DevOps Capstone Project

DEPLOYING A MOVIE LISTING WEBSITE USING AWS CLOUD

Team no: 5

Team Members:

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

ABOUT THE PROJECT:

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

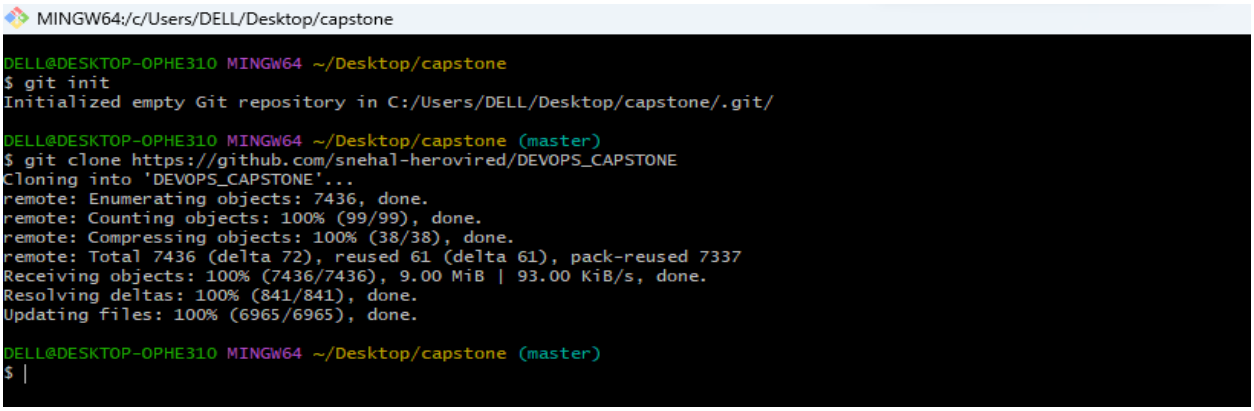
The deployment process involves several steps, including replacing local storage with S3, migrating the database to MongoDB Atlas, deploying the backend in an EC2 instance using Docker, modifying the frontend code to fetch data from the backend, deploying the frontend using Docker into an EC2 instance,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

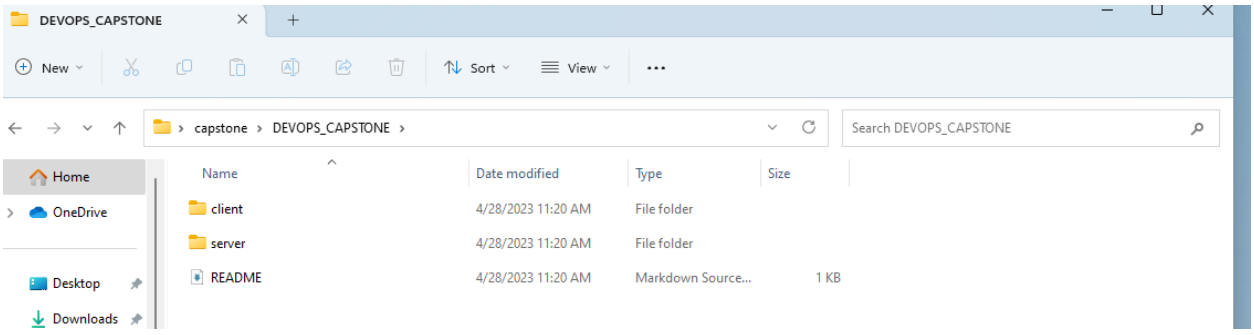
The deployment on AWS ensures that the website is scalable and can handle increasing traffic. The use of an EC2 instance and load balancer ensures that traffic is distributed among multiple instances, and the use of MongoDB Atlas ensures that the database is scalable and highly available. The use of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

TOOLS AND TECHNOLOGIES USED IN THIS PROJECT:

IMPLEMENTATION

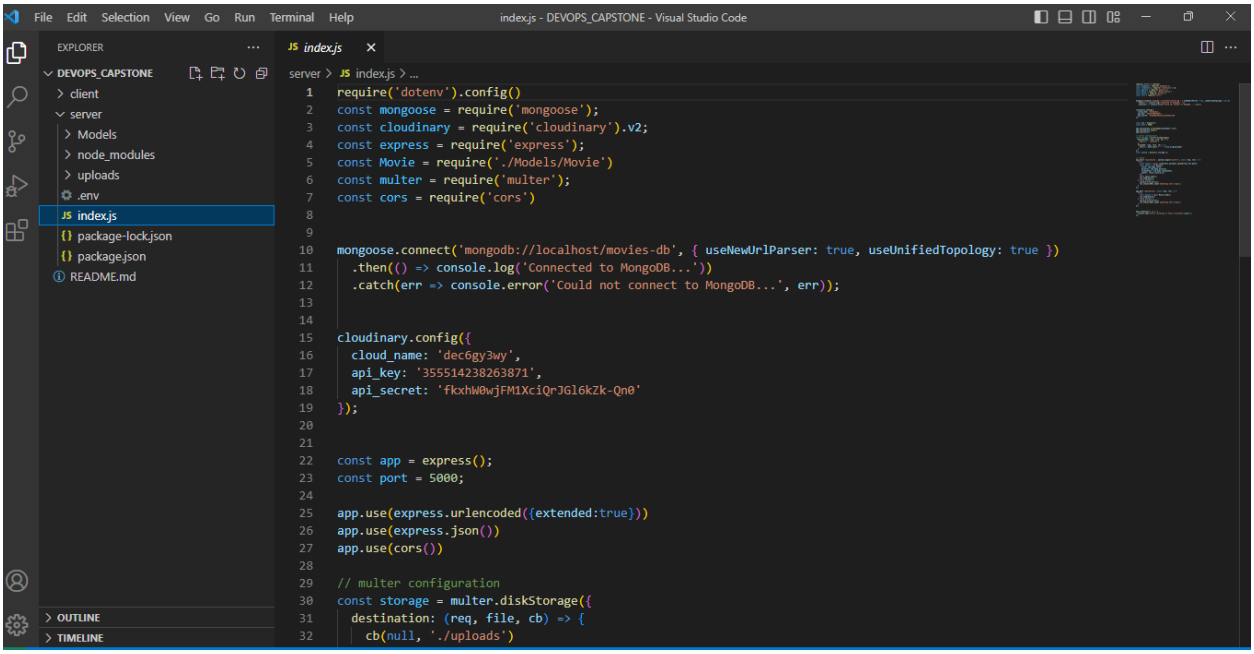
STEP 1: clone the GitHub repository provided by initiating a new repository by using **git init** and clone it by using **git clone <link>.**





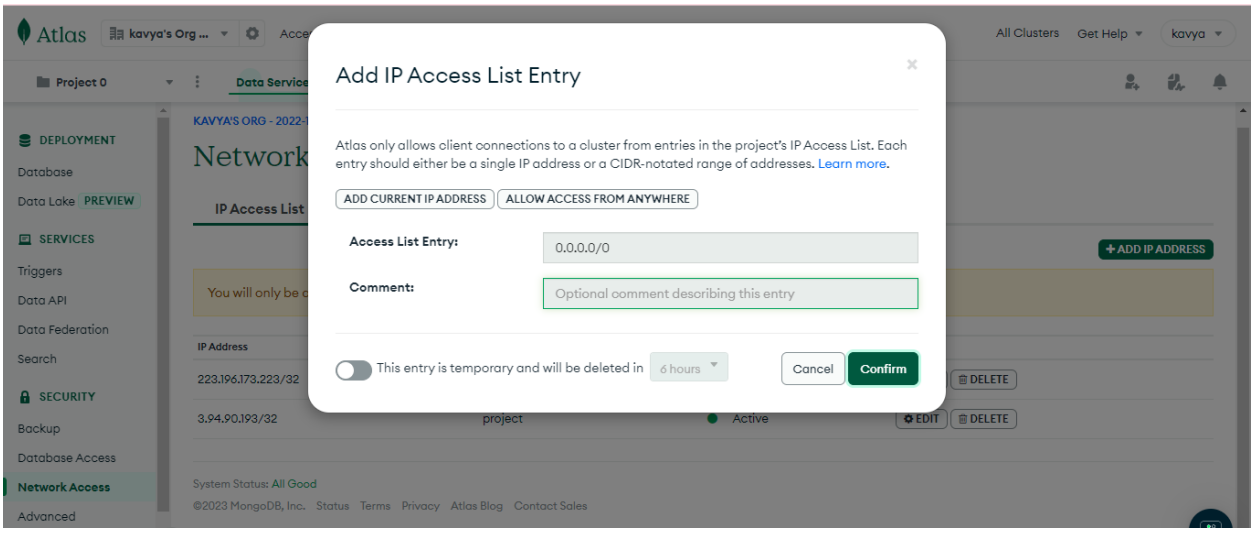
STEP 2: we must replace local database with Atlas MongoDB cloud infrastructure to take the database into the cloud. For that, we create Atlas MongoDB account and create a new free-tier cluster.

