Movie Magic–Smart Movie Ticket Booking System

Project Overview :

Movie Magic is a cloud-enabled web application that allows users to browse movies, select show timings, and book movie tickets online. The system is powered by Python Flask for backend logic, AWS DynamoDB for storing booking data, AWS SNS for sending real-time booking confirmations, and AWS EC2 for deployment. The project demonstrates how cloud services can be used to build scalable, reliable, and real-time web applications.

Scenario 1: User Browses and Books a Ticket

- User visits the Movie Magic homepage.
- The system displays a list of available movies with show timings.
- User fills in their name and email, selects a movie and timing, and clicks "Book Now."
- Flask backend saves the booking in AWS DynamoDB.
- The user receives a confirmation email via AWS SNS.
- The confirmation page is displayed with their booking details.

Scenario 2: User Enters Invalid Email Address

- User inputs an invalid or misspelled email.
- Form submission fails validation or AWS SNS can't deliver the email.
- Flask may still save the booking, but the email notification does not reach the user.

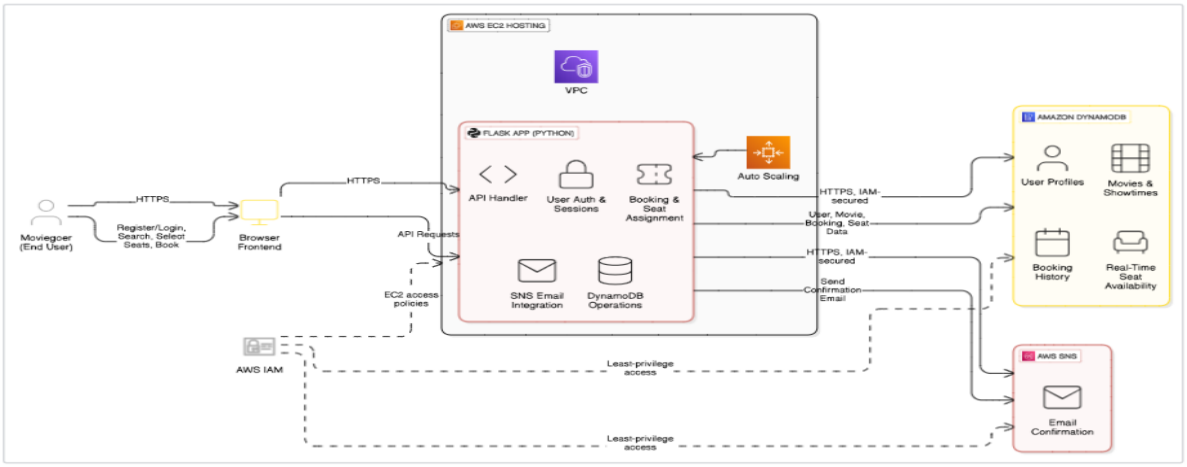
Scenario 3: Admin Views Booking Records (Future Enhancement)

- Admin logs in to a protected route (not implemented yet).
- Flask app fetches all booking records from DynamoDB.
- Admin can view or export booking data.

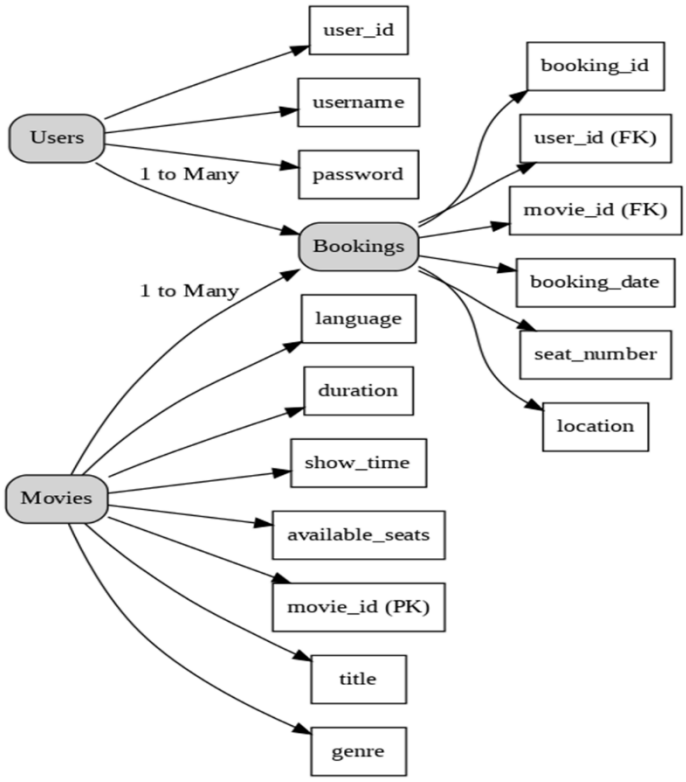
Scenario 4: SMS Booking Confirmation (Optional via SNS)

- User selects phone number option and enters mobile number.
- AWS SNS sends SMS confirmation instead of email.

AWS ARCHITECTURE



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: Set up an AWS account if not already done**.**

Activity 1.2: Log in to the AWS Management Console

2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.

Activity 2.2: Configure Attributes for User Data and Book Requests.

3. SNS Notification Setup

Activity 3.1: Create SNS topics for book request notifications.

Activity 3.2: Subscribe users and library staff to SNS email notifications.

