Movie Magic–Smart Movie Ticket Booking System

Project Description:

In many cities, moviegoers face fragmented booking experiences across multiple platforms, leading to confusion over seat availability and missed showtimes. The lack of personalized features and real-time updates often disrupts the excitement of planning a cinematic outing.To solve this, a cloud-based Movie Booking Platform was developed using Flask for backend development, AWS EC2 for hosting, and DynamoDB for real-time data management. The system allows users to browse movie listings, securely register/login, select seats in real time, and complete bookings smoothly. Designed to support future scalability, the platform lays the foundation for integrating multiple theaters, payment gateways, and personalized recommendations. IAM ensures role-based access control, and error resilience is achieved with robust transaction handling logic.This modern solution enhances convenience for users and operational efficiency for theaters by streamlining the entire booking experience into a single, cloud-native interface.

Scenario 1: Real-Time Seat Selection for Moviegoers

The Movie Booking Platform utilizes AWS EC2 to provide a highly available backend infrastructure, handling multiple users exploring seat layouts concurrently. For instance, a user logs in, navigates to a movie listing, and views a live seating chart for a specific showtime. Flask orchestrates backend logic to lock seats momentarily while the user completes selection. DynamoDB leverages conditional writes to prevent double booking during peak hours, allowing real-time updates to be reflected instantly across users' screens. The responsive experience ensures high satisfaction and prevents booking conflicts.

Scenario 2: Secure User Registration and Booking Flow

The platform ensures secure onboarding via IAM-based policies and Flask-authenticated login flows. For example, a new user signs up, browses current movies, selects available seats, and confirms the booking—all within an intuitive interface. AWS DynamoDB stores session data and ticket status, enabling persistent and fast retrieval for future interactions. IAM manages role-based permissions, separating administrative operations from user-level access. The booking flow validates payment readiness and ensures transaction resilience, minimizing failure points during confirmation.

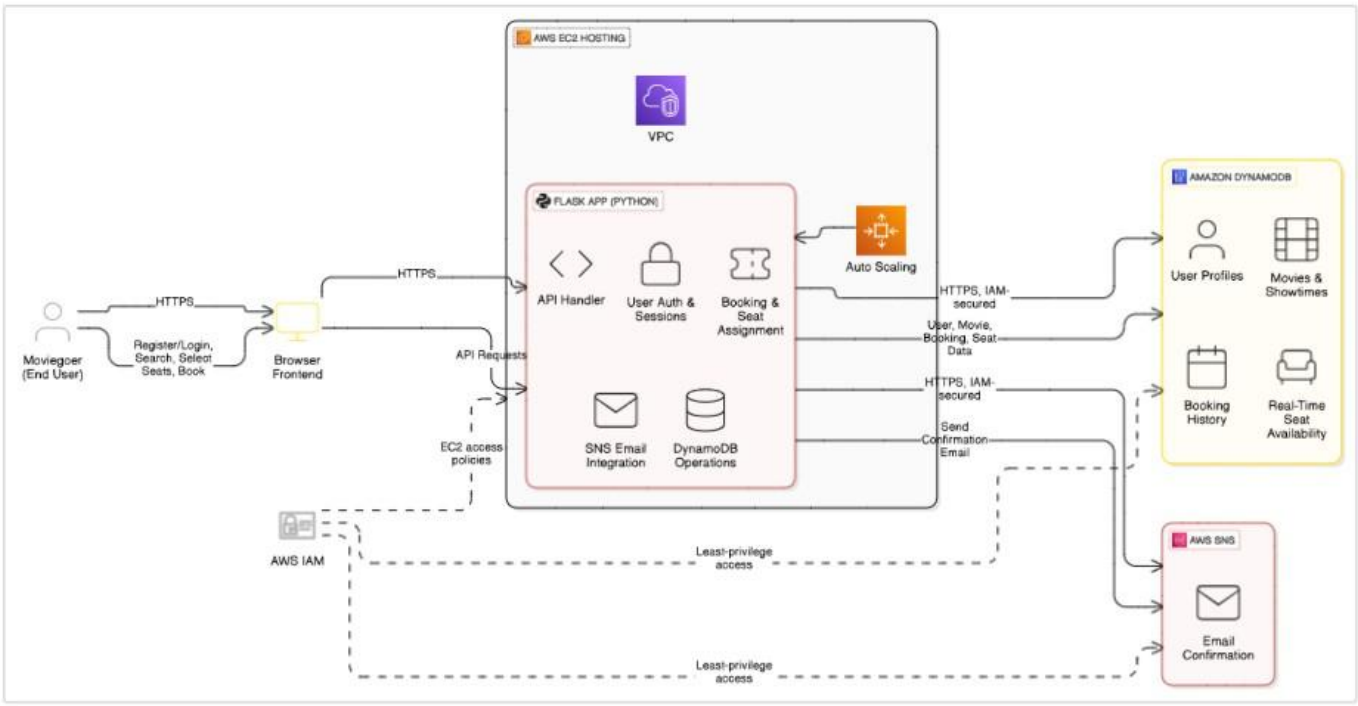
Scenario 3: Theater Efficiency and Backend Resilience

The movie theater benefits from streamlined operations with minimal manual coordination. Admins can log in to update movie schedules, track seat occupancy, and analyze booking trends. Flask handles these backend operations, while EC2 provides reliable deployment support for simultaneous user access. DynamoDB enables fast queries about occupancy rates and movie popularity. Real-time insights from the data allow theaters to optimize show timings and predict

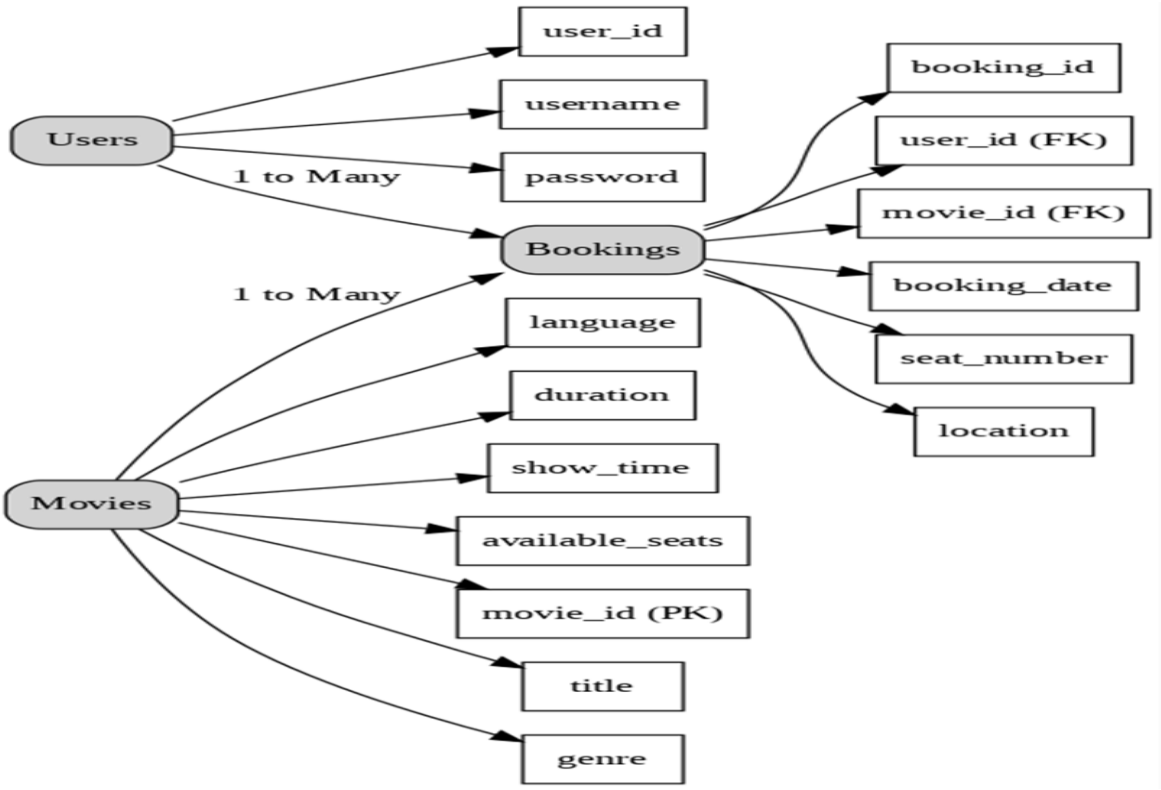
demand. The seat selection mechanism ensures concurrency-safe updates, avoiding overbooking during high-traffic periods.

AWS ARCHITECTURE

This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for hosting the backend, with a lightweight framework like Flask handling core logic. Application data is stored in Amazon DynamoDB, ensuring fast, reliable access, while user access is managed through AWS IAM for secure authentication and control. Real-time alerts and system notifications are enabled via Amazon SNS.



Entity Relationship (ER)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:

1. AWS Account Setup and Login

Activity 1.1: Create an AWS account at aws.amazon.com.

Activity 1.2: Log in to the AWS Management Console

2. DynamoDB Database Creation and Setup

Activity 2.1: Create a table Movie_Bookings, Partition Key: id

Activity 2.2: Create a table Movie_Users, Partition Key: email

3. SNS Notification Setup

Activity 3.1: Create SNS topics for movie bookings

Activity 3.2: Subscribe users to SNS email notifications, You will get SNS_TOPIC_ARN.

sns_topic_arn = 'arn:aws:sns:us-east-1:605134439175:MovieMagicNotifications:259e3be3-5864-4985-ab9d-edfc09ca6300'

4. Backend Development and Application Setup

Activity 4.1:Develop the Backend Using Flask.

Activity 4.2: Integrate AWS Services Using boto3.

5. IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies
SNSFullAccess, DynamoDBFullAccess, EC2FullAccess

6. EC2 Instance Setup

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

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

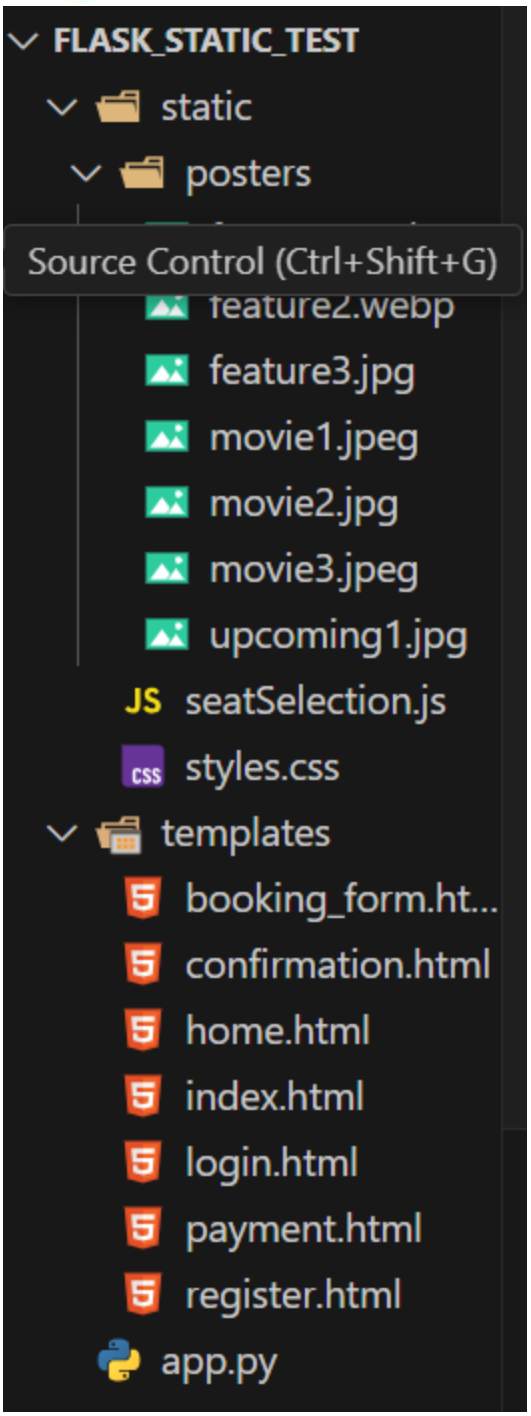
7.Deployment on EC2

Activity 7.1:Upload Flask Files

Activity 7.2: Run the Flask App

Backend Development and Application Setup focuses on establishing the core structure of the application. This includes configuring the backend framework, setting up routing, and integrating database connectivity. It lays the groundwork for handling user interactions, data management, and secure access.