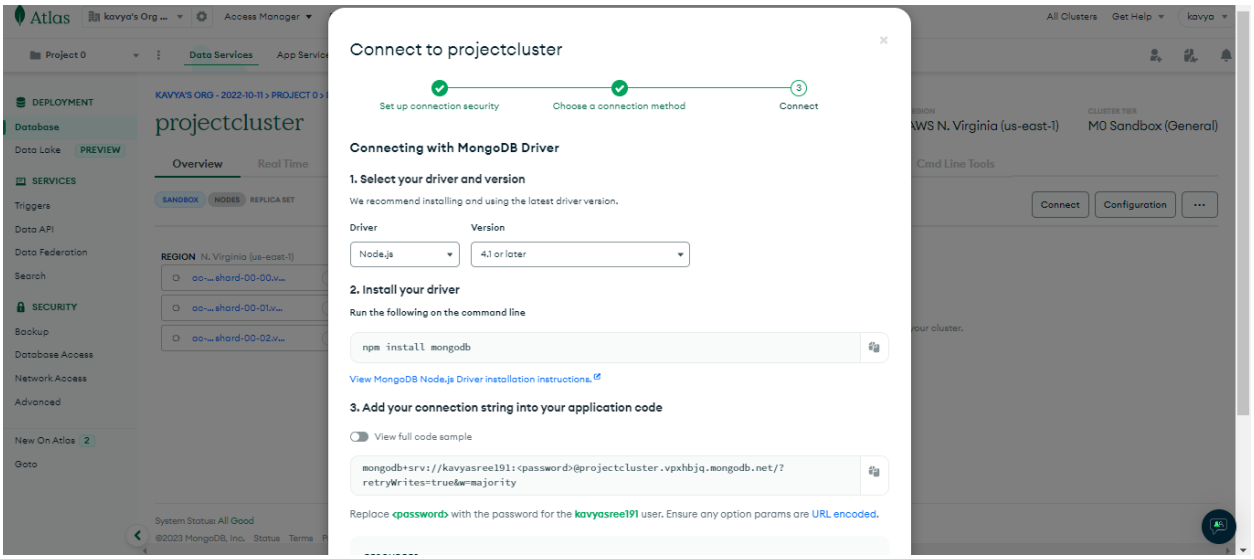
Before changing the code



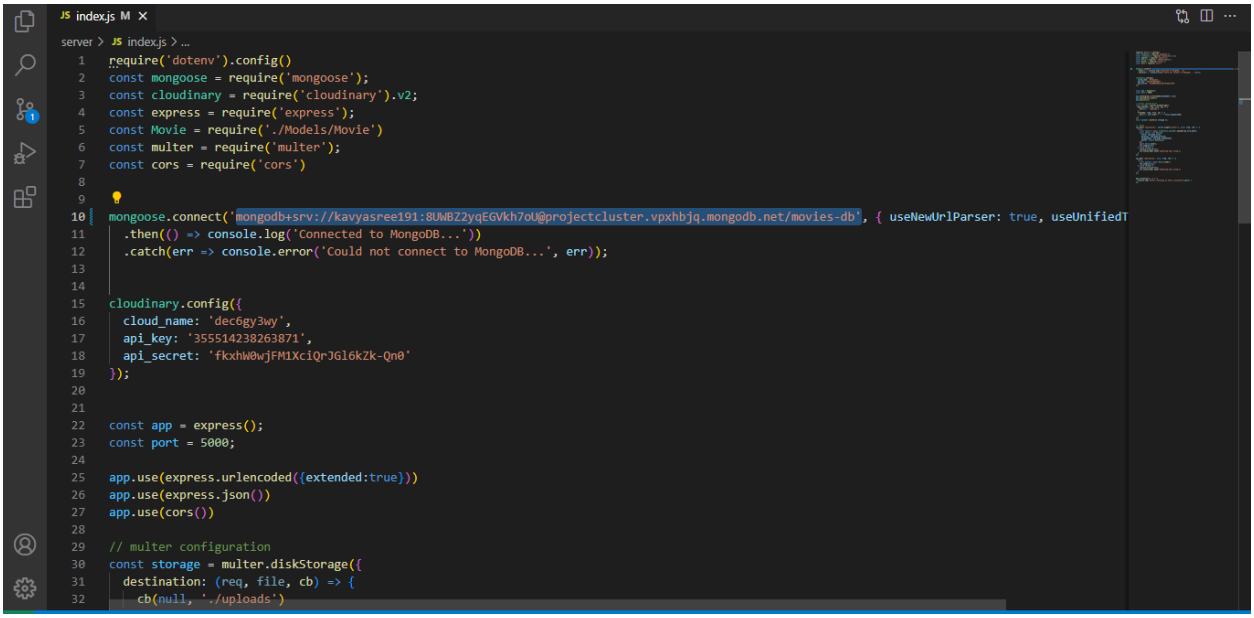
Give the network access (instance IP) to allow access and connect it by selecting your driver and version. Copy the connection string provided.

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

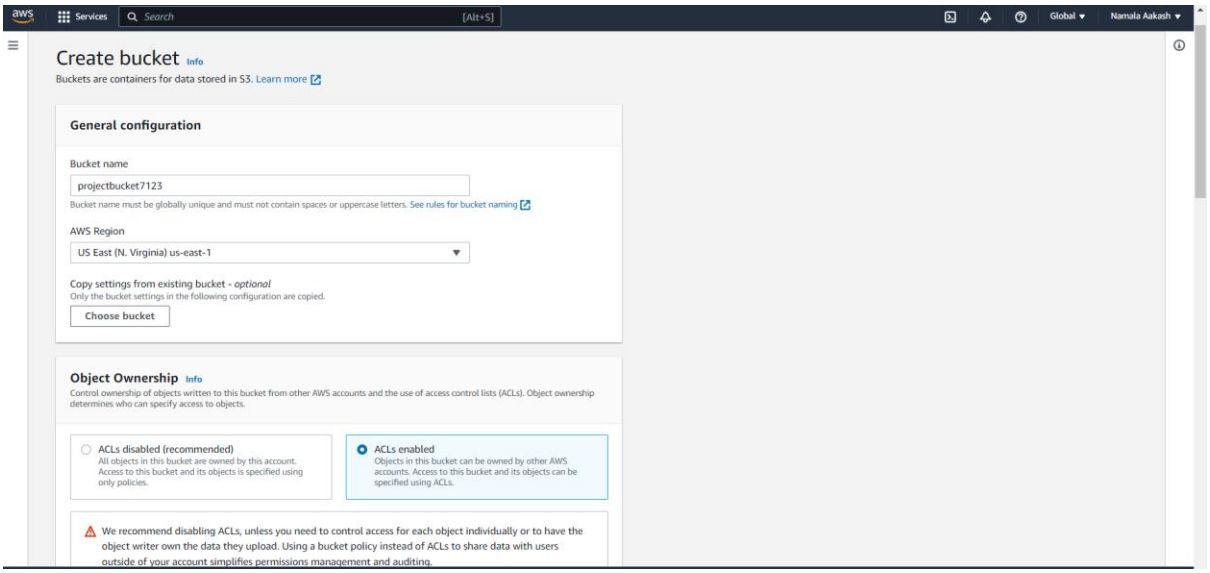


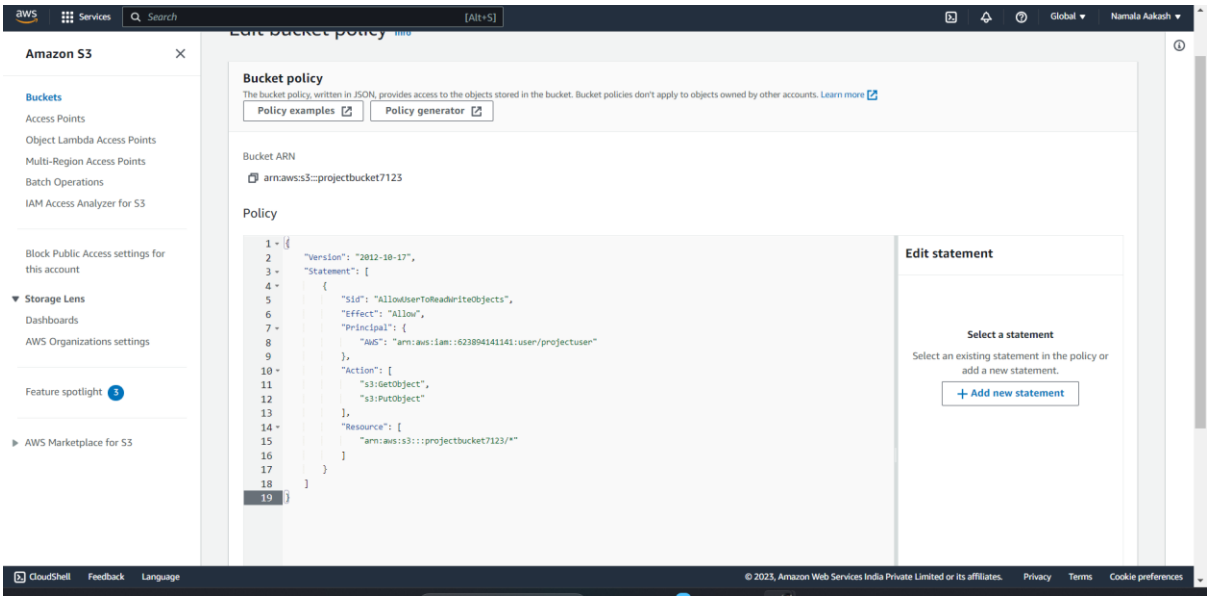


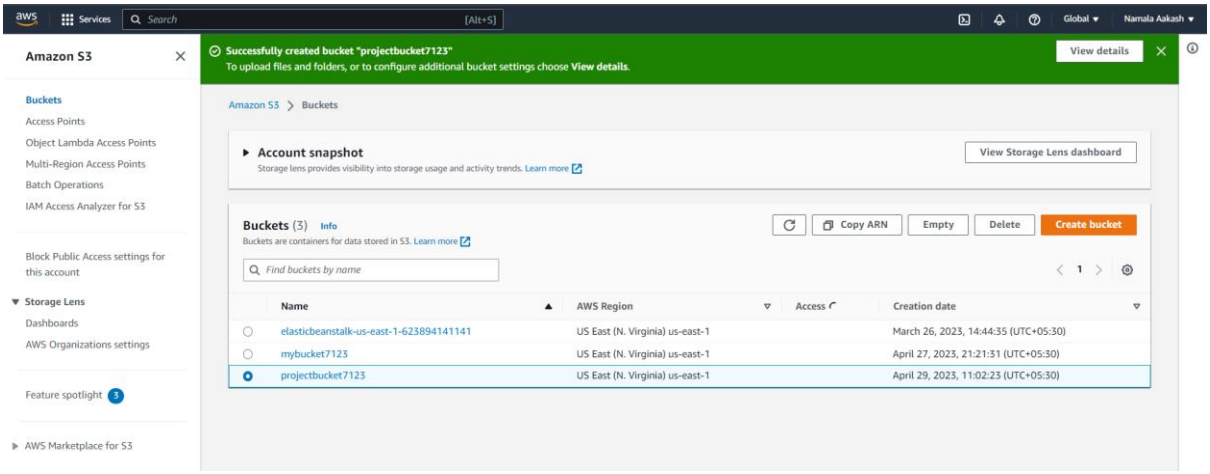
mongodb+srv://kavyasree191:<password>@projectcluster.vpxhbjq.mongodb.net/?retryWrites=true&w=majority



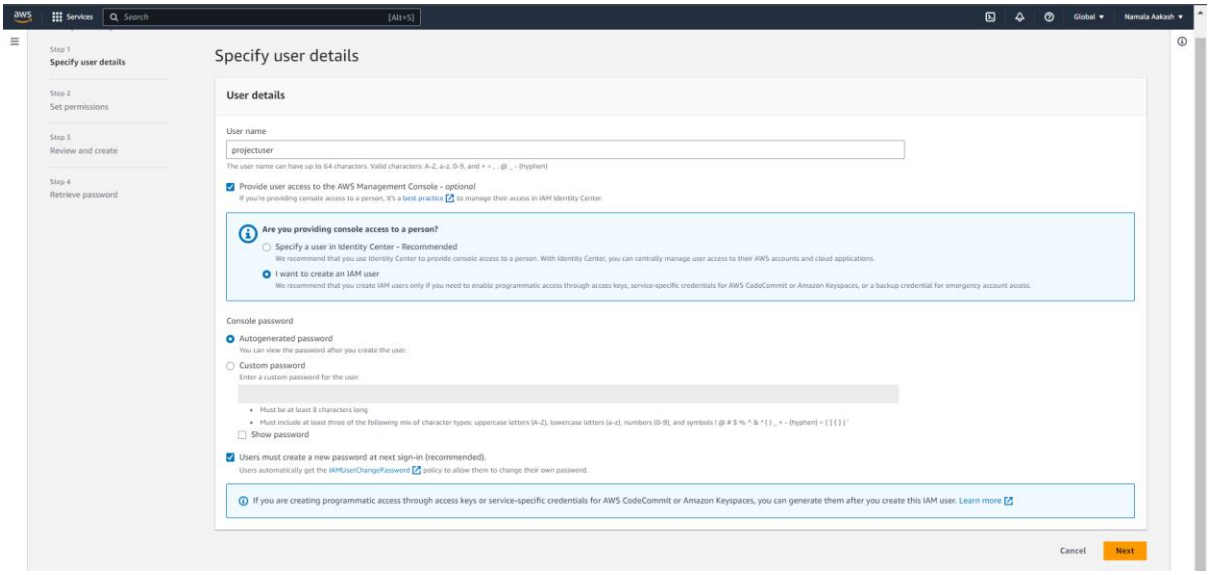
STEP 3: create a S3 bucket to store the images uploaded by the user. Create the bucket by giving necessary policy.