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

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

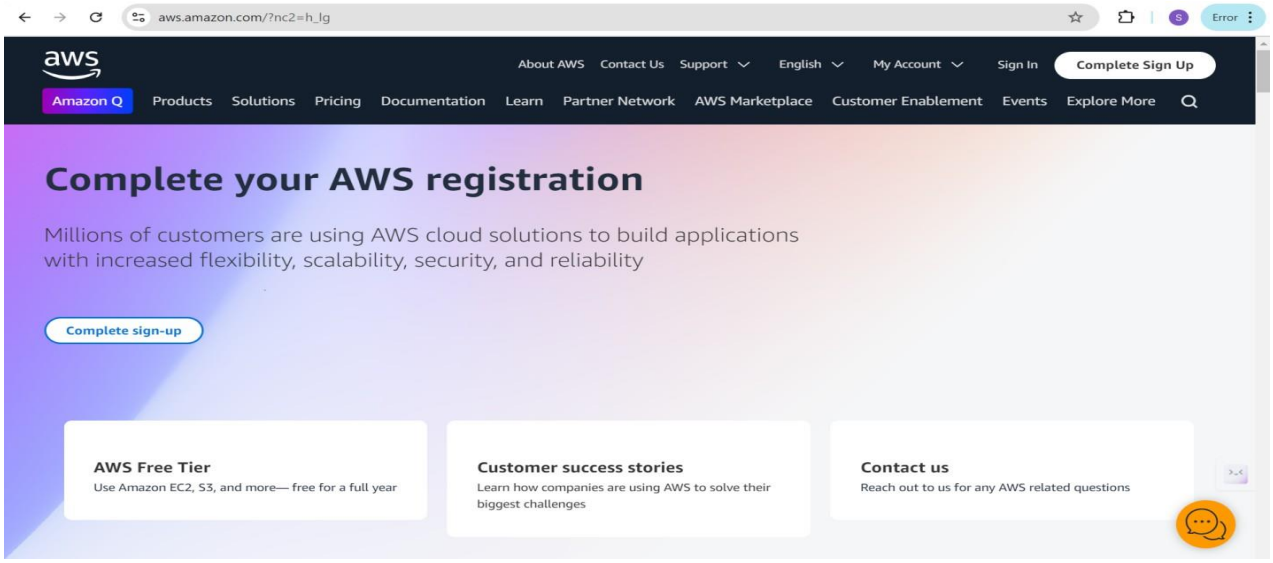
Activity 7.1:Upload Flask Files

Activity 7.2: Run the Flask App

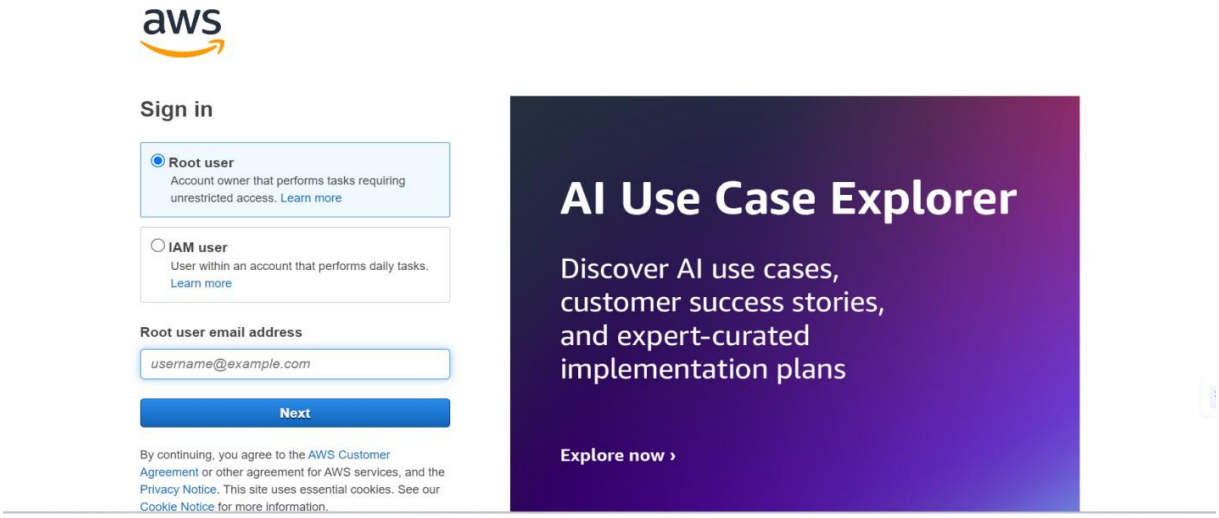
8. Testing and Deployment

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

1: AWS Account Setup and Login



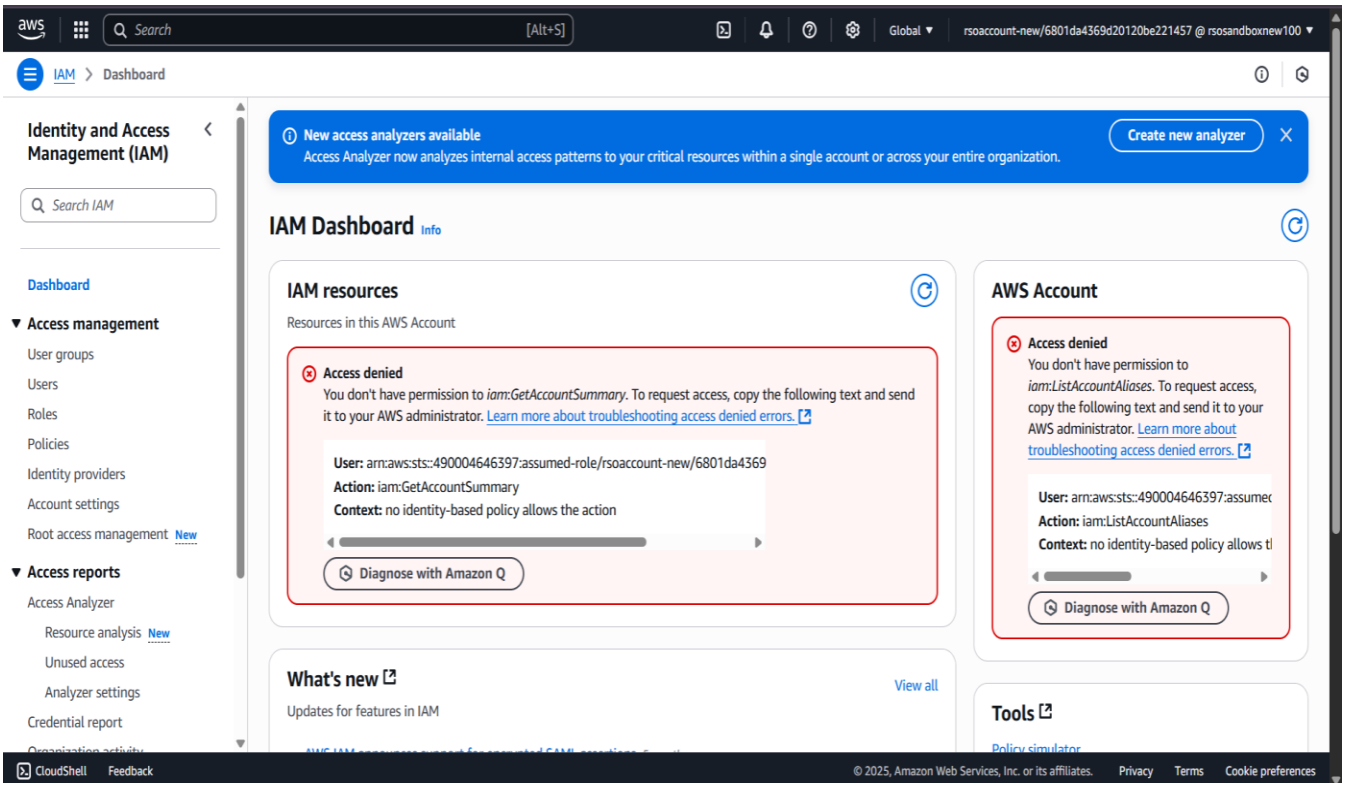
2: Log in to the AWS Management Console.

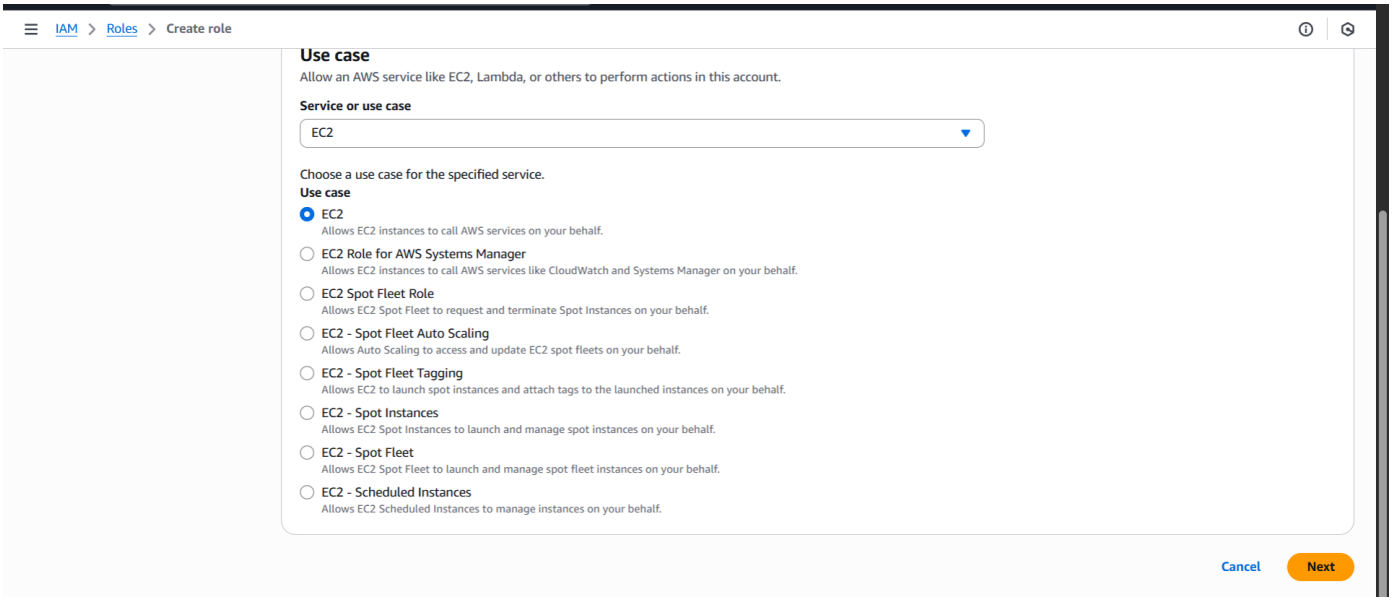


After logging into the AWS Console:

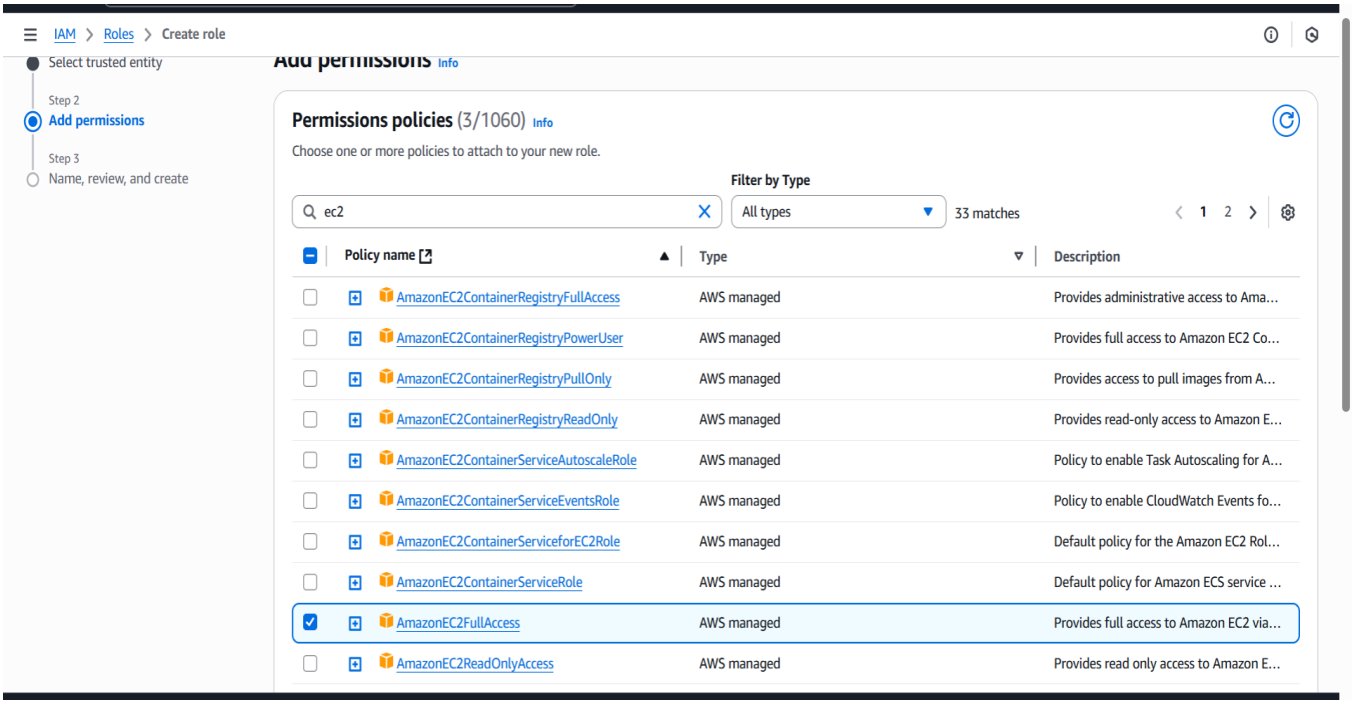
Once you're logged in, you can access a wide range of AWS services from the dashboard. Use the search bar at the top to quickly navigate to services like EC2, DynamoDB, SNS, or IAM as needed for your project setup.

