Movie Magic-Smart Movie Ticket Booking System

Project Description:

With the increasing demand for a seamless and modern movie-watching experience, traditional ticket booking methods often fall short due to long queues, limited availability, and inconsistent service. To address this, the development team introduced Movie Magic—a smart, cloud-based movie ticket booking system. Built using Flask for backend development, hosted on AWS EC2, and integrated with DynamoDB for dynamic data management, the platform allows users to register, log in, and book movie tickets online with ease. Users can search for movies and events based on location, view real-time seat availability, and complete their bookings in just a few clicks. Upon booking, AWS SNS sends instant email notifications confirming ticket details, enhancing user engagement and trust. This cloud-native solution streamlines the entire movie ticketing process, ensuring fast, scalable, and user-friendly access to entertainment for all.

Scenario 1: Efficient Booking system for users

In the Movie Magic System, AWS EC2 provides a reliable infrastructure capable of handling multiple users accessing the platform simultaneously. For example, a user can log in, navigate to the movie selection page, and seamlessly browse available shows and events in their city. They can then select a showtime, pick their preferred seats using an interactive layout, and confirm the booking—all in real-time. Flask manages backend processes, ensuring smooth data flow and quick response times even during high-traffic periods such as weekends or blockbuster releases.

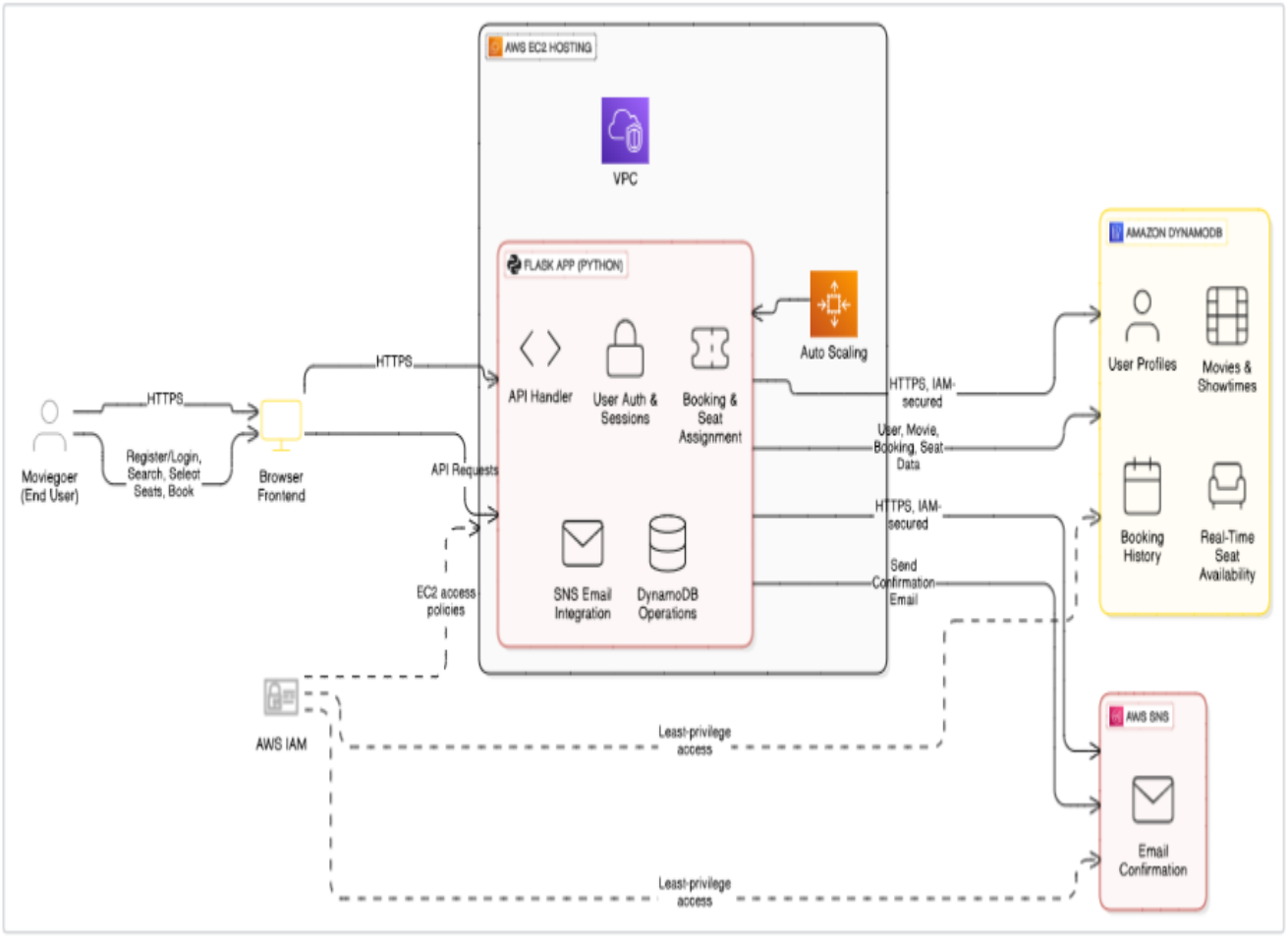
Scenario 2: Seamless Booking confirmation Notifcations

When a user completes a ticket booking, the Movie Magic System leverages AWS SNS to send instant email notifications to confirm the booking. For instance, once the booking is submitted, Flask processes the transaction, and SNS sends a customized email to the user with all ticket details, including movie name, date, time, and seat numbers. This real-time notification system enhances the customer experience and reduces uncertainty, while DynamoDB securely stores the booking records for both users and admins to manage and track.

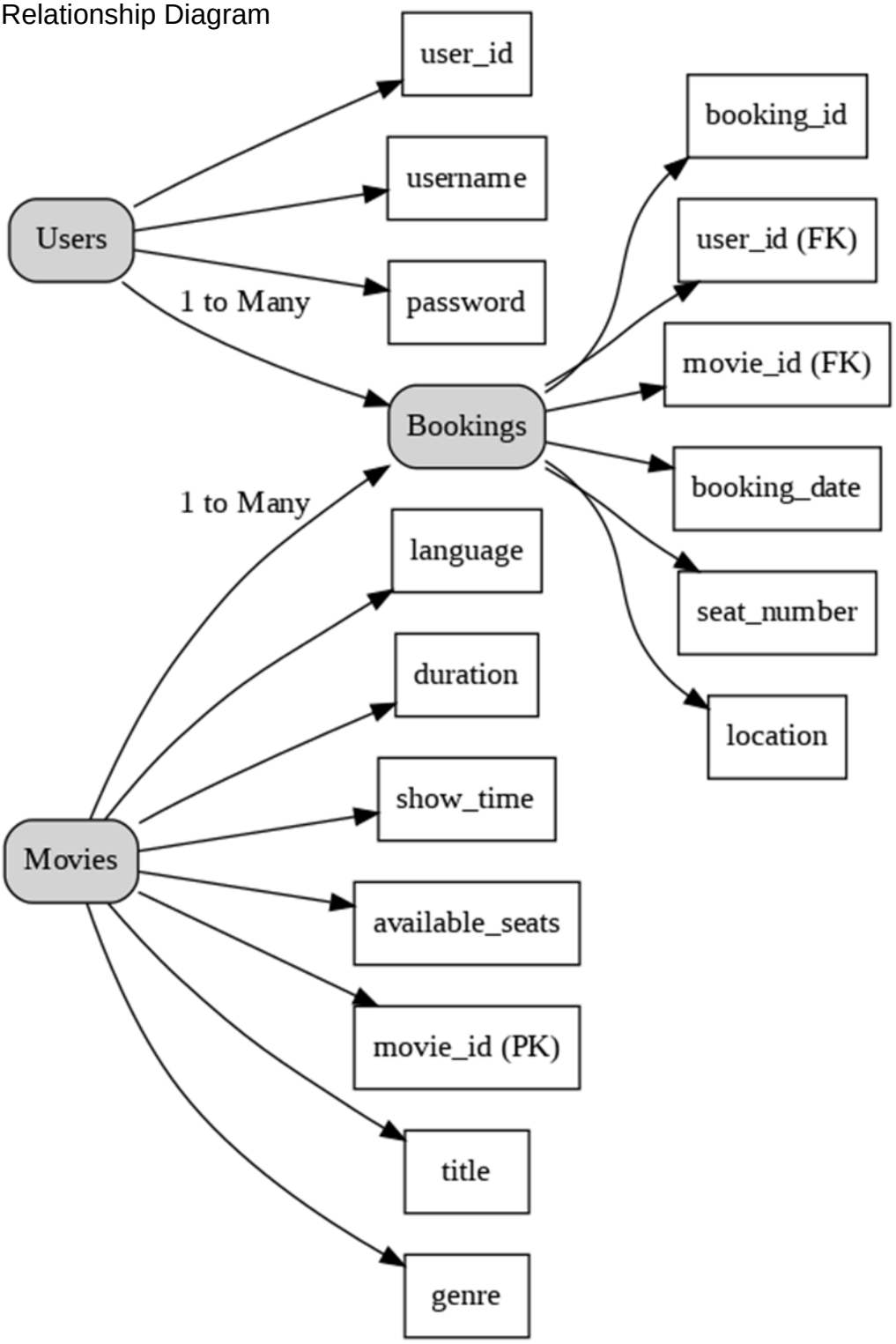
Scenario 3: Easy Access to Movies and Events

The Movie Magic platform offers users a seamless interface to explore currently running movies and upcoming live events. After logging in, a user can search by location or genre and instantly view listings with showtimes, ratings, and available seats. Flask dynamically fetches this data from DynamoDB, ensuring real-time updates on seat availability and event information. Meanwhile, the EC2-hosted application remains stable and responsive, even during traffic spikes, providing users with an uninterrupted and enjoyable booking experience.

AWS ARCHITECTURE



Entity Relationship Diagram



Pre-requisites:

1. .**AWS Account Setup**: AWS Account Setup
2. **Understanding IAM**: IAM Overview
3. **Amazon EC2 Basics**: EC2 Tutorial
4. **DynamoDB Basics**: DynamoDB Introduction
5. **SNS Overview**: SNS Documentation
6. **Git Version Control**: Git Documentation

Project WorkFlow:

Milestone 1. Backend Development and Application Setup

Develop the Backend Using Flask.

Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login

Set up an AWS account if not already done.

Log in to the AWS Management Console

Milestone 3. DynamoDB Database Creation and Setup

Create a DynamoDB Table.

Configure Attributes for User Data and Book Requests.

Milestone 4. SNS Notification Setup

Create SNS topics for book request notifications.

Subscribe users and library staff to SNS email notifications.

Milestone 5. IAM Role Setup

Create IAM Role

Attach Policies

Milestone 6. EC2 Instance Setup

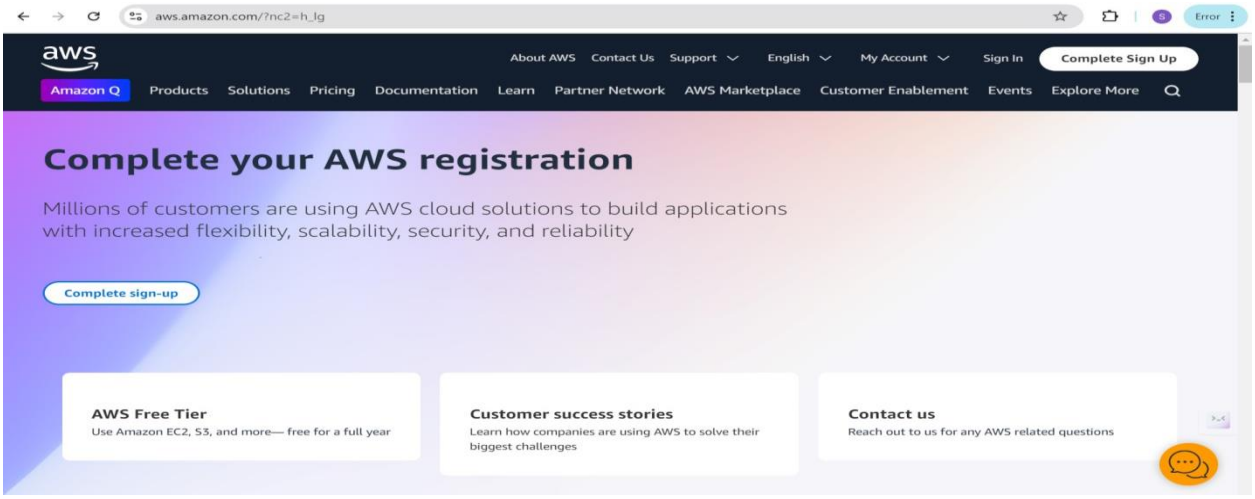
- Launch an EC2 instance to host the Flask application.
- Configure security groups for HTTP, and SSH access.

Milestone 7. Deployment on EC2

- Upload Flask Files
- Run the Flask App

Milestone 8. Testing and Deployment

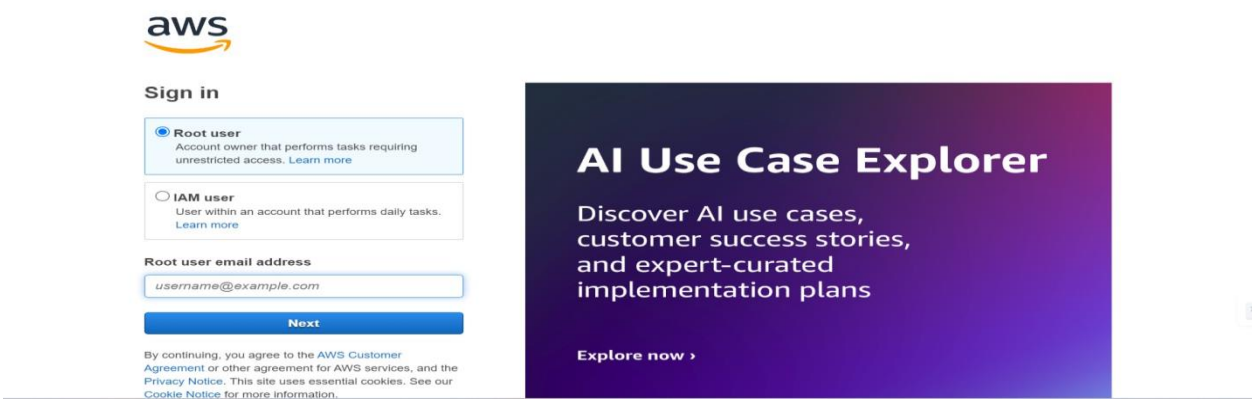
- Conduct functional testing to verify user registration, login, book requests, and notifications.



- **Activity 1.2: Log in to the AWS Management Console**
    - After setting up your account, log in to the AWS Management Console.

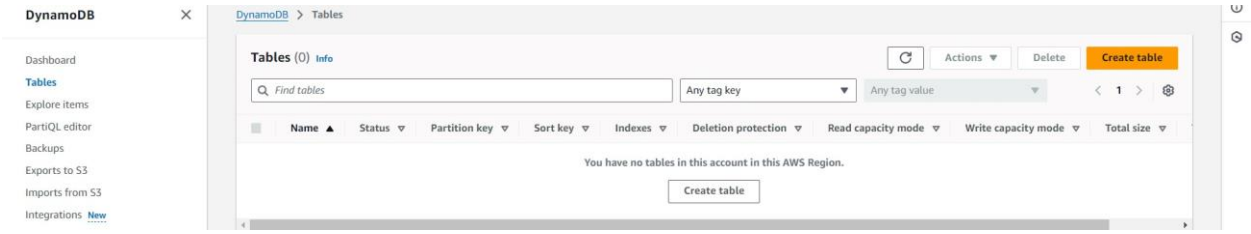
● Activity 1.2: Log in to the AWS Management Console

○ After setting up your account, log in to the AWS Management Console.



Milestone 2: DynamoDB Database Creation and Setup

● Activity 2.1:Navigate to the DynamoDB



- **Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**
    - Create Users table with partition key “Email” with type String and click on create tables.

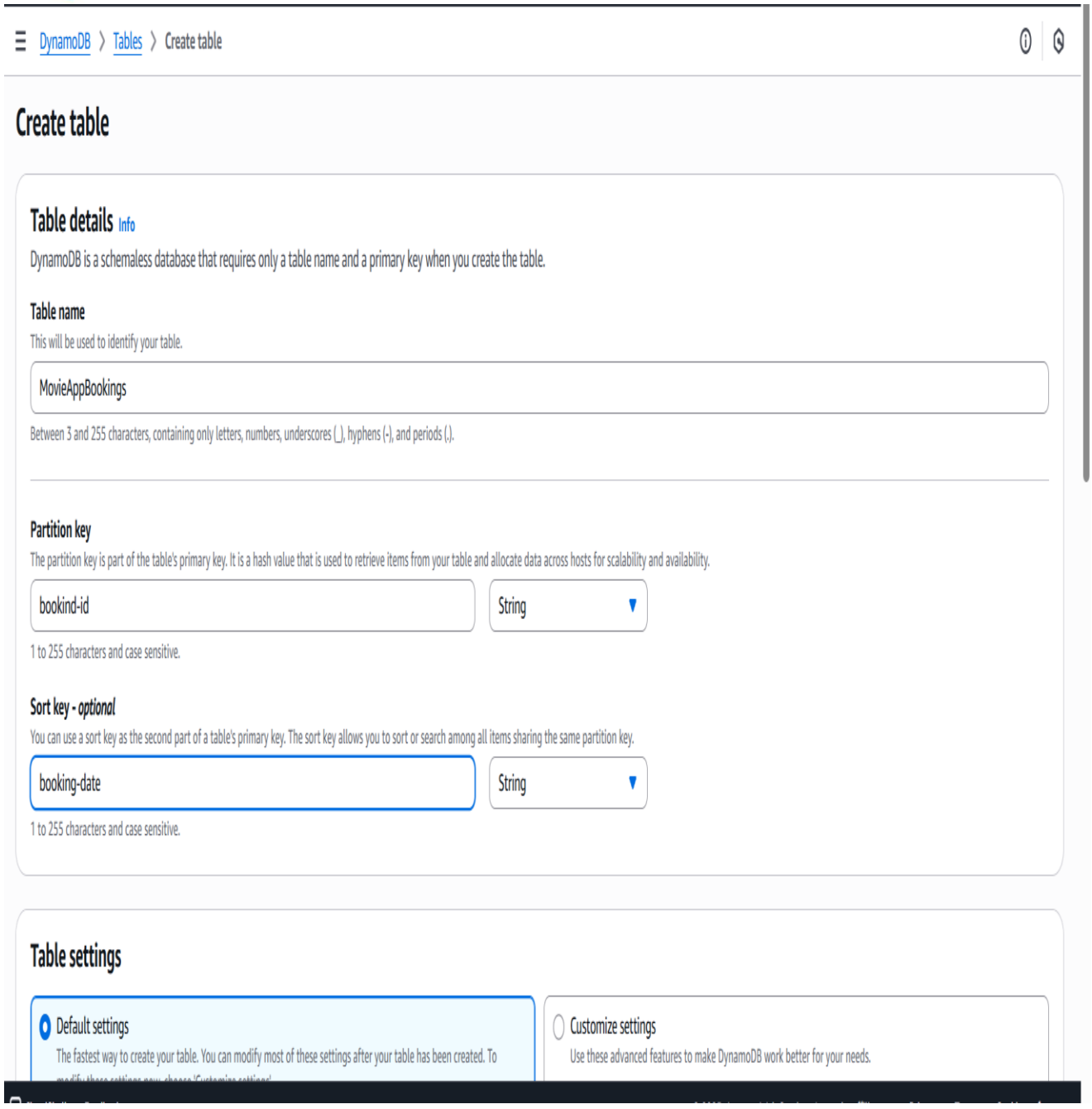

DynamoDB
Tables
Create table
Create table
Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.
Table name
This will be used to identify your table.
MovieAppBookings
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).
Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
bookind-id
String
1 to 255 characters and case sensitive.
Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
booking-date
String
1 to 255 characters and case sensitive.
Table settings
Default settings
The fastest way to create your table. You can modify most of these settings after your table has been created. To
Customize settings
Use these advanced features to make DynamoDB work better for your needs.

Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags

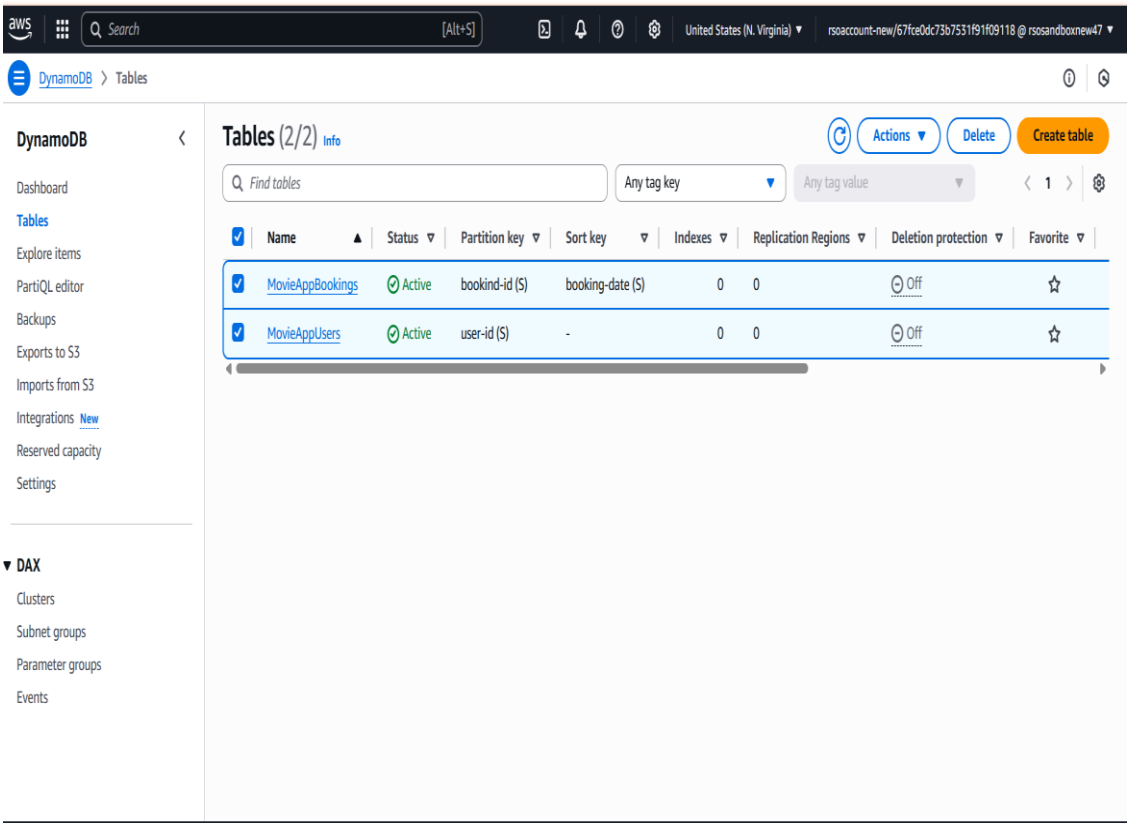
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

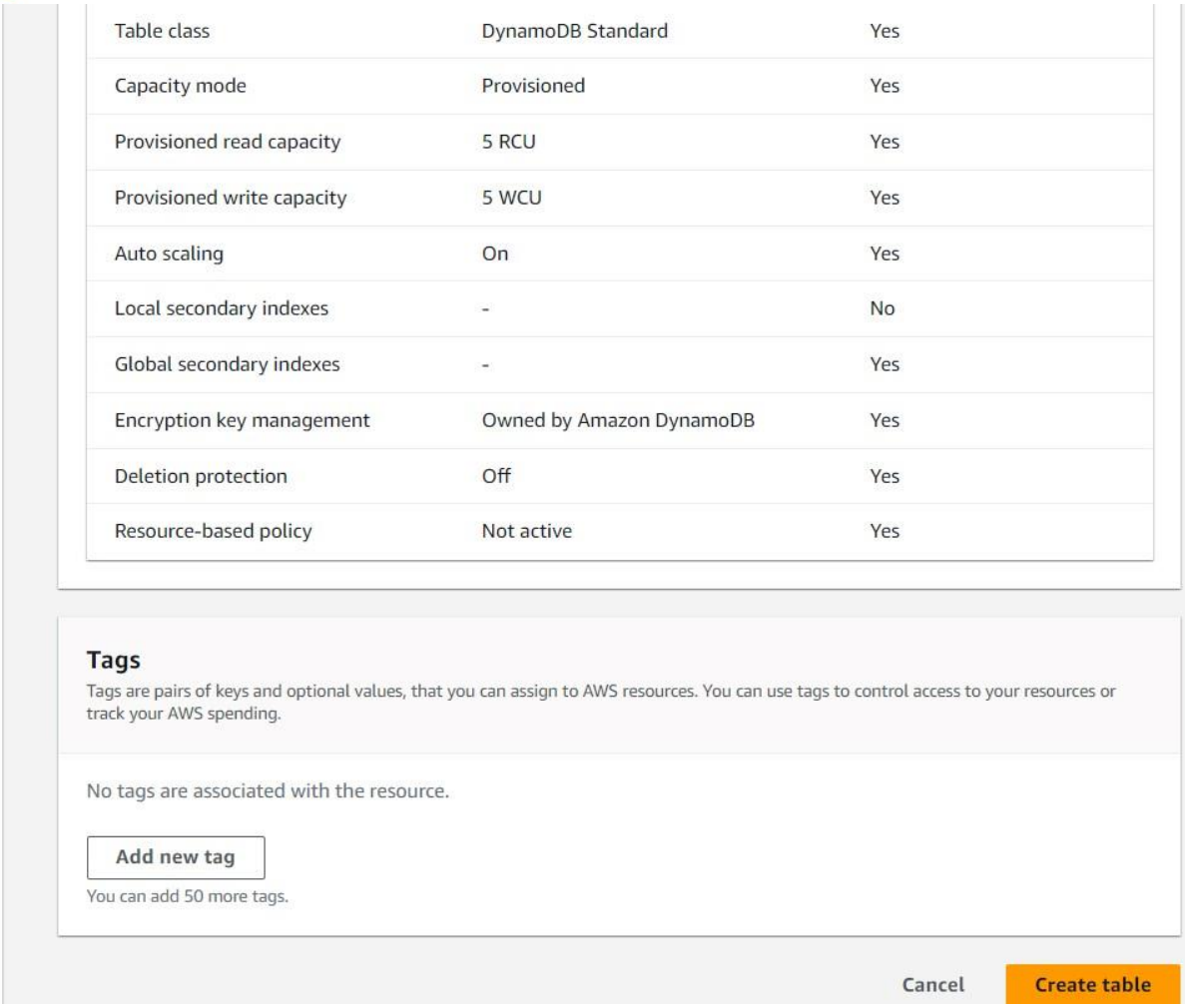
Add new tag

You can add 50 more tags.

Cancel Create table



- Follow the same steps to create a requests table with Email as the primary key for book requests data.



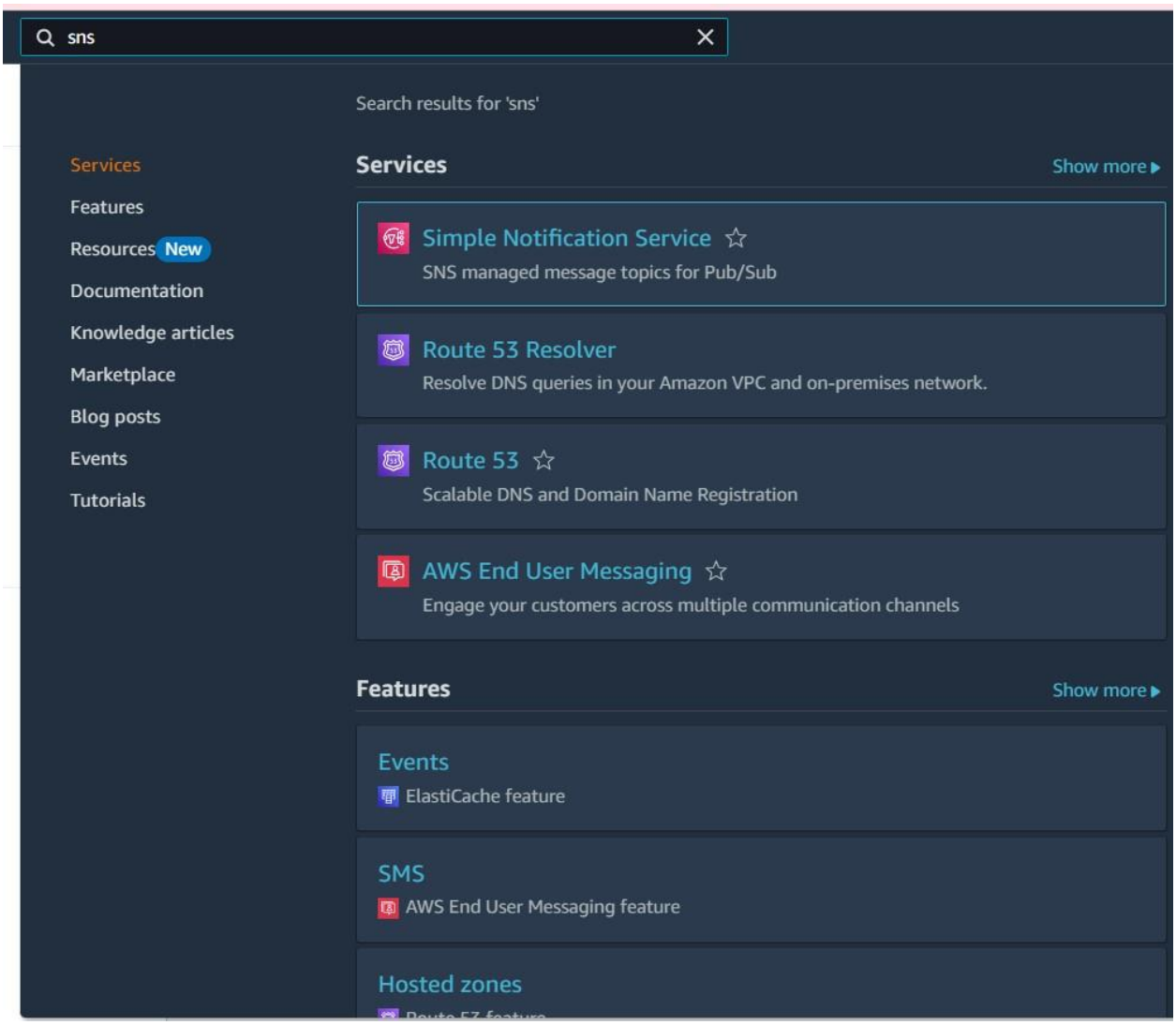
Milestone 3: SNS Notification Setup

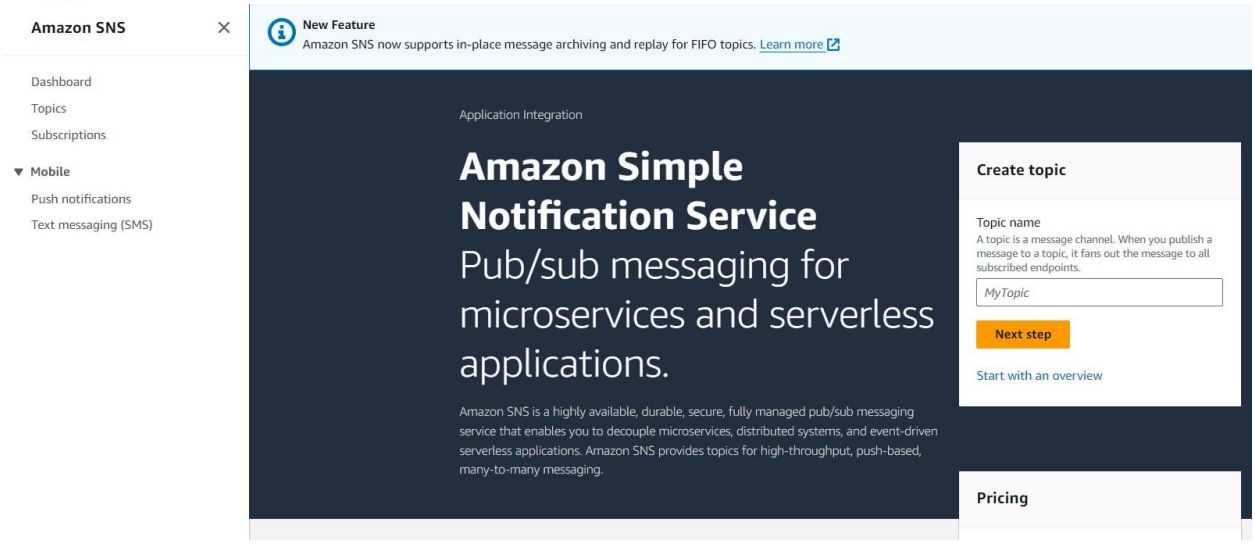
- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**

  - In the AWS Console, search for SNS and navigate to the SNS Dashboard.

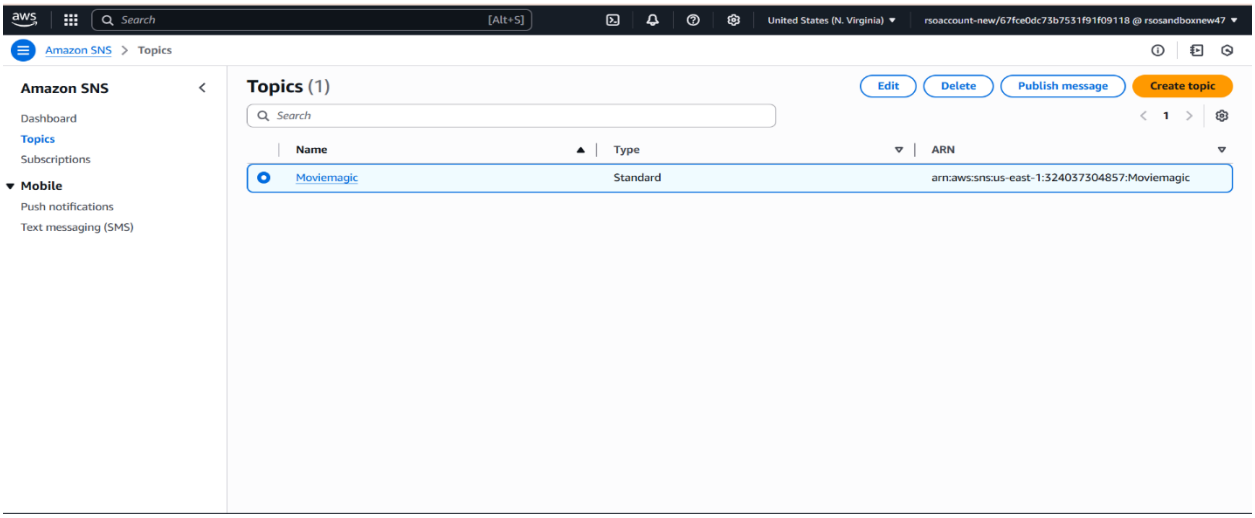
Milestone 3: SNS Notification Setup

- Activity 3.1: Create SNS topics for sending email notifications to users and library staff.

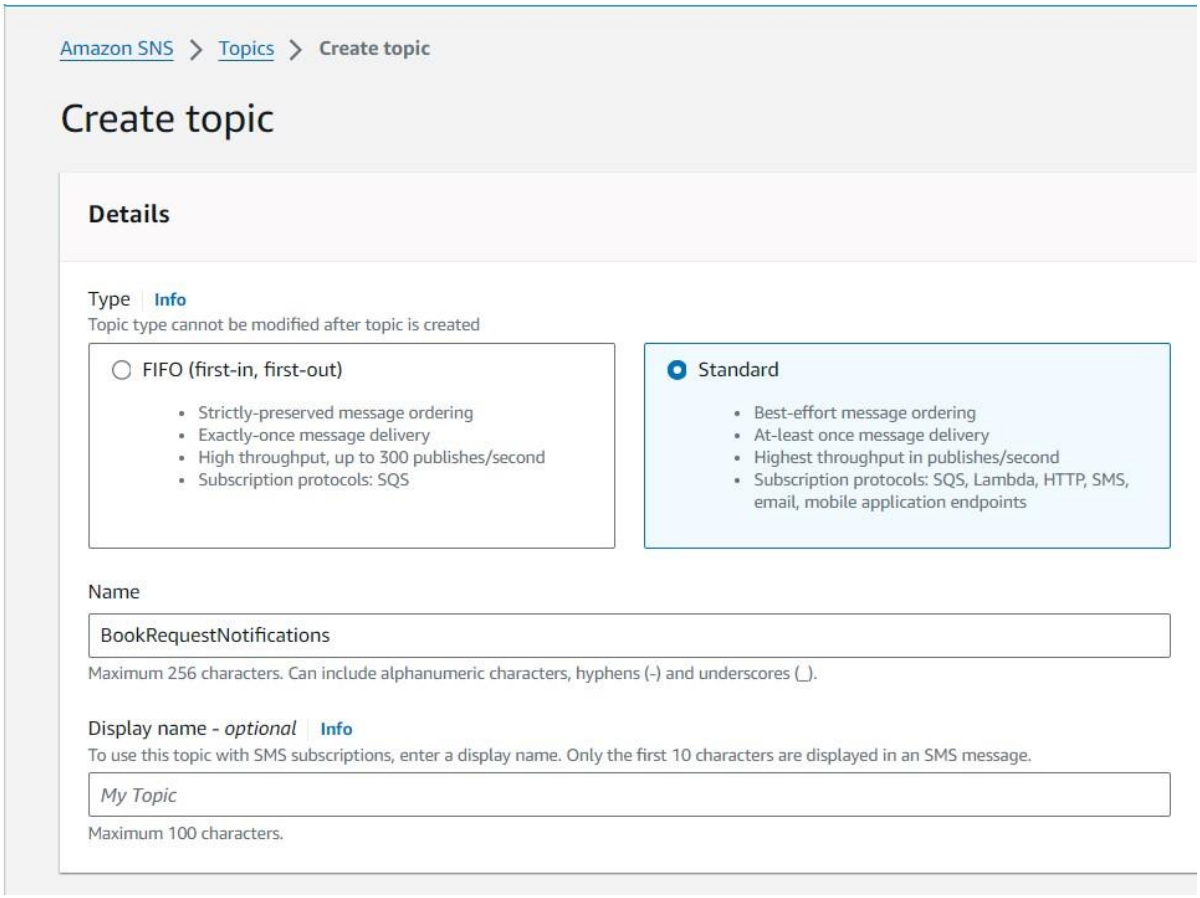


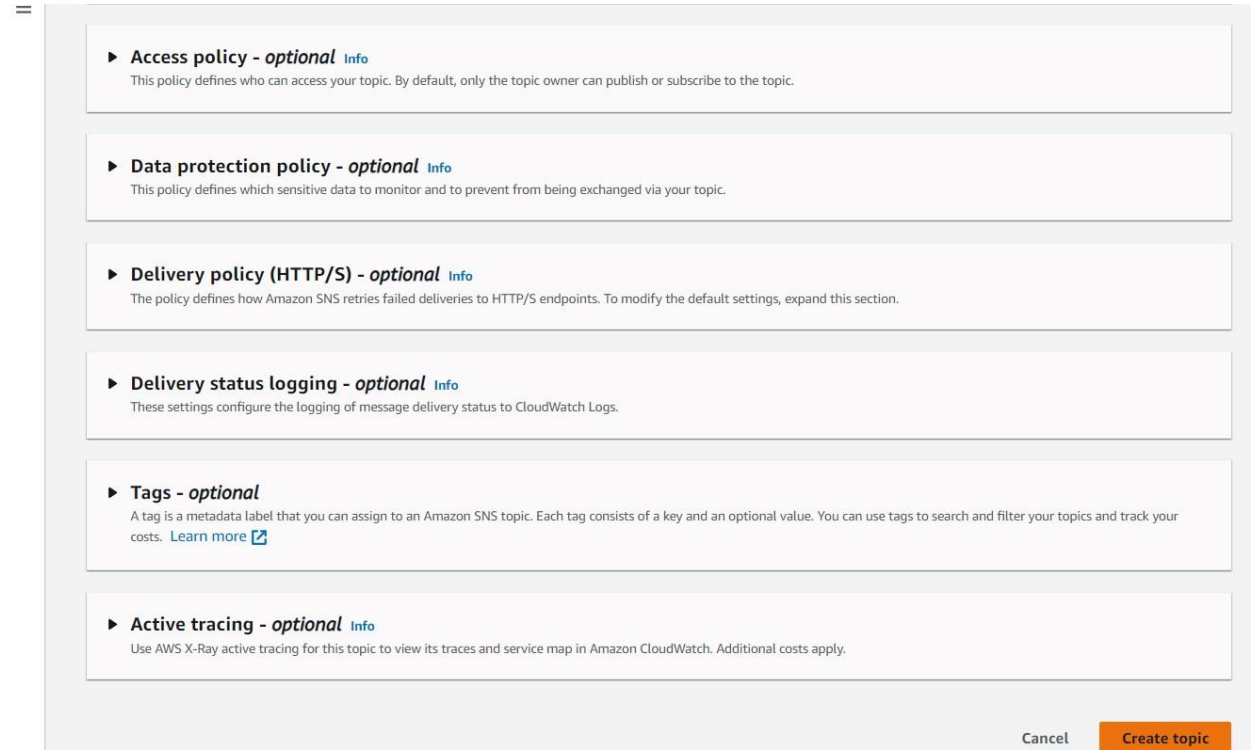


- Click on **Create Topic** and choose a name for the topic.

