5 to update!
npm notice
[ec2-user@ip-10-0-0-74 server]$ node index.js
Server listening at http://localhost:5000
Could not connect to MongoDB... MongooseServerSelectionError: Could not connect to any servers in your MongoDB Atlas cluster. One common reason is that you're trying to access the data base from an IP that isn't whitelisted. Make sure your current IP address is on your Atlas cluster's IP whitelist.
at _handleConnectionError (/home/ec2-user/Capstone/server/node_modules/mongoose/lib/connection.js:755:11)
at NativeConnection.openUri (/home/ec2-user/Capstone/server/node_modules/mongoose/lib/connection.js:730:11)
at process.processTicksAndRejections (node:internal/process/task_queues:95:5)
Show desktop
UNREGISTERED VERSION - Please support MobaTerm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
29°C Party sunny Search EN IN 13:01 30-04-2023

```

```
3.236.232.165 (ec2-user)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect... 3.236.232.165 (ec2-user) 3.236.232.165 (ec2-user)
X server Exit

[{"$": "star"}, {"$": "star"}, {"$": "star"}, {"$": "star"}, {"$": "star"}]
heatbeatFrequencyMS: 10000,
localThresholdMS: 15,
setName: 'atlas-1g87jy-shard-0',
maxElectionId: null,
maxSetVersion: null,
commonWireVersion: 0,
logicalSessionTimeoutMinutes: null
},
code: undefined
}
{
[ec2-user@ip-19-0-0-74 server]$ node index.js
Server listening at http://localhost:5080
Could not connect to MongoDB... MongooseserverSelectionError: Could not connect to any servers in your MongoDB Atlas cluster. One common reason is that you're trying to access the data base from an IP that isn't whitelisted. Make sure your current IP address is on your Atlas cluster's IP Whitelist: https://www.mongodb.com/docs/atlas/security-whitelist/
    at handleConnectionErrors (/home/ec2-user/Capstone/server/node_modules/mongoose/lib/connection.js:75:11)
    at NativeConnection.openUri (/home/ec2-user/Capstone/server/node_modules/mongoose/lib/connection.js:730:11)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)
  reason: TopologyDescription {
    type: 'ReplicasetNoPrimary',
    servers: Map(3) {
      'ac-udby8sy-lshard-00-01.sfvmu6.mongodb.net:27017' => [ServerDescription],
      'ac-udby8sy-lshard-00-02.sfvmu6.mongodb.net:27017' => [ServerDescription],
      'ac-udby8sy-lshard-00-03.sfvmu6.mongodb.net:27017' => [ServerDescription]
    },
    stale: false,
    compatible: true,
    heartbeatFrequencyMS: 10000,
    localThresholdMS: 15,
    setName: 'atlas-1g87jy-shard-0',
    maxElectionId: null,
    maxSetVersion: null,
    commonWireVersion: 0,
    logicalSessionTimeoutMinutes: null
  },
  code: undefined
}
[ec2-user@ip-19-0-0-74 server]$ node index.js
Server listening at http://localhost:5080
Connected to MongoDB...
```

STEP 7: Deploying client on EC2 using docker

- Creating EC2 instance to deploy the frontend with the help of docker.

- Follow the commands as shown:
 1. Sudo yum update -y
 2. Sudo yum Install docker -y
 3. Sudo service docker start