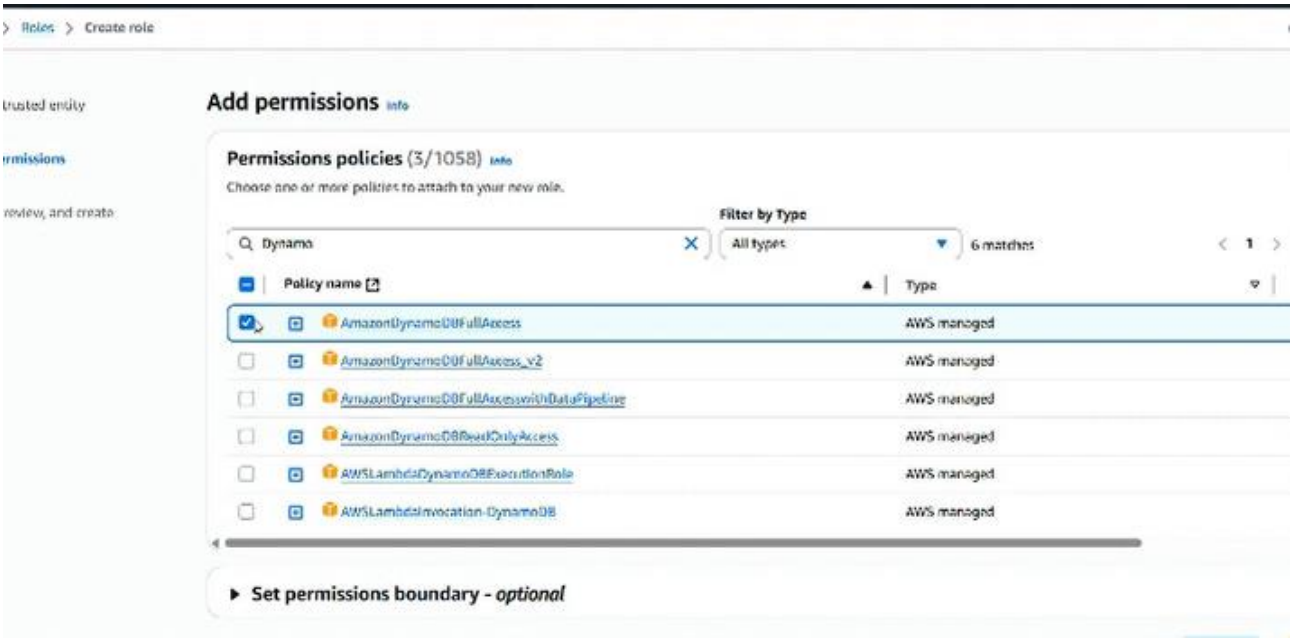
3. IAM Roles Creation

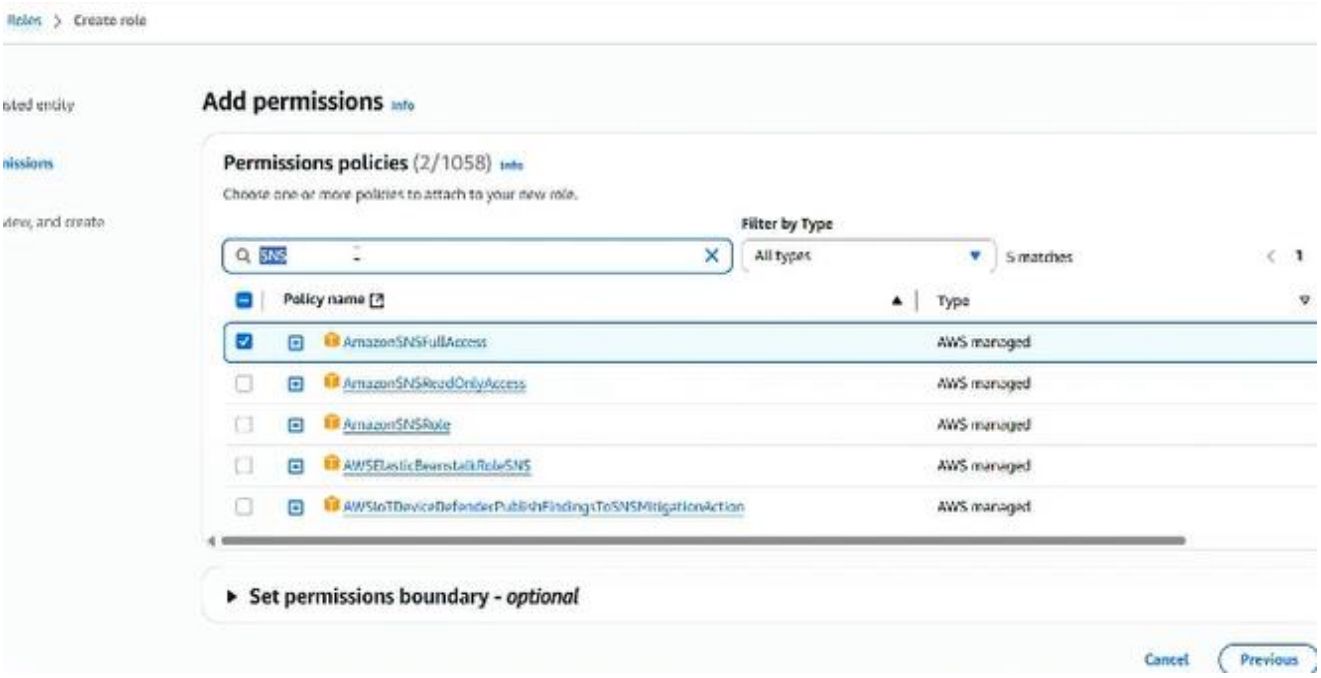




Before assigning permissions, ensure that the IAM role is created and ready to be attached with appropriate AWS-managed policies for each required service.