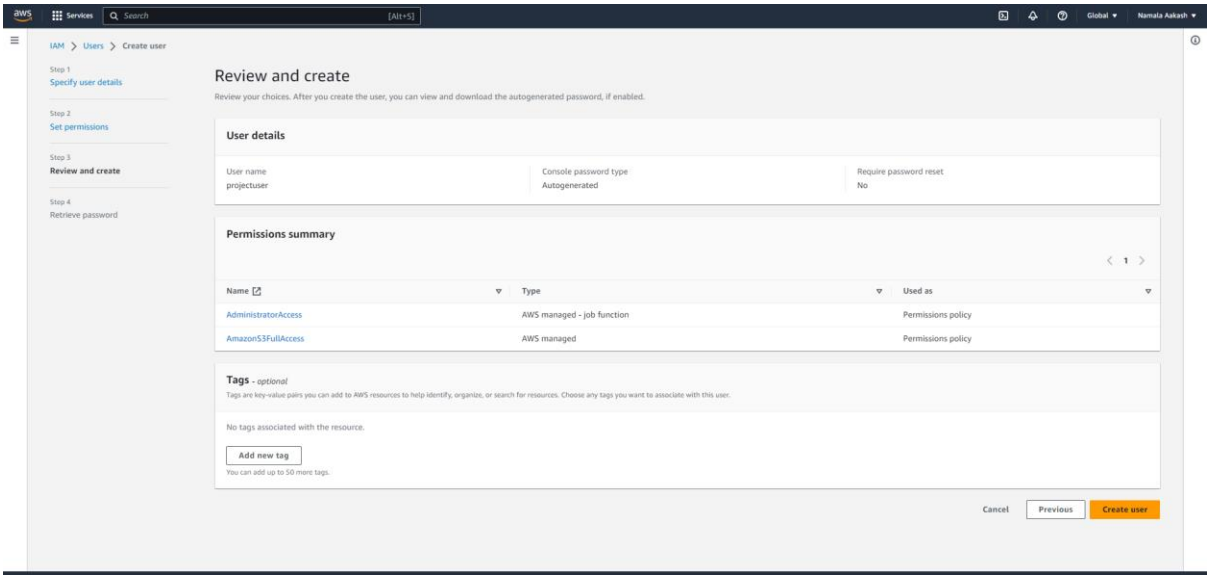

Services
Search
[Alt+S]
Global
Namala Aakash
Create bucket Info
Buckets are containers for data stored in S3. Learn more
General configuration
Bucket name
projectbucket7123
Bucket name must be globally unique and must not contain spaces or uppercase letters. See rules for bucket naming
AWS Region
US East (N. Virginia) us-east-1
Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.
Choose bucket
Object Ownership Info
Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.
ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.
We recommend disabling ACLs, unless you need to control access for each object individually or to have the object writer own the data they upload. Using a bucket policy instead of ACLs to share data with users outside of your account simplifies permissions management and auditing.


Services
Search
[Alt+S]
Global
Namala Aakash
Amazon S3
Buckets
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
AWS Organizations settings
Feature spotlight 3
AWS Marketplace for S3
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
Policy examples
Policy generator
Bucket ARN
arn:aws:s3:::projectbucket7123
Policy
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "AllowUserToReadWriteObjects",
"Effect": "Allow",
"Principal": {
"AWS": "arn:aws:iam::623894141141:user/projectuser"
},
"Action": [
"s3:GetObject",
"s3:PutObject"
],
"Resource": [
"arn:aws:s3:::projectbucket7123/*"
]
}
]
}
Edit statement
Select a statement
Select an existing statement in the policy or add a new statement.
+ Add new statement
CloudShell
Feedback
Language
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences



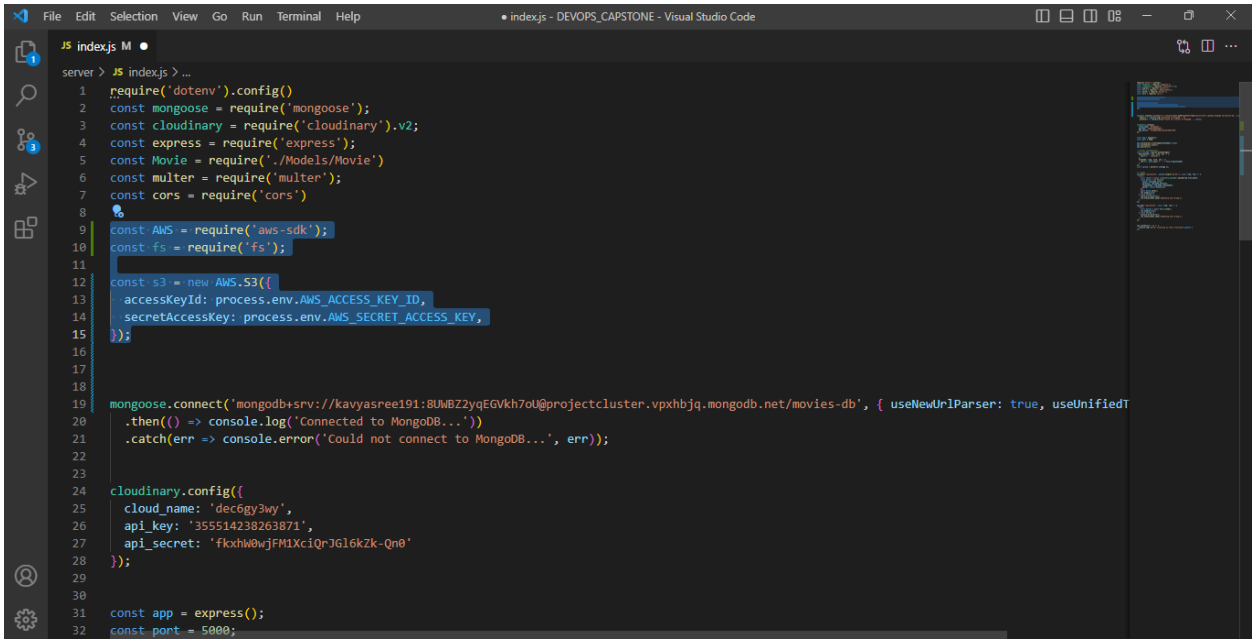
STEP 4: create an IAM user by giving administration access and S3FullAccess generate an access key.

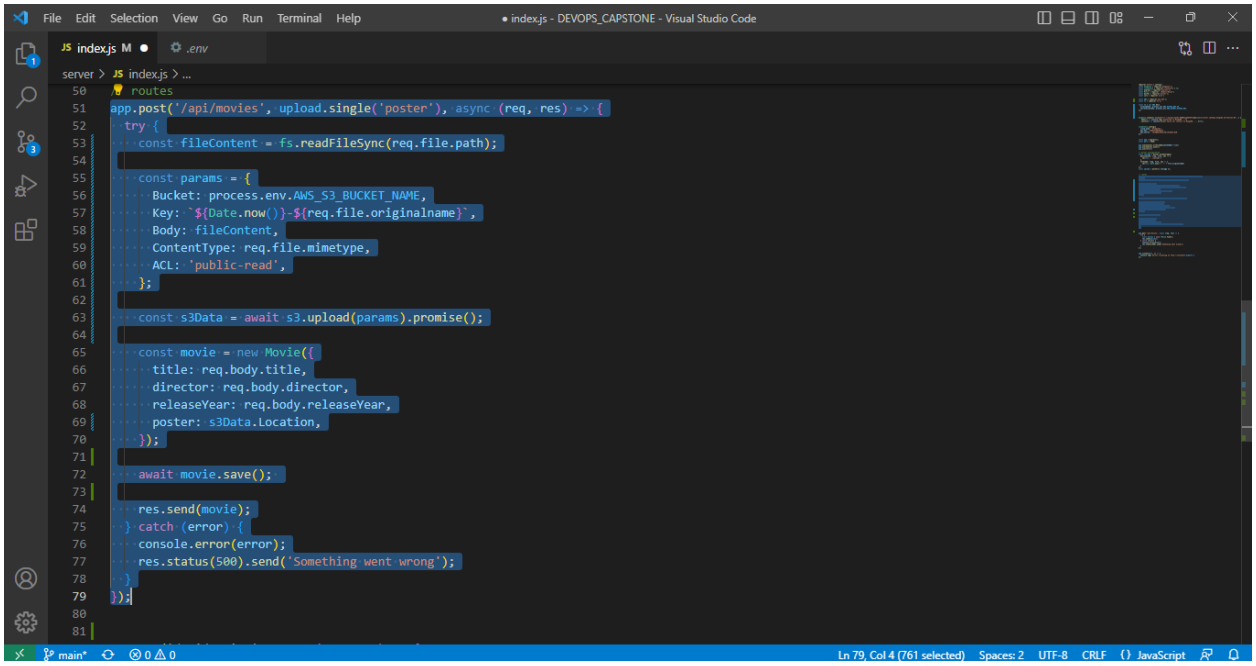




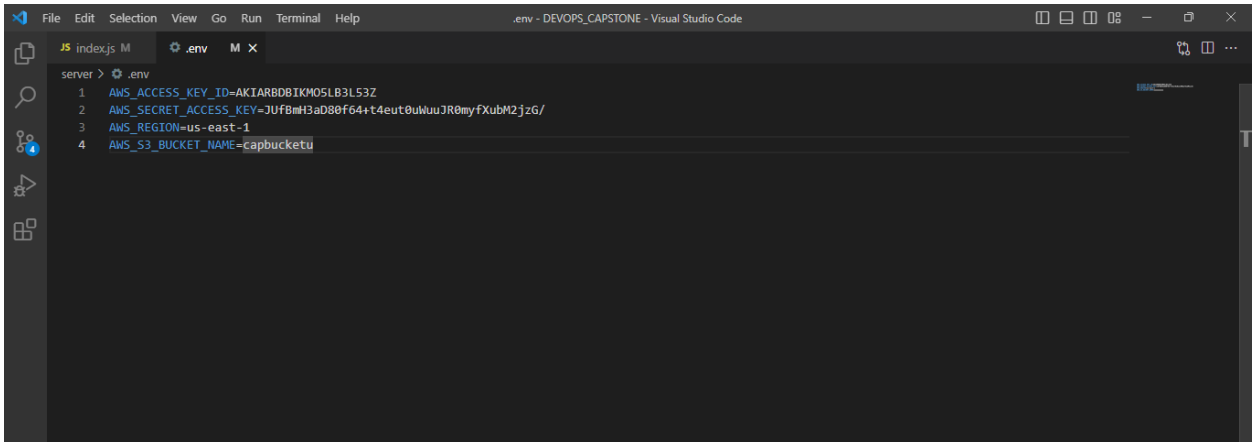
STEP 5: now we have to change the code for further implementation. Replace the existing multer code with multer-s3 with AWS access key, s3 bucket name.