LOCAL DEPLOYMENT



Description of the code :

- **Flask App Initialization**

  Imports and Configuration:

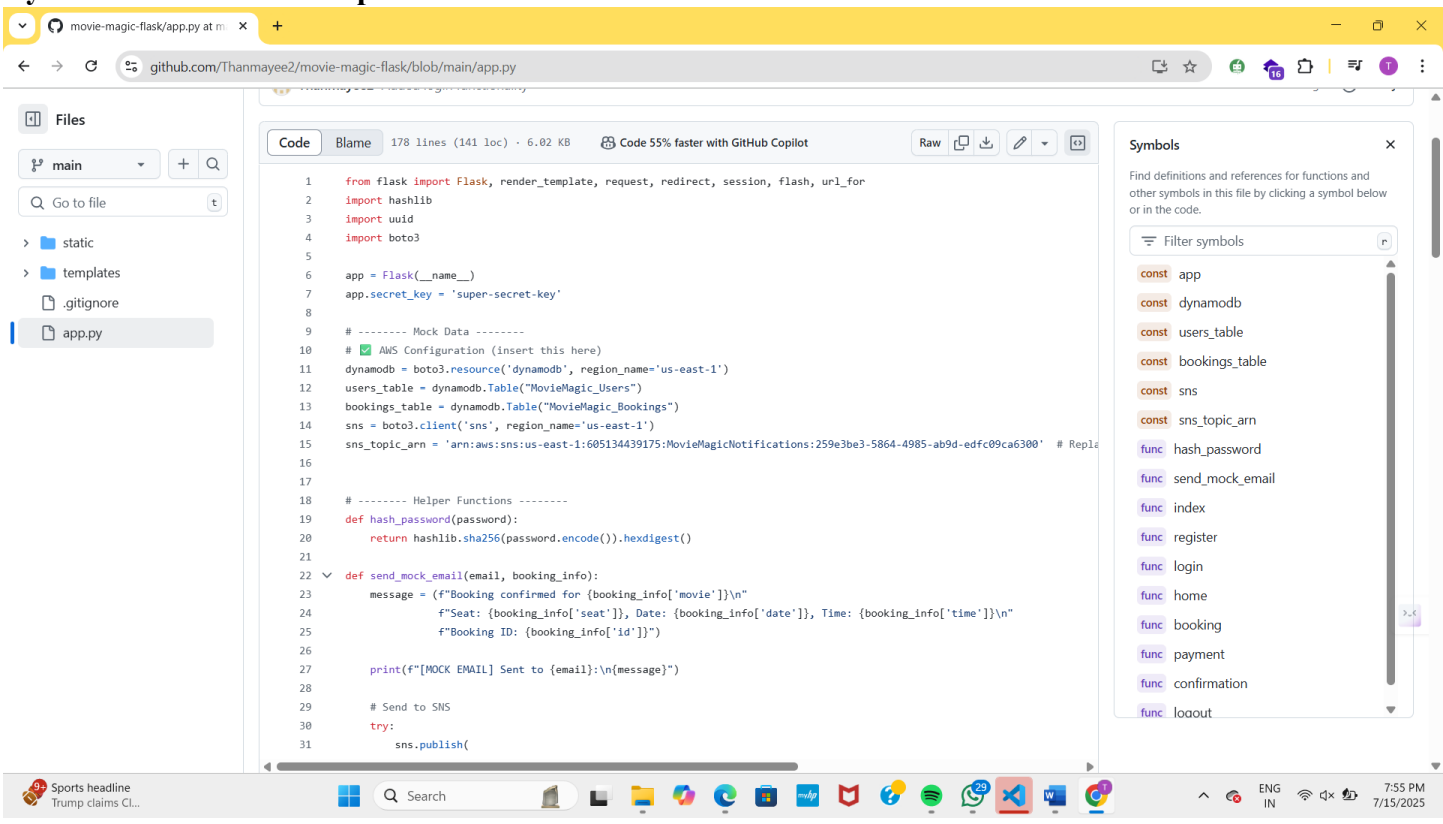
```
from flask import Flask, render_template, request, redirect, session, flash, url_f
import hashlib
import uuid
```

Description: This project uses Flask for routing, session management, and user authentication with secure password hashing. It integrates AWS services via Boto3 for handling data storage, notifications, and unique user operations.

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

Description:A new Flask application instance is initialized, and a secret key is set to securely manage user sessions and protect against cookie tampering.

Dynamodb and SNS Setup:

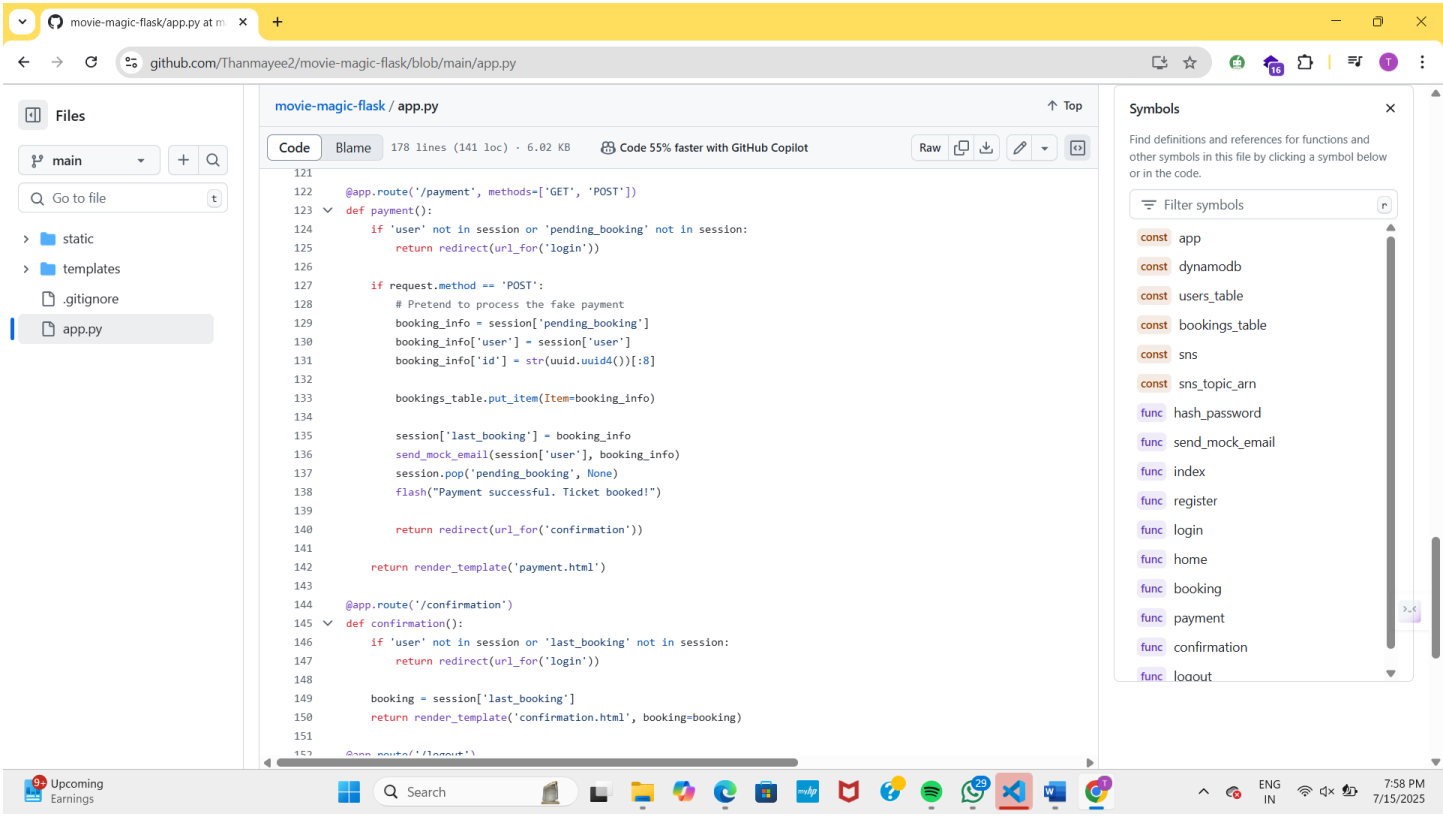


Description: Use **boto3** to connect to **DynamoDB** for handling user registration, movie bookings database operations and also mention region_name where Dynamodb tables are created.

- **SNS Connection** o **Description:** Configure **SNS** to send notifications when a movie ticket is booked. 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.

- **Function to send the Notifications:**

Description: This function sends a booking confirmation email using AWS SNS. It formats the booking details into a message and publishes it to a specified SNS topic, notifying the user via email about their successful movie ticket booking.



- **Routes for Web Pages**
- **Register User:** Collecting registration data, hashes the password, and stores user details in the database.

login Route (GET/POST): Verifies user credentials, increments login count, and redirects to the dashboard on success.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        hashed = hash_password(password)

        response = users_table.get_item(Key={'email': email})
        user = response.get('Item')

        if user and user['password'] == hashed:
            session['user'] = email
            return redirect(url_for('home'))
        else:
            flash("Invalid email or password.")
            return render_template('login.html')
    return render_template('login.html')
```

- These Flask routes handle key navigation in the app: /logout logs out the user by clearing the session and showing a flash message; /home1 is a protected route accessible only to logged-in users; /about and /contact_us render static pages with information about the app and ways to get in touch.

```
@app.route('/logout')
def logout():
    session.clear()
    flash("You have been logged out.")
    return redirect(url_for('index'))
```

Booking Page Route:

Description: This route displays the booking page (b1.html) with movie, theater, address, and price details passed as query parameters. It ensures only logged-in users can access the page.

```
@app.route('/booking', methods=['GET', 'POST'])
def booking():
    if 'user' not in session:
        return redirect(url_for('login'))

    if request.method == 'POST':
        # Temporarily store booking form in session
        session['pending_booking'] = {
            'movie': 'Example Movie',
            'seat': request.form['seat'],
            'date': request.form['date'],
            'time': request.form['time']
        }
        return redirect(url_for('payment'))

    return render_template('booking_form.html', movie='Example Movie')
```

Home Page :

Description: This route processes movie ticket bookings by collecting form data, generating a unique booking ID, storing details in DynamoDB, and sending a confirmation email via AWS SNS. It then displays the booking details on the tickets page.

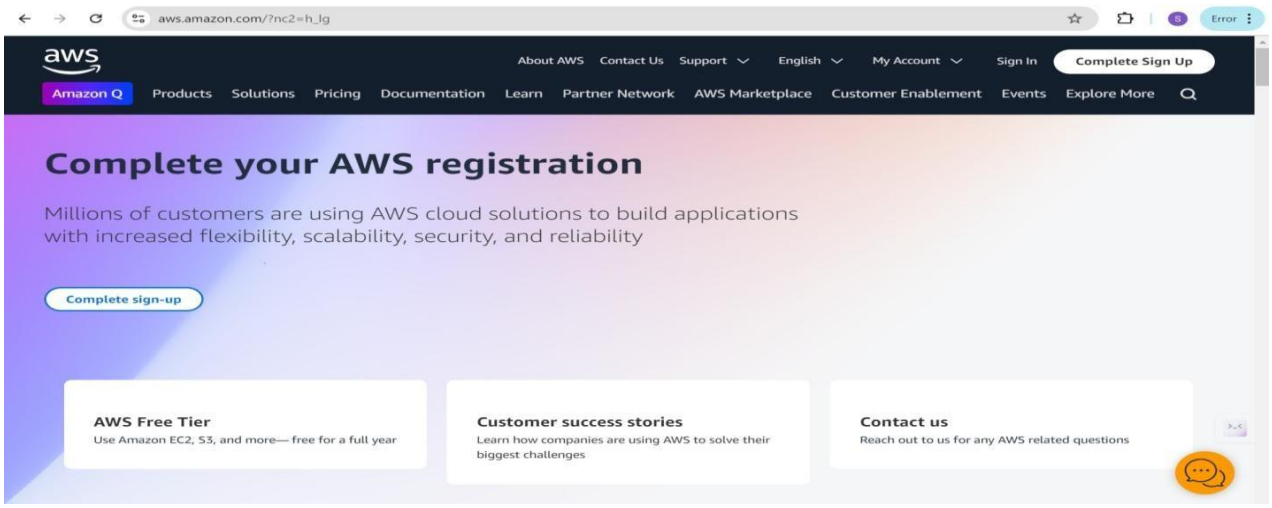
```python
@app.route('/home')
def home():
    if 'user' not in session:
        return redirect(url_for('login'))

    now_showing = [
        {"title": "The Grand Premiere", "genre": "Drama", "poster": "posters/movie1.jpeg", "duration": "2h 10m", "rating": "4.5", "synopsis": "A heartfelt journey of dreams and destiny.
        {"title": "Engaging", "genre": "Drama", "poster": "posters/movie2.jpg", "duration": "1h 45m", "rating": "4.2", "synopsis": "A hilarious ride through everyday chaos."}
    ]
    coming_soon = [
        {"title": "Future Flick", "genre": "Sci-Fi", "poster": "posters/upcoming1.jpg", "duration": "2h 20m", "rating": "N/A", "synopsis": "A mind-bending tale of time and technology."}
    ]
    top_rated = [
        {"title": "Edge of Tomorrow", "genre": "Action", "poster": "posters/movie3.jpeg", "duration": "2h", "rating": "4.8", "synopsis": "A soldier relives the same day in a war against
    ]
    return render_template('home.html', now_showing=now_showing, coming_soon=coming_soon, top_rated=top_rated)
```

Application Entry point:

```python
if __name__ == '__main__':
    print("🚀 Mock MovieMagic running at http://127.0.0.1:5000")
    app.run(debug=True, host='0.0.0.0', port=5000)
```

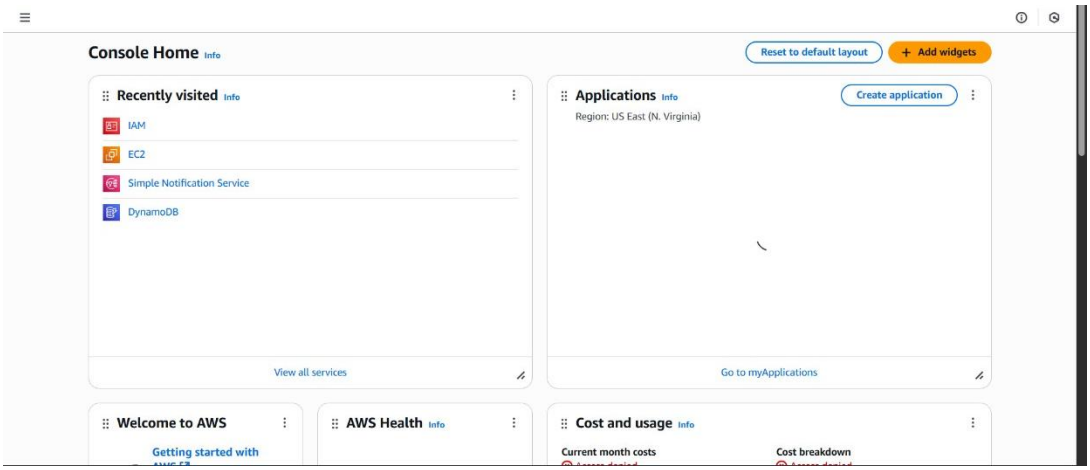
- **Description**: This block starts the Flask application using the built-in development server, setting the host, port, and enabling debug mode for easier development and testing.

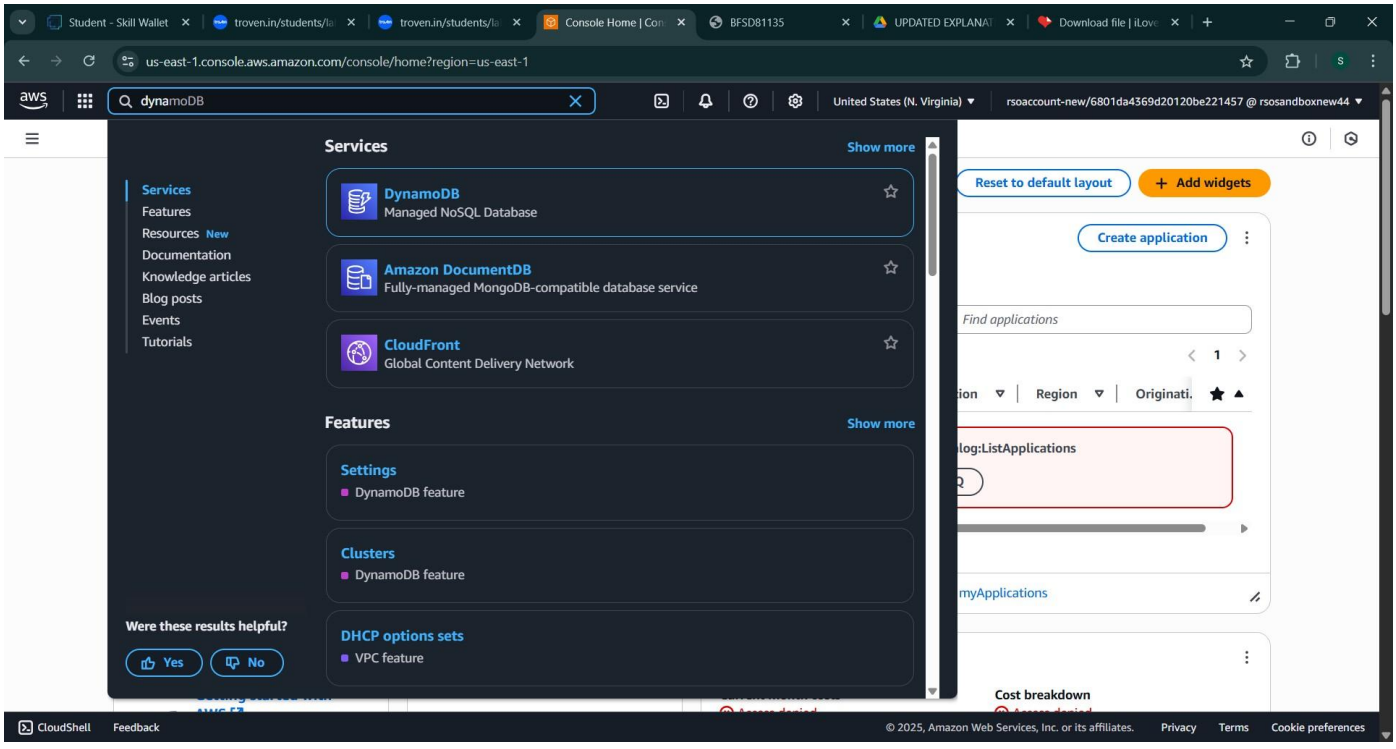
AWS Account Setup



- **Activity 2.2: Log in to the AWS Management Console**

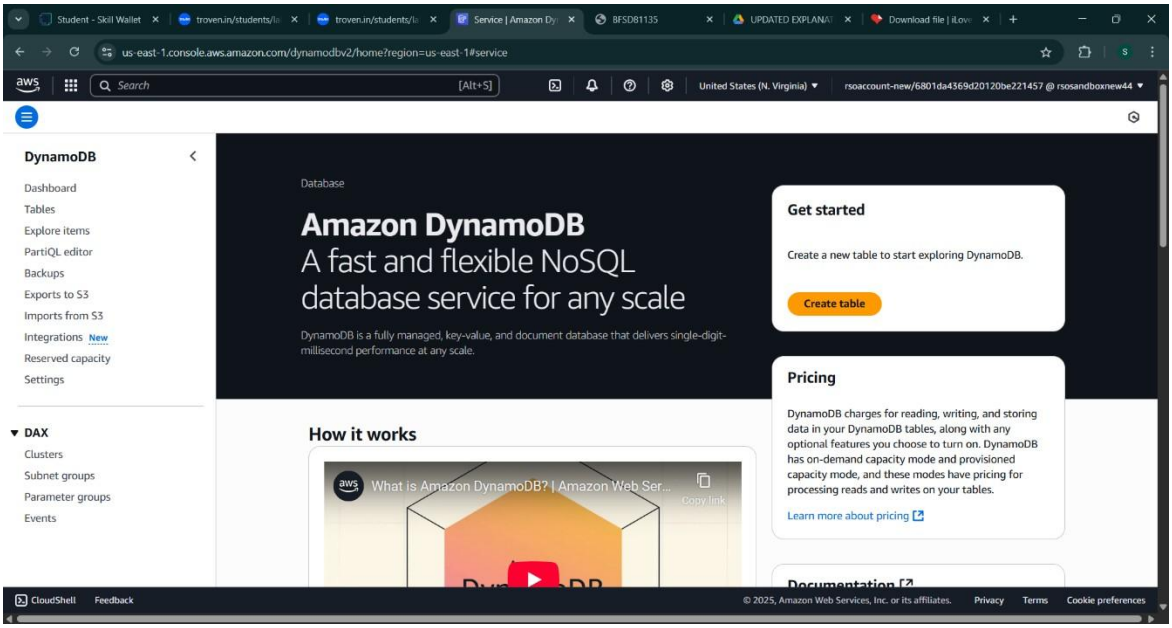
    - After setting up your account, log in to the AWS Management Console.





DynamoDB Database Creation and Setup

- **Activity 3.1:Navigate to the DynamoDB**
    - In the AWS Console, navigate to DynamoDB and click on create tables.



2.

3.

○

Activity 3.2:Create a DynamoDB table for storing registration details and book requests.

— Create Users table with partition key “user_email” with type String and click on create tables.