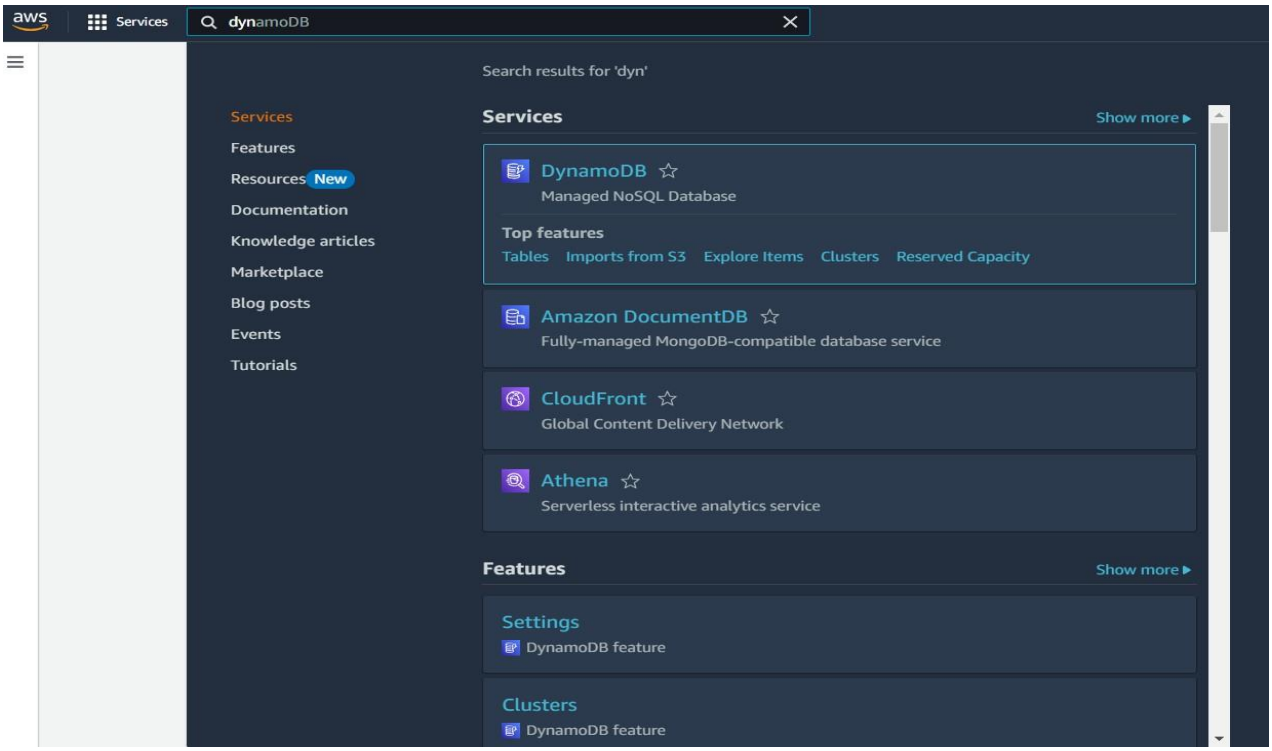




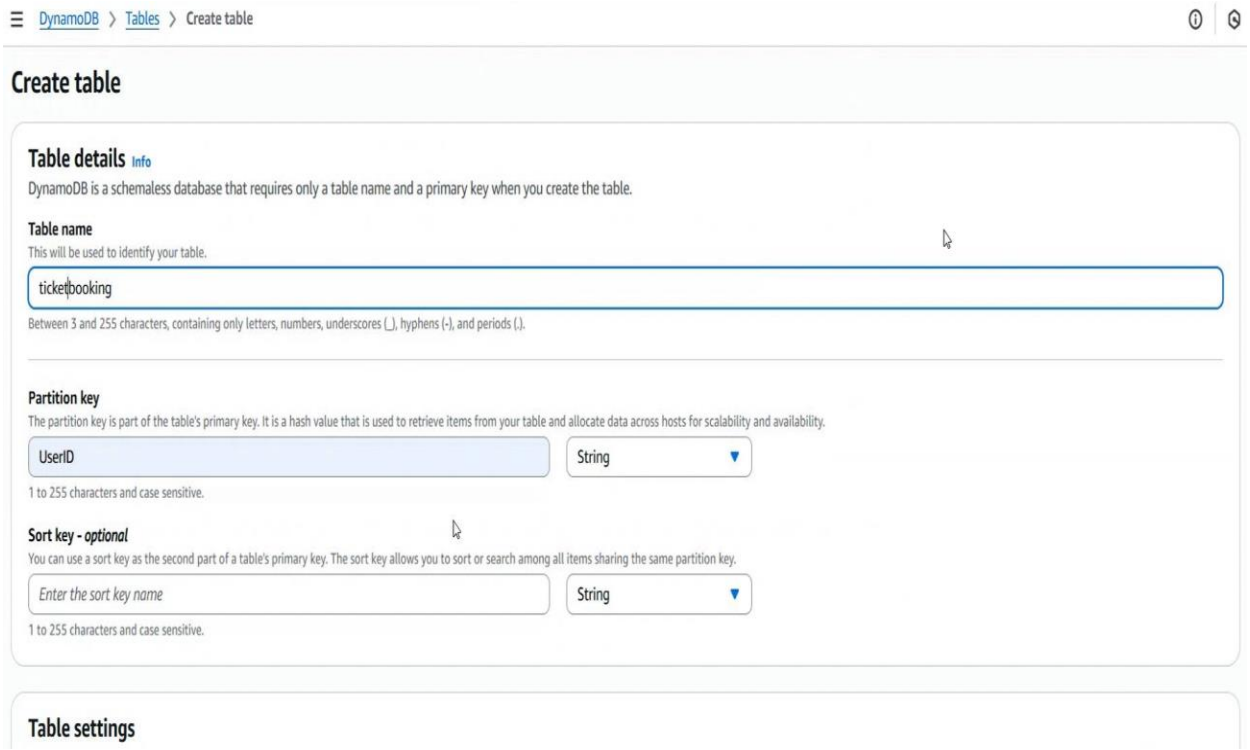


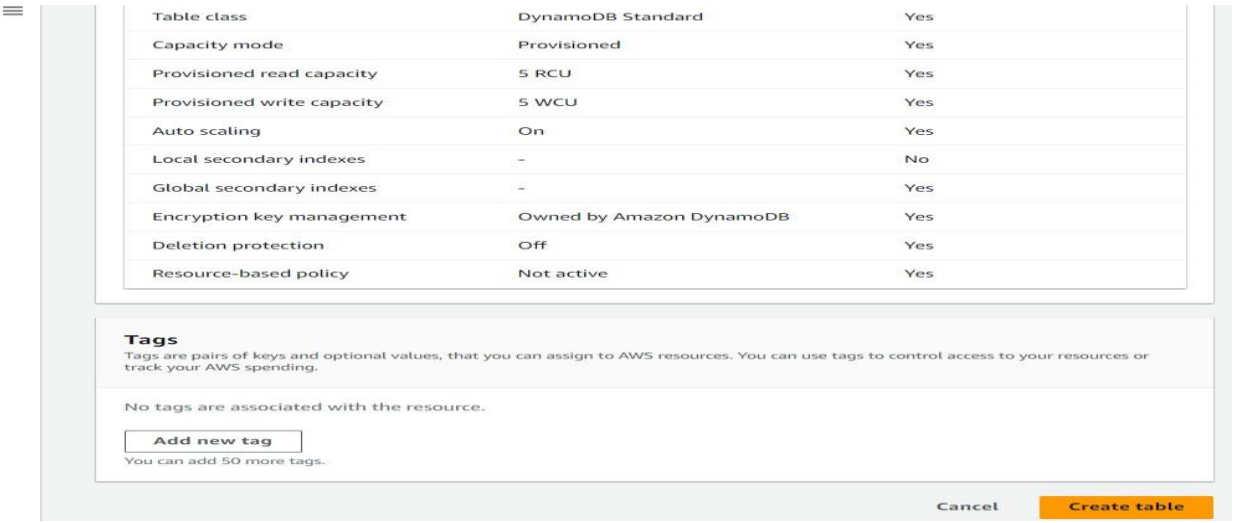
DynamoDB Database Creation and Setup

- In the AWS Console, navigate to DynamoDB and click on create tables



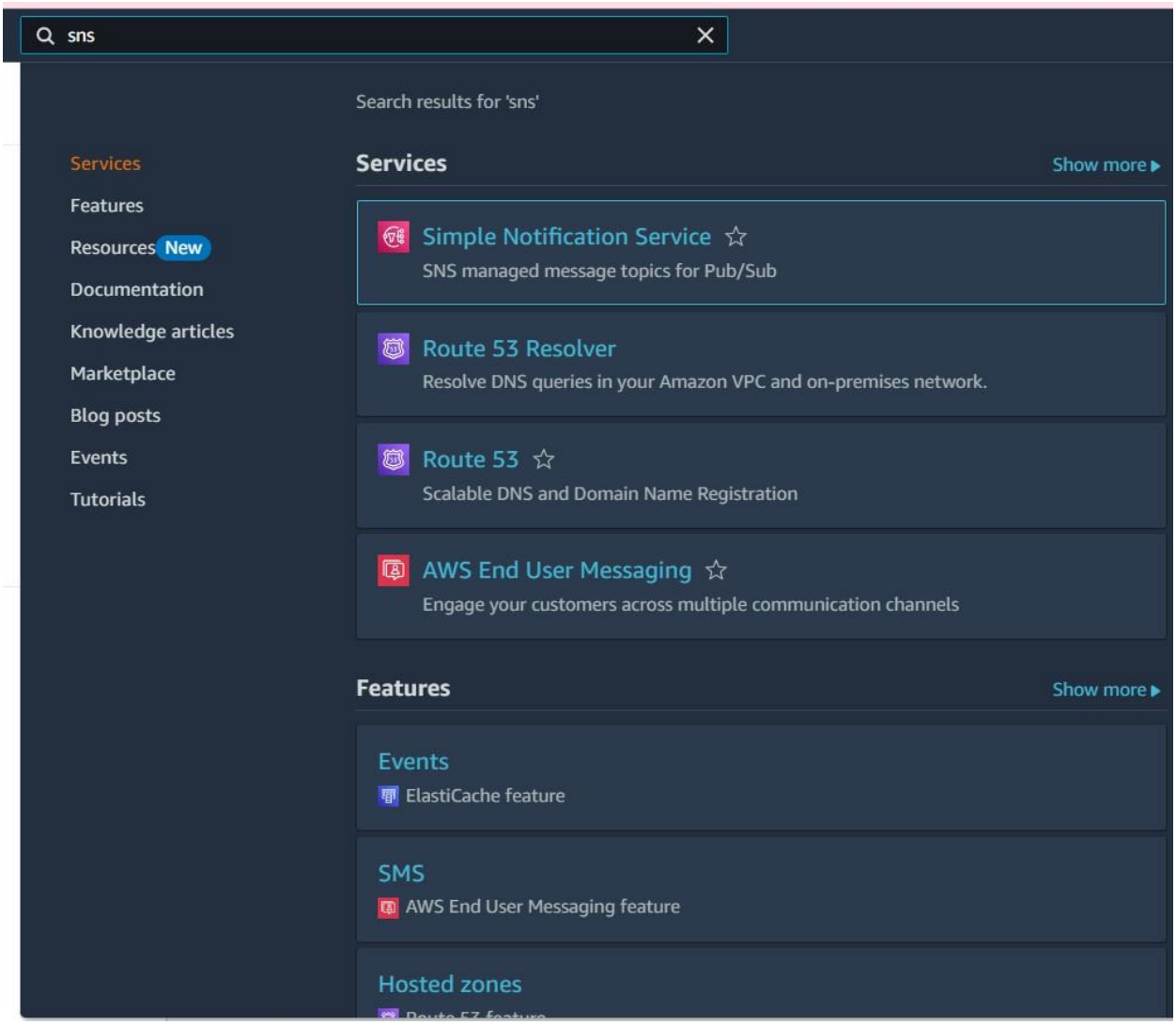
Create Users table with partition key “Email” with type String and click on create tables.