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

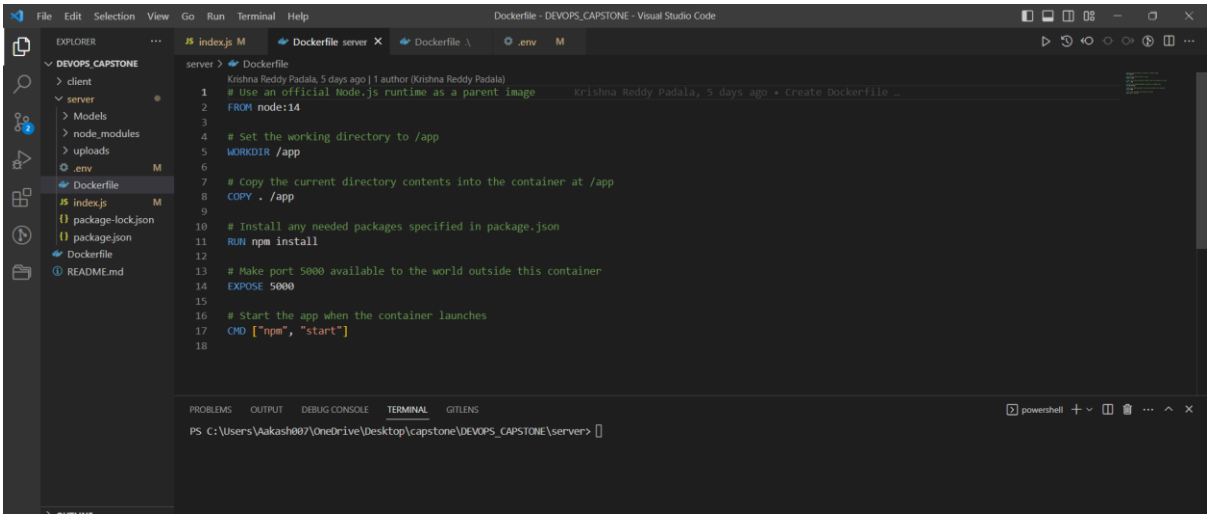




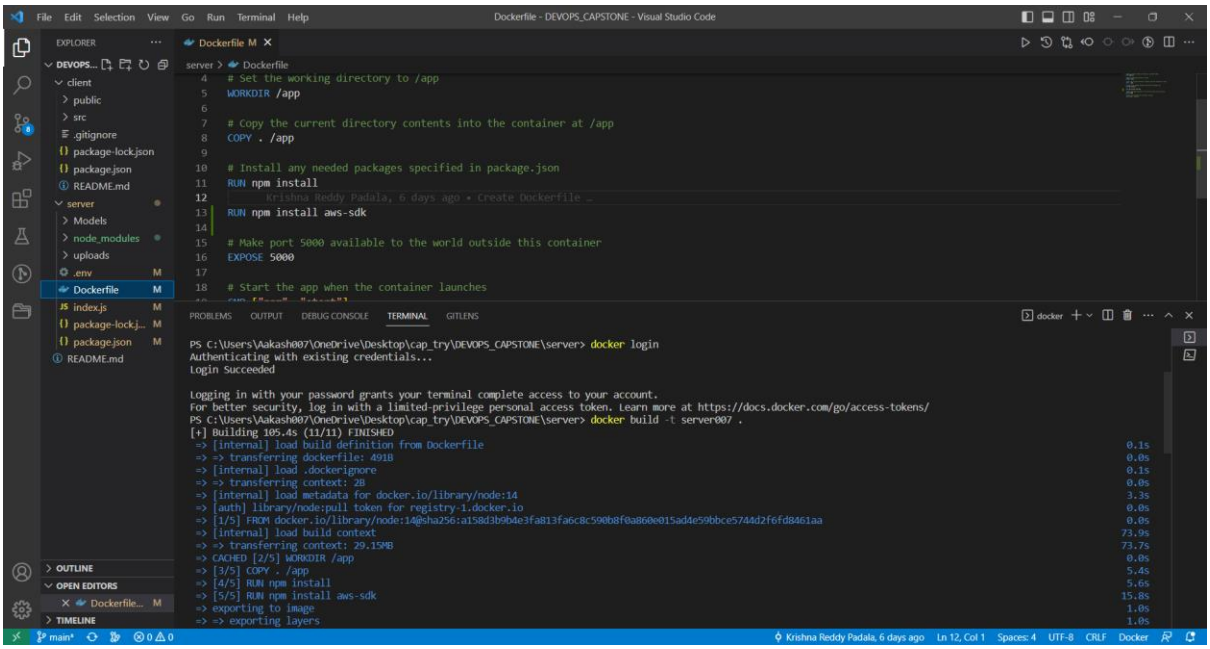
To perform this, multer requires AWS user access information such as access key, bucket name, region. Create an .env file and upload the keys in it.



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



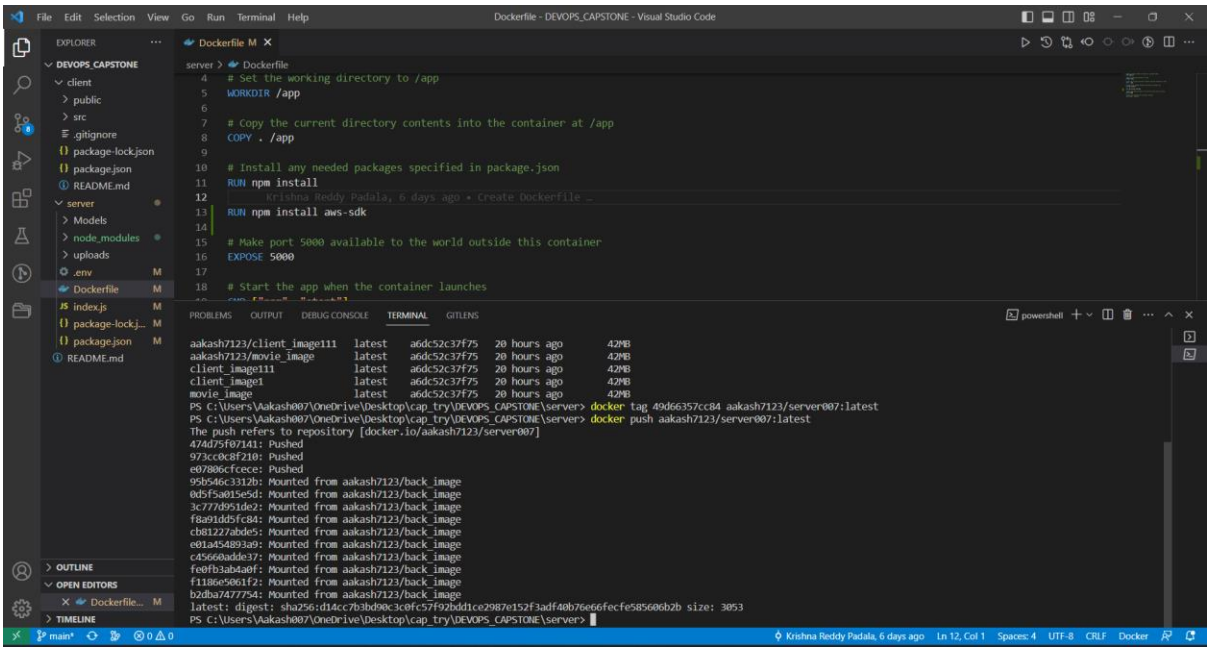
Now, we have to create a docker image to execute the code in the container. To build an image use command **docker build -t <image name> .**



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

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

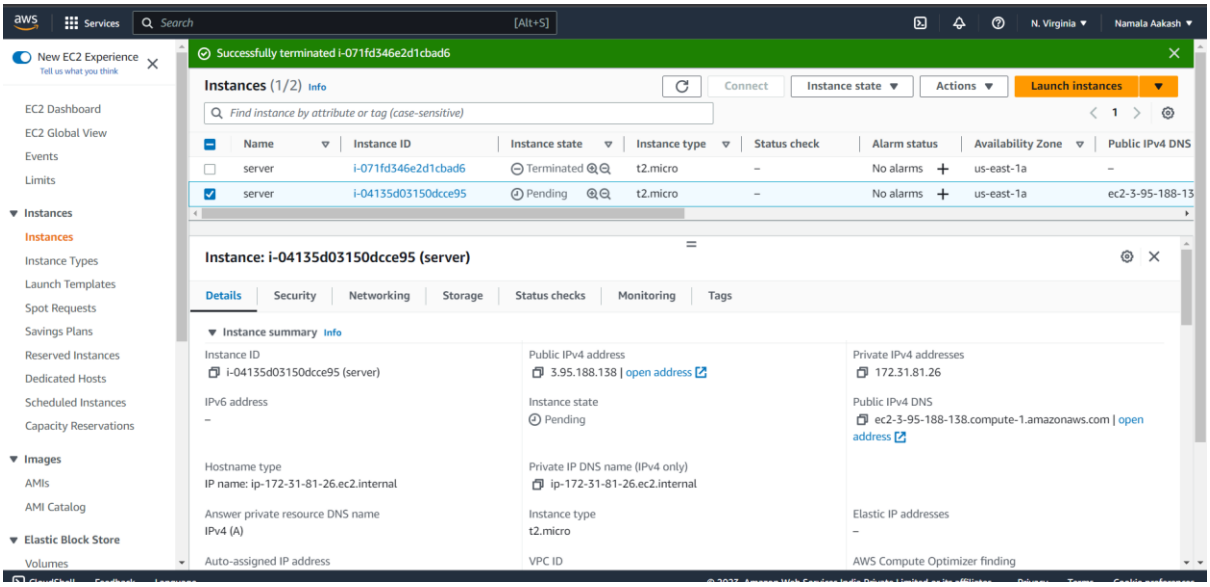
Docker tag <image_id> <username>/<image_name> and push the image to the docker hub by using command **docker push <username>/<image_name>**

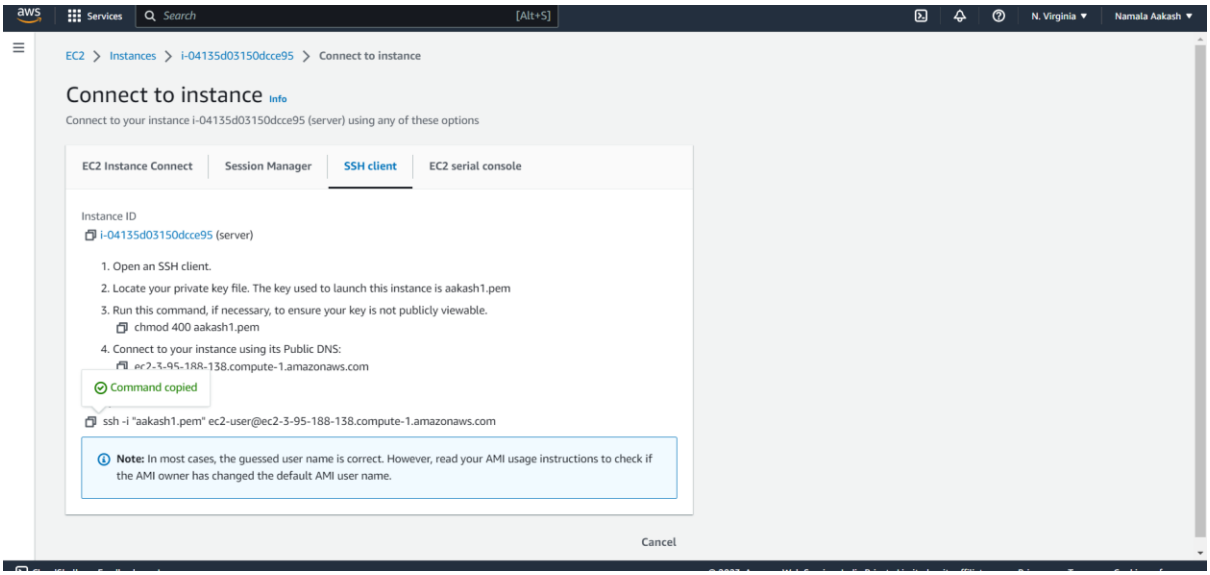


You can view the images in the Docker Hub.

Step 7: Deploying the server in the EC2 instance.