4.

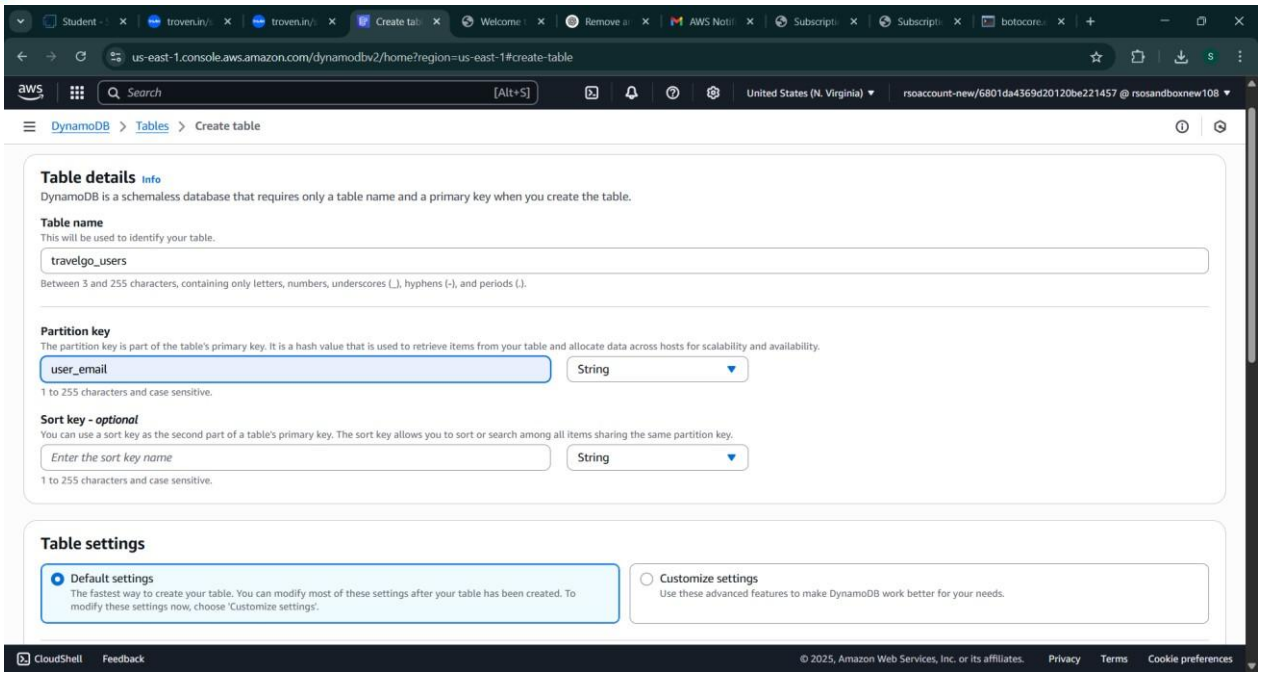


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based 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.

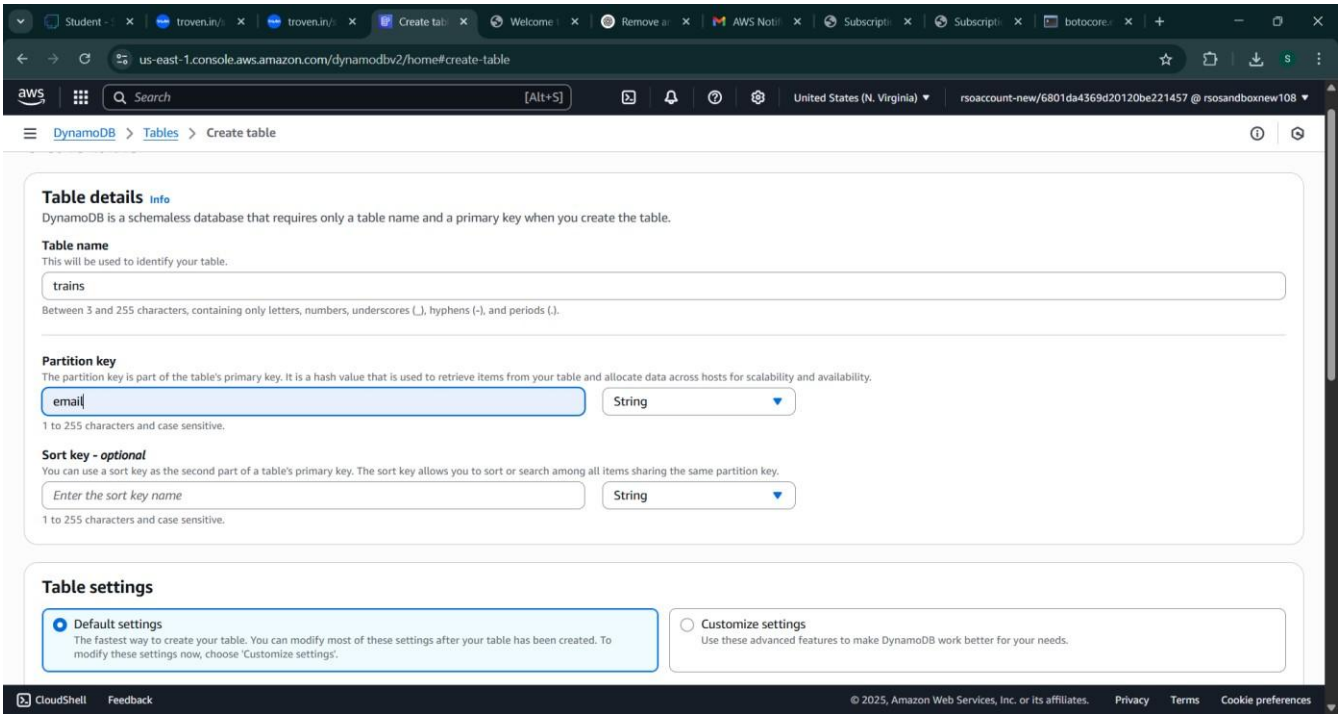
No tags are associated with the resource.

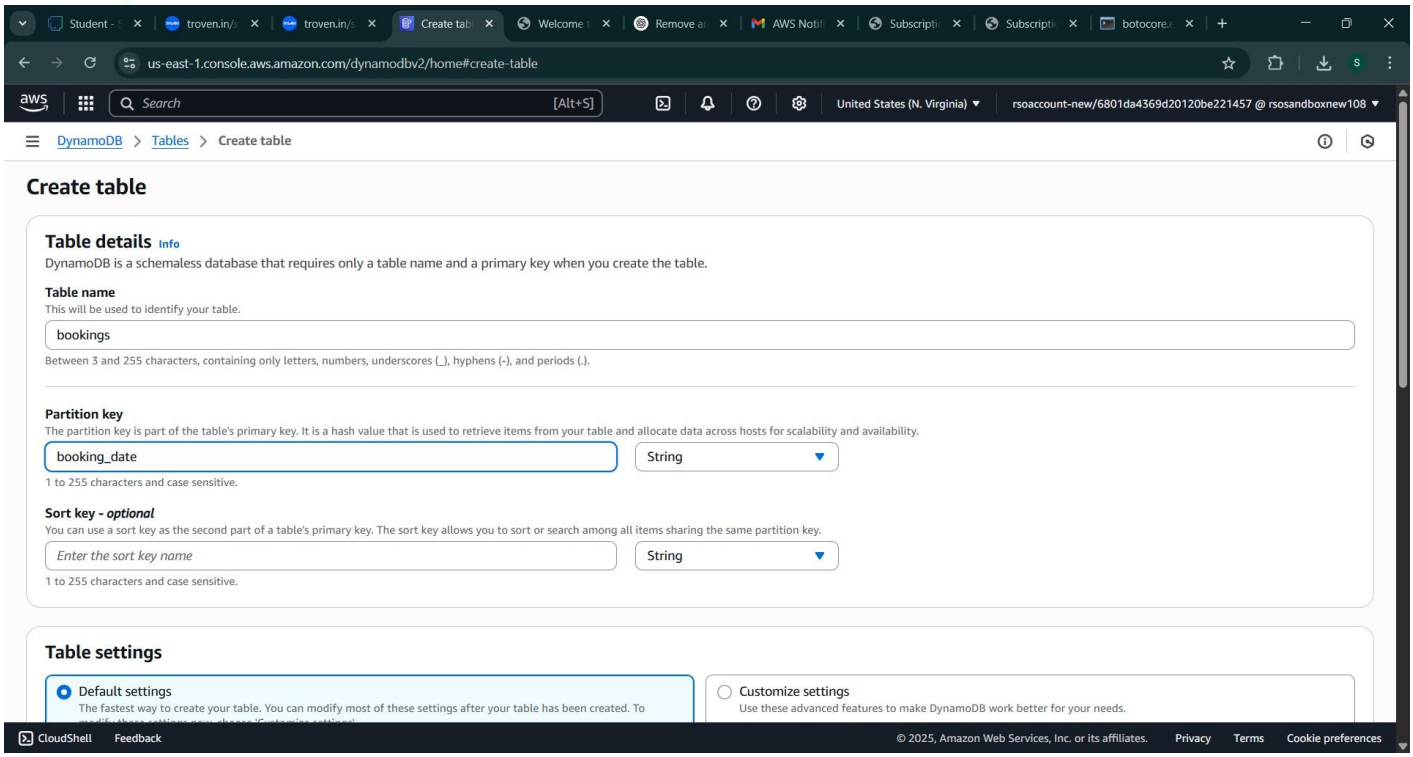
Add new tag

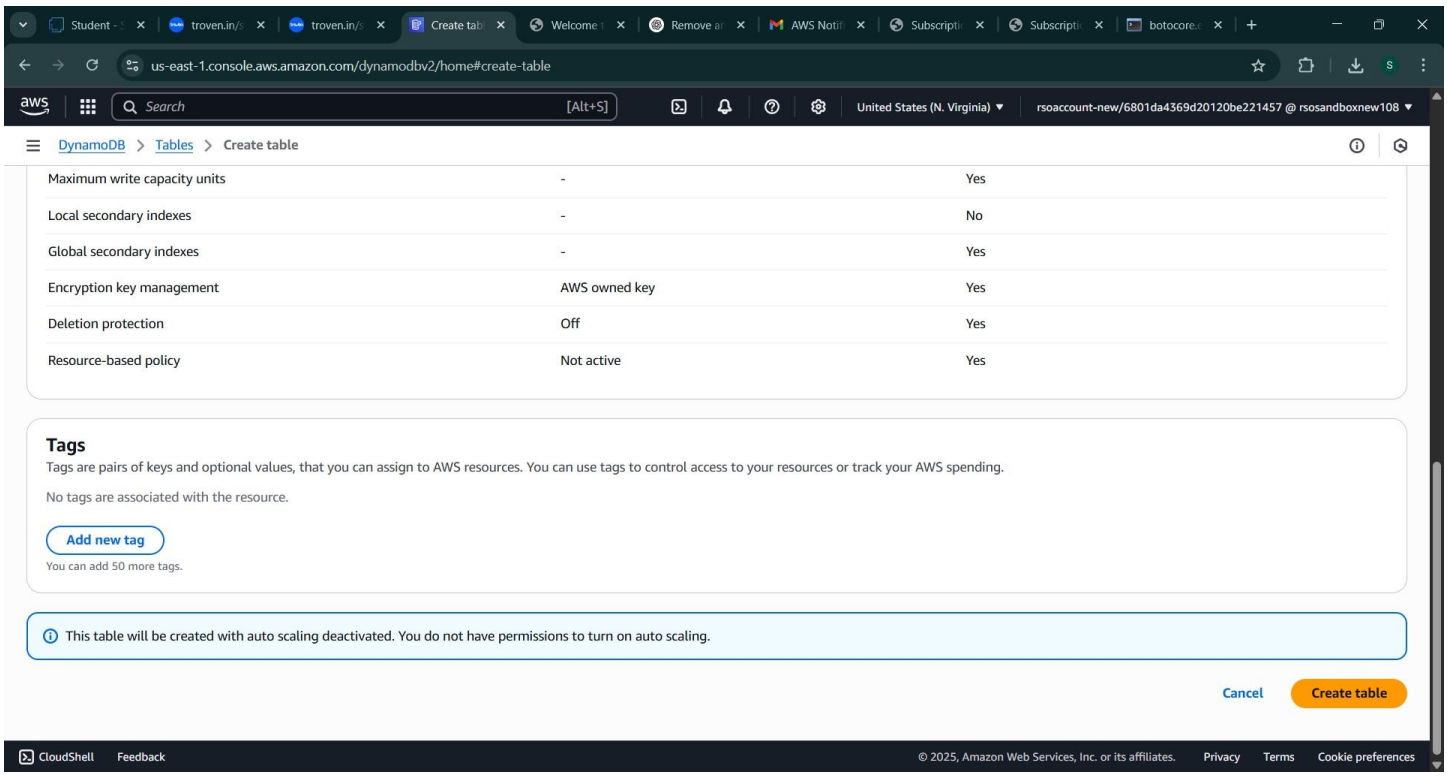
You can add 50 more tags.

Cancel Create table

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

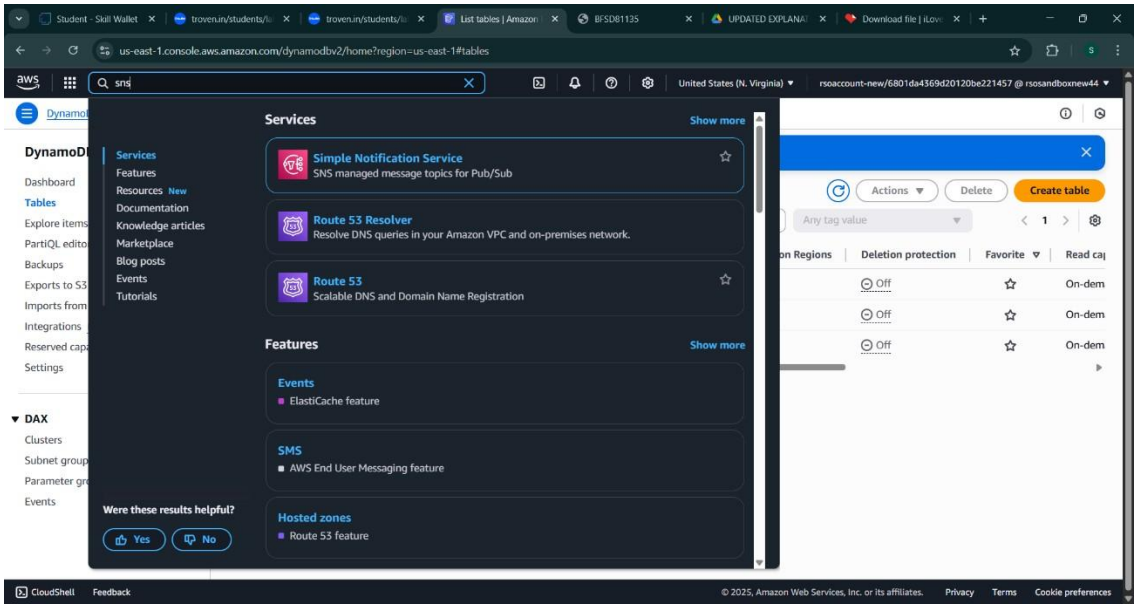




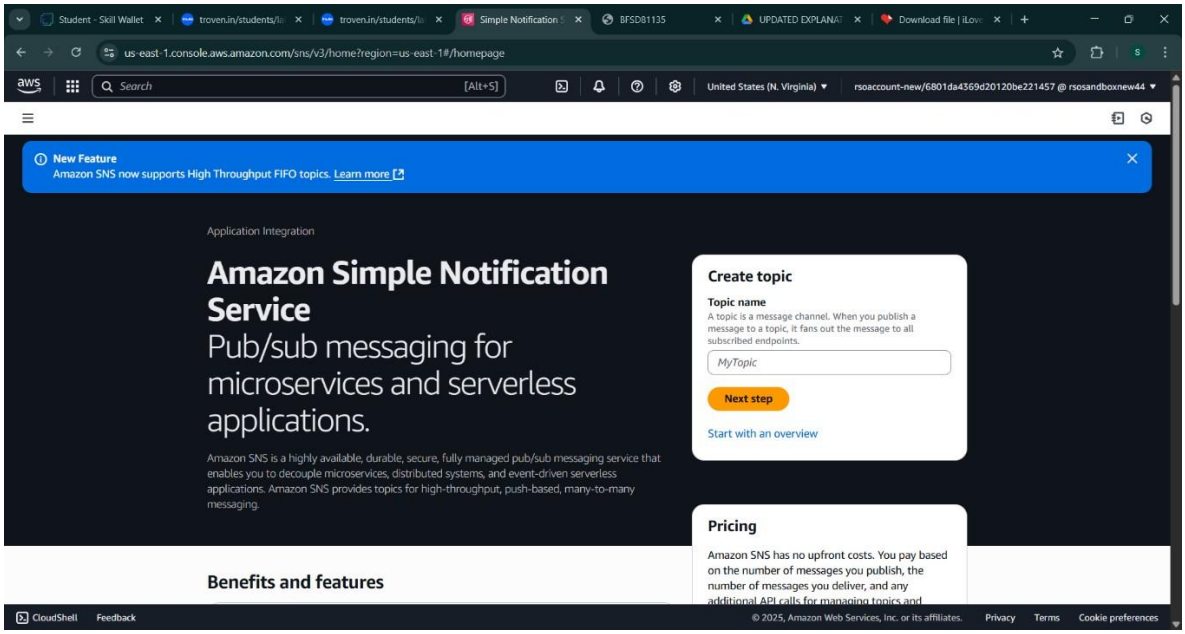


Milestone 4: SNS Notification Setup

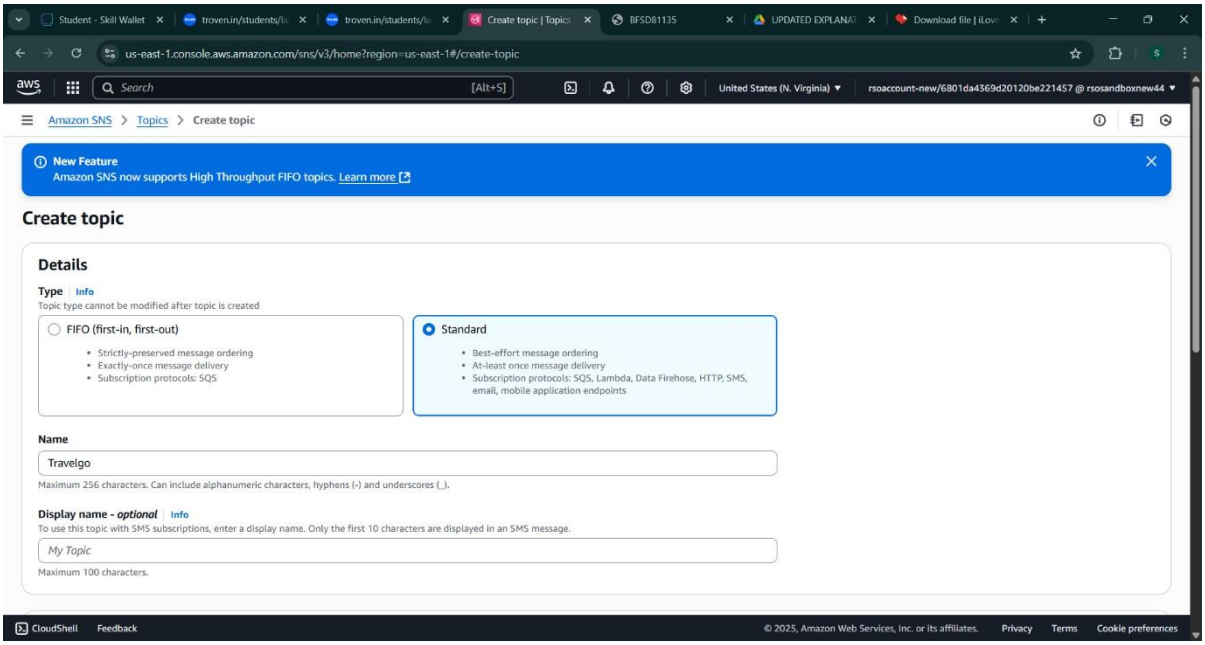
- **Activity 4.1: Create SNS topics for sending email notifications to users**



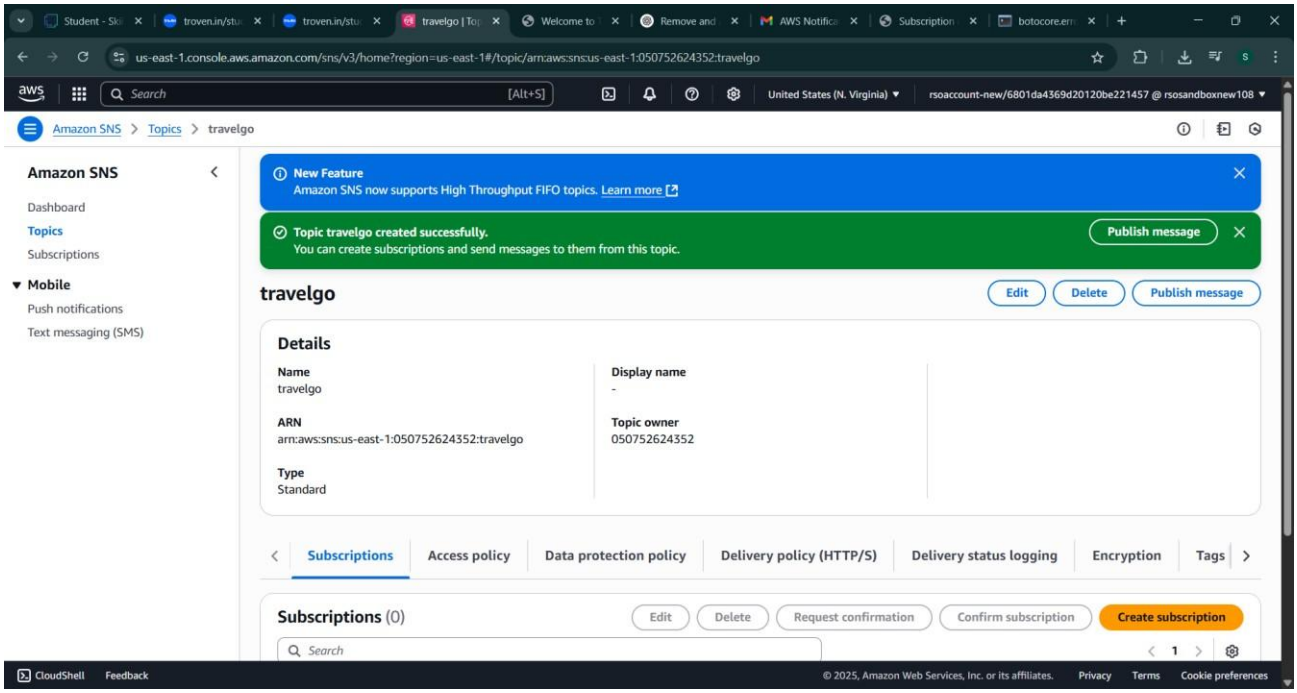
  - In the AWS Console, search for SNS and navigate to the SNS Dashboard.
-



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

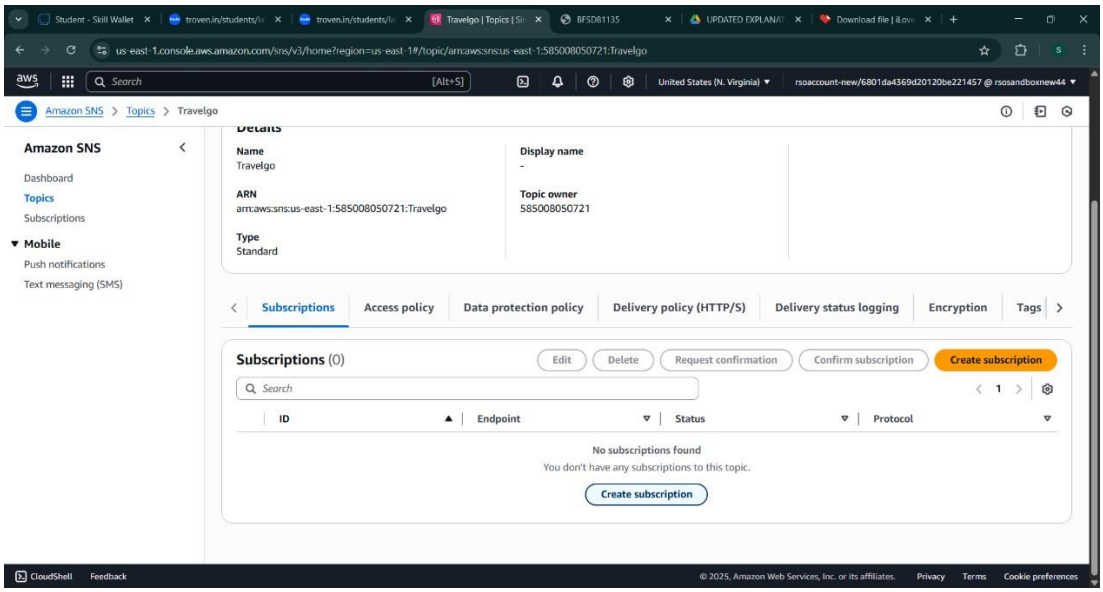