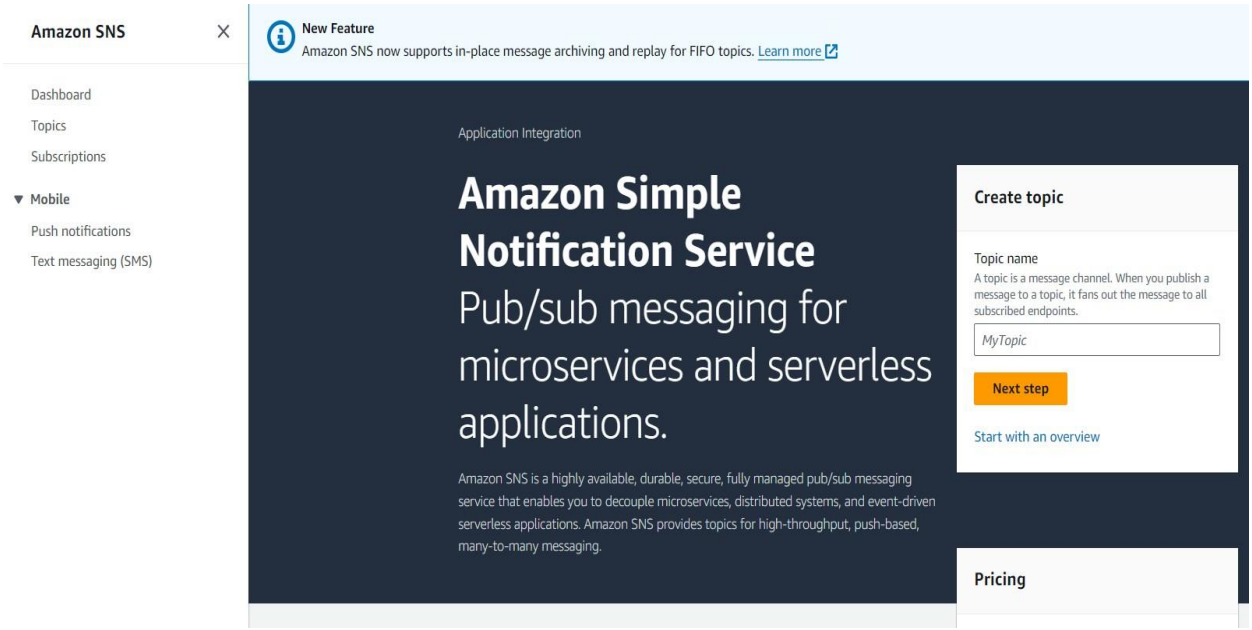




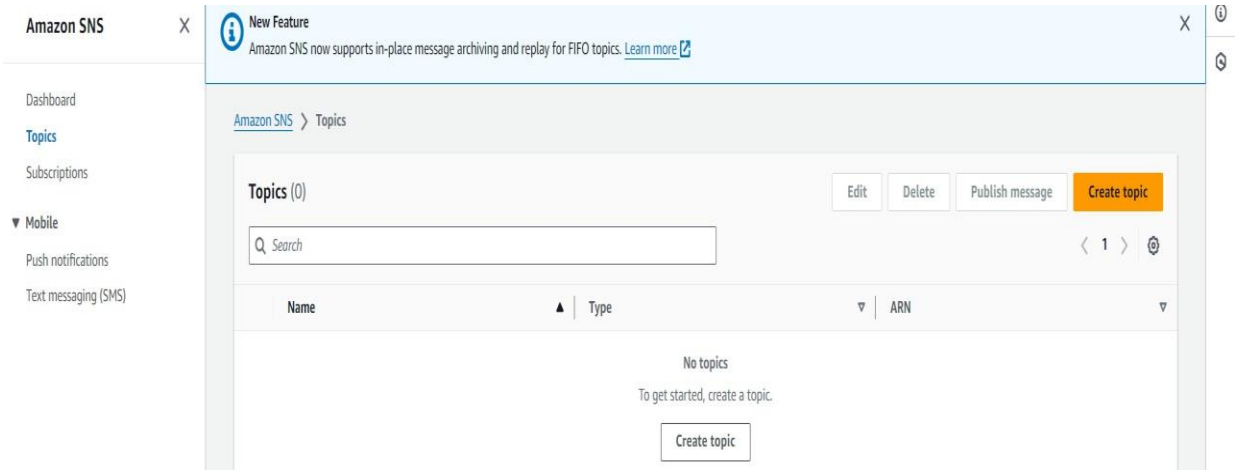
SNS Notification Setup

1. Create SNS topics for sending email notifications to users.

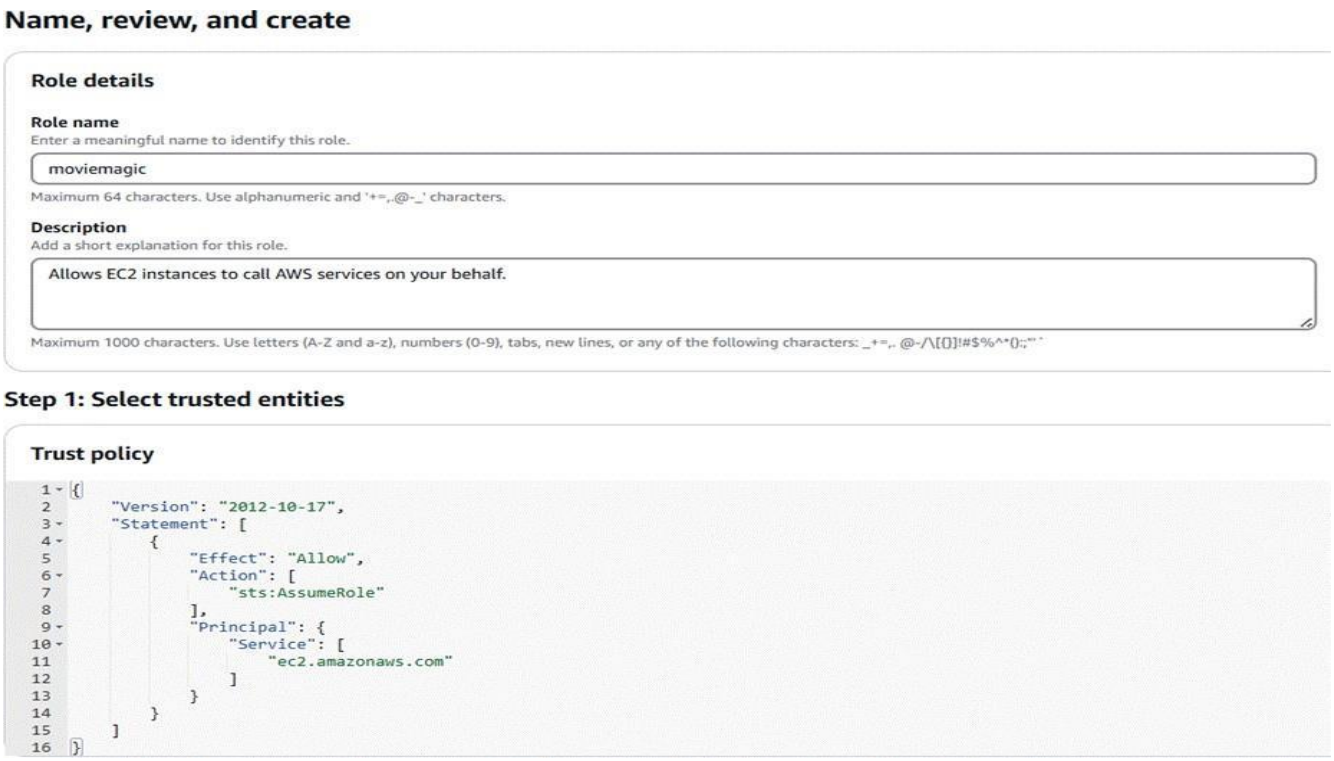




- Click on **Create Topic** and choose a name for the topic.