4. Sudo usermod -a -G docker ec2-user
5. Sudo chmod 777 /var/run/docker.sock
6. Sudo yum install git -y
7. Git clone url
8. Docker build -t frontend .
9. Docker run -d -p 3000:3000 frontend

```

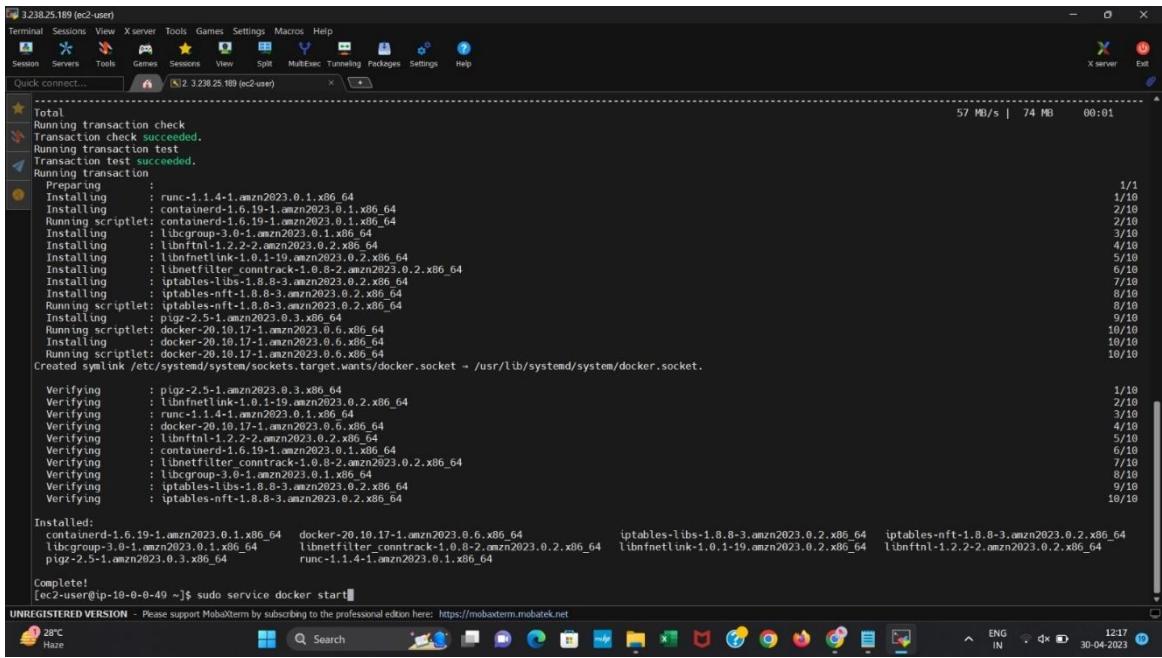
3.238.25.189 (ec2-user)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect... [ 2 3.238.25.189 (ec2-user) ] X server Exit
[ ec2-user@ip-10-0-0-49 ~]$ sudo yum update -y
Last metadata expiration check: 0:02:39 ago on Sun Apr 30 06:46:40 2023.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-10-0-0-49 ~]$ sudo yum install docker -y
Last metadata expiration check: 0:03:01 ago on Sun Apr 30 06:46:40 2023.
Dependencies resolved.
=====
Package           Architecture      Version          Repository    Size
=====
Installing:
docker            x86_64          20.10.17-1.amzn2023.0.6   amazonlinux  39 M
=====
Installing dependencies:
containerd        x86_64          1.6.19-1.amzn2023.0.1   amazonlinux  31 M
iptables          x86_64          1.8.8-3.amzn2023.0.2   amazonlinux  401 k
iptables-nft     x86_64          1.8.8-3.amzn2023.0.2   amazonlinux  163 k
libcgroup         x86_64          3.0.3-1.amzn2023.0.1   amazonlinux  75 K
libnftnl-filter   x86_64          1.0.8-2.amzn2023.0.2   amazonlinux  58 K
libnftnl          x86_64          1.0.1-19.amzn2023.0.2  amazonlinux  30 k
libnftnl          x86_64          1.2.2-2.amzn2023.0.2  amazonlinux  84 k
pigz              x86_64          2.5-1.amzn2023.0.3   amazonlinux  83 k
runc              x86_64          1.1.4-1.amzn2023.0.1  amazonlinux  3.1 M
=====
Transaction Summary
Install 10 Packages
Total download size: 74 M
Installed size: 287 M
UNREGISTERED VERSION - Please support MobaTerm by subscribing to the professional edition here: https://mobaterm.mobatek.net
28°C Maze Search ENG IN 12:16 30-04-2023

```

```

3.238.25.189 (ec2-user)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect... [ 2 3.238.25.189 (ec2-user) ] X server Exit
[ ec2-user@ip-10-0-0-49 ~]$ sudo yum install docker -y
Last metadata expiration check: 0:03:01 ago on Sun Apr 30 06:46:40 2023.
Dependencies resolved.