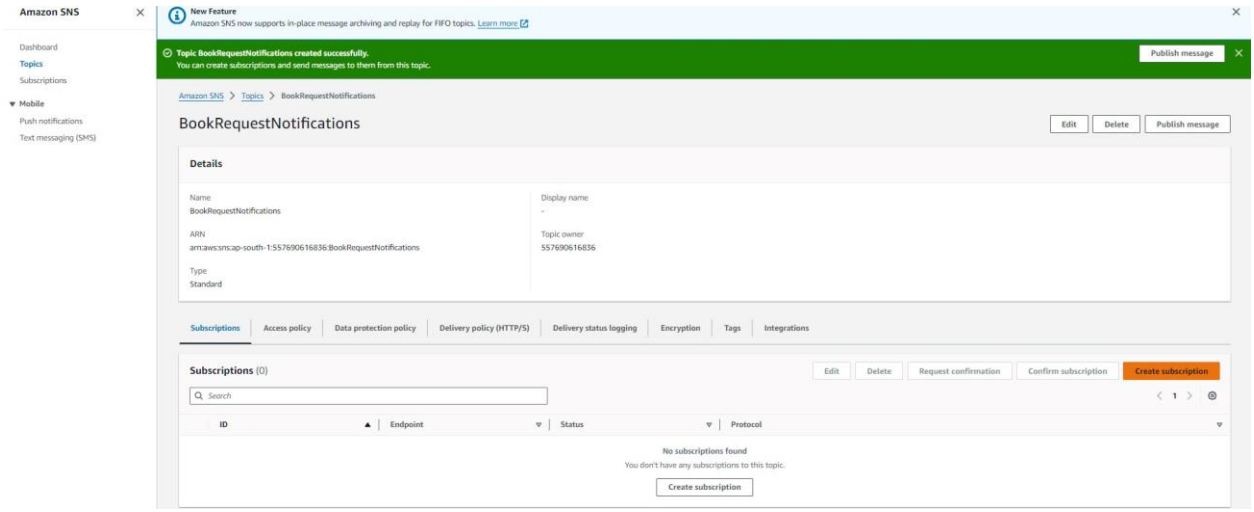


O Choose Standard type for general notification use cases and Click on Create Topic.





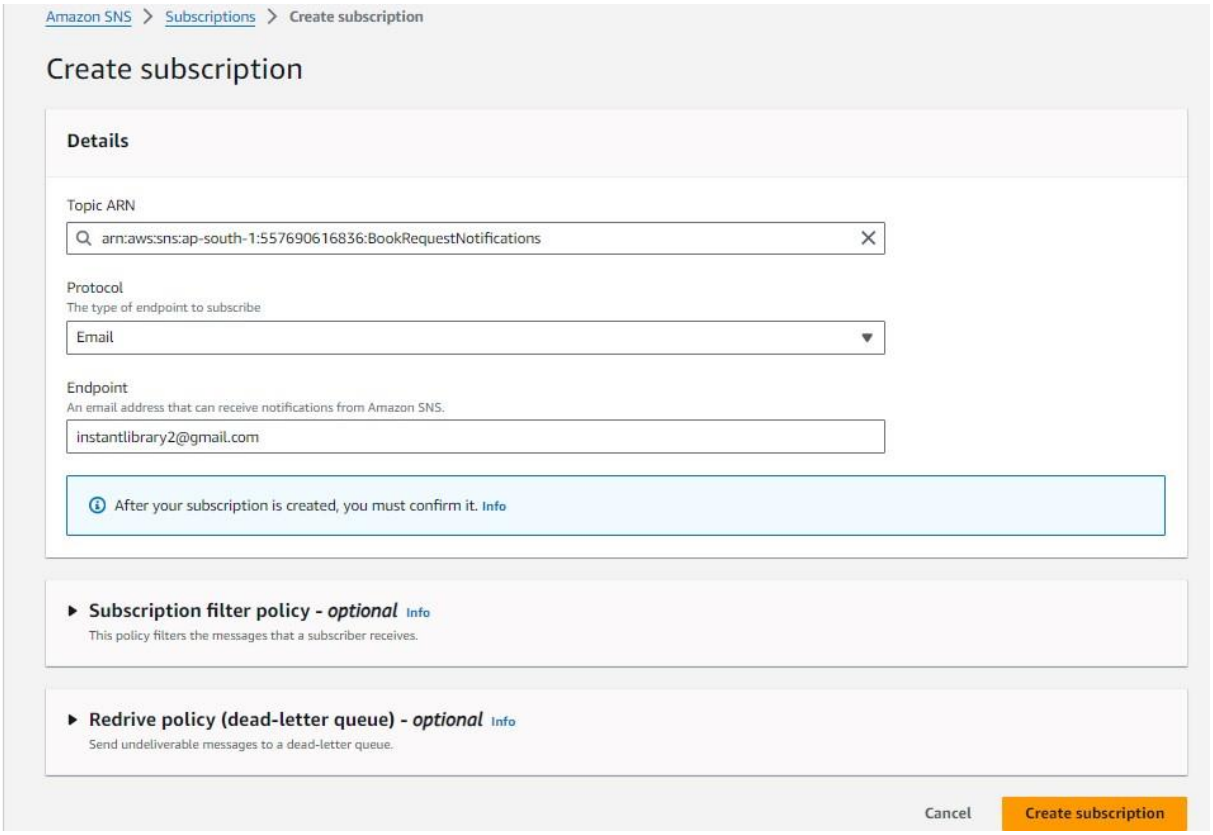
- Configure the SNS topic and note down the **Topic ARN**.

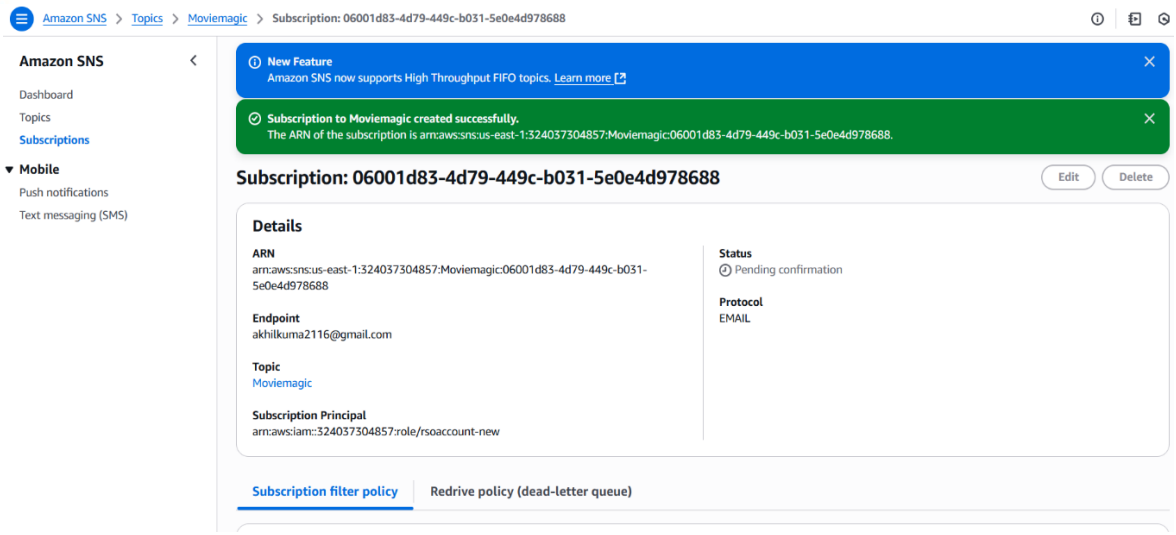


- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a movie booking is made.**

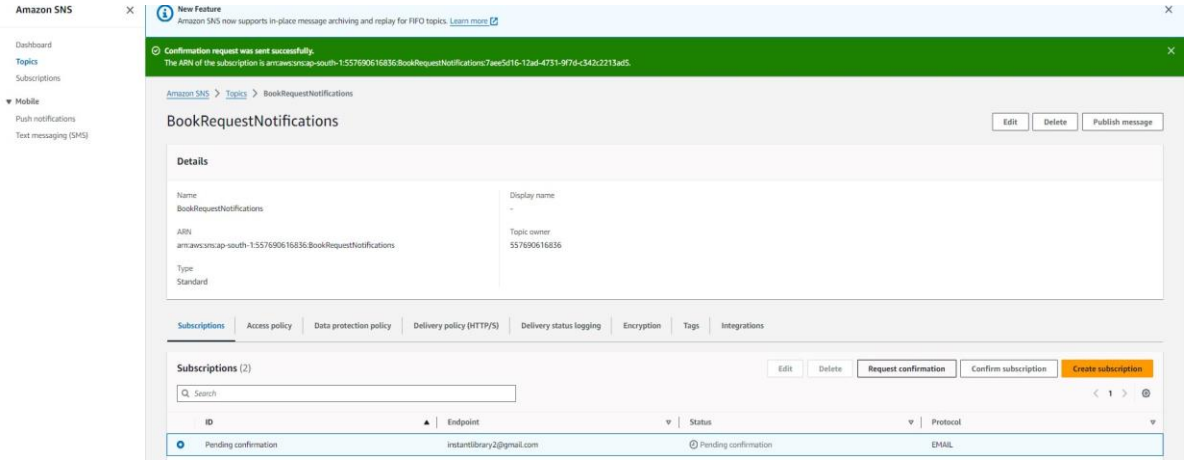
Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails

.

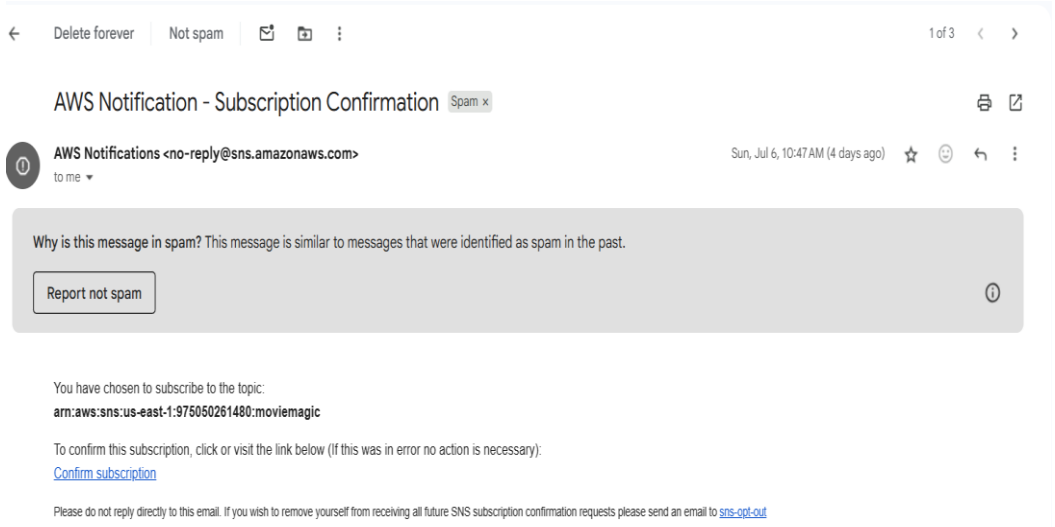




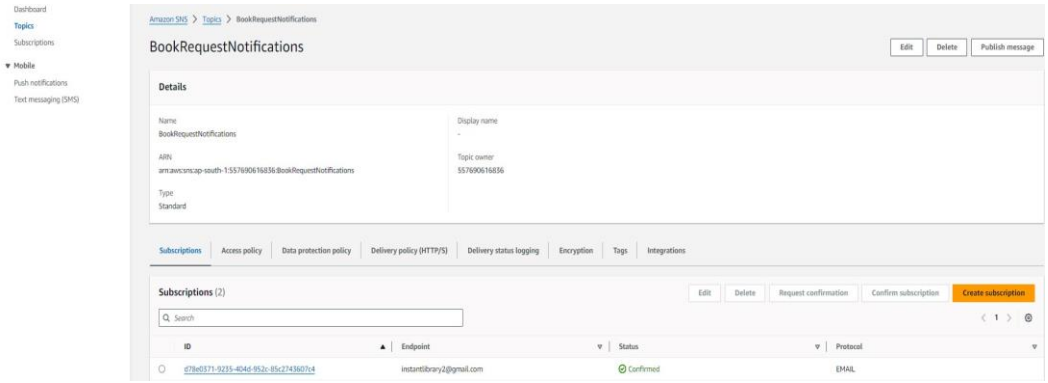
○ After subscr iption r equest for the mail confir mation



○ Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.

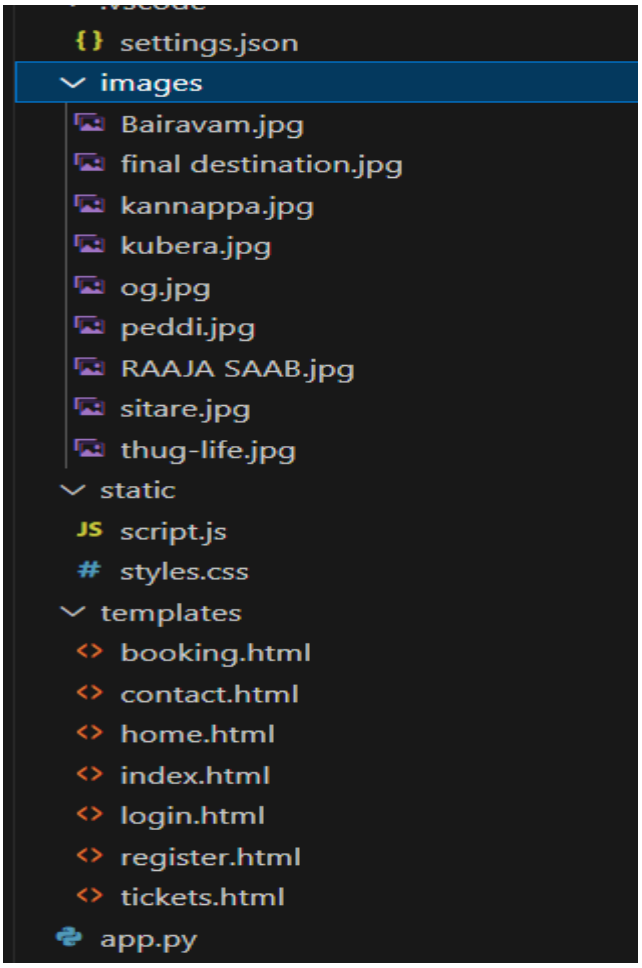


○ Successfully d one with the SNS mail subscr iption and setup, now stor e the AR N link.



Milestone 4:Backend Development and Application Setup

- **Activity 4.1: Develop the backend using Flask**
  - File Explorer Structure



Description: set up the INSTANT LIBRARY project with an app.py file, a static/ folder for assets, and a templates/ directory containing all required HTML pages like home, login, register, booking,tickets,contact pages (e.g., home.html,login.html), and css and js files (e.g., styles.css,script.js etc).

Description of the code :

- **Flask App Initialization**

Description of the code :

- **Flask App Initialization**

```
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
from werkzeug.security import generate_password_hash, check_password_hash
import boto3
from botocore.exceptions import ClientError
import os
```

Description: import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification

```
app = Flask(__name__)
```

Description: initialize the Flask application instance using Flask(__name__) to start building the web app.

- **Dynamodb Setup:**

```
dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION)
users_table = dynamodb.Table(DYNAMODB_USERS)
bookings_table = dynamodb.Table(DYNAMODB_BOOKINGS)
sns_client = boto3.client('sns', region_name=AWS_REGION)
```

Description: initialize the DynamoDB resource for the ap-south-1 region and set up access to the Users and Requests tables for storing user details and book requests.

- **SNS Connection**

```
# AWS Services setup
AWS_REGION = os.environ.get('AWS_REGION', 'us-east-1')
DYNAMODB_USERS = os.environ.get('DYNAMODB_USERS', 'MovieAppUsers')
DYNAMODB_BOOKINGS = os.environ.get('DYNAMODB_BOOKINGS', 'MovieAppBookings')
SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN', 'arn:aws:sns:us-east-1:975050261480:moviemagic:84b8
```

Description: Configure **SNS** to send notifications when a book request is submitted. Paste your stored ARN link in the sns_topic_arn space, along with the region_name where the SNS topic is created. Also, specify the chosen email service in SMTP_SERVER (e.g., Gmail, Yahoo, etc.) and enter the subscribed email in the SENDER_EMAIL section. Create an 'App password' for the email ID and store it in the SENDER_PASSWORD section.

● Routes for Web Pages

- **Home Route:**

```python
# Home route redirects to Registration page
@app.route('/')
def home():
    return redirect(url_for('register'))
```

Description: define the home route / to automatically redirect users to the register page when they access the base URL.

- **Register Route:**

```python
@app.route('/check-auth')
def check_auth():
    return jsonify({'authenticated': 'username' in session})

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        data = request.get_json()
        username = data['username']
        password = data['password']
        email = data['email']
        try:
            resp = users_table.get_item(Key={'username': username})
            if 'Item' in resp:
                return jsonify({'success': False, 'message': 'Username already exists'}), 400
            users_table.put_item(Item={
                'username': username,
                'password': generate_password_hash(password),
                'email': email
            })
            return jsonify({'success': True})
        except ClientError as e:
            return jsonify({'success': False, 'message': str(e)}), 500
    return render_template('register.html')
```

Description: define /register route to validate registration form fields, hash the user password using decrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

:

- **login Route (GET/POST):**

```
def login():
    if request.method == 'POST':
        data = request.get_json()
        username = data['username']
        password = data['password']
        try:
            resp = users_table.get_item(Key={'username': username})
            user = resp.get('Item')
            if user and check_password_hash(user['password'], password):
                session['username'] = username
                return jsonify({'success': True})
            else:
                return jsonify({'success': False, 'message': 'Invalid username or password'}), 401
        except ClientError as e:
            return jsonify({'success': False, 'message': str(e)}), 500
    return render_template('login.html')
```

Description: define /login route to validate user credentials against DynamoDB, check the password using Bcrypt, update the login count on successful authentication, and redirect users to the home page

- **Home Page**

```
@app.route('/home')
def home():
    if 'username' not in session:
        flash('Please login first', 'error')
        return redirect(url_for('login'))
    return render_template('home.html', movies=movies)
```

Description: define /home-page to render the main homepage, / it will redirect to the homepage after the user logins

- **Booking Form:**

```
@app.route('/booking', methods=['POST'])
def make_booking():
    if 'username' not in session:
        return jsonify({'success': False, 'message': 'Not authenticated'}), 401
    data = request.get_json()
    movie_id = int(data['movie_id'])
    movie = movies.get(movie_id)
    if not movie:
        return jsonify({'success': False, 'message': 'Movie not found'}), 404
    showtime = data['showtime']
    if showtime not in movie['showtimes']:
        return jsonify({'success': False, 'message': 'Invalid showtime'}), 400
    seats = data['seats']
    if not isinstance(seats, int) or seats <= 0:
        return jsonify({'success': False, 'message': 'Invalid number of seats'}), 400

    booking_data = {
        'username': session['username'],
        'movie_id': str(movie_id),
        'movie_title': movie['title'],
        'showtime': showtime,
        'seats': seats
    }

    try:
        # Store booking in DynamoDB
        bookings_table.put_item(Item=booking_data)

        # Send SNS notification
        send_booking_confirmation(
            username=session['username'],
            movie_title=movie['title'],
            showtime=showtime,
            seats=seats
        )

        return jsonify({'success': True})
    except ClientError as e:
        return jsonify({'success': False, 'message': str(e)}), 500
```

Description: define /request-form route to capture movie bookings of users, store the request in DynamoDB, send a thank-you email to the user, notify the admin, and confirm submission with a success message.