○ Choose Standard type for general notification use cases and Click on Create Topic.



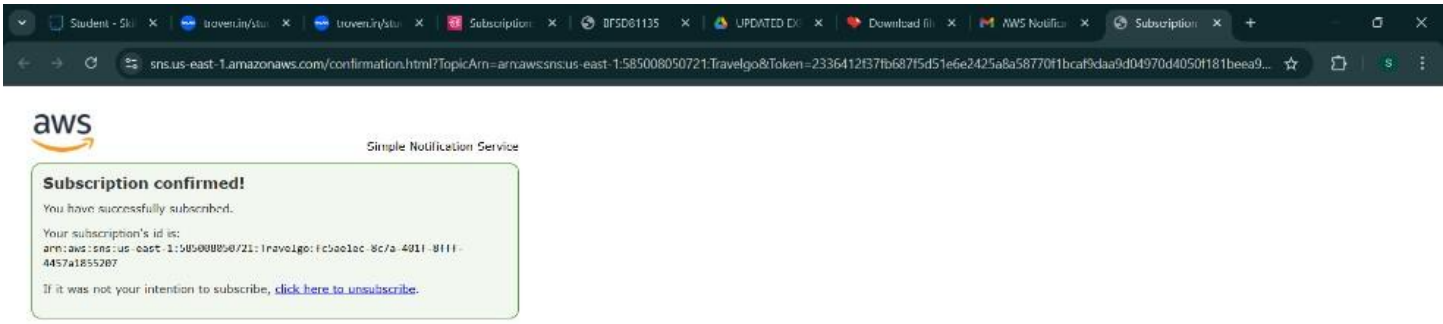
○ Configure the SNS topic and note down the **Topic ARN**.

- **Activity 4.2: Subscribe users relevant SNS topics to receive real-time notifications when a book request is made.**



- Subscribe users to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails**.**

After subscription request for the mail confirmation

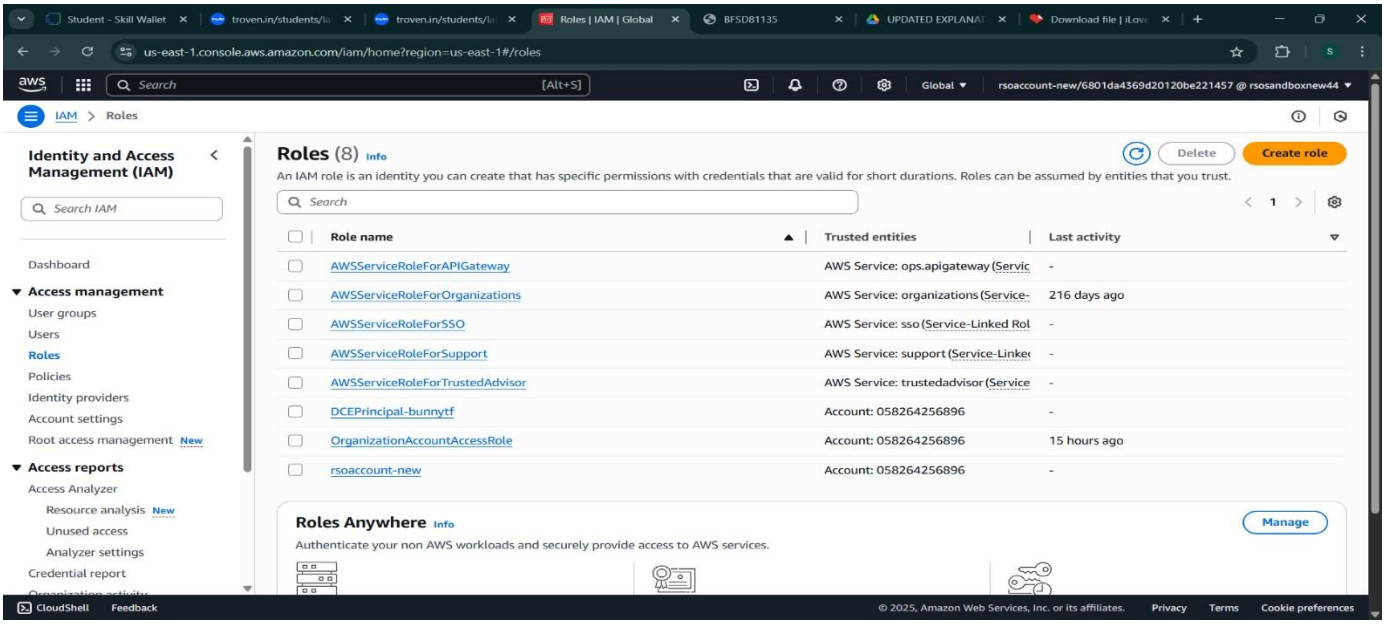


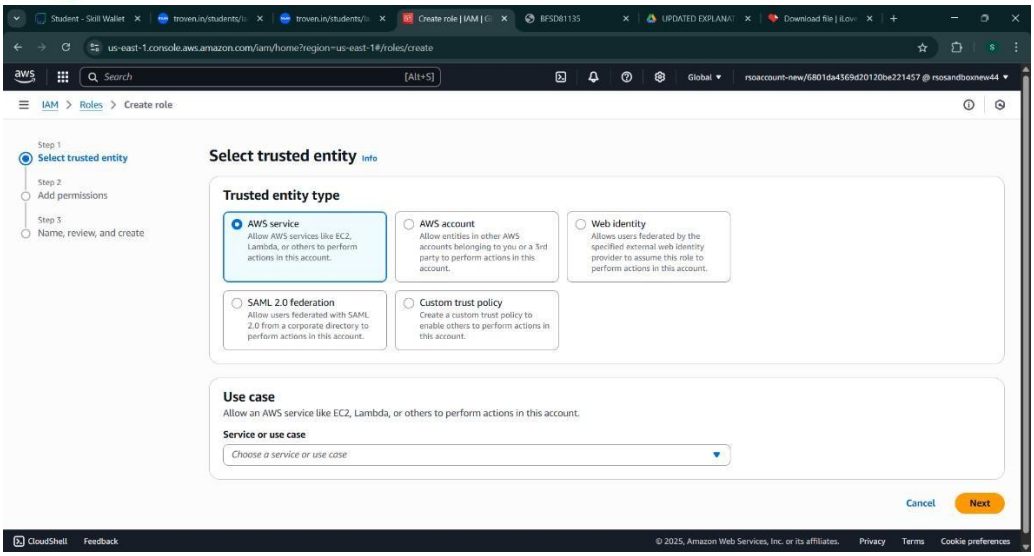
Successfully done with the SNS mail subscription and setup, now store the ARN link

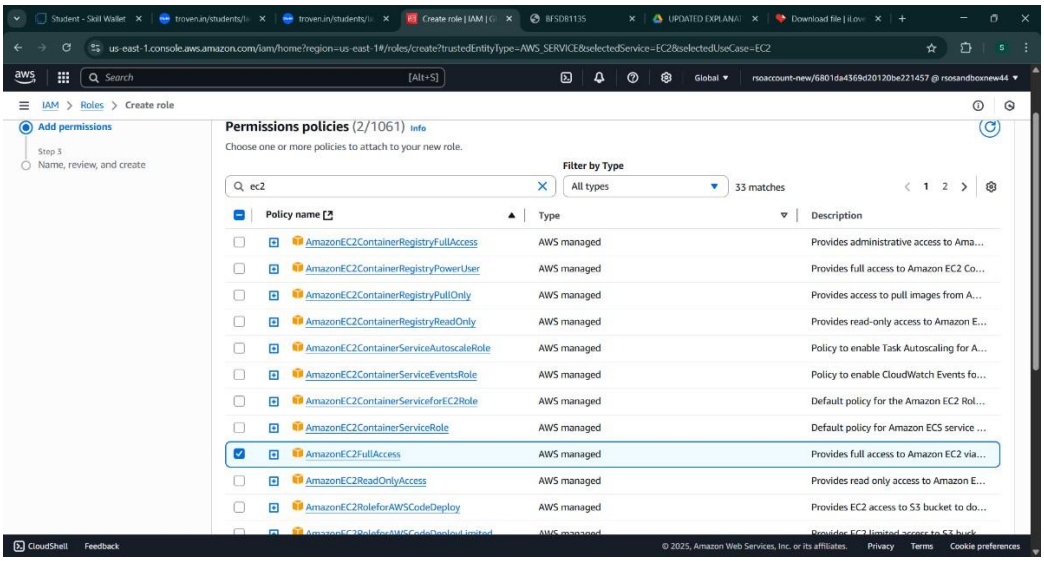
IAM Role Setup

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.



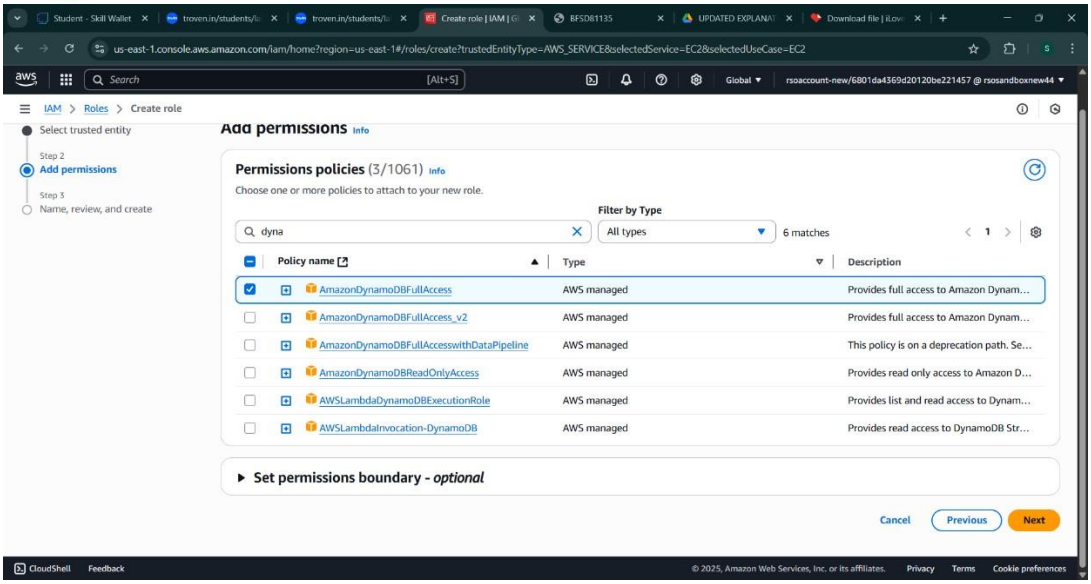


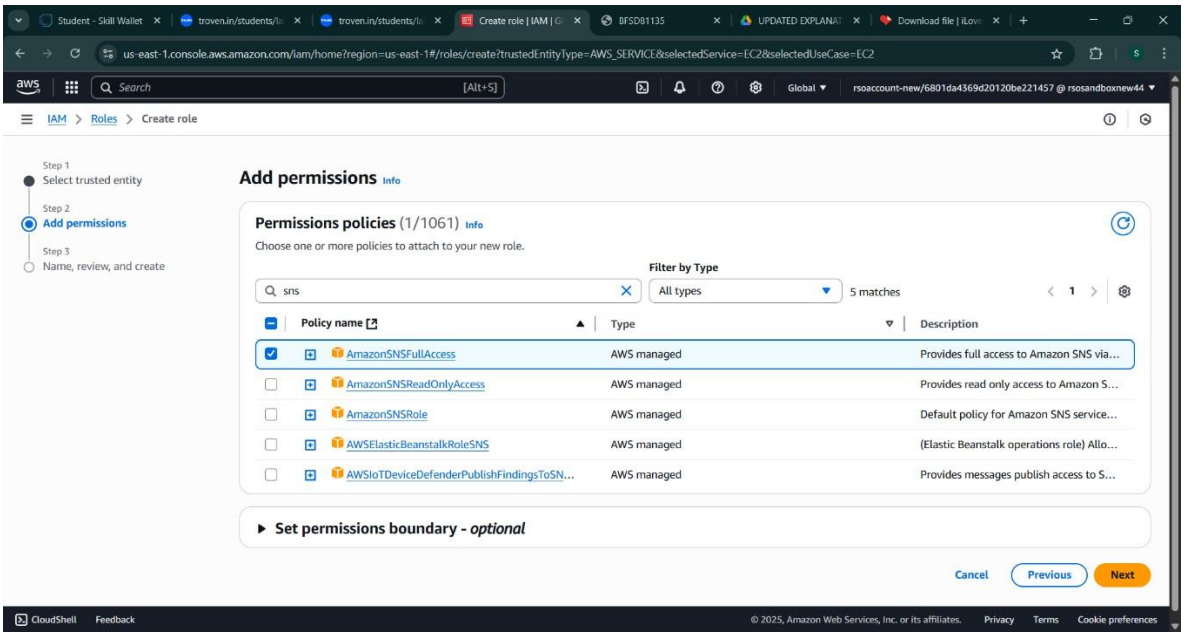


- **Activity 5.2: Attach Policies.**

  Attach the following policies to the role:

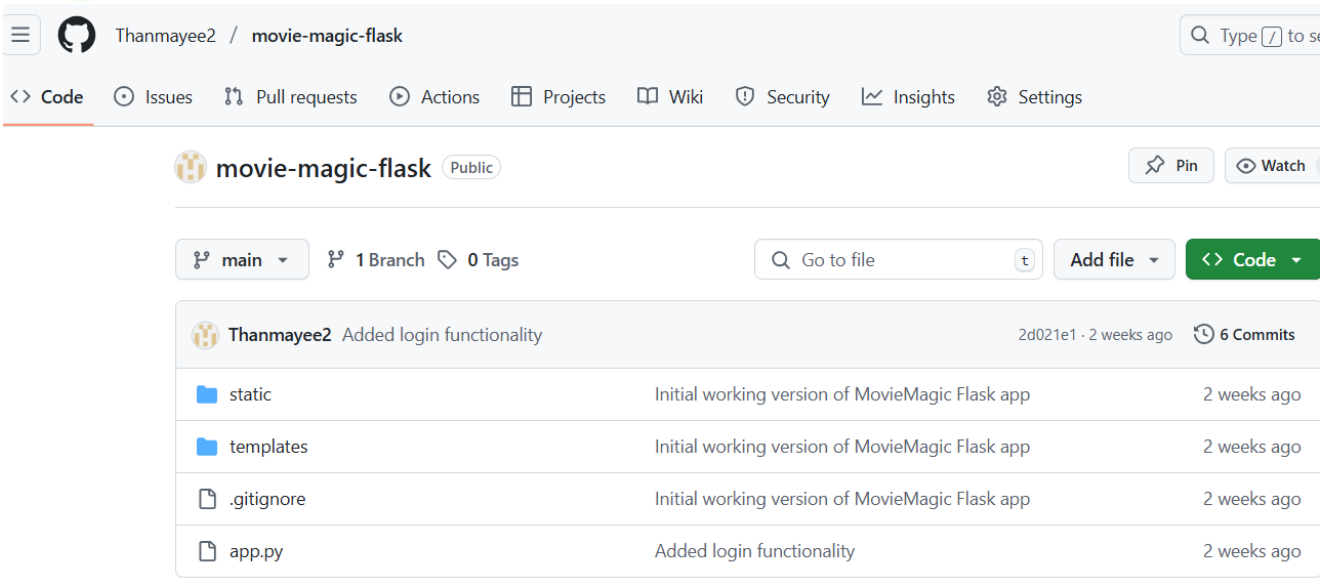
- AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
- AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.





EC2 Instance Setup

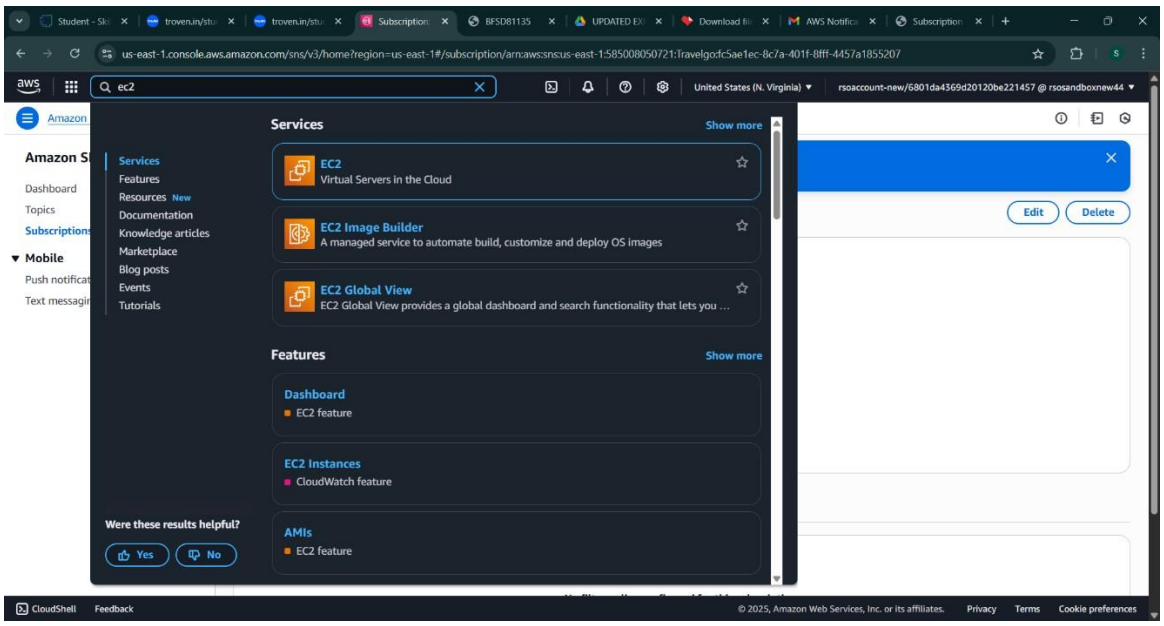
- **Note: Load your Flask app and Html files into GitHub repository.**

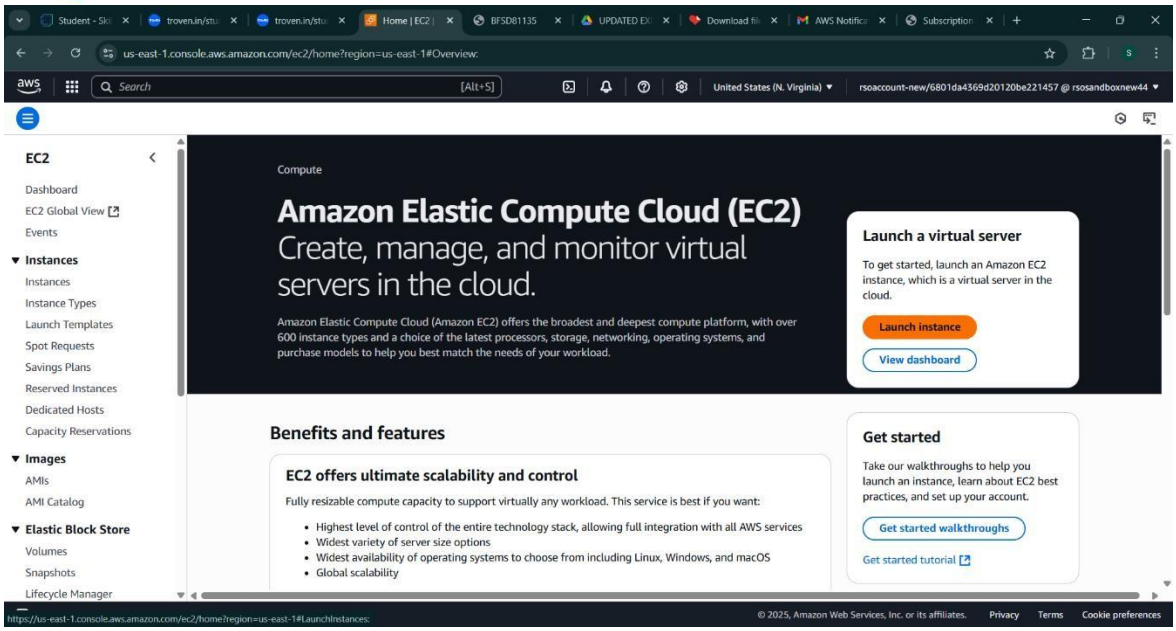


Activity 6.1: Launch an EC2 instance to host the Flask application.

- **Launch EC2 Instance**

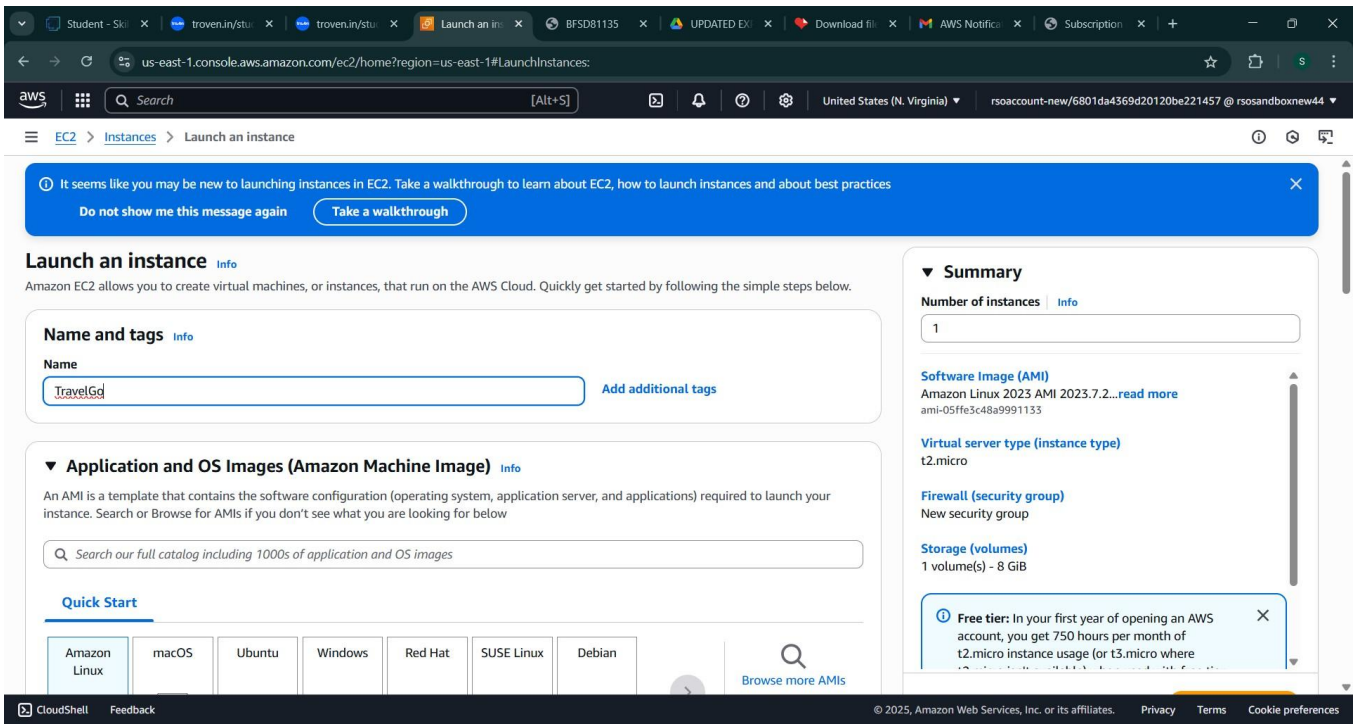
In the AWS Console, navigate to EC2 and launch a new instance.



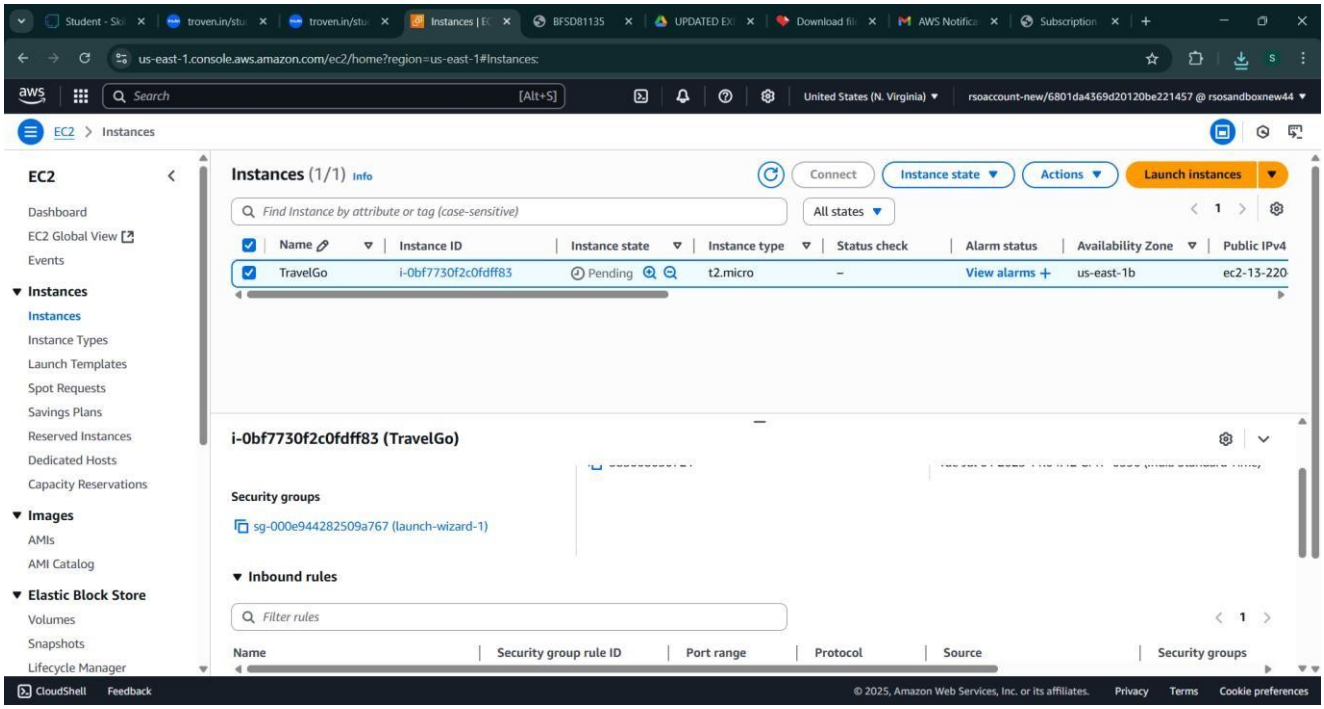


- Click on Launch instance to launch EC2 instance

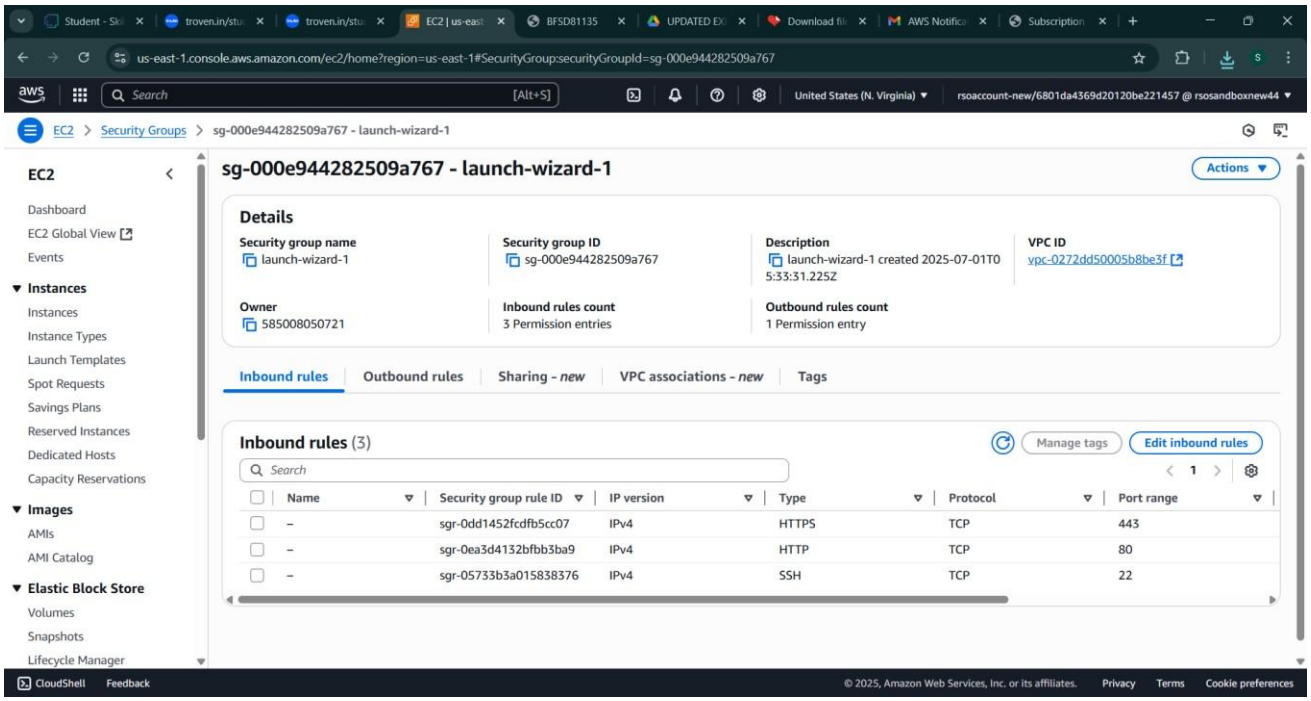
- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



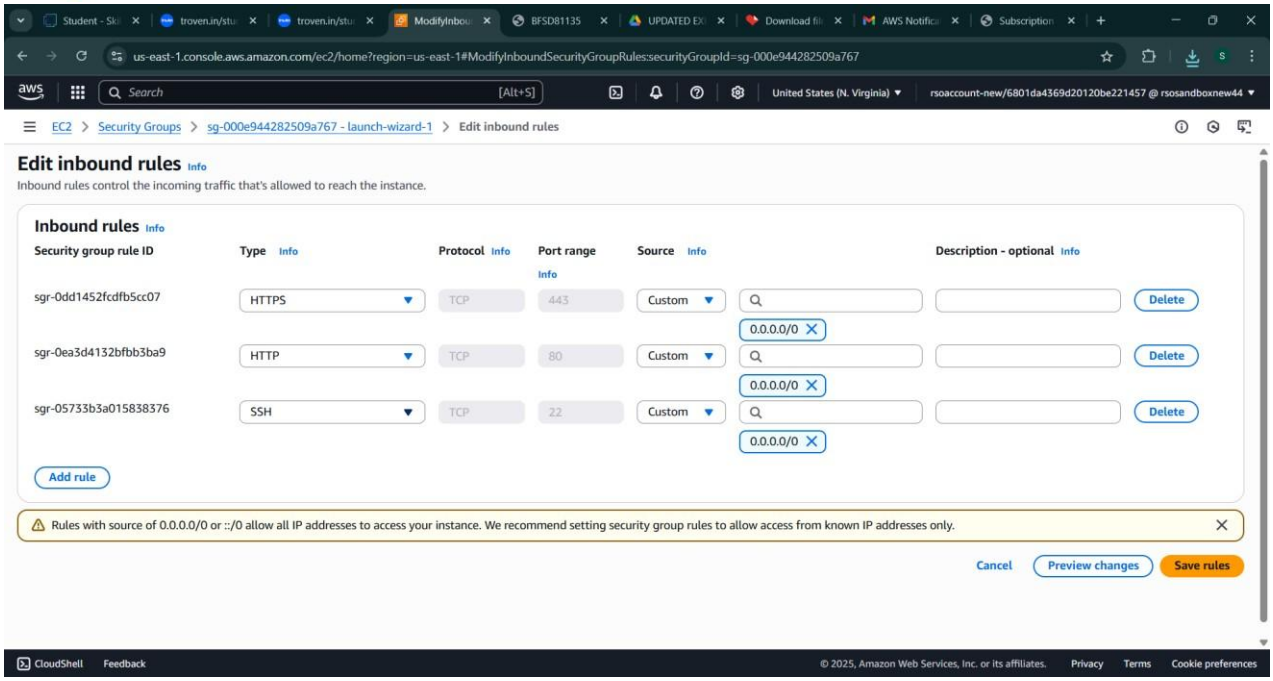
- Create and download the key pair for Server access.

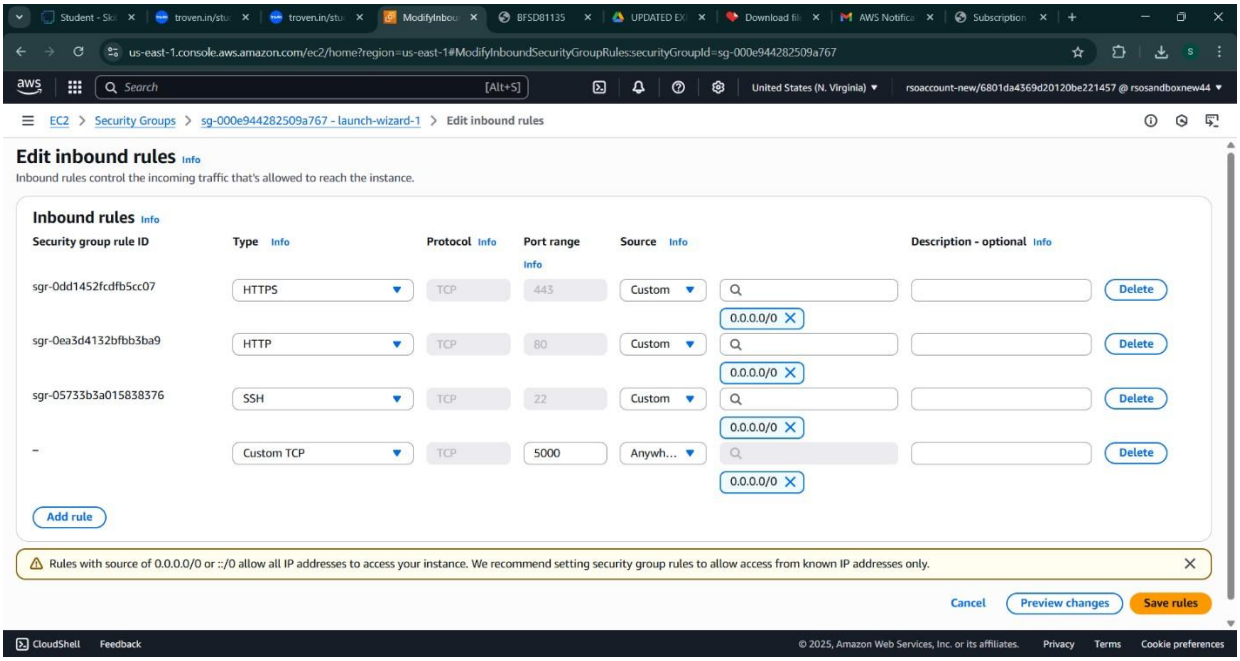


- **Activity 6.2:Configure security groups for HTTP, and SSH access.**