=====
Package           Architecture      Version          Repository    Size
=====
Installing:
docker            x86_64          20.10.17-1.amzn2023.0.6   amazonlinux  39 M
=====
Installing dependencies:
containerd        x86_64          1.6.19-1.amzn2023.0.1   amazonlinux  31 M
iptables          x86_64          1.8.8-3.amzn2023.0.2   amazonlinux  401 k
iptables-nft     x86_64          1.8.8-3.amzn2023.0.2   amazonlinux  163 k
libcgroup         x86_64          3.0.3-1.amzn2023.0.1   amazonlinux  75 K
libnftnl-filter   x86_64          1.0.8-2.amzn2023.0.2   amazonlinux  58 K
libnftnl          x86_64          1.0.1-19.amzn2023.0.2  amazonlinux  30 k
libnftnl          x86_64          1.2.2-2.amzn2023.0.2  amazonlinux  84 k
pigz              x86_64          2.5-1.amzn2023.0.3   amazonlinux  83 k
runc              x86_64          1.1.4-1.amzn2023.0.1  amazonlinux  3.1 M
=====
Transaction Summary
Install 10 Packages
Total download size: 74 M
Installed size: 287 M
Downloading Packages:
(1/10): libnftnl-link-1.0.1-19.amzn2023.0.2.x86_64.rpm 445 kB/s | 39 kB 00:00
(2/10): libcgroup-3.0.1-0.3.x86_64.rpm 683 kB/s | 83 kB 00:00
(3/10): runc-1.1.4-1.amzn2023.0.1.x86_64.rpm 16 MB/s | 3.1 MB 00:00
(4/10): libnftnl-1.2.2-2.amzn2023.0.2.x86_64.rpm 1.2 MB/s | 84 kB 00:00
(5/10): libnftnl-filter-comptrack-1.0.8-2.amzn2023.0.2.x86_64.rpm 1.9 MB/s | 58 kB 00:00
(6/10): libcgroup-3.0-1.amzn2023.0.1.x86_64.rpm 1.7 MB/s | 75 kB 00:00
(7/10): iptables-lib-1.8-3.amzn2023.0.2.x86_64.rpm 6.4 MB/s | 481 kB 00:00
(8/10): iptables-1.8-3.amzn2023.0.2.x86_64.rpm 2.3 MB/s | 187 kB 00:00
(9/10): docker-20.10.17-1.amzn2023.0.6.x86_64.rpm 41 kB/s | 39 kB 00:00
(10/10): containerd-1.6.19-1.amzn2023.0.1.x86_64.rpm 30 MB/s | 31 MB 00:01
=====
Total
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing :
  Installing : runc-1.1.4-1.amzn2023.0.1.x86_64 1/10
  Installing : containerd-1.6.19-1.amzn2023.0.1.x86_64 2/10