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



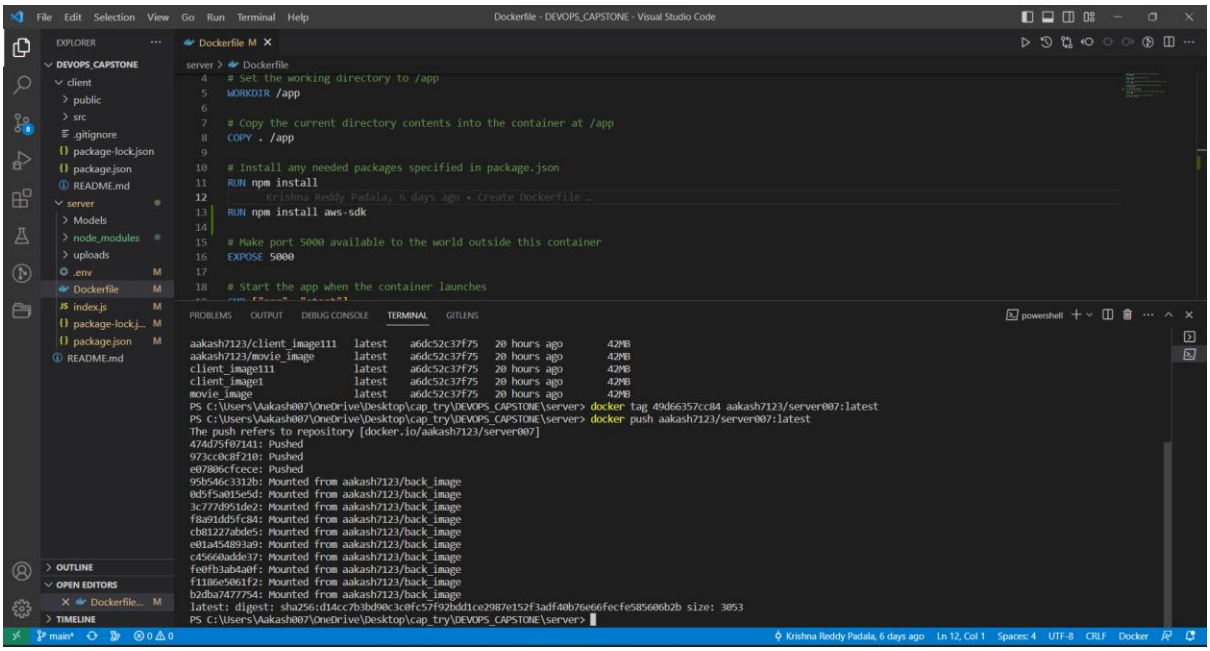


Install and start the docker

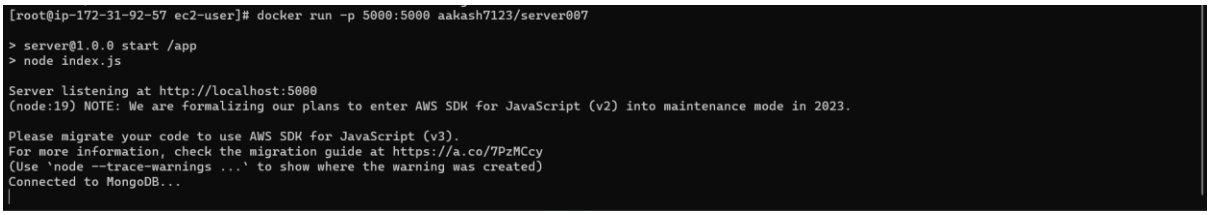
```
[ec2-user@ip-172-31-81-26 ~]$
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

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

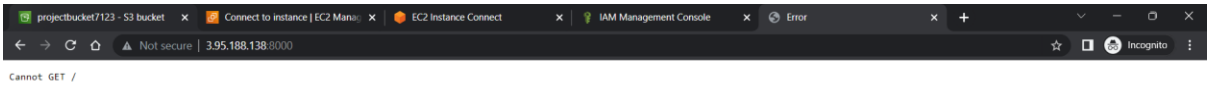
```
[root@ip-172-31-81-26 ec2-user]# systemctl start docker
[root@ip-172-31-81-26 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: active (running) since Sat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

Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.278012551Z" level=info msg="ClientConn switching balancer to...le=grpc
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.320249232Z" level=warning msg="Your kernel does not support ...weight"
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.320672057Z" level=warning msg="Your kernel does not support ...device"
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.321139174Z" level=info msg="Loading containers: start."
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.531901068Z" level=info msg="Default bridge (docker0) is assi...ddress"
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.582762052Z" level=info msg="Loading containers: done."
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.598482031Z" level=info msg="Docker daemon" commit=6051f14 gr...0.10.23
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.598488850Z" level=info msg="Daemon has completed initialization"
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal systemd[1]: Started Docker Application Container Engine.
Apr 29 14:46:38 ip-172-31-81-26.ec2.internal dockerd[3466]: time="2023-04-29T14:46:38.622214739Z" level=info msg="API listen on /run/docker.sock"
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-172-31-81-26 ec2-user]#
```

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



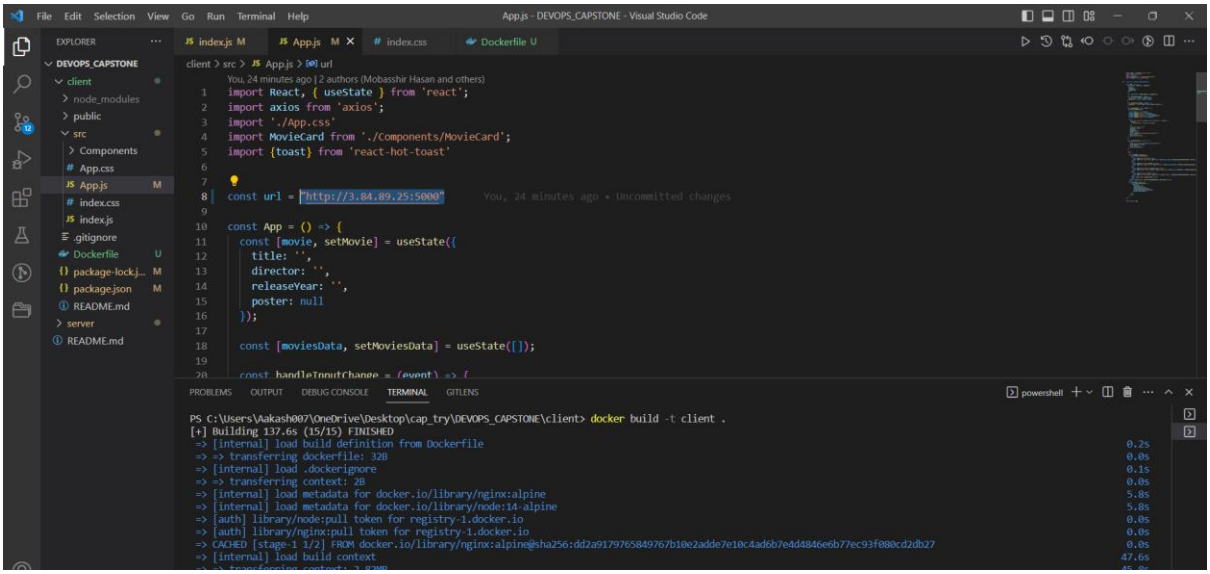
Now, run the image with port 5000 using **docker run -p 5000:5000 <username>/<image_name>**



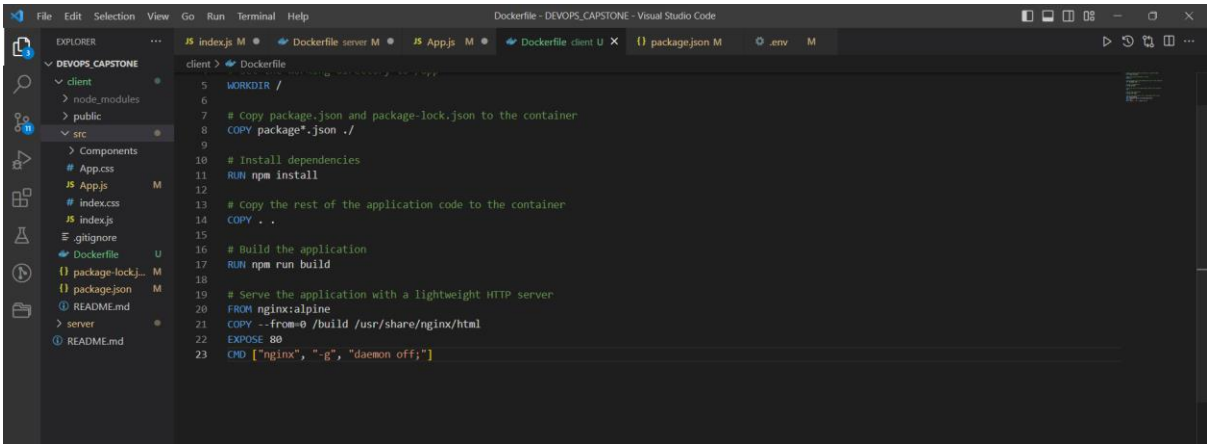


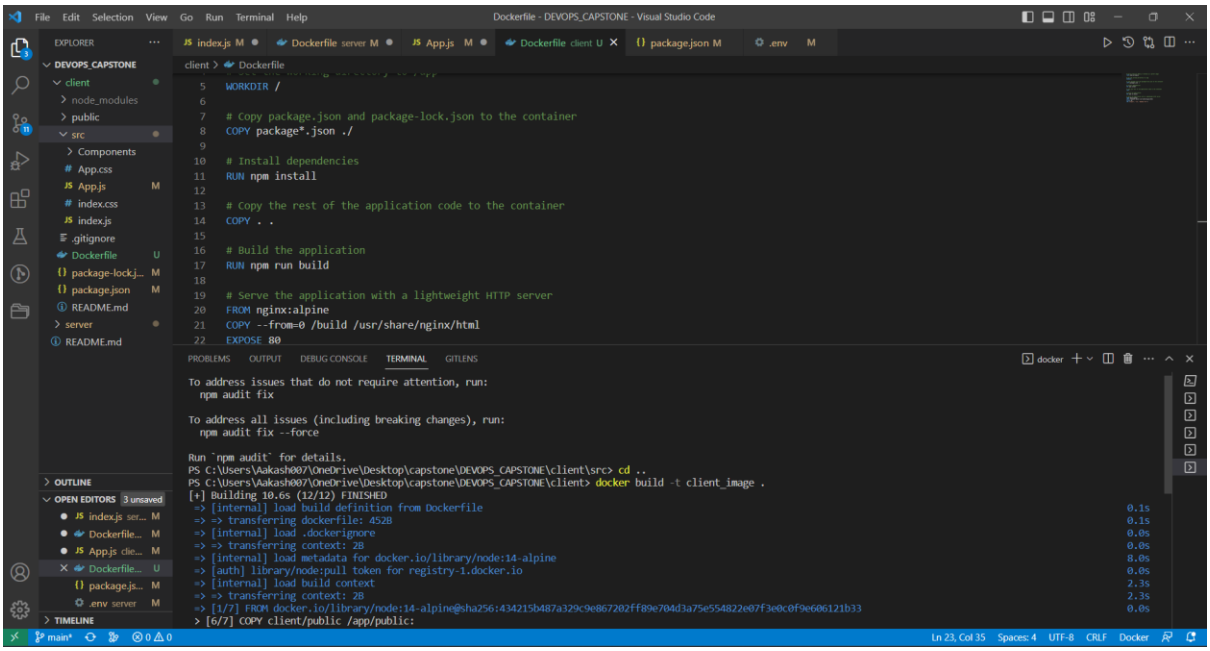
Step 8: modifying frontend code.

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

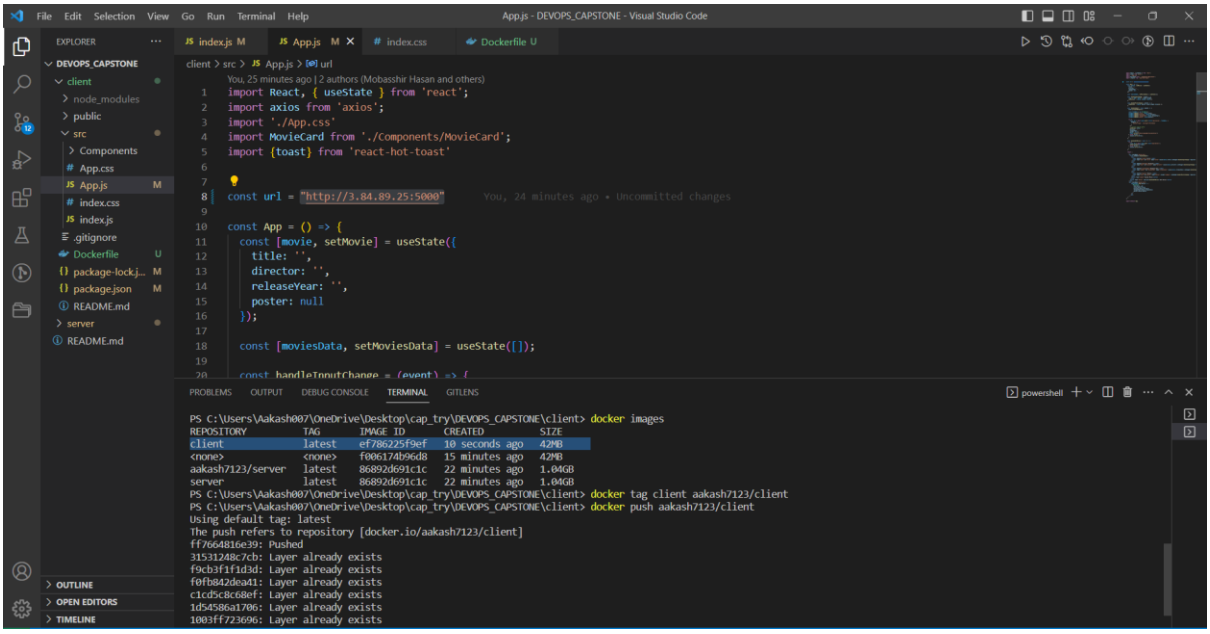


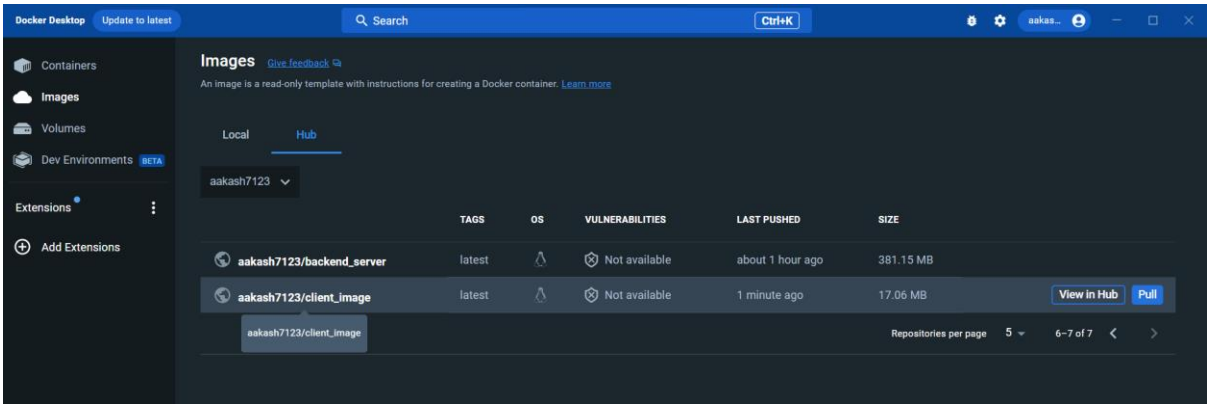
Create a docker file in the client to run the commands and build a docker image.





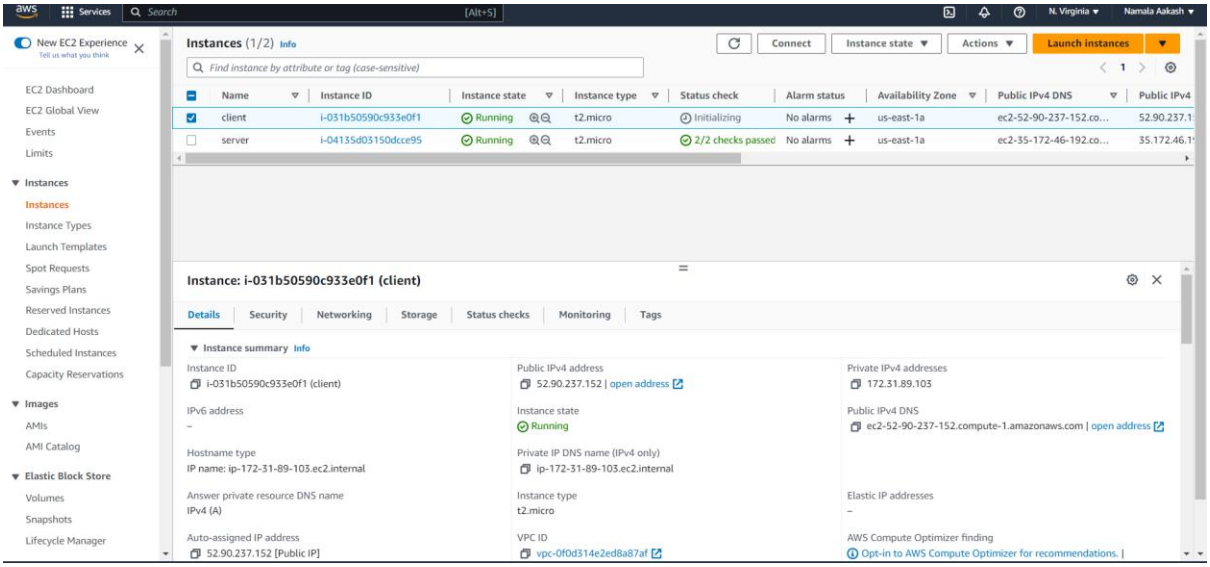
Tag and push the docker image to the Docker Hub.





Step 9: Deploying the client in the EC2 instance.

Create an EC2 instance and connect through SSH.



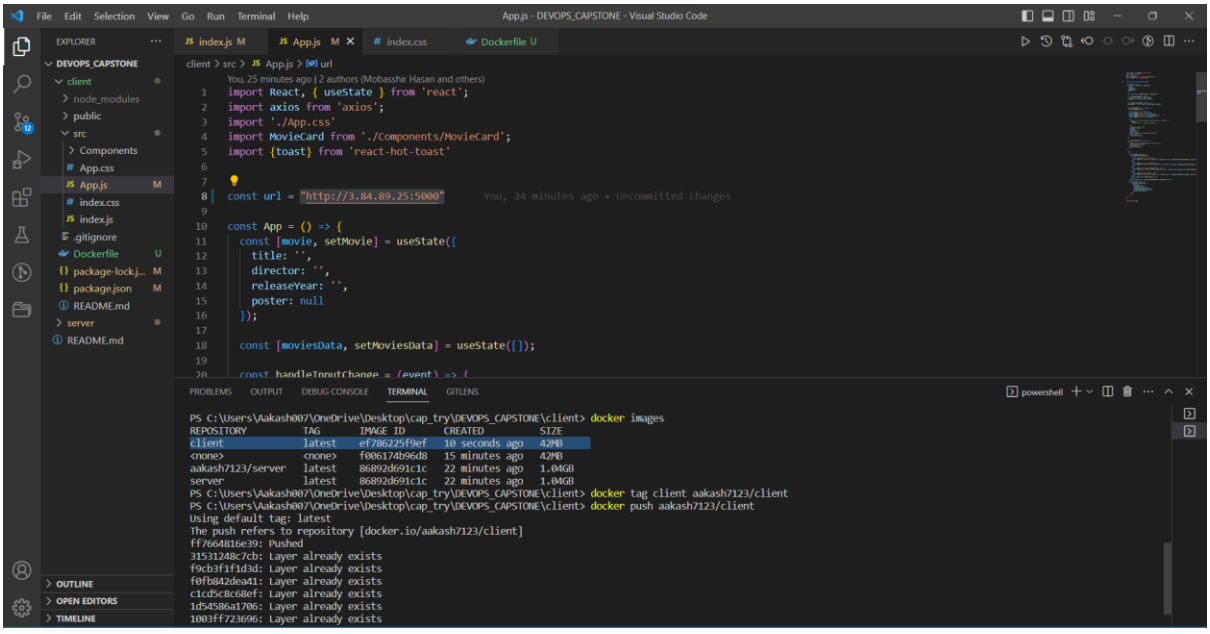