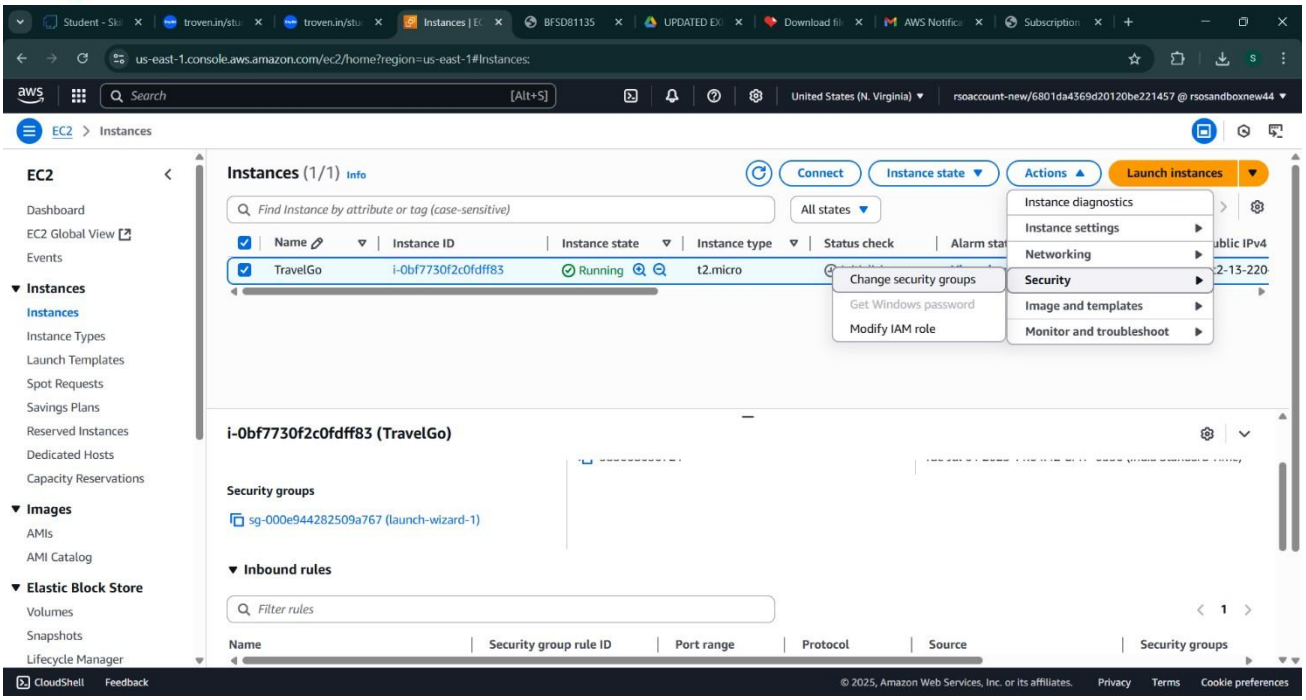


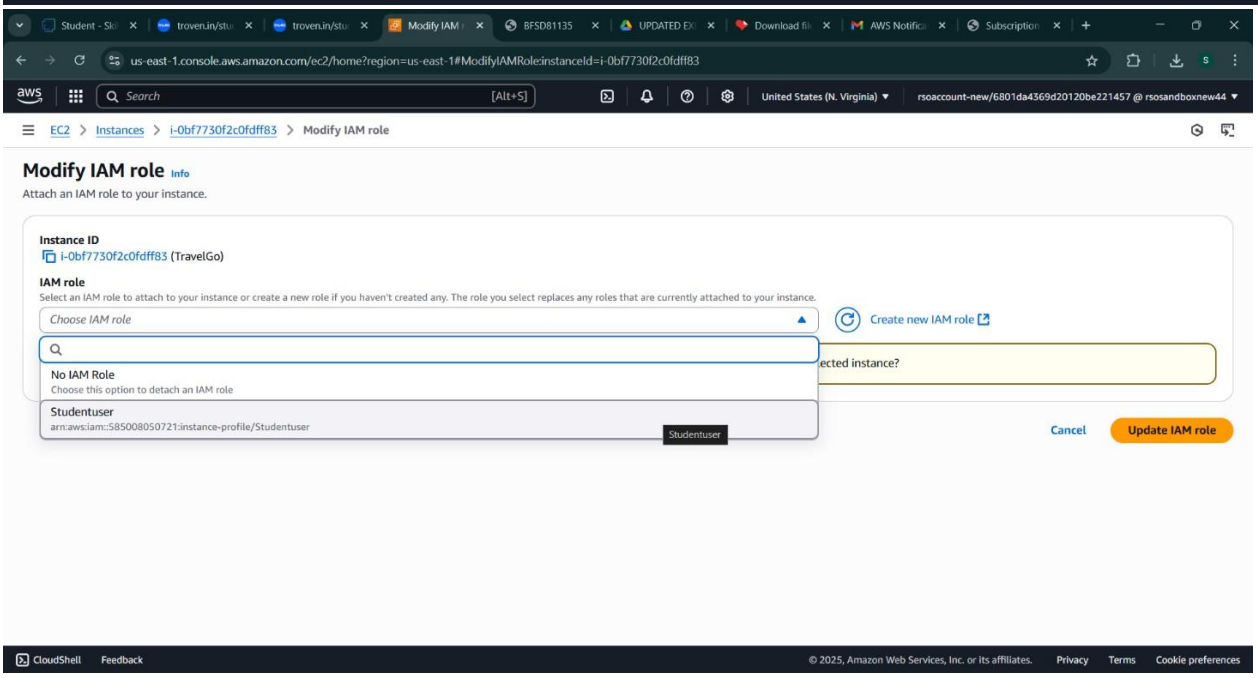
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**



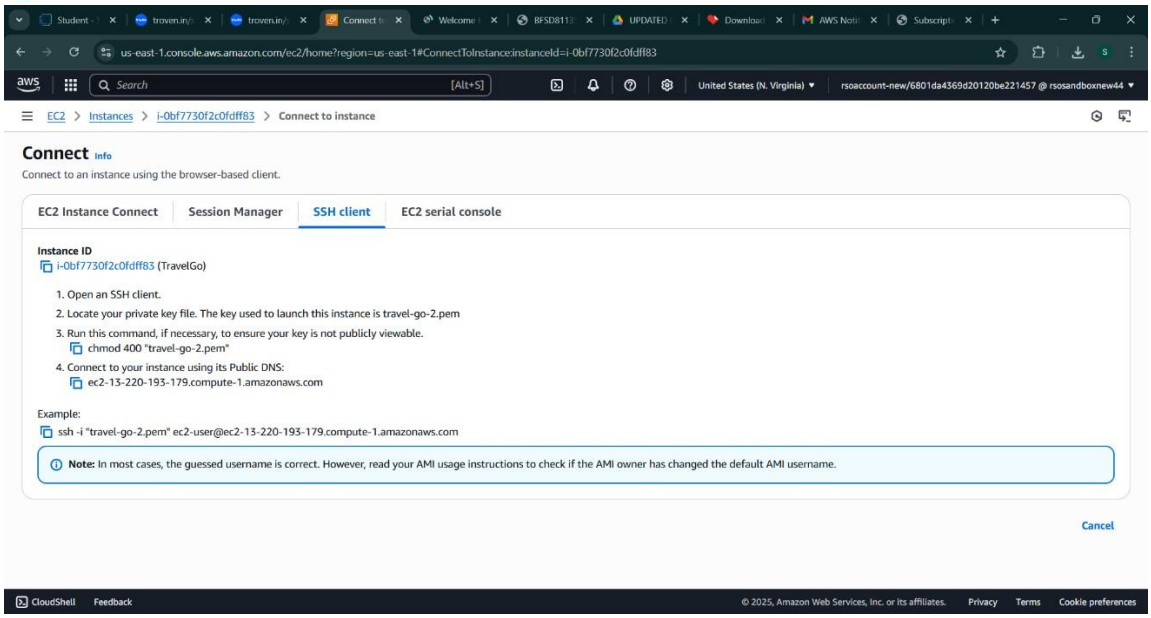


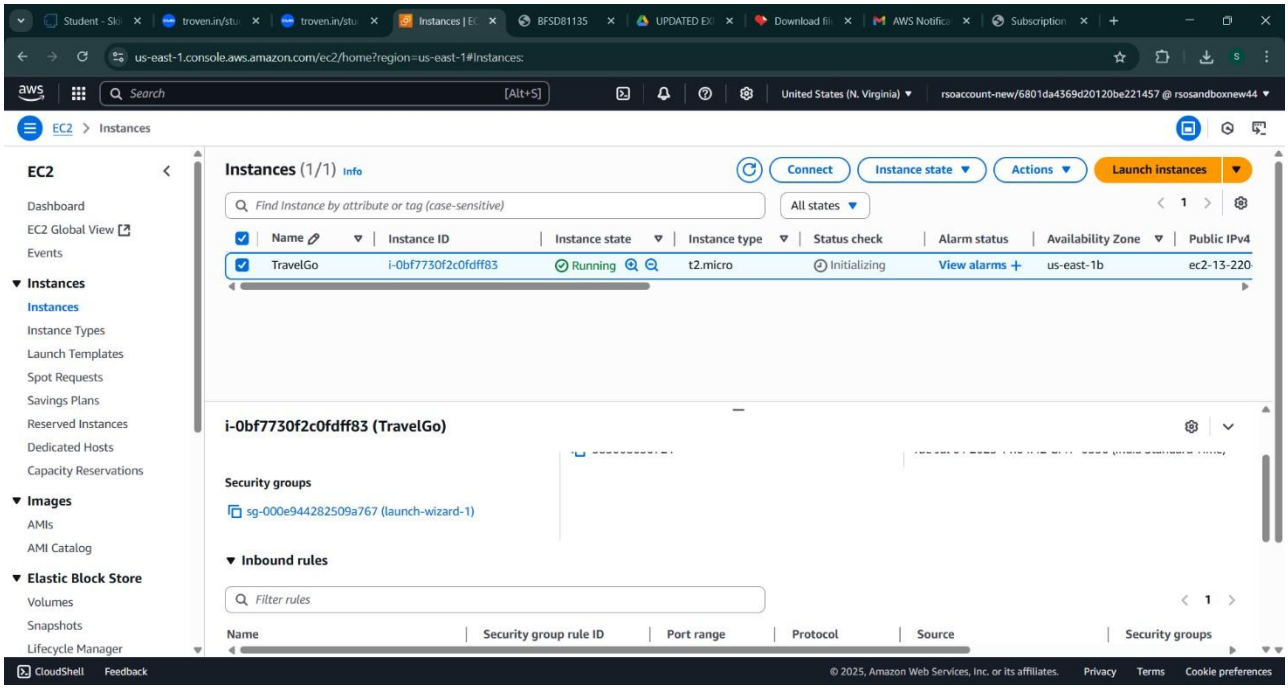
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the 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.





- Now connect the EC2 with the files





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/your-github-username/your-repository-name.git'

Note: change your-github-username and your-repository-name with your credentials here:

'git clone https://github.com/Thanmayee2/movie-magic-flask' ● This will download your project to the EC2 instance.

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

cd InstantLibrary

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**

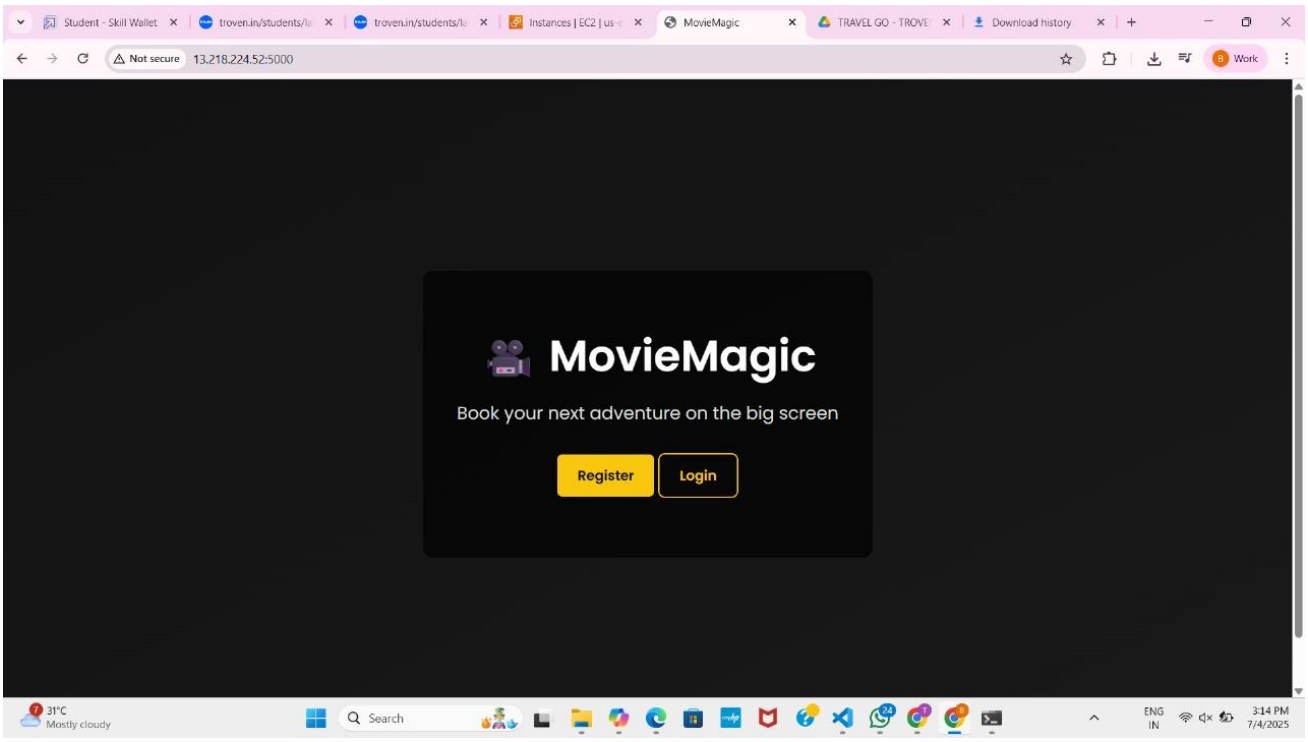
app is running:

http://your-ec2-public-ip

○ Run the Flask app on the EC2 instance