  Running scriptlet: containerd-1.6.19-1.amzn2023.0.1.x86_64 2/10
  Installing : libcgroup-3.0-1.amzn2023.0.1.x86_64 3/10
  Installing : libnftnl-1.2.2-2.amzn2023.0.2.x86_64 4/10
  Installing : libnftnl-filter-1.0.1-19.amzn2023.0.2.x86_64 5/10
  Installing : libnftnl-link-1.0.8-2.amzn2023.0.2.x86_64 6/10
  Installing : iptables-lib-1.8-3.amzn2023.0.2.x86_64 7/10
  Installing : iptables-1.8-3.amzn2023.0.2.x86_64 8/10
  Running scriptlet: iptables-1.8-3.amzn2023.0.2.x86_64 8/10
  Installing : pigz-2.5-1.amzn2023.0.3.x86_64 9/10
  Running scriptlet: docker-20.10.17-1.amzn2023.0.6.x86_64 10/10
  Installing : docker-20.10.17-1.amzn2023.0.6.x86_64 [=====] 10/10
=====
UNREGISTERED VERSION - Please support MobaTerm by subscribing to the professional edition here: https://mobaterm.mobatek.net
28°C Maze Search ENG IN 12:16 30-04-2023

```



```

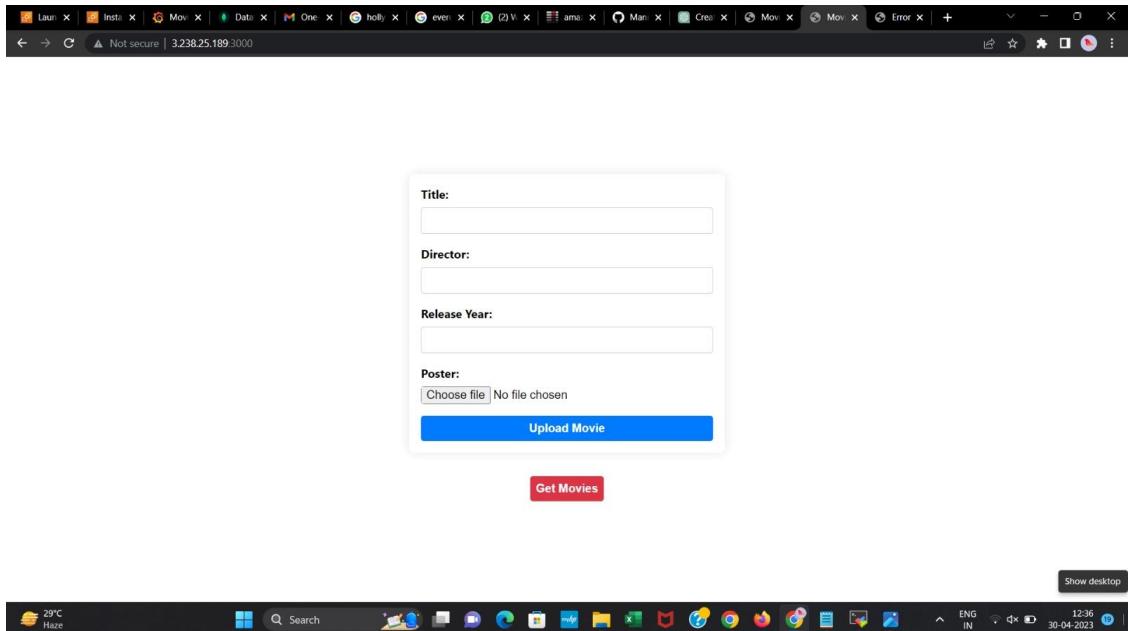
3.238.25.189 (ec2-user)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect... 3.238.25.189 (ec2-user) 57 MB/s | 74 MB 00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : containerd-1.6.19-1.amzn2023.0.1.x86_64 2/10
Running scriptlet:containerd-1.6.19-1.amzn2023.0.1.x86_64
Installing : libcgroup-3.0-1.amzn2023.0.1.x86_64 3/10
Installing : libnftnl-1.2.2-2.amzn2023.0.2.x86_64 4/10
Installing : libnftnl-link-1.0.1-19.amzn2023.0.2.x86_64 5/10
Installing : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64 6/10
Installing : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64 7/10
Installing : libnftnl-link-1.0.1-19.amzn2023.0.2.x86_64 8/10
Running scriptlet:iptables-nft-1.0.8-3.amzn2023.0.2.x86_64
Installing : pigz-2.5-1.amzn2023.0.3.x86_64 9/10
Running scriptlet:docker-20.10.17-1.amzn2023.0.6.x86_64 10/10
Installing : docker-20.10.17-1.amzn2023.0.6.x86_64 10/10
Running scriptlet:docker-20.10.17-1.amzn2023.0.6.x86_64 10/10
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket -> /usr/lib/systemd/system/docker.socket.