```
C:\Users\Aakash007\Downloads>ssh -i "aakash1.pem" ec2-user@ec2-52-90-237-152.compute-1.amazonaws.com
The authenticity of host 'ec2-52-90-237-152.compute-1.amazonaws.com (52.90.237.152)' can't be established.
ED25519 key fingerprint is SHA256:ek/9UfsGmbhbrg8txyEvw+TpbfVxcvKA+tls/QMeltA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-52-90-237-152.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-89-103 ~]$ sudo su
[root@ip-172-31-89-103 ec2-user]# yum update
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
No packages marked for update
[root@ip-172-31-89-103 ec2-user]#
```

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

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

Dependencies Resolved

================================================================================
```

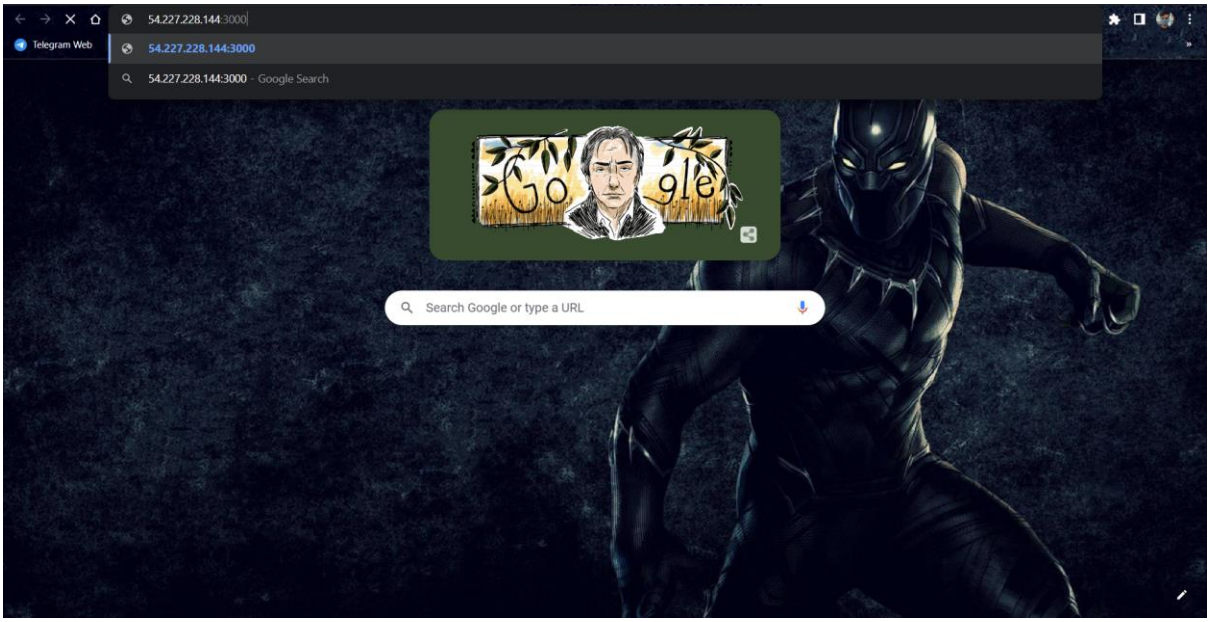
```
[root@ip-172-31-89-103 ec2-user]# systemctl start docker
[root@ip-172-31-89-103 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: active (running) since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

Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.351863280Z" level=info msg="ClientConn switching balancer t...le=grpc
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.388231397Z" level=warning msg="Your kernel does not support...weight"
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.388767025Z" level=warning msg="Your kernel does not support...device"
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.389348207Z" level=info msg="Loading containers: start."
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.661984254Z" level=info msg="Default bridge (docker0) is ass...ddress"
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.716713537Z" level=info msg="Loading containers: done."
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.735015291Z" level=info msg="Docker daemon" commit=6051f14 g...0.10.23
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.735601013Z" level=info msg="Daemon has completed initialization"
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal systemd[1]: Started Docker Application Container Engine.
Apr 29 17:47:41 ip-172-31-89-103.ec2.internal dockerd[3468]: time="2023-04-29T17:47:41.760964940Z" level=info msg="API listen on /run/docker.sock"
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-172-31-89-103 ec2-user]#
```