Tickets Route:

```
@app.route('/tickets')
def tickets():
    if 'username' not in session:
        return redirect(url_for('login'))
    try:
        resp = bookings_table.scan(
            FilterExpression='username = :username',
            ExpressionAttributeValues={':username': session['username']}
        )
        user_bookings = resp.get('Items', [])
    except Exception:
        user_bookings = []
    return render_template('tickets.html', bookings=user_bookings)

@app.route('/contact')
def contact():
    return render_template('contact.html')
```

Description: it navigates to selected tickets section

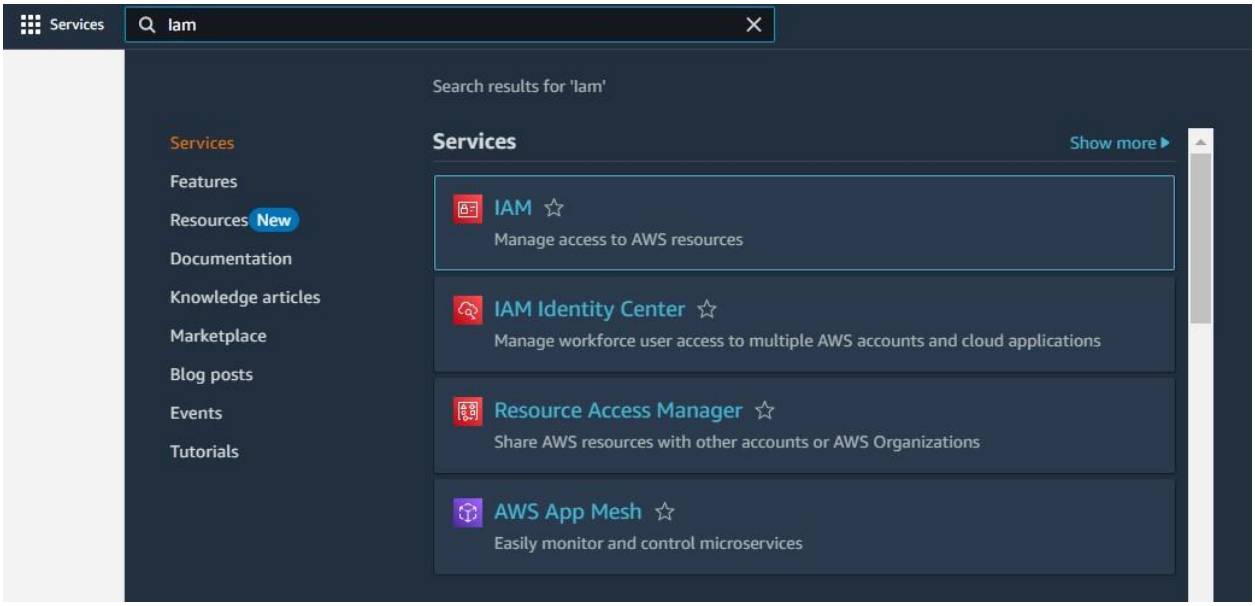
Deployment Code:

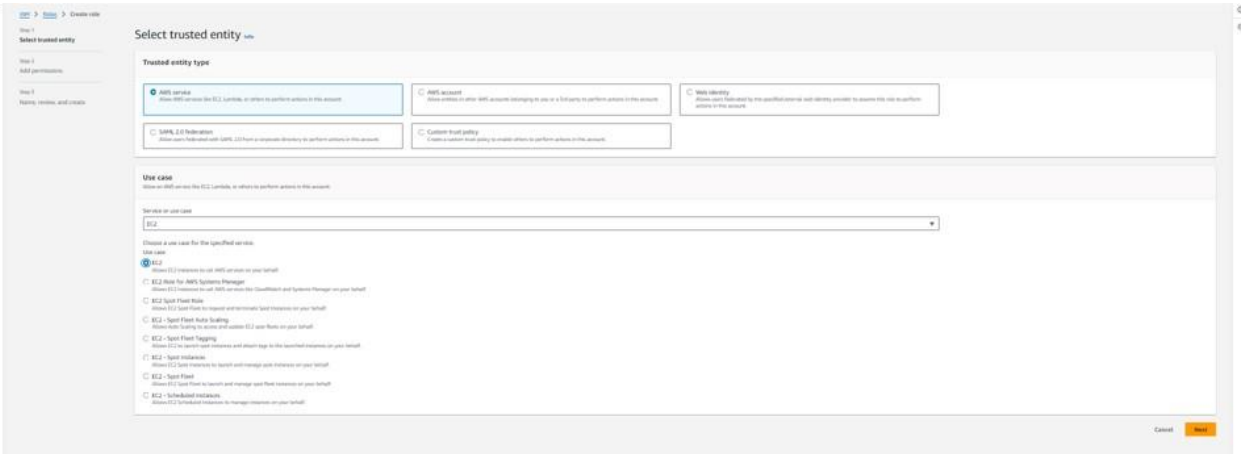
```
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

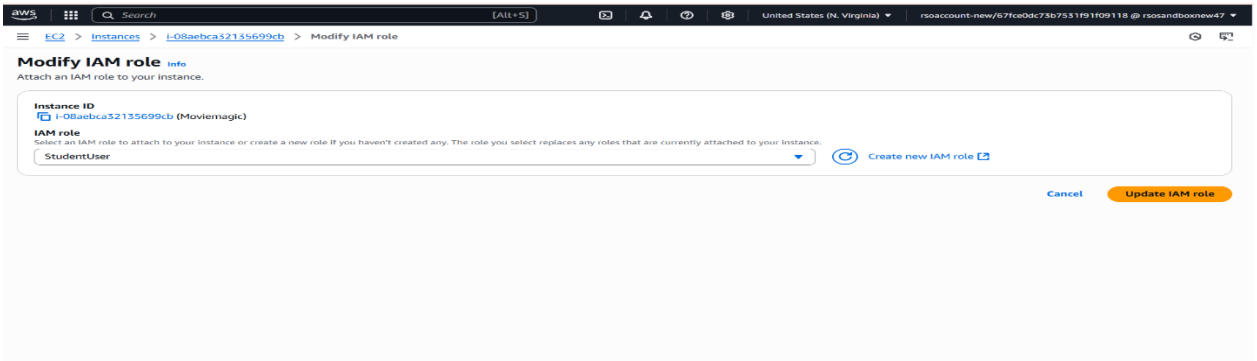
Description: start the Flask server to listen on all network interfaces (0.0.0.0) at port 80 with debug mode enabled for development and testing.

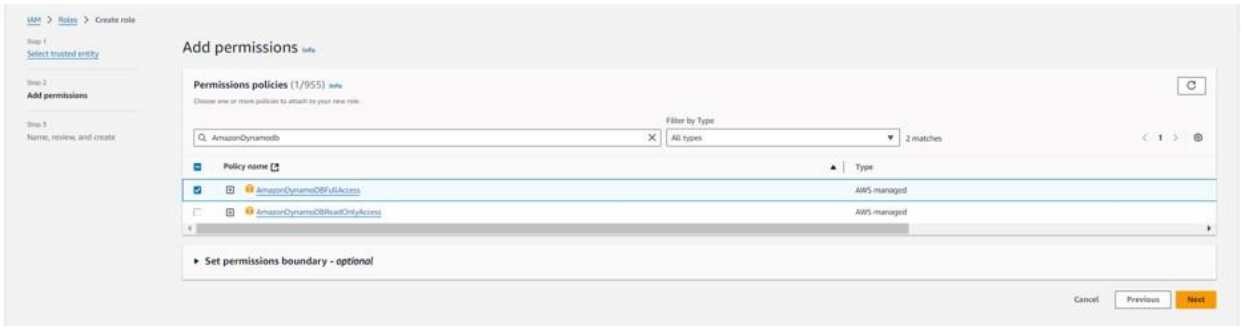
Milestone 5: IAM Role Setup

- **Activity 5.1:Create IAM Role.**
    - In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.


Services
Iam
Search results for 'Iam'
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Services
Show more
IAM
Manage access to AWS resources
IAM Identity Center
Manage workforce user access to multiple AWS accounts and cloud applications
Resource Access Manager
Share AWS resources with other accounts or AWS Organizations
AWS App Mesh
Easily monitor and control microservices


Select trusted entity
Trusted entity type
Use case

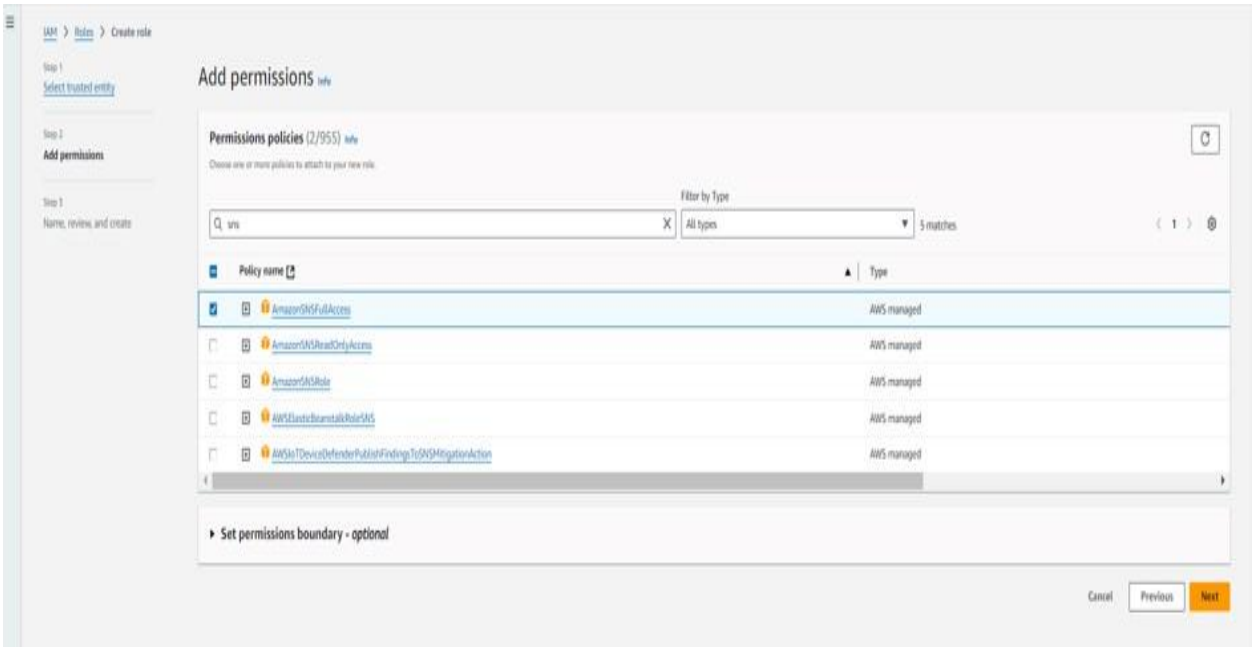

EC2 > Instances > i-08aebca32135699cb > Modify IAM role
Modify IAM role
Info
Attach an IAM role to your instance.
Instance ID
i-08aebca32135699cb (Moviemagic)
IAM role
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.
StudentUser
Create new IAM role
Cancel
Update IAM role

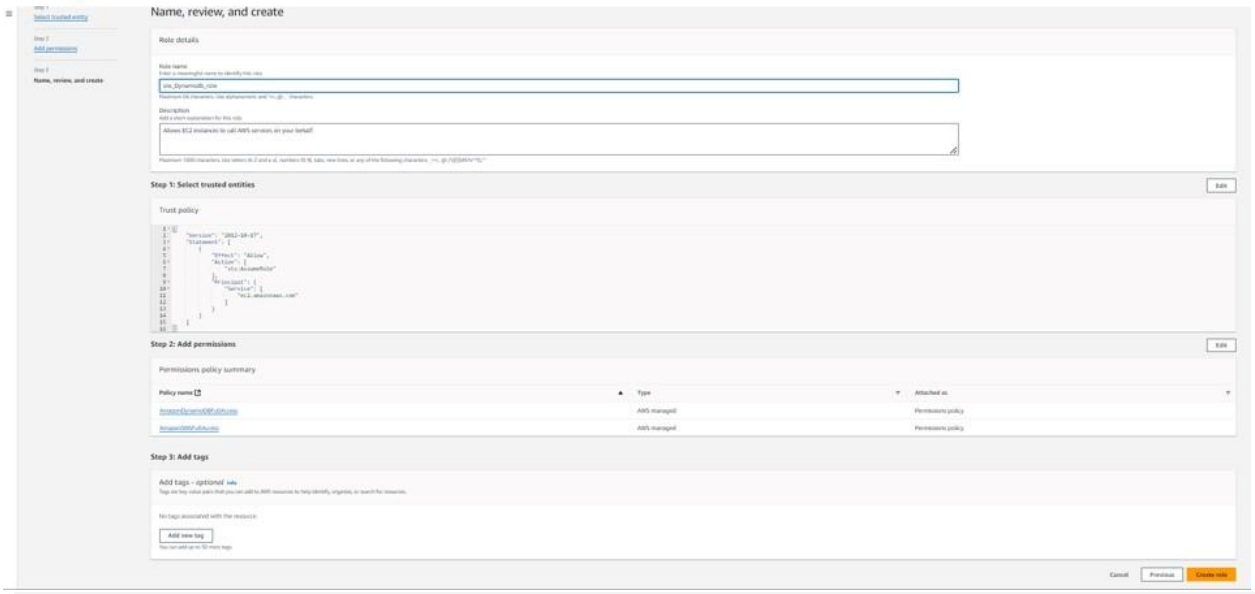


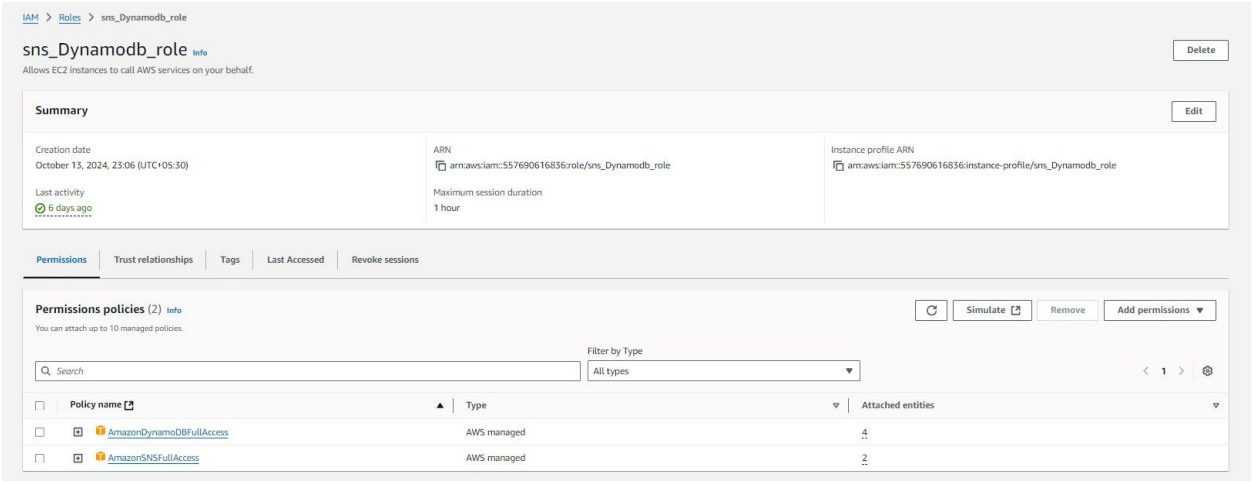
- **Activity 5.2: Attach Policies.**

  Attach the following policies to the role:

- AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
- AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.



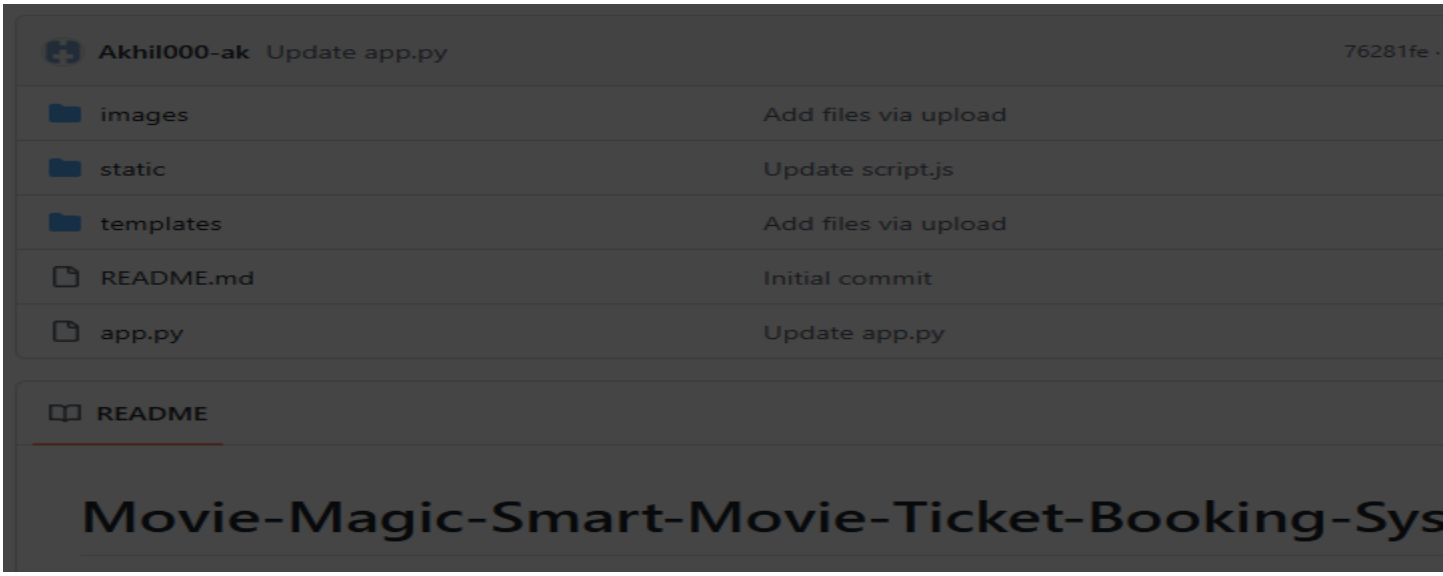

Name, review, and create
Role details
Role name
sns_Dynamodb_role
Description
Allows EC2 instances to call AWS services on your behalf.
Step 1: Select trusted entities
Edit
Trust policy
Step 2: Add permissions
Edit
Permissions policy summary
Policy name
Type
Attached as
Step 3: Add tags
Add tags - optional Info
No tags associated with the resource.
Add new tag
Cancel
Previous
Create role

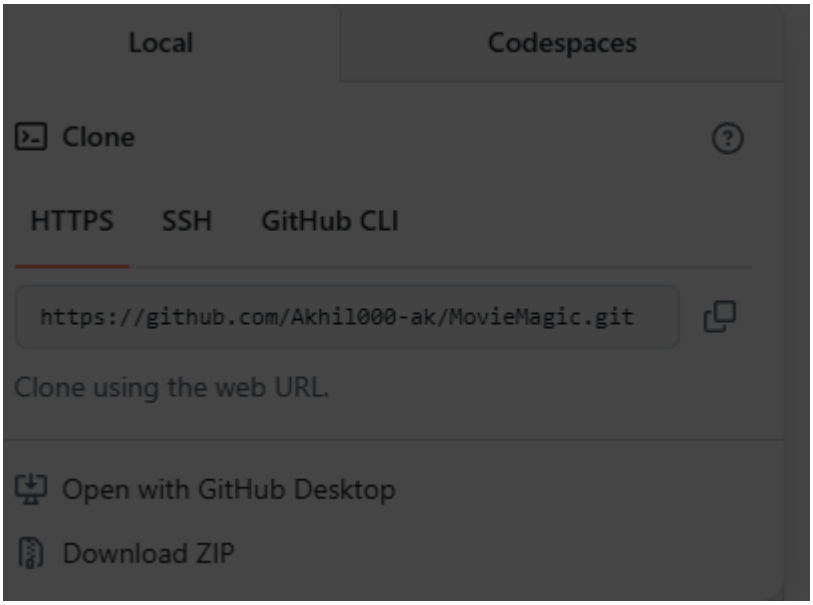

IAM > Roles > sns_Dynamodb_role
sns_Dynamodb_role Info
Allows EC2 instances to call AWS services on your behalf.
Delete
Summary
Edit
Creation date
October 13, 2024, 23:06 (UTC+05:30)
ARN
arn:aws:iam::557690616836:role/sns_Dynamodb_role
Instance profile ARN
arn:aws:iam::557690616836:instance-profile/sns_Dynamodb_role
Last activity
6 days ago
Maximum session duration
1 hour
Permissions