Verifying : 1/18
Verifying : libnftnl-link-1.0.1-19.amzn2023.0.2.x86_64 2/18
Verifying : runc-1.1.4-4.amzn2023.0.1.x86_64 3/10
Verifying : docker-20.10.17-1.amzn2023.0.6.x86_64 4/10
Verifying : libnftnl-1.2.2-2.amzn2023.0.2.x86_64 5/10
Verifying : containerd-1.6.19-1.amzn2023.0.1.x86_64 6/10
Verifying : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64 7/10
Verifying : libnftnl-link-1.0.1-19.amzn2023.0.2.x86_64 8/10
Verifying : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64 9/10
Verifying : iptables-nft-1.0.8-3.amzn2023.0.2.x86_64 10/10

Installed:
containerd-1.6.19-1.amzn2023.0.1.x86_64 docker-20.10.17-1.amzn2023.0.6.x86_64 iptables-libs-1.8.8-3.amzn2023.0.2.x86_64 iptables-nft-1.0.8-3.amzn2023.0.2.x86_64
libcgroup-3.0-1.amzn2023.0.1.x86_64 libnftnlfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64 libnftnl-link-1.0.1-19.amzn2023.0.2.x86_64 libnftnl-1.2.2-2.amzn2023.0.2.x86_64
pigz-2.5-1.amzn2023.0.3.x86_64 runc-1.1.4-4.amzn2023.0.1.x86_64

Complete!
[ec2-user@ip-10-0-0-49 ~]$ sudo service docker start
UNREGISTERED VERSION - Please support Mobaxterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
```

- After deploying to check whether the website we the public IP address of instance and port number.



The screenshot shows a web browser window with multiple tabs open. The active tab displays a form for uploading a movie. The form fields include:

- Title: (input field)
- Director: (input field)
- Release Year: (input field)
- Poster: (input field labeled "Choose file") with a note "No file chosen".
- Upload Movie: (blue button)
- Get Movies: (red button)

The browser's status bar at the bottom shows the date and time as 30-04-2023 12:17.

- The web page has been deployed, then enter the details of the given parameters and then your data is been stored in database and your images are stored in S3 bucket.