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

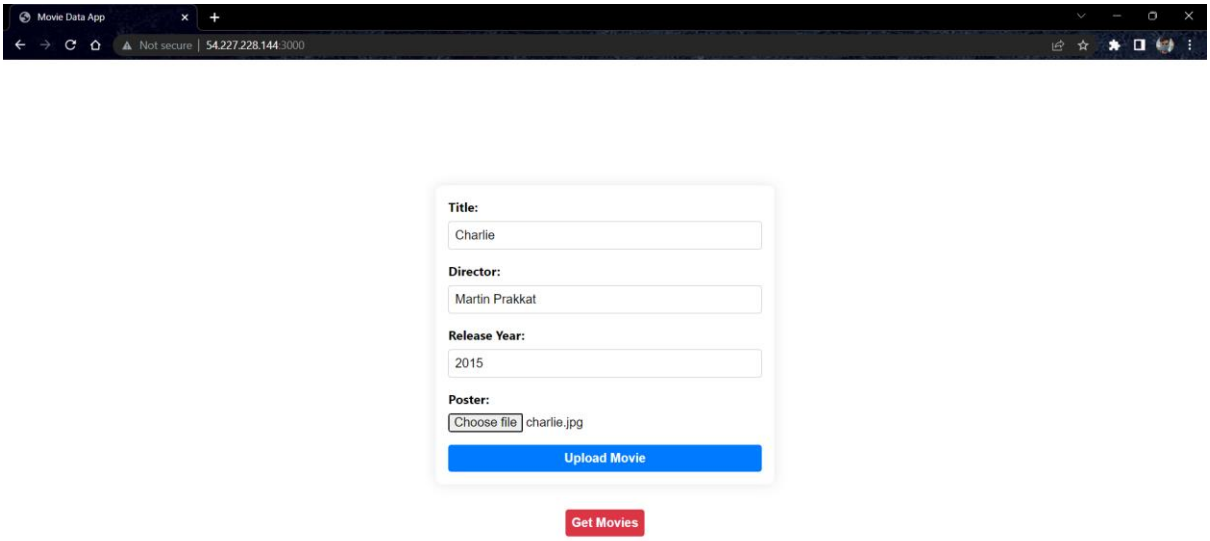


To view the website, use the public IP address of client instance with the port.

http://54.227.228.144:3000

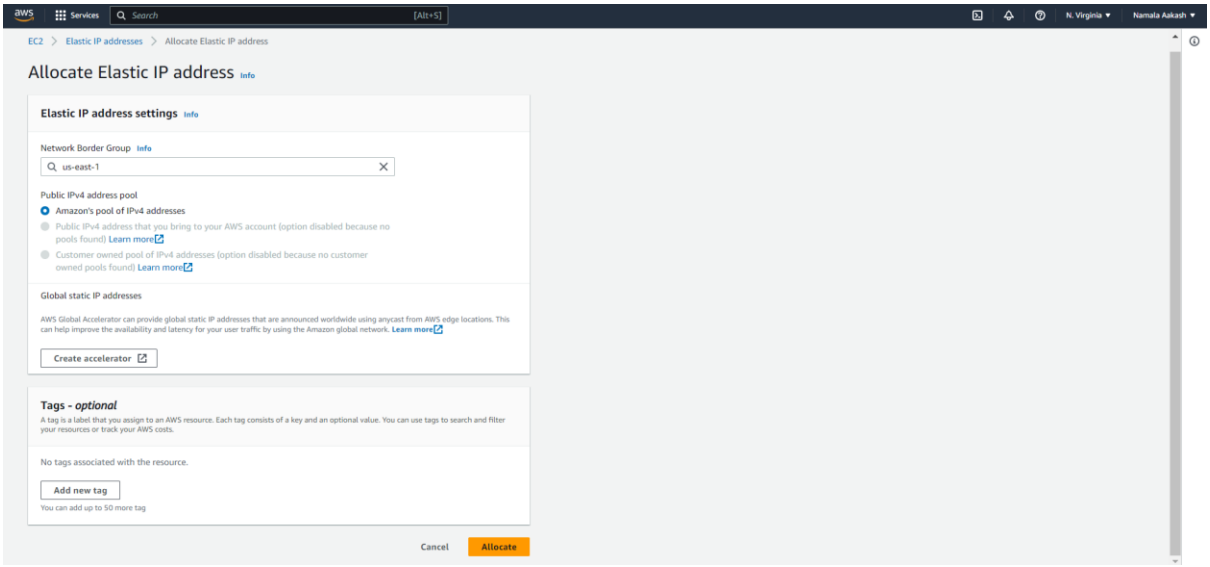


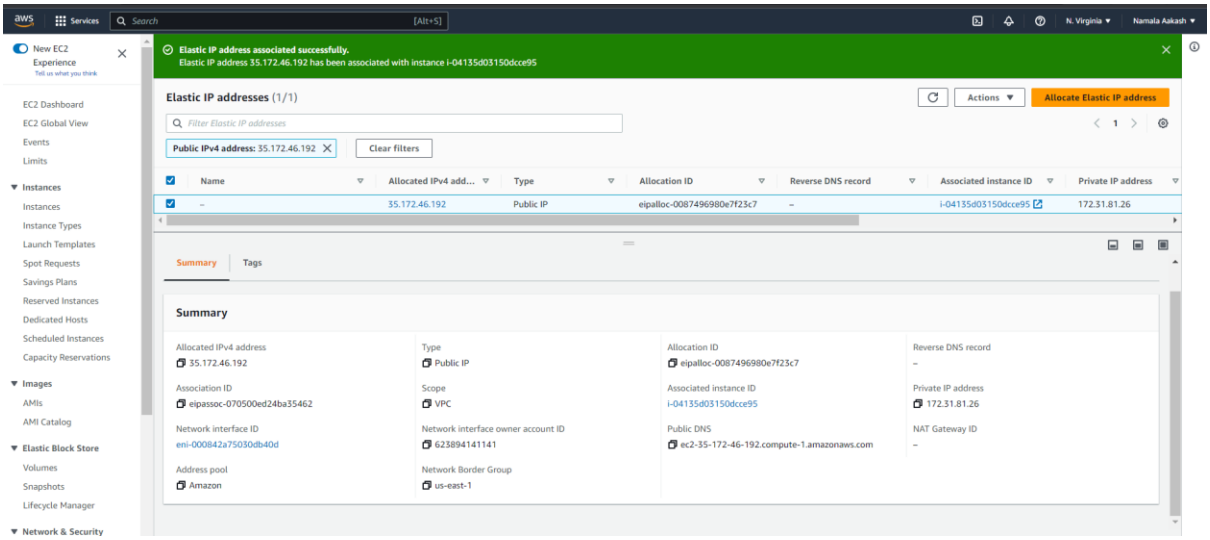
you can view the website. Enter the details and images and submit it. The details are uploaded to the database server and images are uploaded to S3 bucket specified.



Step 10: Allocating Elastic IP to instance.

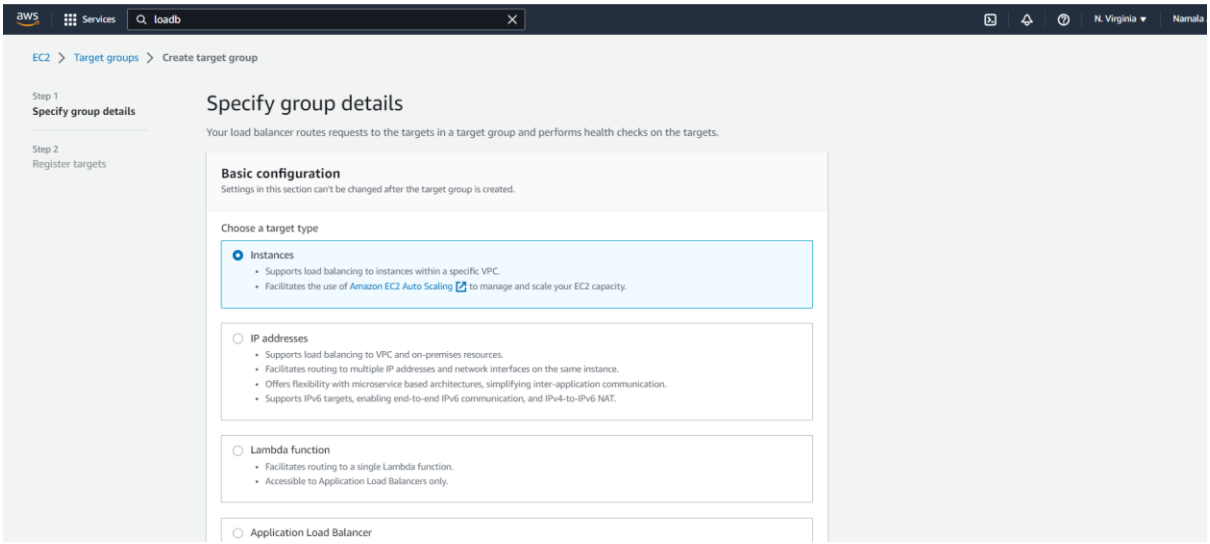
By using elastic IP we can move all the attributes of the network interface to anther instance in a single step. We can allocate by specifying the region, resource type and IP address.