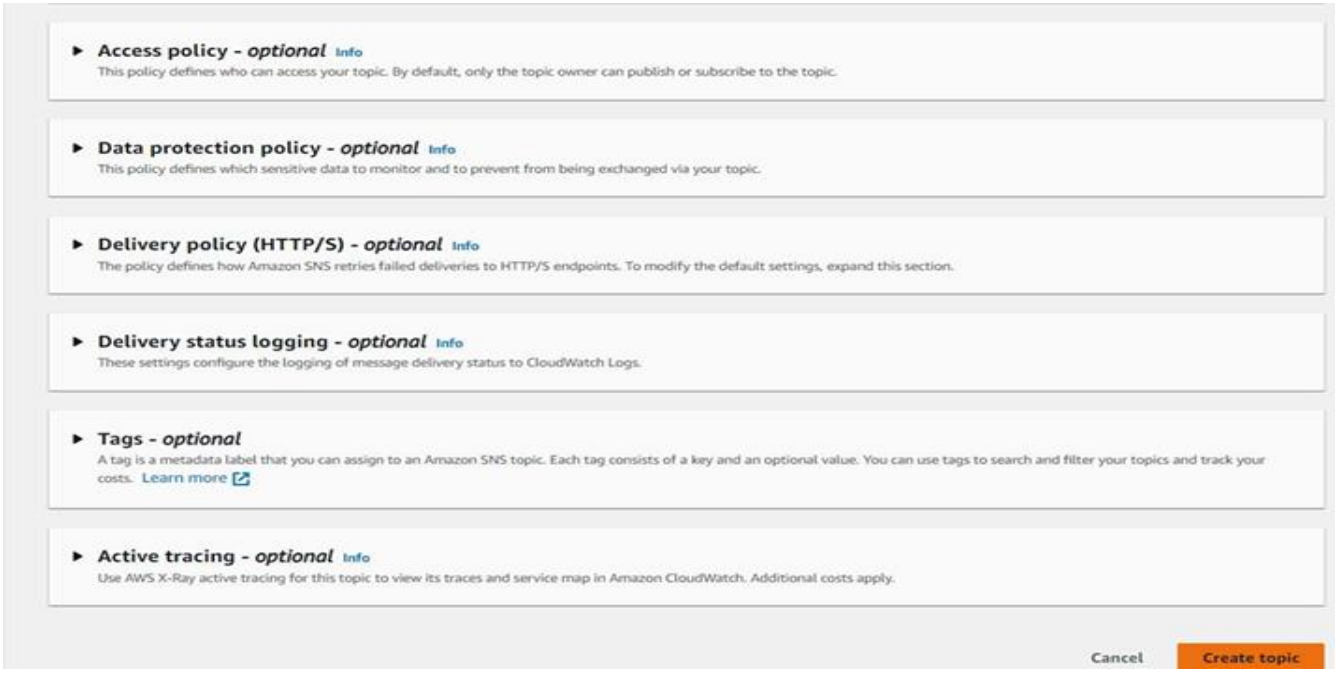


- Choose Standard type for general notification use cases and Click on Create Topic.



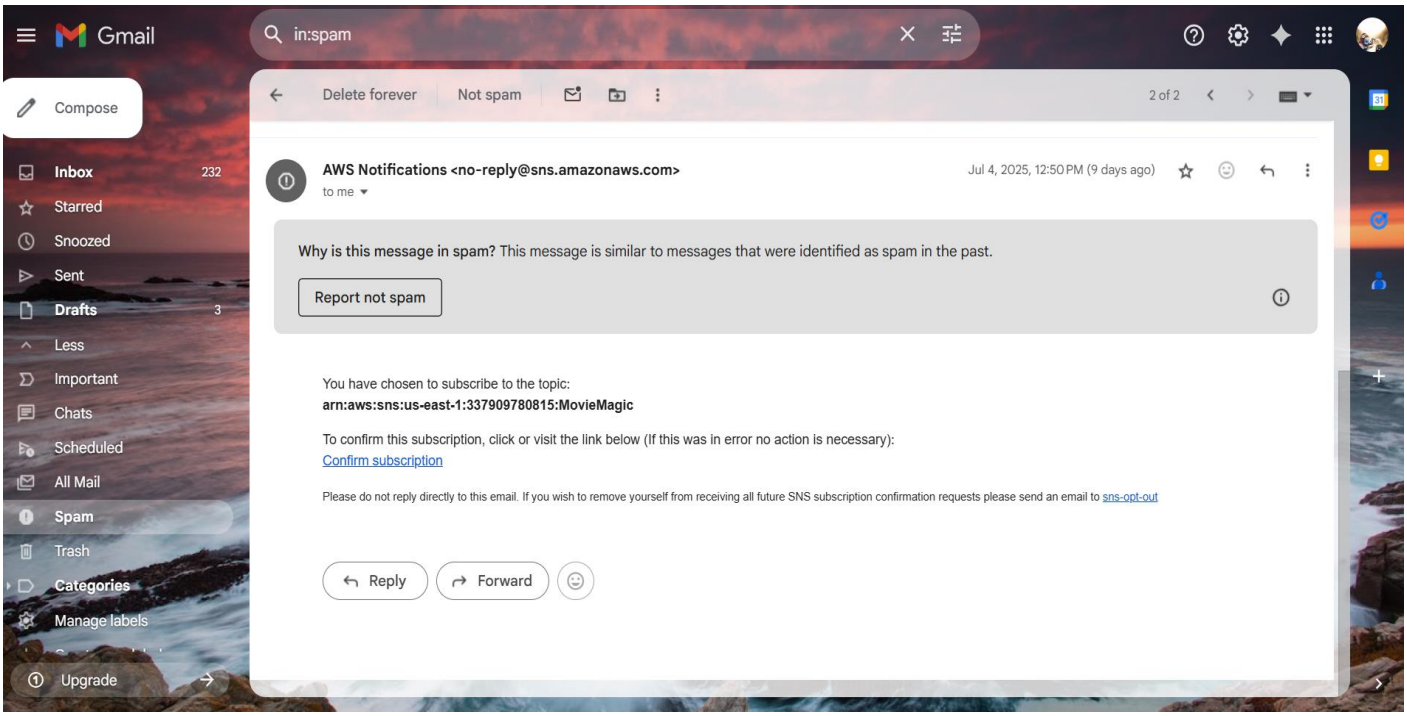
- Click on create topic

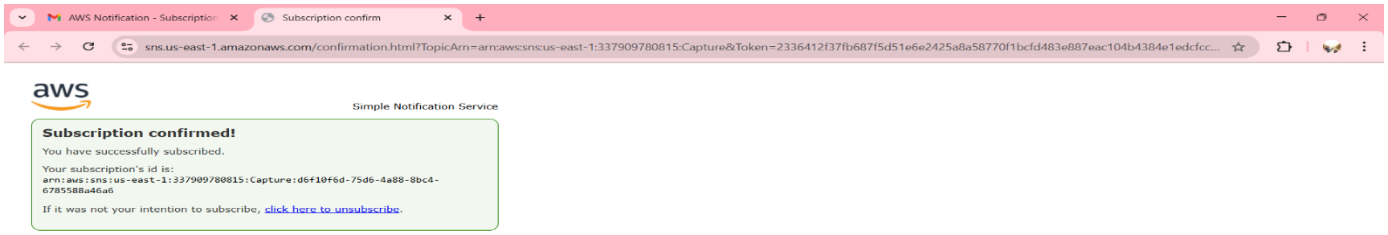
To begin configuring notifications, navigate to the SNS dashboard and click on “CreateTopic.” Choose the topic type (Standard or FIFO) based on your requirements. Provide a meaningful name for the topic that reflects its purpose (e.g., MovieMagic). This topic will serve as the communication channel for sending notifications to subscribed users



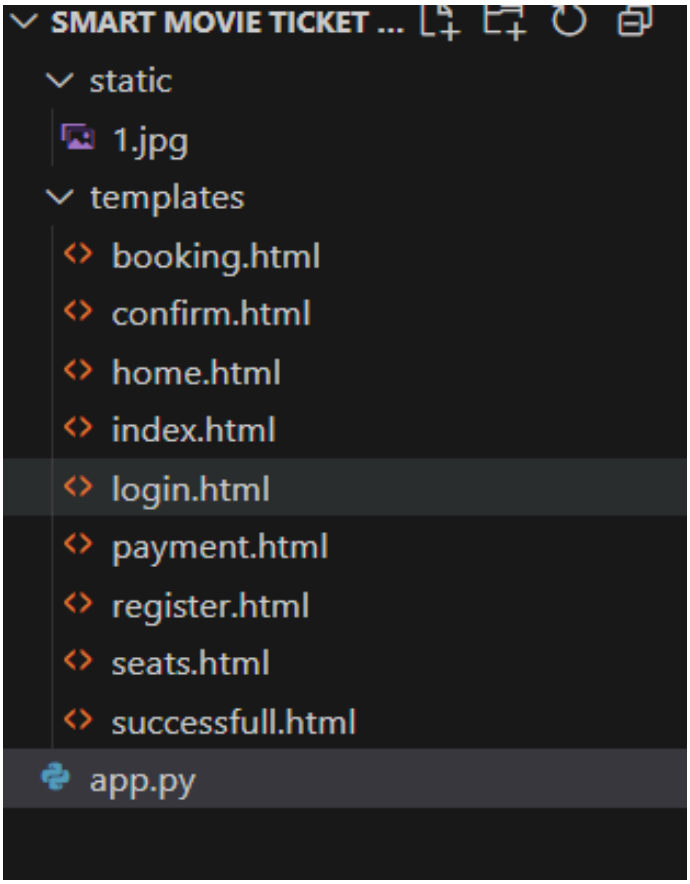
Click on create Subscription:

Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail
Once the email subscription is confirmed, the endpoint becomes active for receiving notifications. This ensures that users will instantly get booking confirmations or alerts.
You can manage or remove subscriptions anytime via the SNS dashboard. It's recommended to test the setup by publishing a sample message to verify successful delivery.