The screenshot shows the MongoDB Atlas Data Services interface. On the left, the sidebar includes sections for Deployment, Database (with 'test' selected), and Services. The main area displays the 'test.movies' collection with the following details:

- STORAGE SIZE: 36KB
- LOGICAL DATA SIZE: 568B
- TOTAL DOCUMENTS: 3
- INDEXES TOTAL SIZE: 36KB

The 'Find' tab is active, showing a query input field: "Type a query: { field: 'value' }". Below it, the results are listed:

```
_id: ObjectId('6447971ddae5f7b6929f8ee9')
title: "THE EXORCIST"
director: "William Friedkin"
releaseYear: 2023
poster: "https://res.cloudinary.com/dcc6gy3wy/image/upload/v1682413527/u0j2ejgy-"
__v: 0

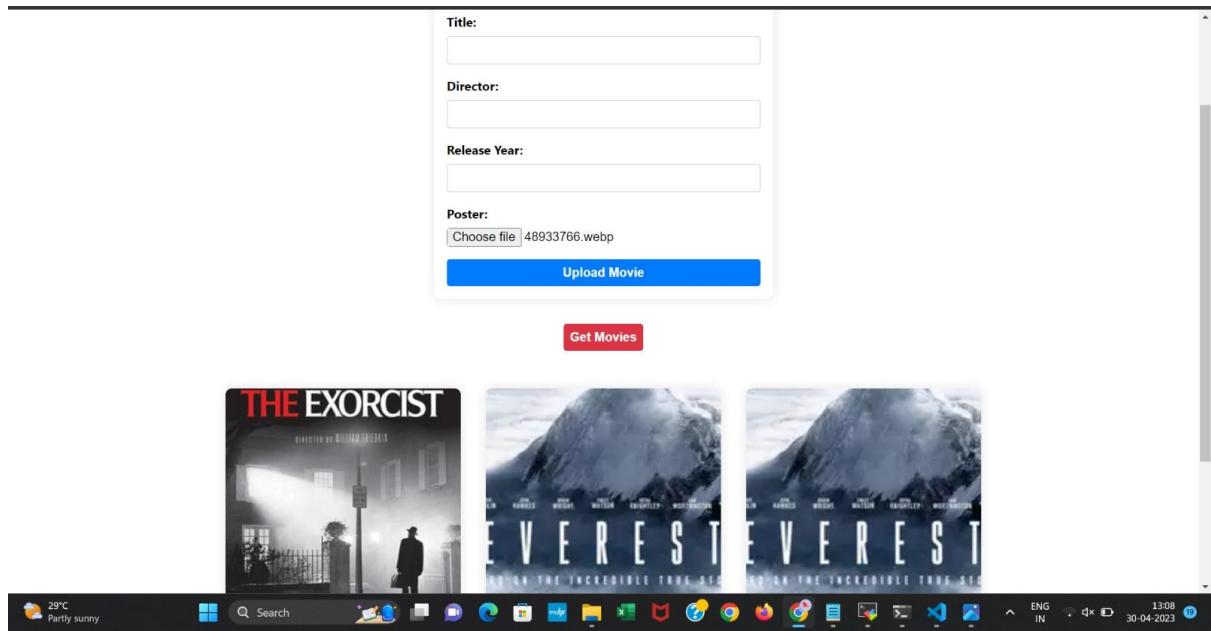
_id: ObjectId('644de4fc134767a83895493c')
title: "Everest"
director: "Baltasar Kormákur"
releaseYear: 2015
poster: "https://projectbuck-1.s3.amazonaws.com/1682826491250-48933766.webp"
__v: 0
```

The screenshot shows the AWS S3 console. The left sidebar includes sections for Buckets, Storage Lens, and AWS Marketplace for S3. The main area shows the 'projectbuck-1' bucket with the following details:

Objects (2)

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

Name	Type	Last modified	Size	Storage class
1682826491250-48933766.webp	webp	April 30, 2023, 09:21:22 (UTC+05:30)	4.3 KB	Standard
1682839299655-48933766.webp	webp	April 30, 2023, 12:54:50 (UTC+05:30)	4.3 KB	Standard

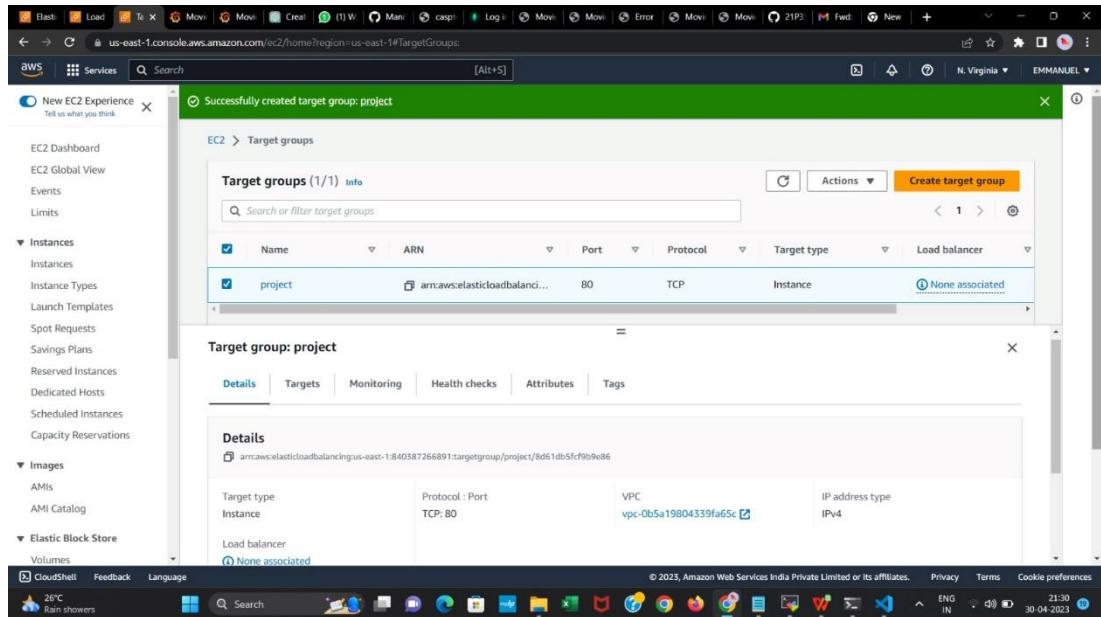


STEP8: Creation of target groups and load balancer

Load balancer: Load balancing is the process of distributing network traffic across multiple servers to ensure that no single server is overloaded with too much traffic. This helps to improve the overall performance, availability, and scalability of the application or service.