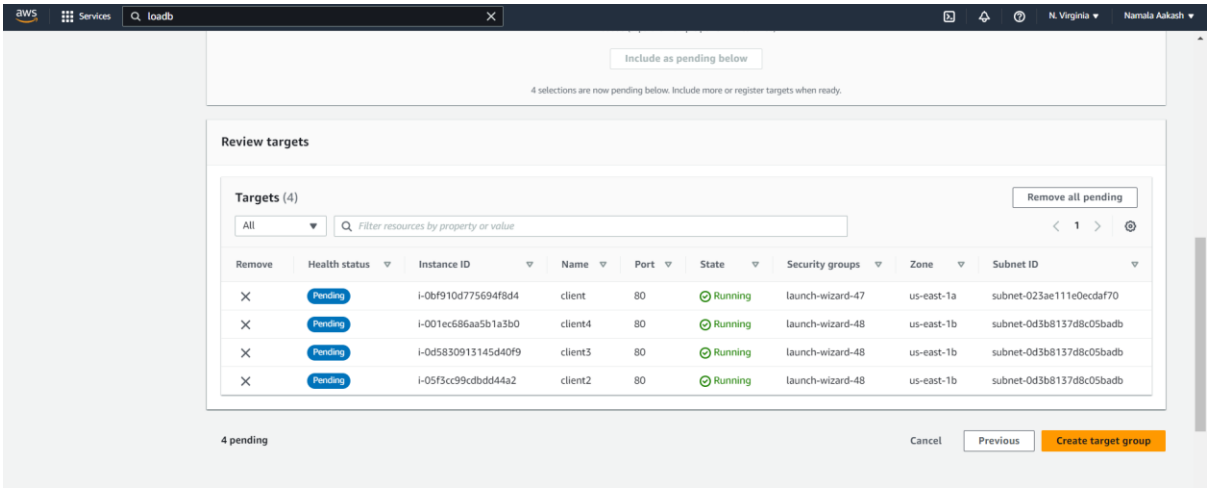


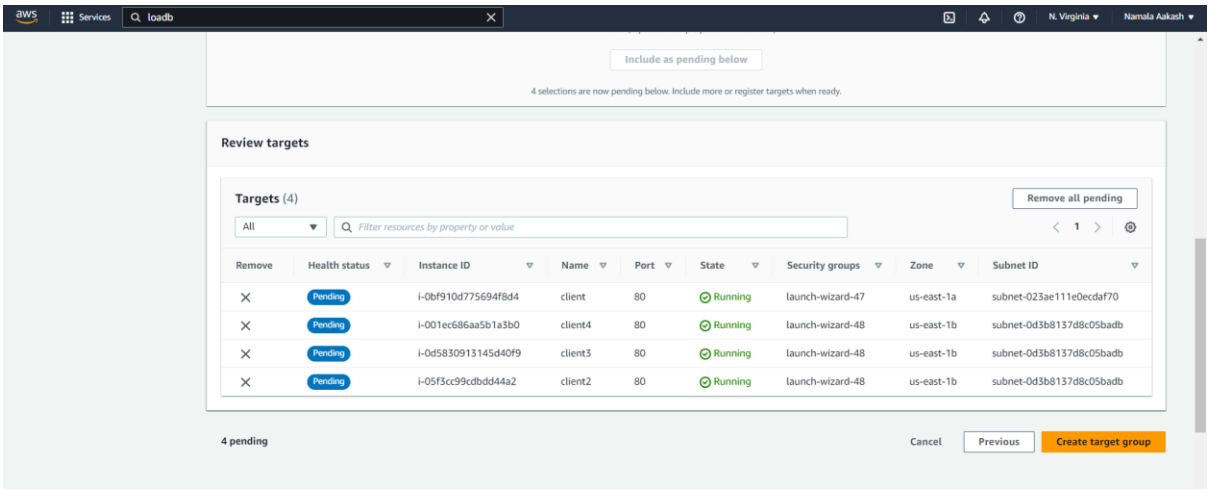


Step 11: creating load balancer and assigning a DNS name

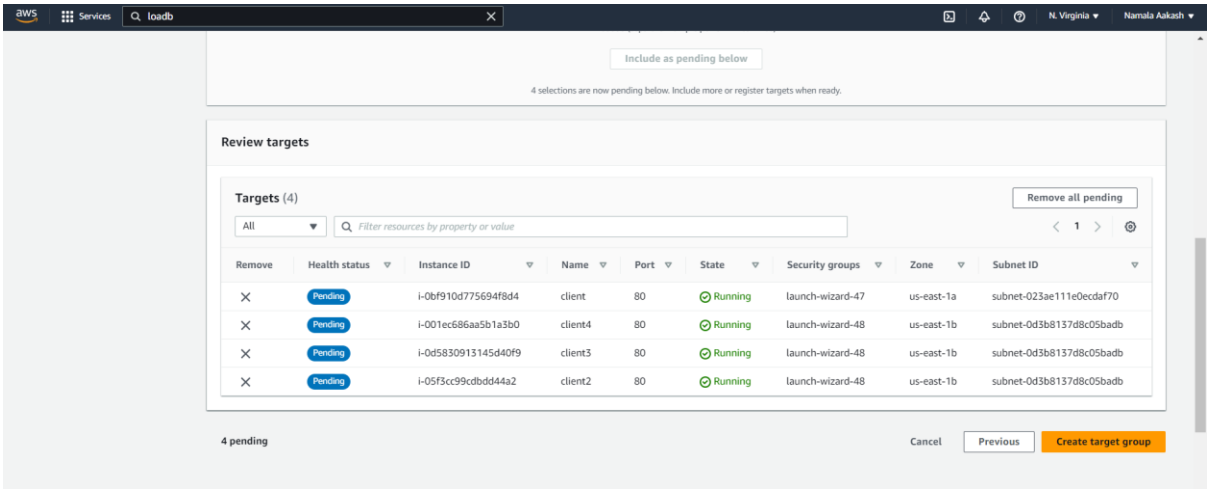
To create a load balancer to redirect the traffic, create target groups to the respective instance with protocol as TCP 80.

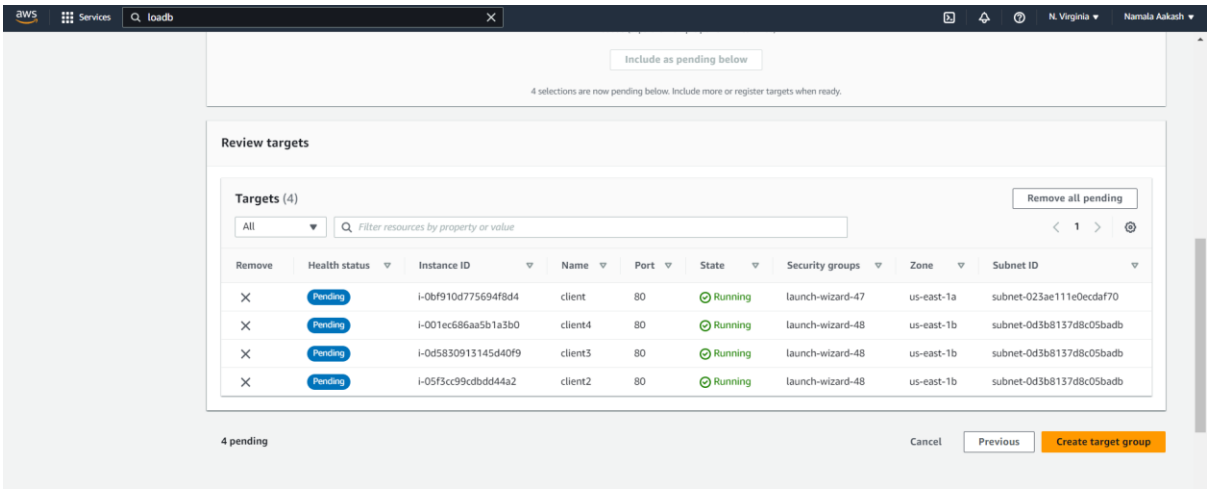






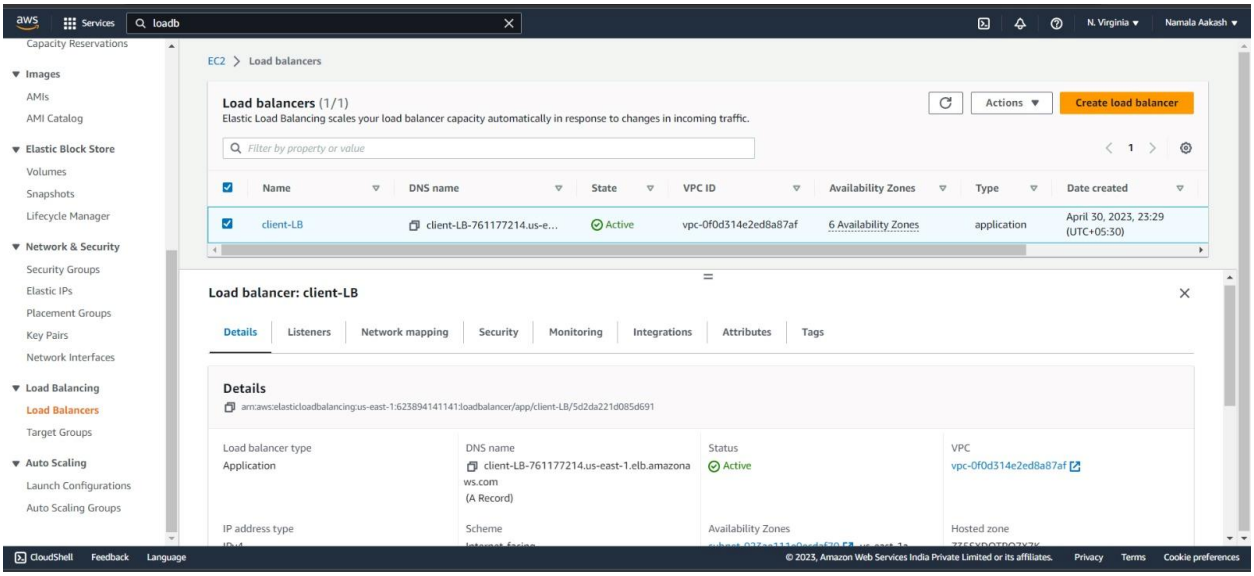
Now create load balancer by giving basic configuration (vpc, region, listener and routing) and mapping listeners to get a DNS name.





Update the existing one to point to the load balancer.

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


aws
Services
loadb
N. Virginia
Namala Aakash
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups
Auto Scaling
Launch Configurations
Auto Scaling Groups
EC2 > Load balancers
Load balancers (1/1)
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.
Actions
Create load balancer
Filter by property or value
Name
DNS name
State
VPC ID
Availability Zones
Type
Date created
client-LB
client-LB-761177214.us-e...
Active
vpc-0f0d314e2ed8a87af
6 Availability Zones
application
April 30, 2023, 23:29 (UTC+05:30)
Load balancer: client-LB
Details
Listeners
Network mapping
Security
Monitoring
Integrations
Attributes
Tags
Details
arn:aws:elasticloadbalancing:us-east-1:623894141141:loadbalancer/app/client-LB/5d2da221d085d691
Load balancer type
Application
DNS name
client-LB-761177214.us-east-1.elb.amazonaws.com
(A Record)
Status
Active
VPC
vpc-0f0d314e2ed8a87af
IP address type
Scheme
Availability Zones
Hosted zone
CloudShell
Feedback
Language
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences

CONTRIBUTONS AND CHALLENGES FACED BY TEAM MEMBERS:

Task 1: Connecting the Server with Atlas MongoDB

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

Task 2: Creating IAM user and s3.

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

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

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

Task 4: Containerization of the code using Docker file.

The task 4 is done by Aakash, Vaishnavi, Pushpa

Task 5: Deploying server on EC2 using Docker.

The task 5 is done by Aakash, Prudhvi, Pushpa

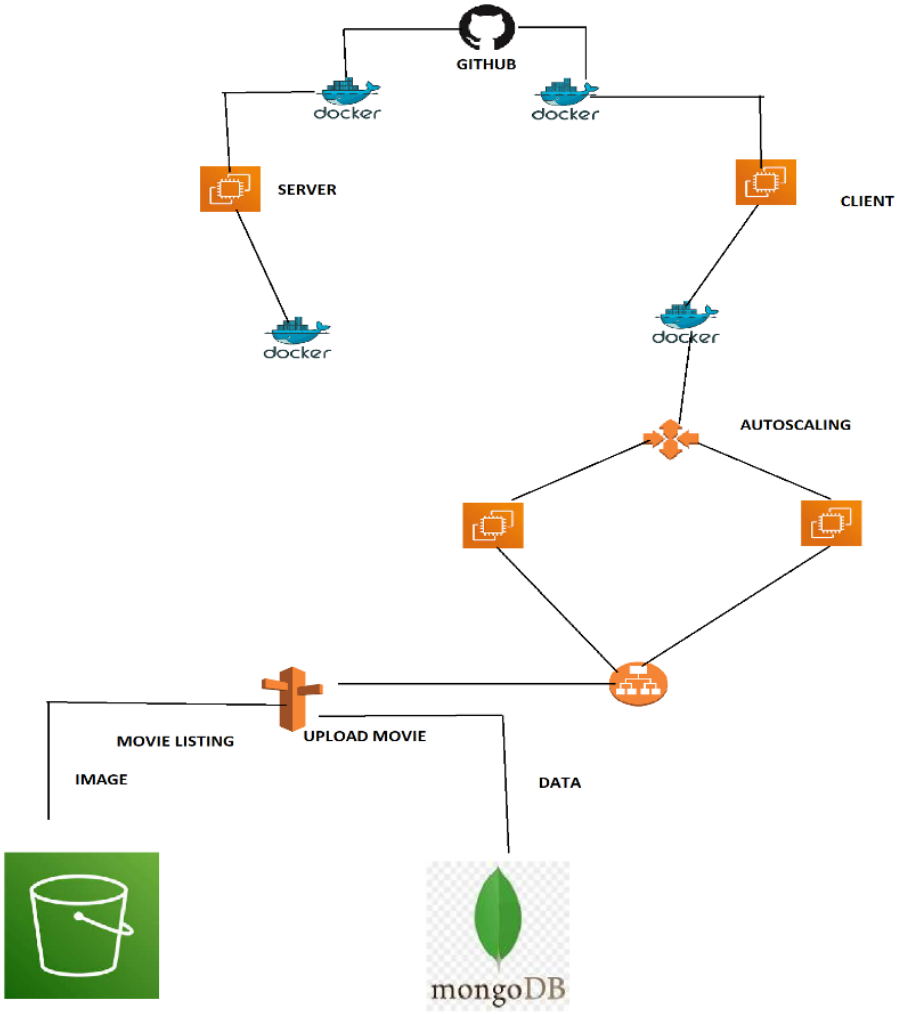
Task 6: load balancing, Elastic IP, DNs: done by Vaishnavi, kavya, Rosy

Task 7: documentation
Done by Pushpa, Vaishnavi, Prudhvi

GIT HUB REPOSITORY LINK :

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

DEPLOYMENT ARCHITECTURE:



Conclusion:

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

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

REFERENCE LINKS :

RESOURCES MongoDB : https://www.mongodb.com/docs/

AWS : https://docs.aws.amazon.com/

S3-multer : https://www.npmjs.com/package/

multer-s3 Docker : https://docs.docker.com/

Git : https://git-scm.com/docs

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