```
  Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: zipp, markupsafe, werkzeug, jinja2, itsdanger
ous, importlib-metadata, click, blinker, flask
Successfully installed blinker-1.9.0 click-8.1.8 flask-3.1.1 importlib-metad
ata-8.7.0 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3 zi
pp-3.23.0
[ec2-user@ip-172-31-26-55 Travel-go]$ pip install boto3
Defaulting to user installation because normal site-packages is not writeabl
e
Collecting boto3
  Downloading boto3-1.39.3-py3-none-any.whl (139 kB)
     |████████████████████████████████| 139 kB 16.3 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
     |████████████████████████████████| 85 kB 7.6 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/
site-packages (from boto3) (0.10.0)
Collecting botocore<1.40.0,>=1.39.3
  Downloading botocore-1.39.3-py3-none-any.whl (13.8 MB)
     |████████████████████████████████| 13.8 MB 39.1 MB/s
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/s
ite-packages (from botocore<1.40.0,>=1.39.3->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/pytho
n3.9/site-packages (from botocore<1.40.0,>=1.39.3->boto3) (2.8.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages
(from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.3->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.3 botocore-1.39.3 s3transfer-0.13.0
[ec2-user@ip-172-31-26-55 Travel-go]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deploym
ent. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.26.55:5000
Press CTRL+C to quit
 * Restarting with stat
```

Testing and Deployment



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.

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

THANK YOU