Target groups: Target groups are a feature of load balancers in Amazon Web Services (AWS) that enable traffic to be directed to a set of instances that are registered with the target group. A target group can contain one or more instances, and multiple target groups can be associated with a single load balancer.

FRONTEND:



EC2 > Load balancers

Load balancers (1/1)

Name	DNS name	Status	VPC ID	Availability Zones	Type	Date created
capstone	capstone-4cc4f98e14f7695d.elb.us-east-1.amazonaws.com (A Record)	Active	vpc-0b5a19804339fa65c	us-east-1f (use1-az5)	network	April 30, 2023 (UTC+05:30)

Load balancer: capstone

Details

Load balancer type	DNS name	Status	VPC
Network	capstone-4cc4f98e14f7695d.elb.us-east-1.amazonaws.com (A Record)	Active	vpc-0b5a19804339fa65c
IP address type	Scheme	Availability Zones	Hosted zone

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

EC2 > Target groups > Create target group

Step 1
Specify group details

Step 2
Register targets

Register targets

This is an optional step to create a target group. However, to ensure that your load balancer routes traffic to this target group you must register your targets.

Available instances (2/4)

Instance ID	Name	Status	Security groups	Zone	Subnet ID
i-049b2c3d2518def65	frontend	Running	launch-wizard-1	us-east-1f	subnet-09e2d790f0a4afdf6f
i-0394cf72dd29b0992	Backend	Running	launch-wizard-2	us-east-1f	subnet-09e2d790f0a4afdf6f
i-06904d8d23d1c949b	frontend1	Running	launch-wizard-1	us-east-1f	subnet-09e2d790f0a4afdf6f
i-0ed32052a31d6be63	backend1	Running	launch-wizard-2	us-east-1f	subnet-09e2d790f0a4afdf6f

2 selected

Ports for the selected instances
Ports for routing traffic to the selected instances.

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

BACKEND:

The screenshot shows the AWS EC2 Target Groups console. A success message at the top says "Successfully created target group: TG1". The main table shows one target group named "TG1" with ARN "arn:aws:elasticloadbalancing:us-east-1:0b5a19804339fa65c", port 80, protocol HTTP, target type Instance, and no load balancer associated. Below the table, a modal window titled "Target group: TG1" shows the registered targets: "frontend1" and "frontend". Both targets are listed with Instance ID, Name, Port, Zone, and Health status (unused). The status details indicate they are not configured to receive traffic from the load balancer.

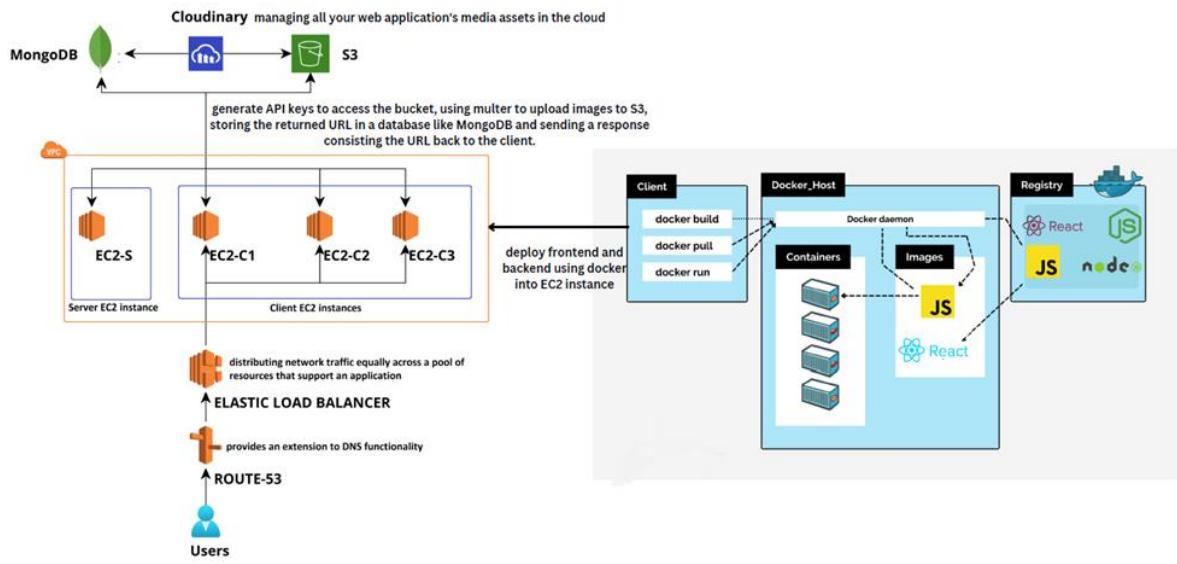
The screenshot shows the AWS EC2 Instances console. A modal window titled "Create target group" is open, showing a list of instances: "frontend", "Backend", "frontend1", and "backend1". The instances "Backend" and "backend1" are selected, indicated by a checked checkbox. Below the list, a section titled "Ports for the selected instances" shows the port number "80" entered into a text input field. At the bottom of the modal, there is a "Review targets" button.