Trust relationships
Tags
Last Accessed
Revoke sessions
Permissions policies (2) Info
You can attach up to 10 managed policies.
Simulate
Remove
Add permissions
Filter by Type
Search
All types
Policy name
Type
Attached entities
AmazonDynamoDBFullAccess
AWS managed
4
AmazonSNSFullAccess
AWS managed
2

-

Milestone 6: EC2 Instance Setup

- Load your Flask app and Html files into GitHub repository.

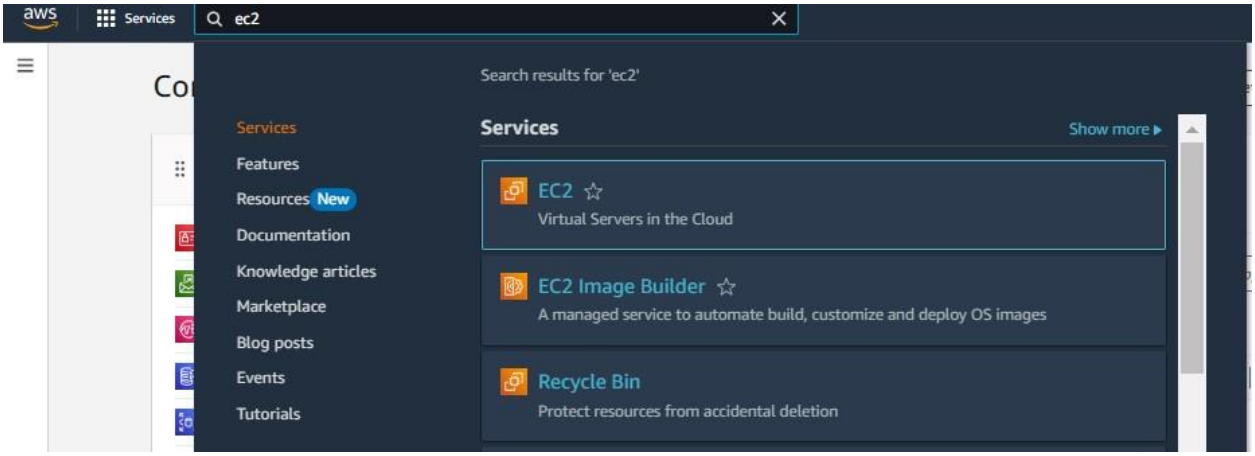




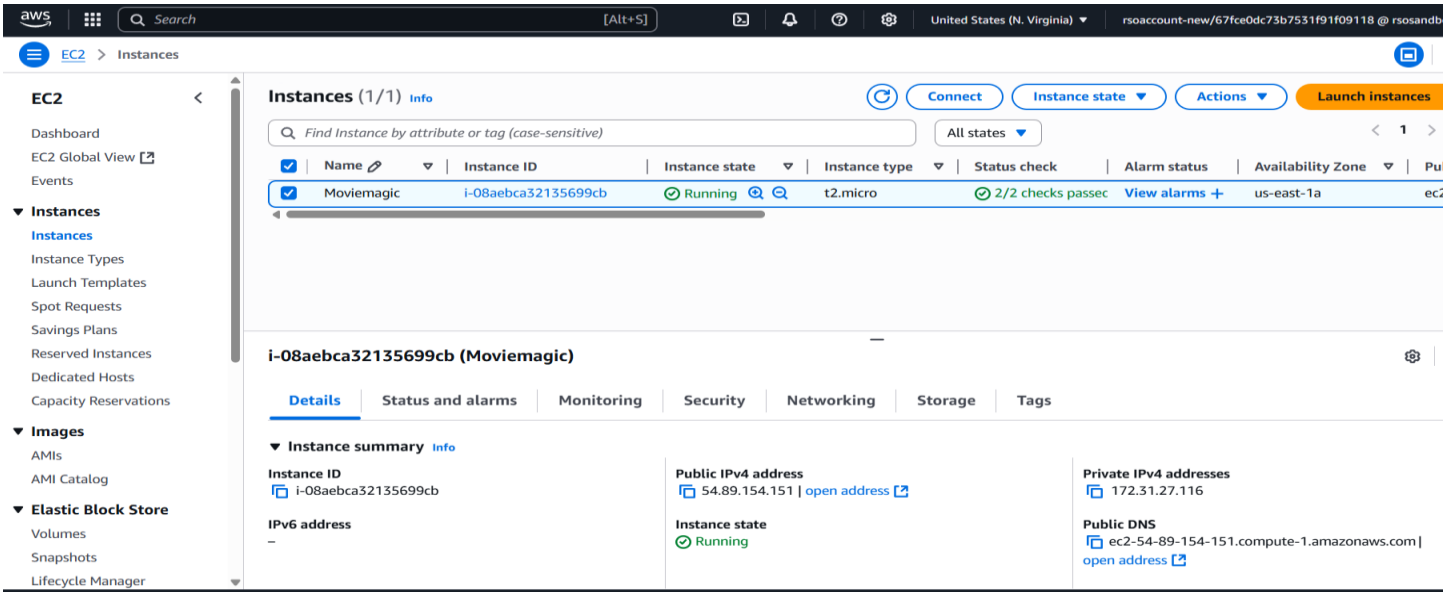
● Activity 6.1: Launch an EC2 instance to host the Flask application.

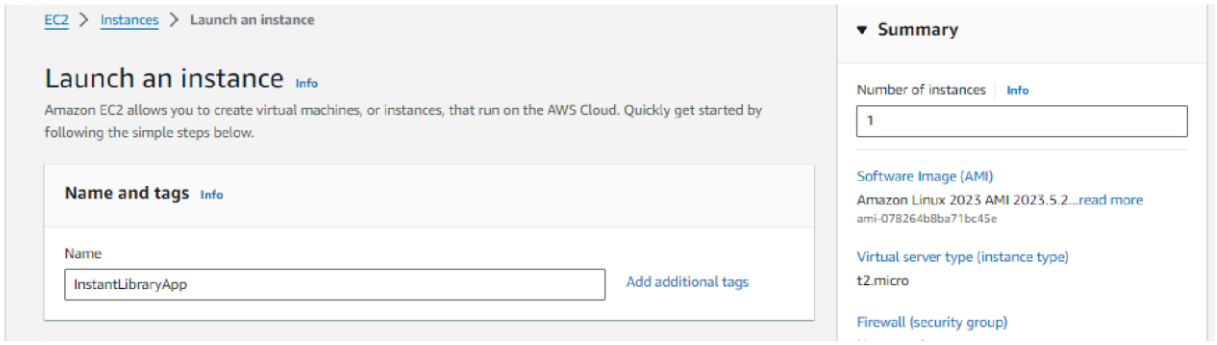
● Launch EC2 Instance

○ In the AWS Console, navigate to EC2 and launch a new instance.

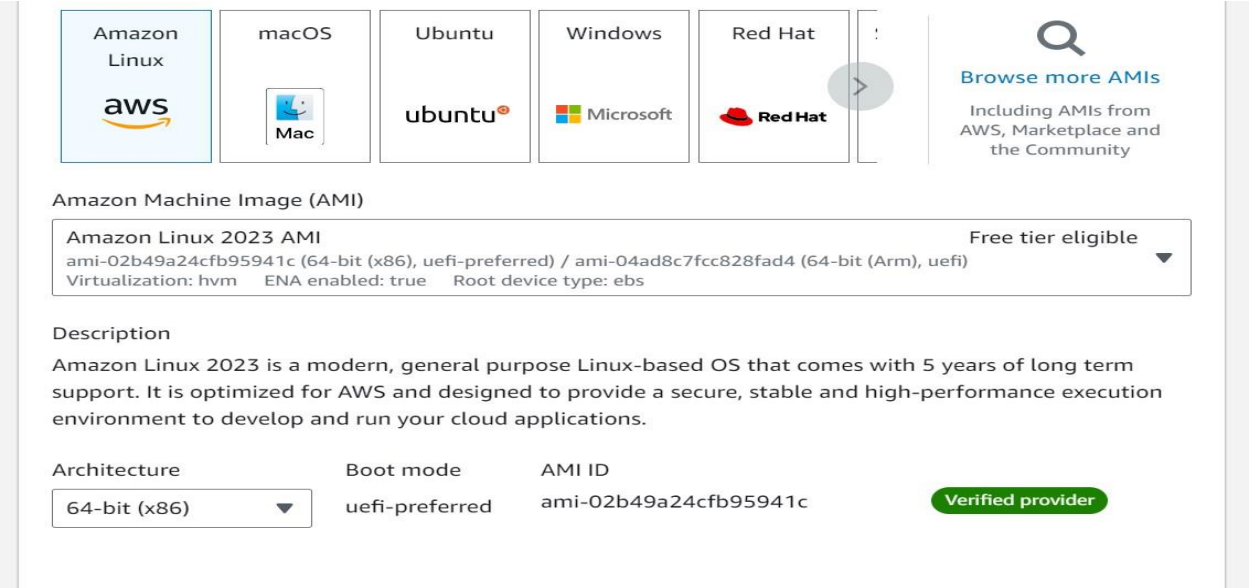


- Click on Launch instance to launch EC2 instance



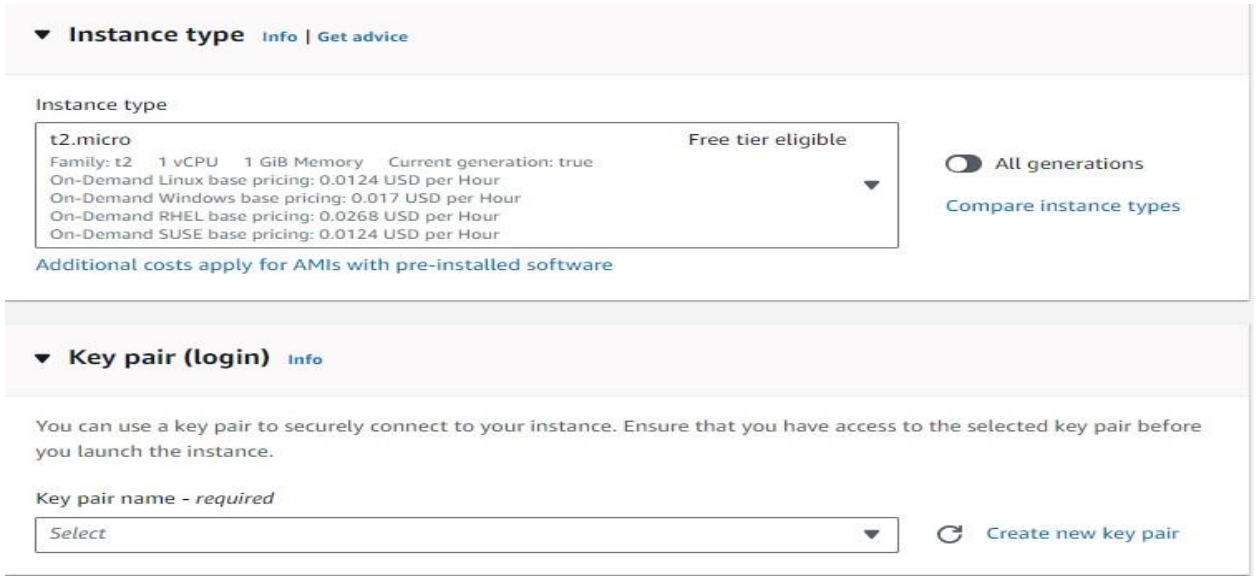


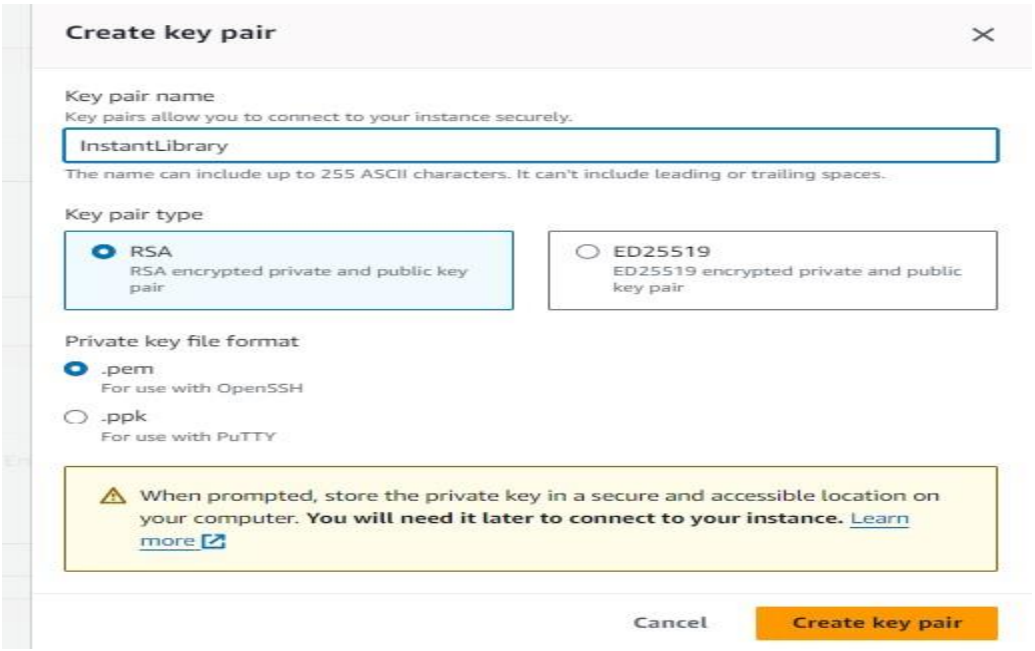
- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



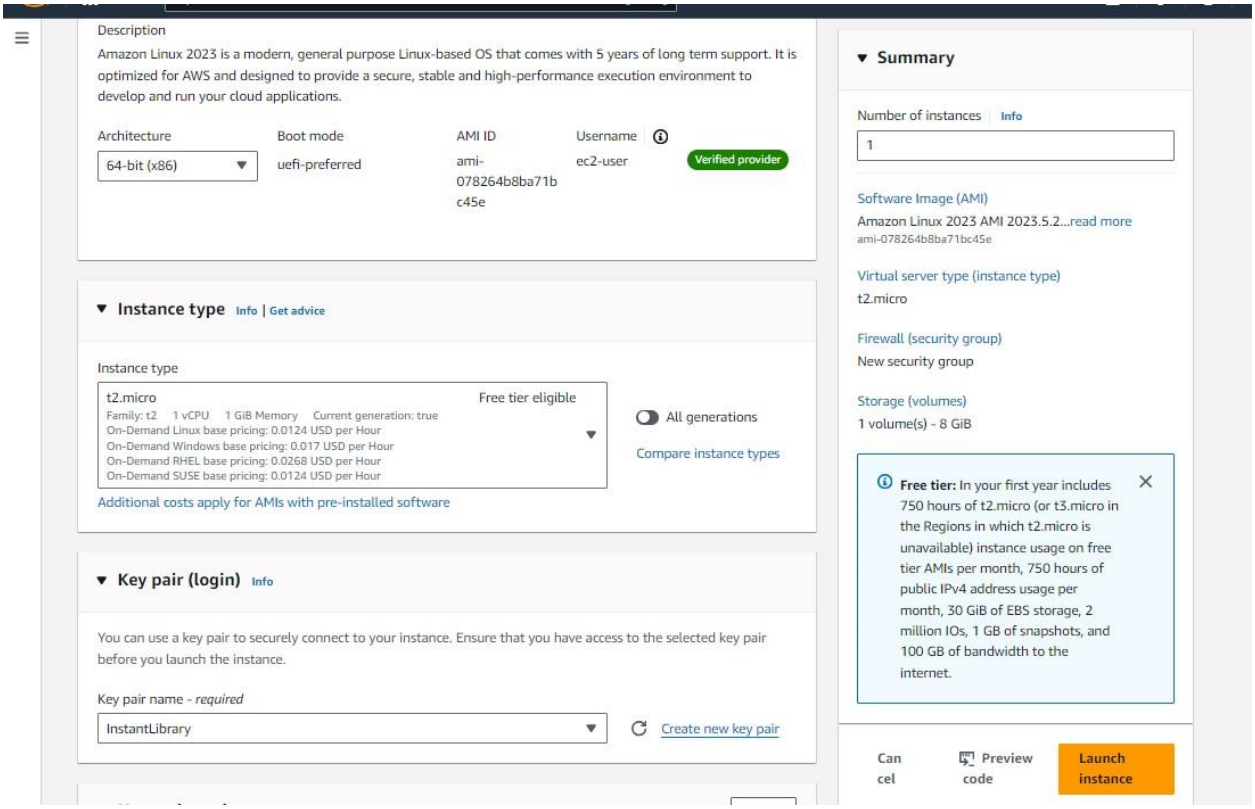
- Create and download the key pair for Server access.

● Create and download the key pair for Server access.

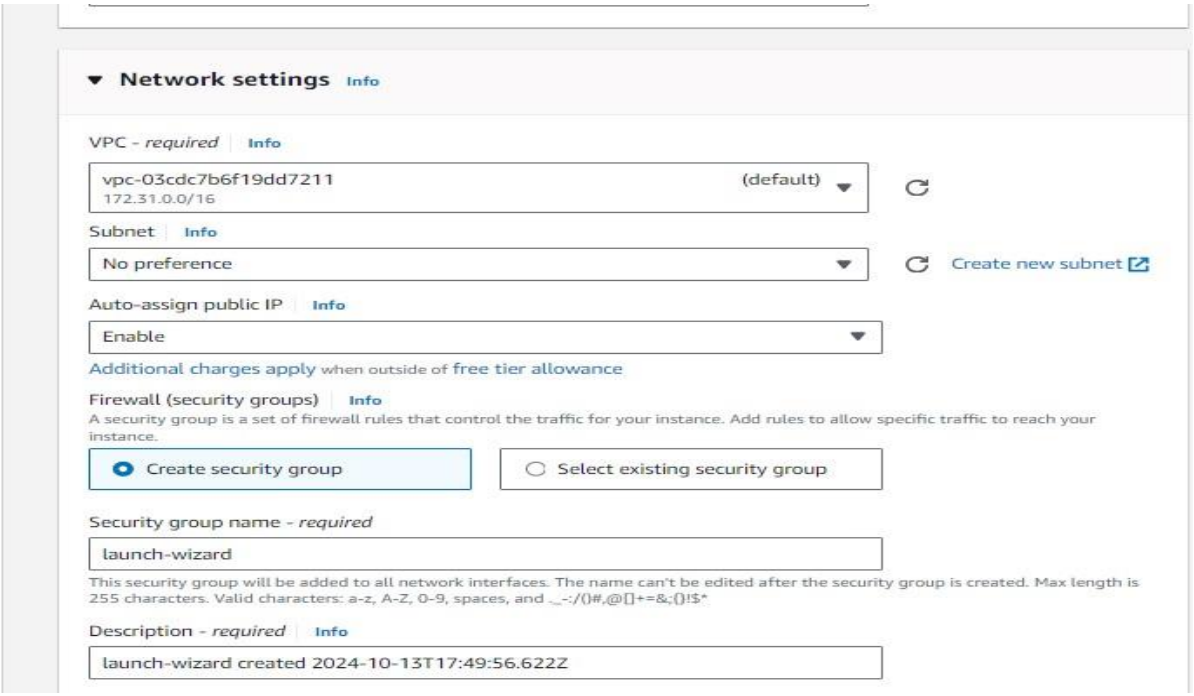


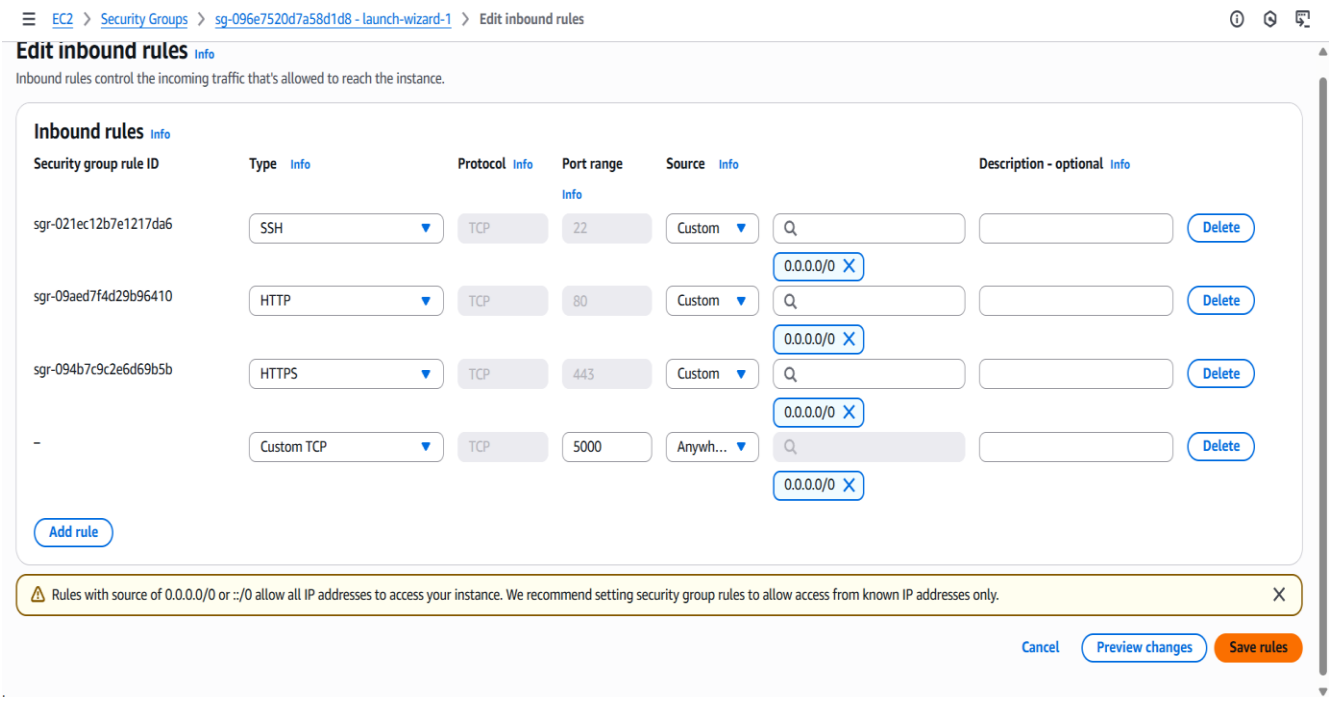


InstantLibrary.pem

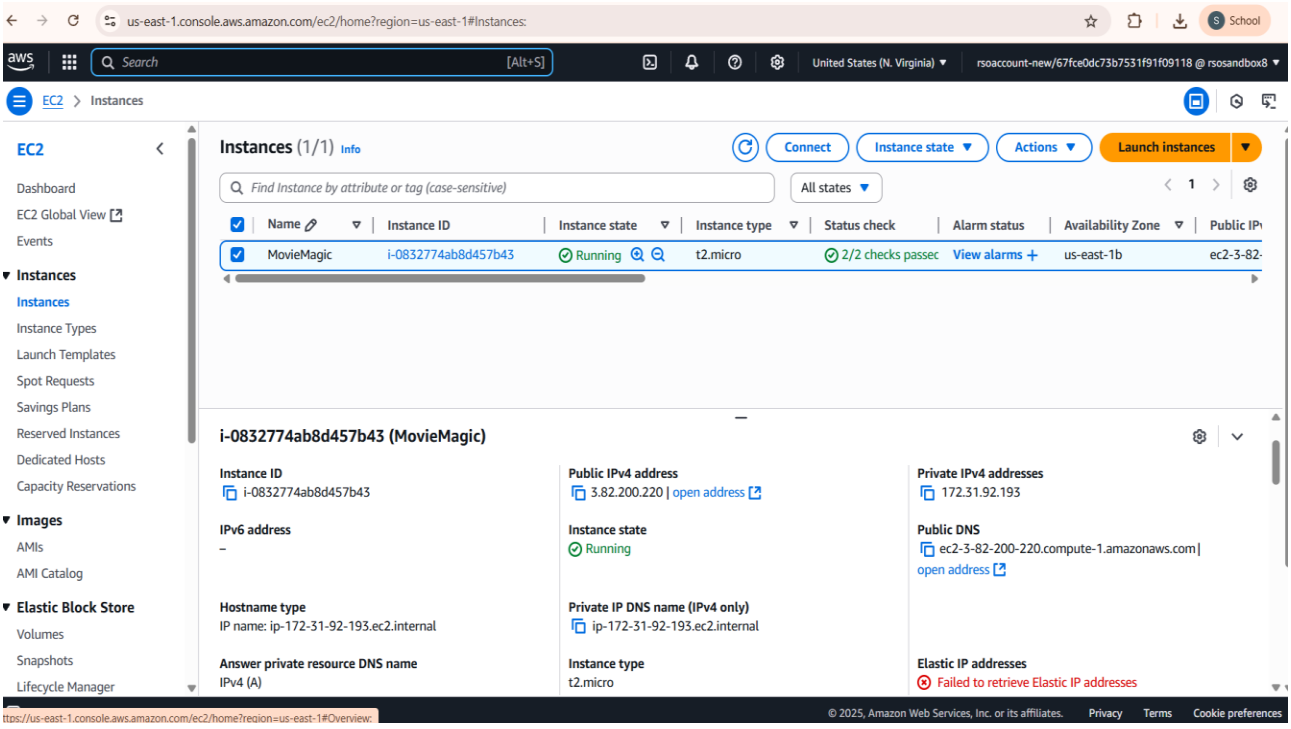


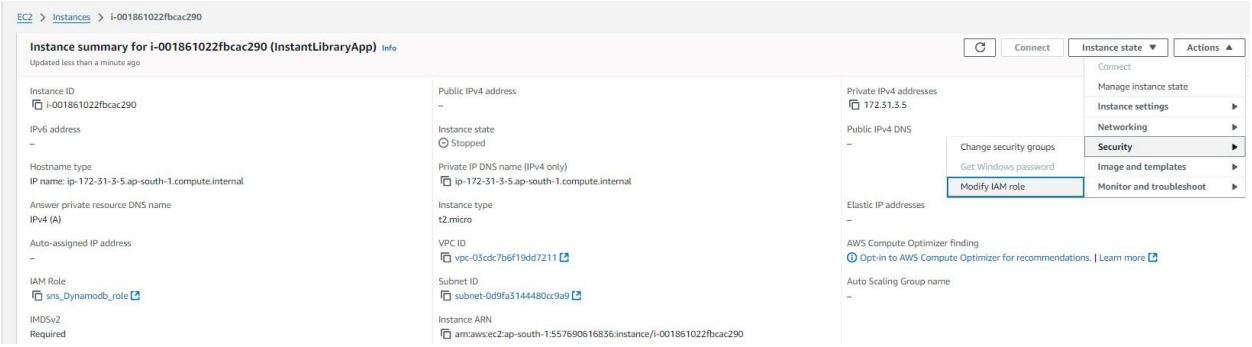
Activity 6.2:Configure security groups for HTTP, and SSH access.

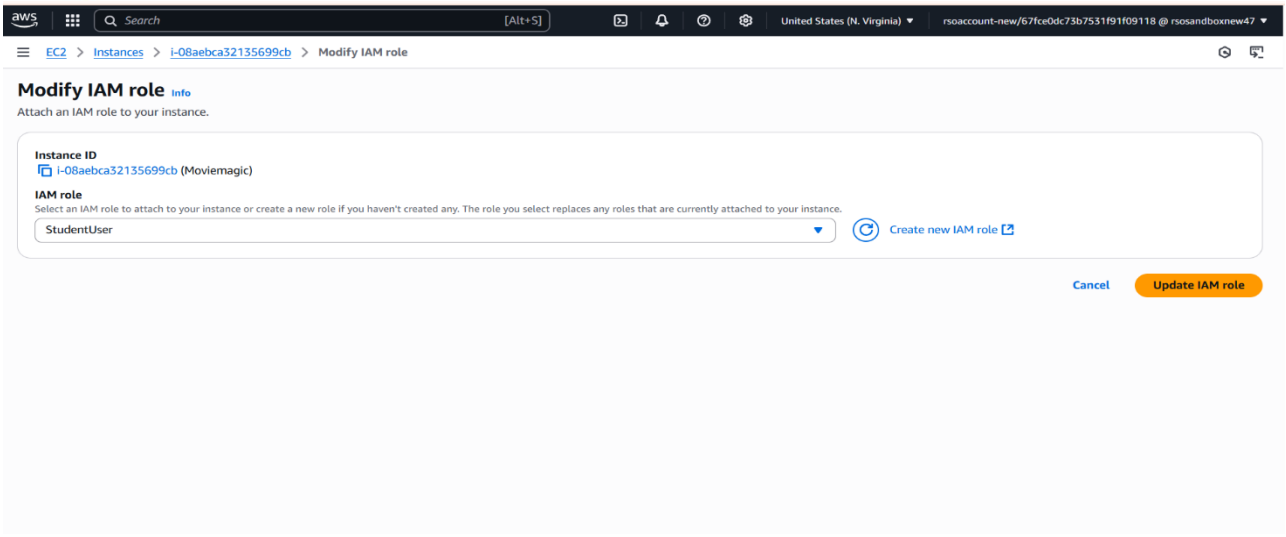




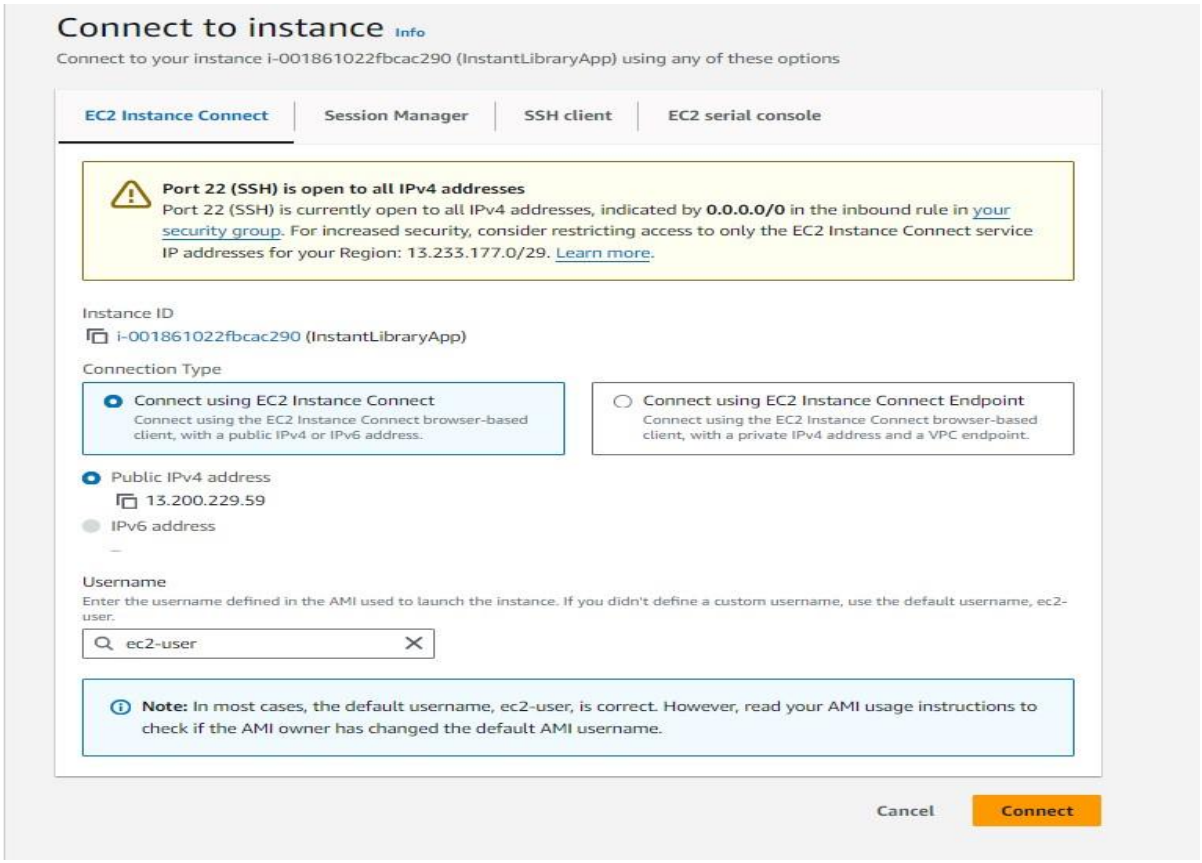
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the IAM role is connected, navigate to the **EC2** section in the **AWS Management Console**. Select the **EC2 instance** you wish to connect to. At the top of the **EC2 Dashboard**, click the **Connect** button. From the connection methods presented, choose **EC2 Instance Connect**. Finally, click **Connect** again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

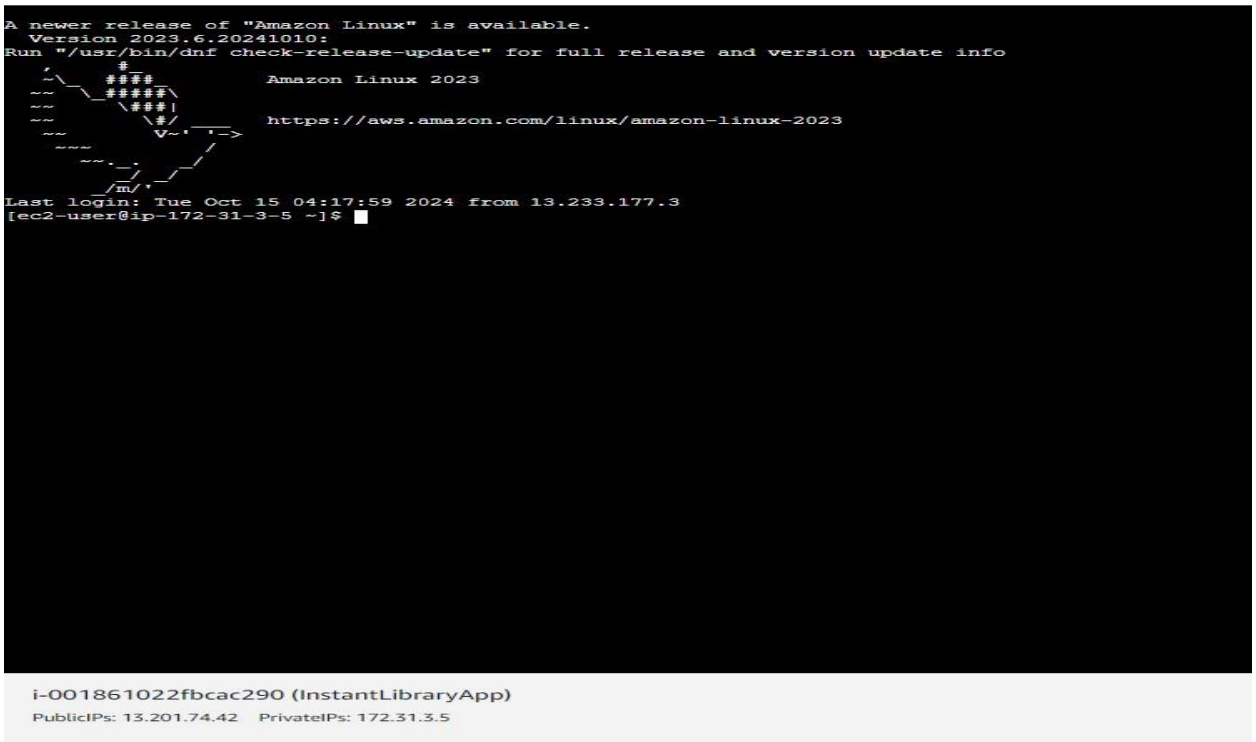





aws
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce0dc73b7531f91f09118 @ rsosandboxnew47
EC2 > Instances > i-08aebca32135699cb > Modify IAM role
Modify IAM role Info
Attach an IAM role to your instance.
Instance ID
i-08aebca32135699cb (Moviemagic)
IAM role
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.
StudentUser
Create new IAM role
Cancel
Update IAM role

● Now connect the EC2 with the files





Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