- **DNS (Domain Name System):** is a system used to translate human-readable domain names into IP addresses that computers can understand.

LINK: <http://Casptone-ac1c7a45280f6a36.elb.us-east1.amazonaws.com>

GIT LINK: https://github.com/21P35A0347/Capstone_project.git

AWS DEPLOYMENT ARCHETECHTURE



METHODS TO IMPROVE AWS DEPLOYMENT

- Use Infrastructure as Code (IaC): Use IaC tools like AWS CloudFormation or AWS Terraform to define and deploy infrastructure. This allows you to define your infrastructure in code and manage it like any other software application.
- Automate deployment: Use continuous integration and continuous deployment (CI/CD) pipelines to automate your deployment process. This ensures that code changes are automatically deployed to production after passing a series of automated tests.
- Implement scalability: Design your infrastructure to scale horizontally or vertically based on usage patterns. This can be achieved by using AWS Auto Scaling groups, which automatically adjust the number of instances running based on demand.
- Use AWS best practices: Follow AWS best practices to ensure your infrastructure is secure, highly available, and cost-effective. This includes using AWS services like AWS Identity and Access Management (IAM) for access control, Amazon Elastic Compute Cloud (EC2) for compute resources, and Amazon Relational Database Service (RDS) for databases.
- Monitor performance: Use AWS CloudWatch to monitor the performance of your infrastructure and applications. This allows you to

identify and troubleshoot performance issues before they impact your users.

- Use AWS Managed Services: Consider using AWS Managed Services like Amazon Elastic Kubernetes Service (EKS), AWS Lambda, or Amazon Aurora to reduce the operational overhead of managing infrastructure.
- Ensure Disaster Recovery: Implement a disaster recovery plan that includes regular backups and replication of your infrastructure and data to another AWS region or third-party provider.

By implementing these methods, you can improve the reliability, scalability, security, and performance of your AWS deployment.

Connecting the server with atlas Mongo DB	EMMANUEL, NEELIMA
Creating IAM user and S3	BHAVYA, PRIYANKA
Configuration the Application Code with S3-multer	VIJAYA, HARI PRIYANKA
Configuration of the code using Docker file	NITHYA SRI, FREINEY
Deploying server on EC2 using Docker	BHAVYA, PRIYANKA
Creation a target group and Load Balancer	EMMANUEL, NEELIMA
Creating AWS Deployment Diagram	VIJAYA, HARI PRIYANKA, NITHYA SRI, FREINEY
Preparing Project Documentation	VIJAYA, HARI PRIYANKA, NITHYA SRI, FREINEY, EMMANUEL, NEELIMA, BHAVYA, PRIYANKA.

CONCLUSION

In conclusion, deploying a movie listing website to the cloud infrastructure using AWS can provide numerous benefits such as scalability, reliability, and availability. The project involved using ReactJS as the frontend, NodeJS as the backend, and MongoDB as the database, with Multer being used for file uploads.

The project scope included deploying the application to the cloud using AWS services like EC2, S3, IAM, and Route53. Additionally, load balancing was implemented using target groups to distribute traffic across instances.

Overall, this project demonstrates how deploying a web application to the cloud infrastructure can provide numerous benefits and is an essential skill for modern software development.