```
sudo yum update -y
sudo yum install python3 git
sudo pip3 install flask boto3
```

Verify Installations:

```
flask --version
git --version
```

Activity 7.2:Clone Your Flask Project from GitHub

Clone your project repository from GitHub into the EC2 instance using Git.

Run: 'git clone https://github.com/Akhil000-ak/MovieMagic.git

- This will download your project to the EC2 instance.

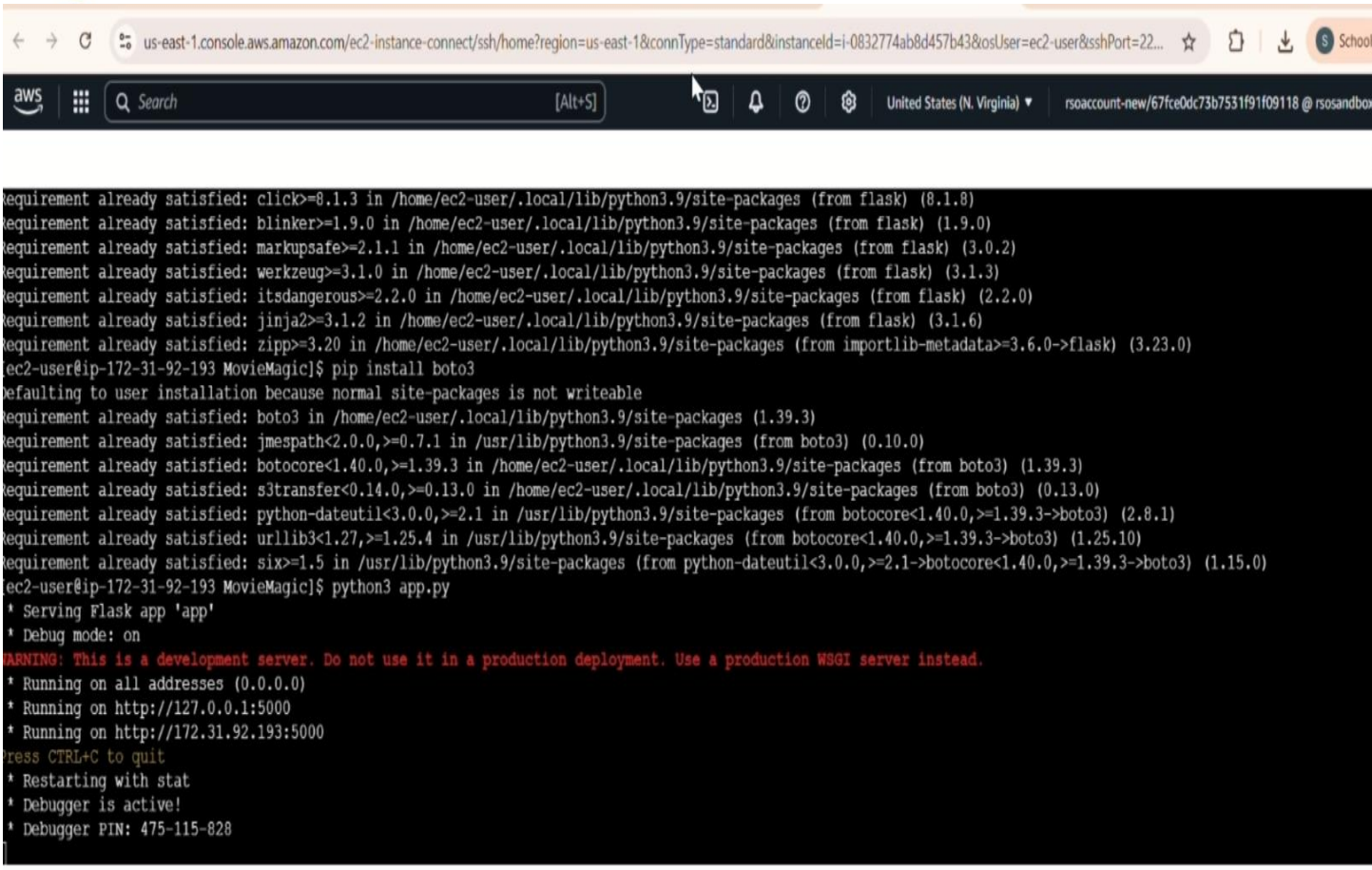
To navigate to the project directory, run the following command:

```
cd MovieMagic
```

Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application

```
sudo flask run --host=0.0.0.0 --port=80
```



Verify the Flask app is running: http://your-ec2-public-ip

- Run the Flask app on the EC2 instance

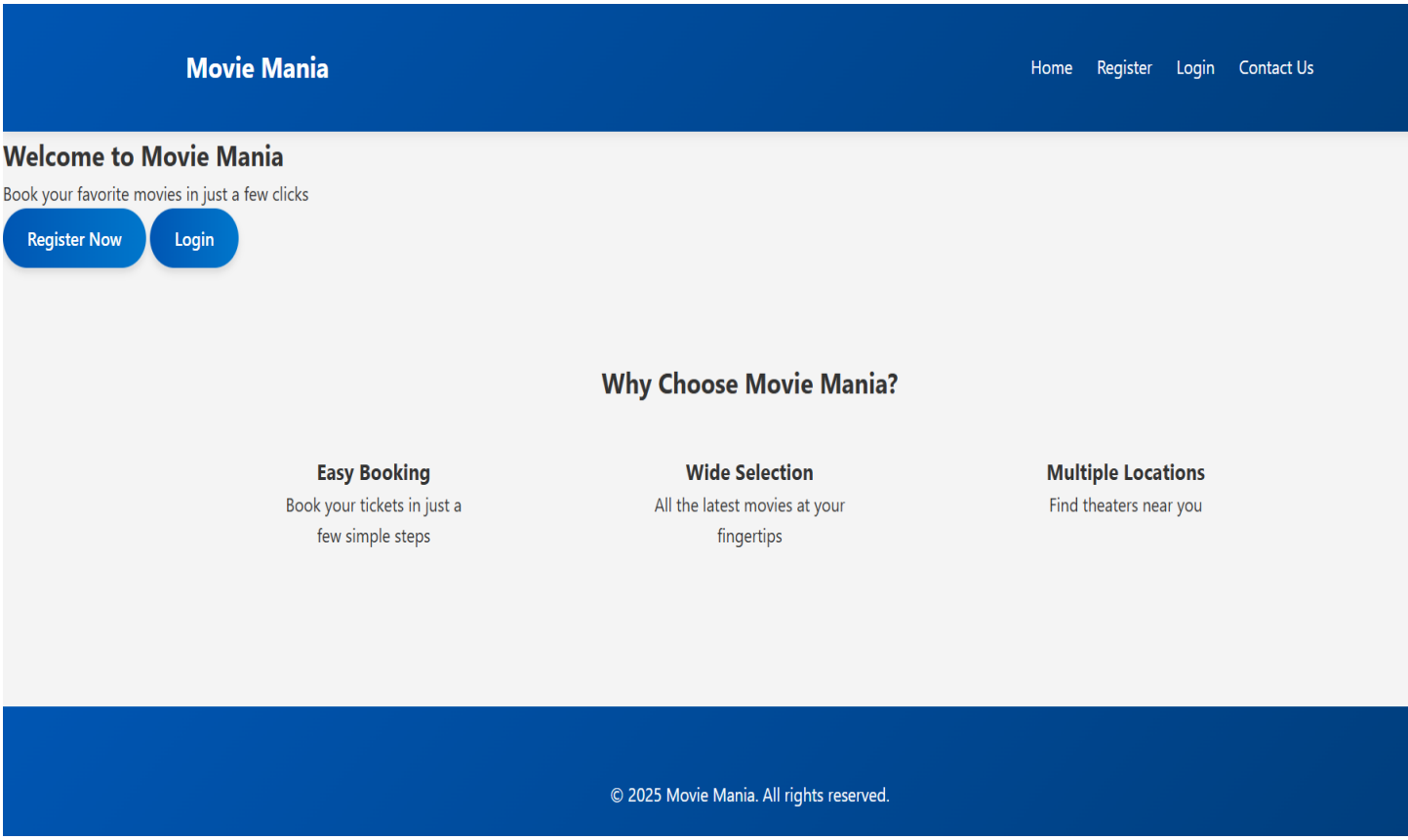
Access the website through:

Public ip: https://3.82.200.220:5000

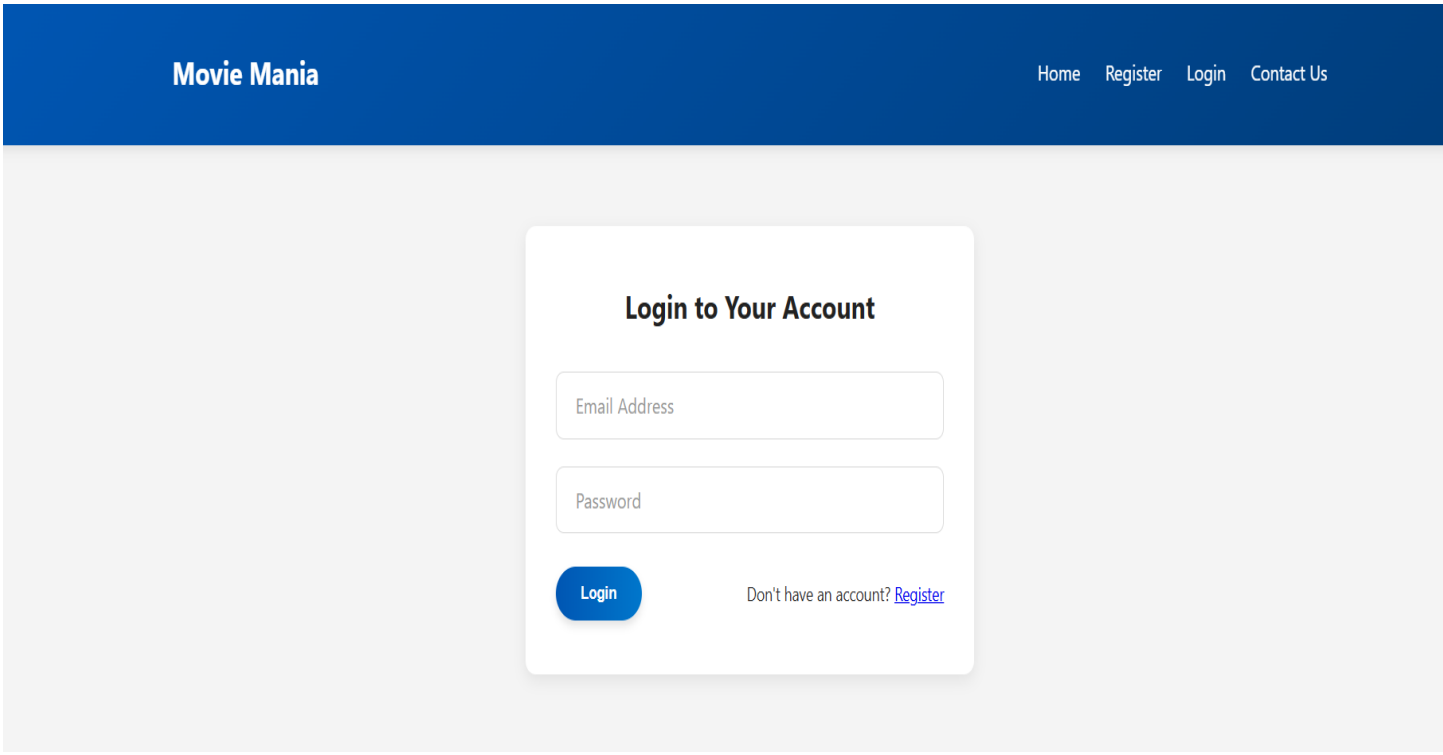
Milestone 8: Testing and Deployment

- **Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.**

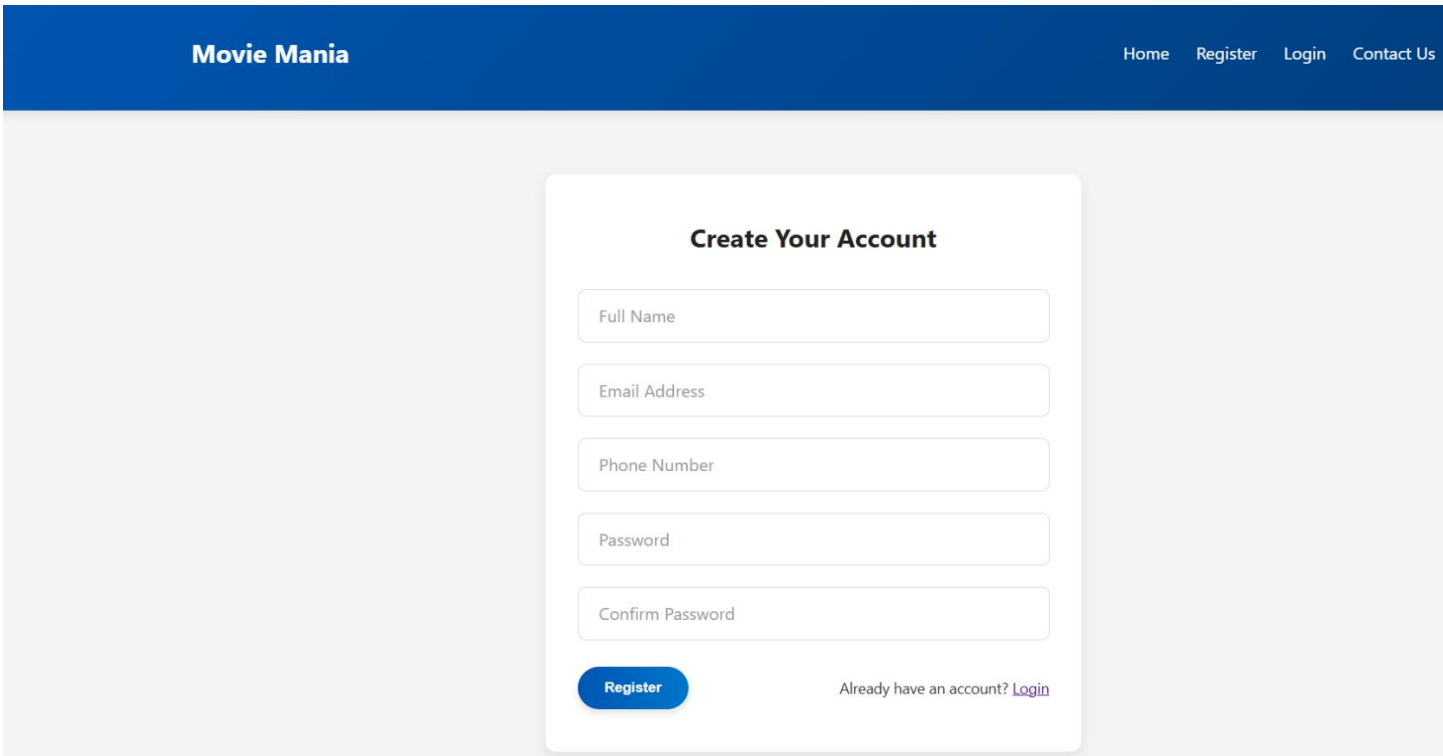
Index Page:



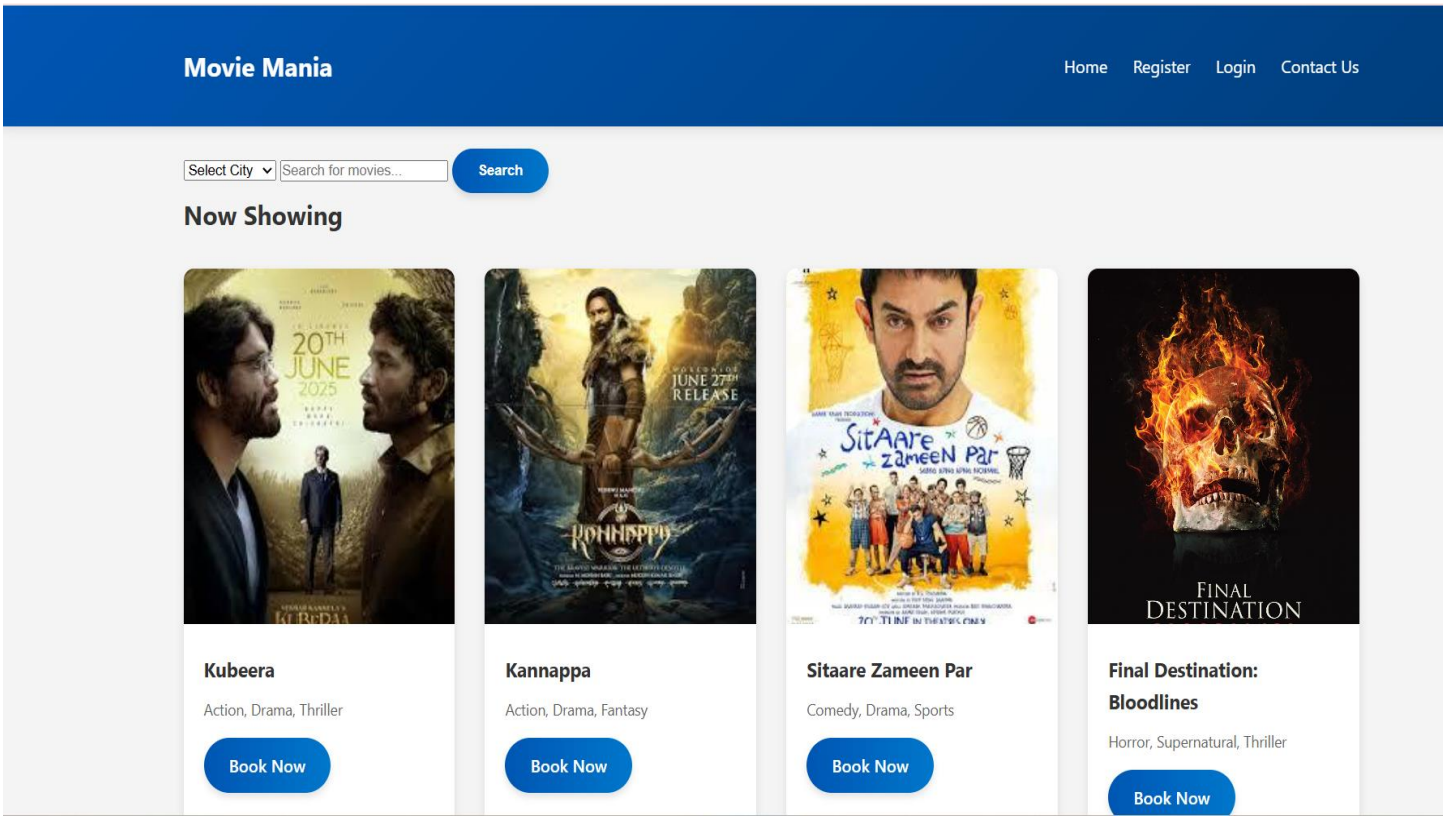
Login page

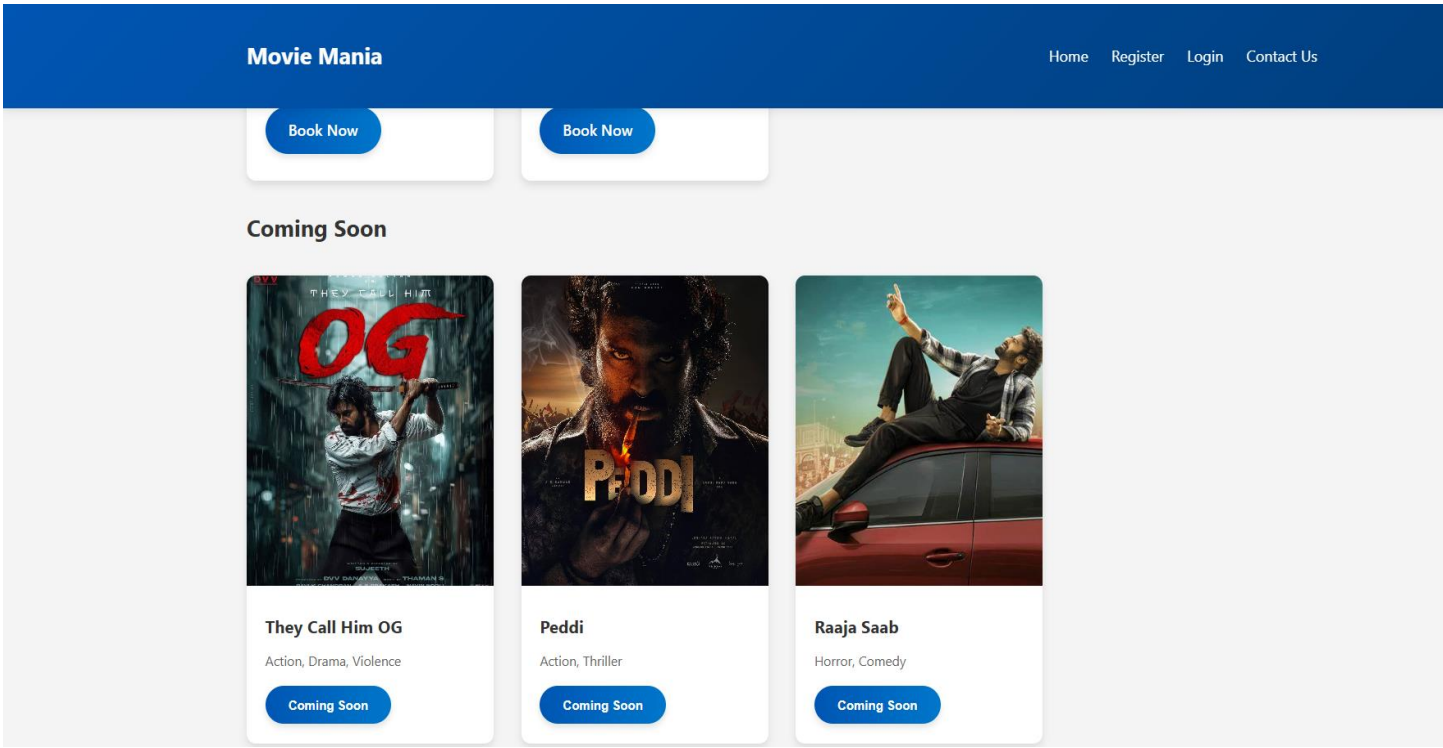


Registration page:

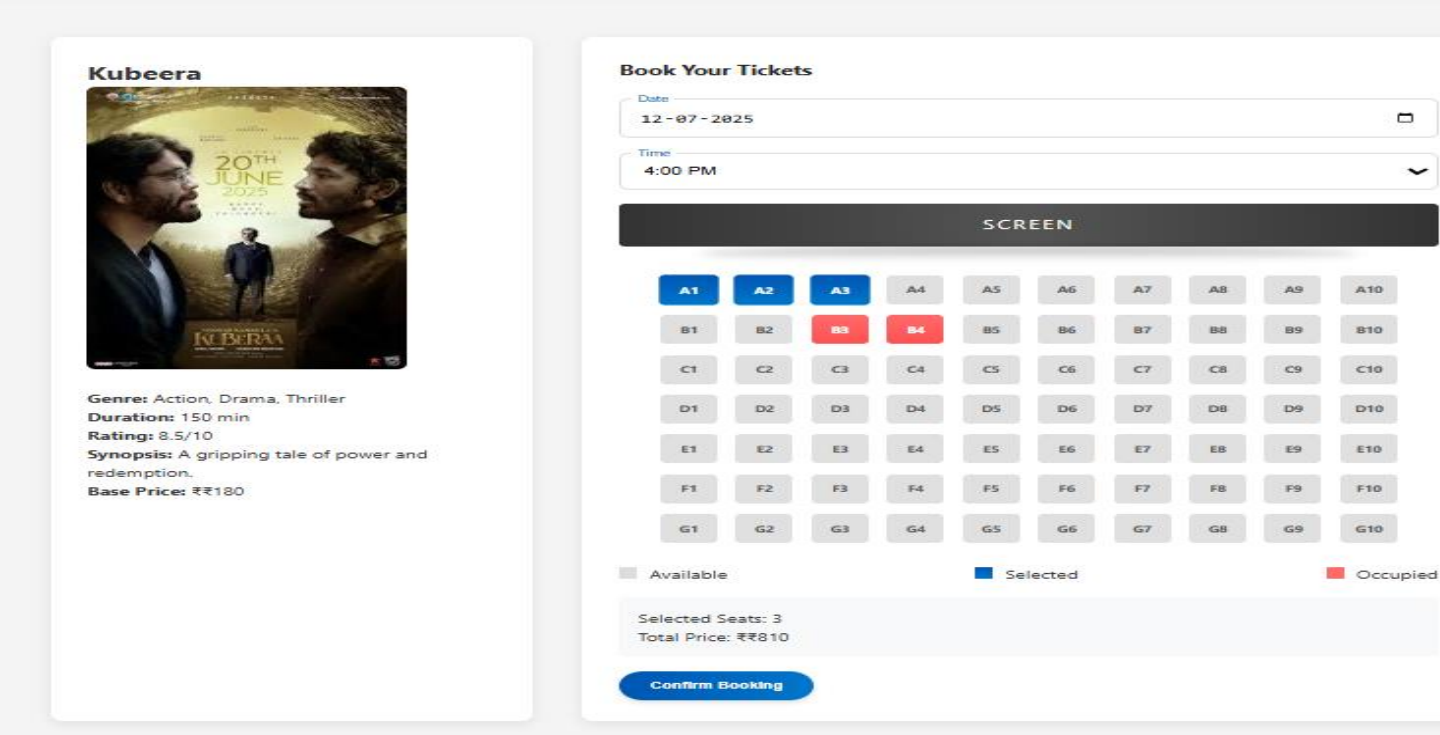


Home page:

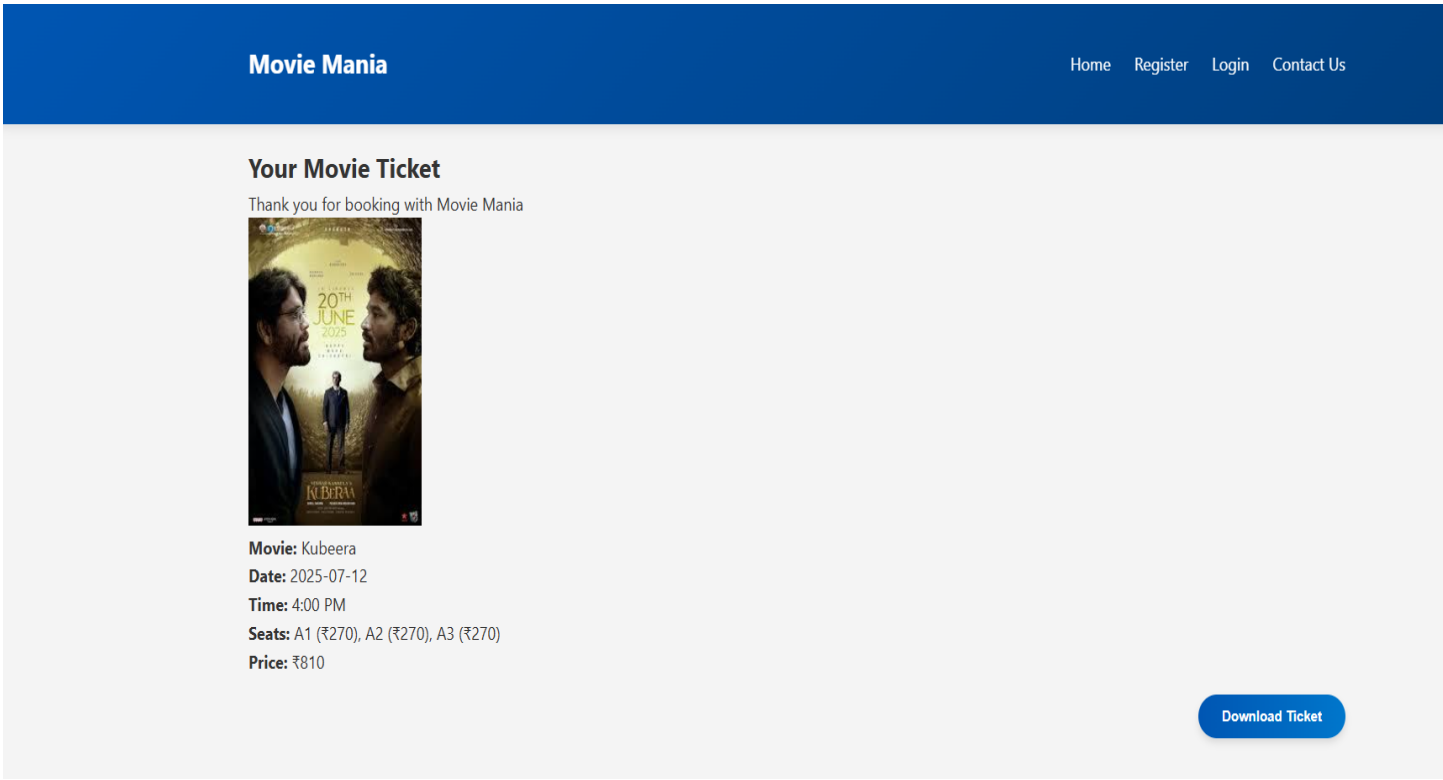

Movie Mania
Home
Register
Login
Contact Us
Select City
Search for movies...
Search
Now Showing
Kubeera
Action, Drama, Thriller
Book Now
Kannappa
Action, Drama, Fantasy
Book Now
Sitaare Zameen Par
Comedy, Drama, Sports
Book Now
Final Destination: Bloodlines
Horror, Supernatural, Thriller
Book Now


Movie Mania
Home
Register
Login
Contact Us
Book Now
Book Now
Coming Soon
They Call Him OG
Action, Drama, Violence
Coming Soon
Peddi
Action, Thriller
Coming Soon
Raaja Saab
Horror, Comedy
Coming Soon

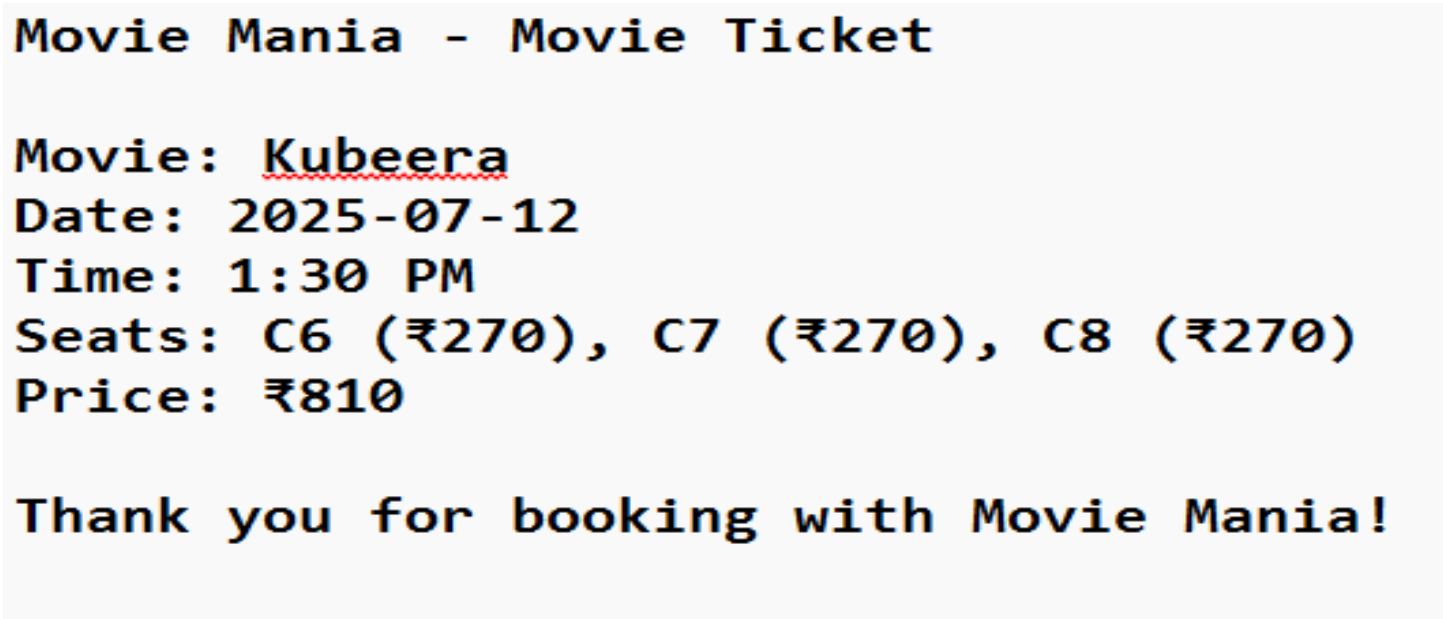
Booking Page:



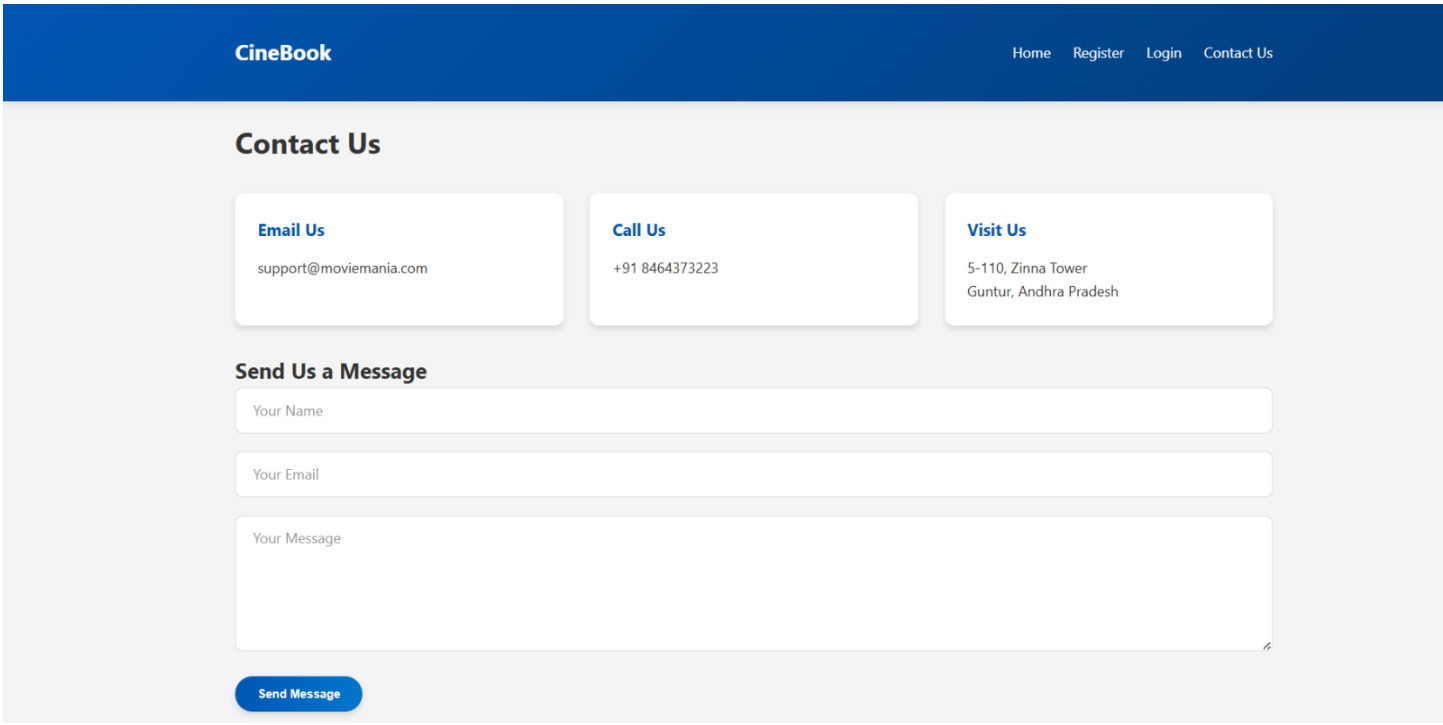
Tickets Page



Download Ticket:

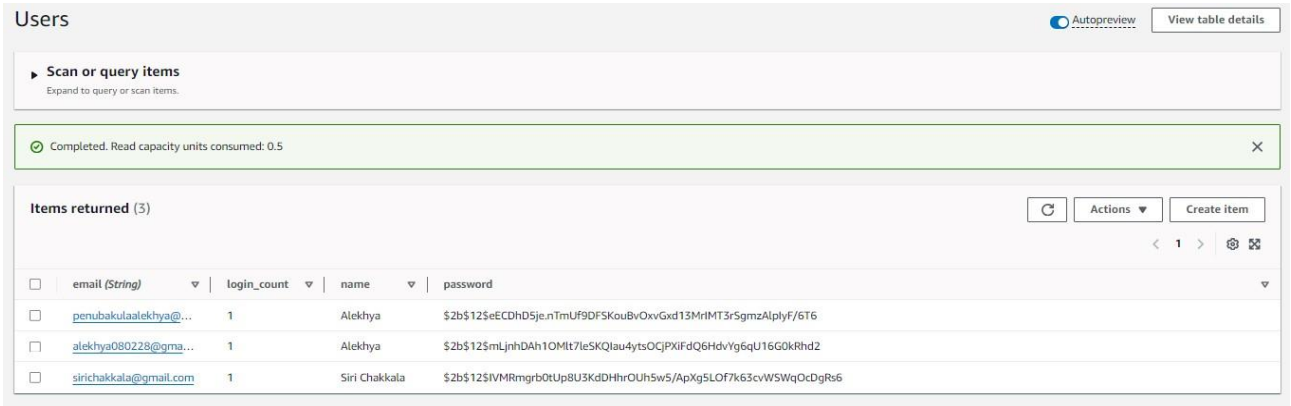


Contact Page

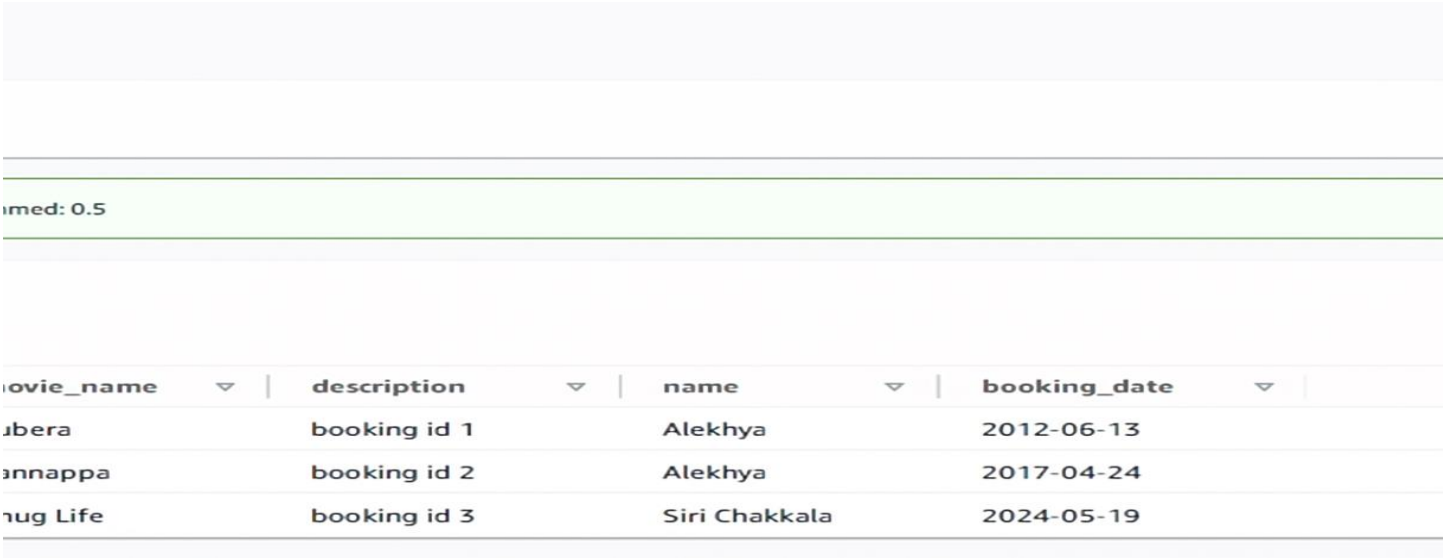


Dynamodb Database updations :

1. **Users table :**



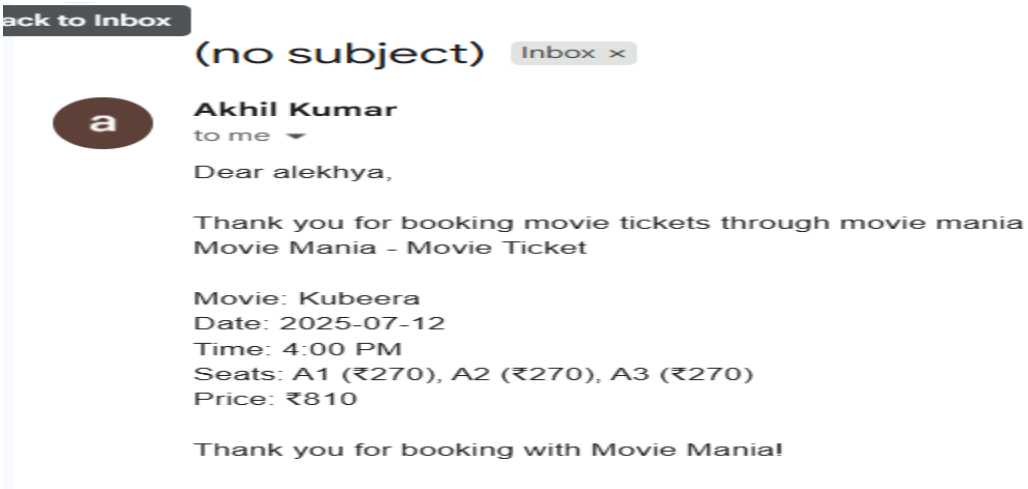
2. Request Table



med: 0.5

ovie_name	description	name	booking_date
ubera	booking id 1	Alekhya	2012-06-13
nnappa	booking id 2	Alekhya	2017-04-24
ug Life	booking id 3	Siri Chakkala	2024-05-19

3. Mail to the User:



ack to Inbox

(no subject) Inbox ×

a

Akhil Kumar
to me

Dear alekhya,

Thank you for booking movie tickets through movie mania
Movie Mania - Movie Ticket

Movie: Kubeera
Date: 2025-07-12
Time: 4:00 PM
Seats: A1 (₹270), A2 (₹270), A3 (₹270)
Price: ₹810

Thank you for booking with Movie Mania!

4. **Mail to the admin:**

Conclusion:

The Movie Magic website has been successfully developed and deployed using a robust, cloud-native architecture. By leveraging key AWS services such as EC2 for hosting, DynamoDB for real-time data management, and SNS for instant booking confirmations, the platform delivers a scalable and user-friendly movie ticketing experience. This solution addresses the limitations of traditional ticket booking systems by enabling users to seamlessly search for movies, select seats, and receive digital tickets—all from a single platform.

The cloud-based infrastructure ensures that the system can handle a high volume of users and transactions, especially during peak movie release times, without compromising speed or reliability. The integration of Flask with AWS services enables smooth backend operations, including user authentication, movie/event browsing, seat availability checks, and secure ticket booking.

Comprehensive testing has confirmed that all core functionalities—from user registration and login to booking confirmation via email—operate seamlessly. The platform's modern interface and responsive design further enhance the user experience, making movie booking quick, efficient, and enjoyable.

In conclusion, Movie Magic stands as a powerful demonstration of how cloud technologies can modernize traditional services. It provides an efficient, scalable, and intuitive solution for movie ticket booking, showcasing the potential of full-stack cloud applications in solving real-world user experience challenges in the entertainment industry.