Backend Development and Application Setup



Develop the backend using Flask

Initialize the Flask application instance using Flask(_name_) to start building the web app.

app.py > ...

```python
from flask import Flask, render_template, request, redirect, url_for
import uuid

app = Flask(__name__)

users = {}

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

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        password = request.form['password']
        if email in users:
            return "User already exists!"
        users[email] = {'name': name, 'password': password}
        return redirect(url_for('login'))
    return render_template('register.html')
```

- Flask: Framework used to build the web server
- render_template: Renders HTML templates from the templates/ folder
- request: Gets form data from user input
- redirect, url_for: Redirects users between pages
- uuid: Generates unique IDs for ticket confirmation

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = users.get(email)
        if user and user['password'] == password:
            return redirect(url_for('home'))
        return "Invalid credentials!"
    return render_template('login.html')

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

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

@app.route('/seats', methods=['GET'])
def seats():
    movie = request.args.get('movie')
    price = request.args.get('price')
    time = request.args.get('time')
    location = request.args.get('location')
    return render_template('seats.html', movie=movie, price=price, time=time, location=location)

@app.route('/confirm', methods=['POST'])
def confirm():
    movie = request.form.get('movie')
    location = request.form.get('location')
    time = request.form.get('time')
    seats = request.form.get('seats')
    price = request.form.get('price')
    print("DEBUG:", movie, seats, price, time, location)
    return render_template('confirm.html', movie=movie, location=location, time=time, seats=seats, price=pri
```

```python
    return render_template('confirm.html', movie=movie, location=location, time=time, seats=seats, price=pri

@app.route('/payment', methods=['POST'])
def payment():
    movie = request.form.get('movie')
    seats = request.form.get('seats')
    price = request.form.get('price')
    time = request.form.get('time')
    location = request.form.get('location')
    if not all([movie, seats, price, time, location]):
        return "Missing required parameters", 400
    return render_template('payment.html', movie=movie, seats=seats, price=price, time=time, location=locati

@app.route('/successfull', methods=['GET'])
def successfull():
    movie = request.args.get('movie')
    seats = request.args.get('seats')
    showtime = request.args.get('showtime')
    location = request.args.get('location')
    booking_id = str(uuid.uuid4())[:8].upper()
    return render_template(
        'successfull.html',
        movie=movie,
        showtime=showtime,
        seats=seats,
        location=location,
        booking_id=booking_id
    )

if __name__ == '__main__':
    app.run(host='0.0.0.0',port=5000,debug=True)
```

Deployment Code:

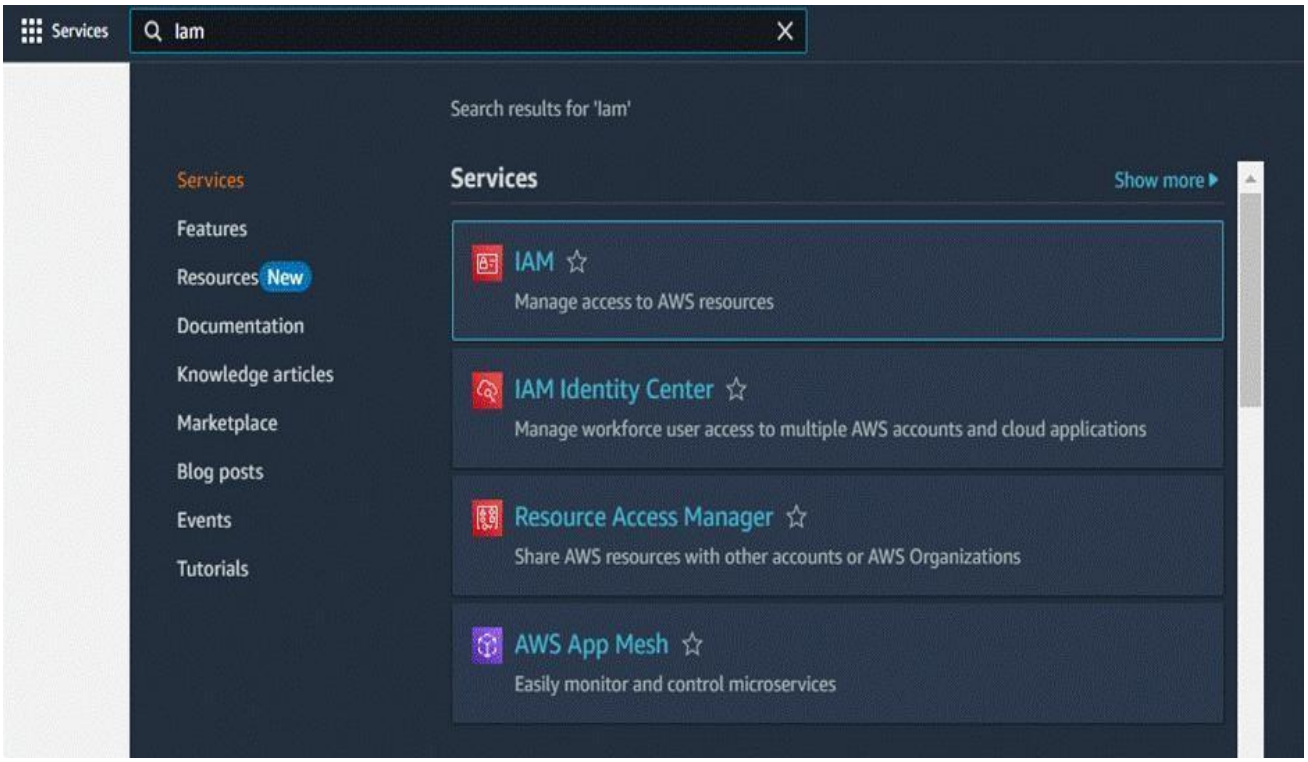
```
if __name__ == '__main__':
    app.run(host='0.0.0.0',port=5000,debug=True)
```

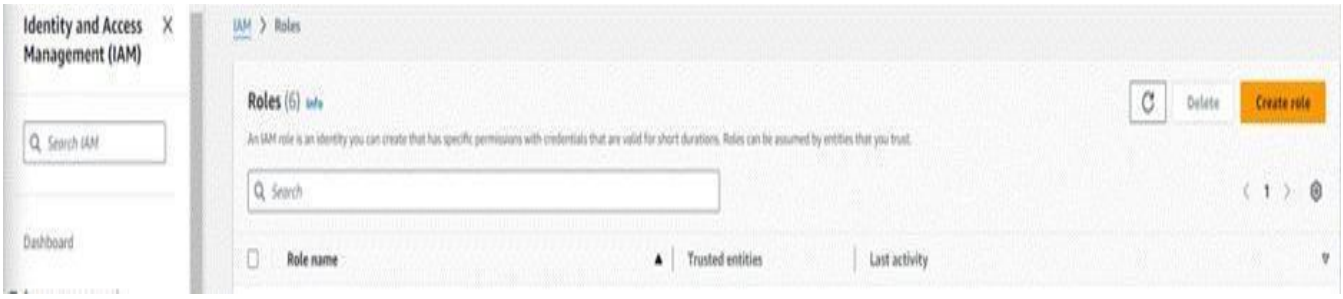
IAM Role Setup:

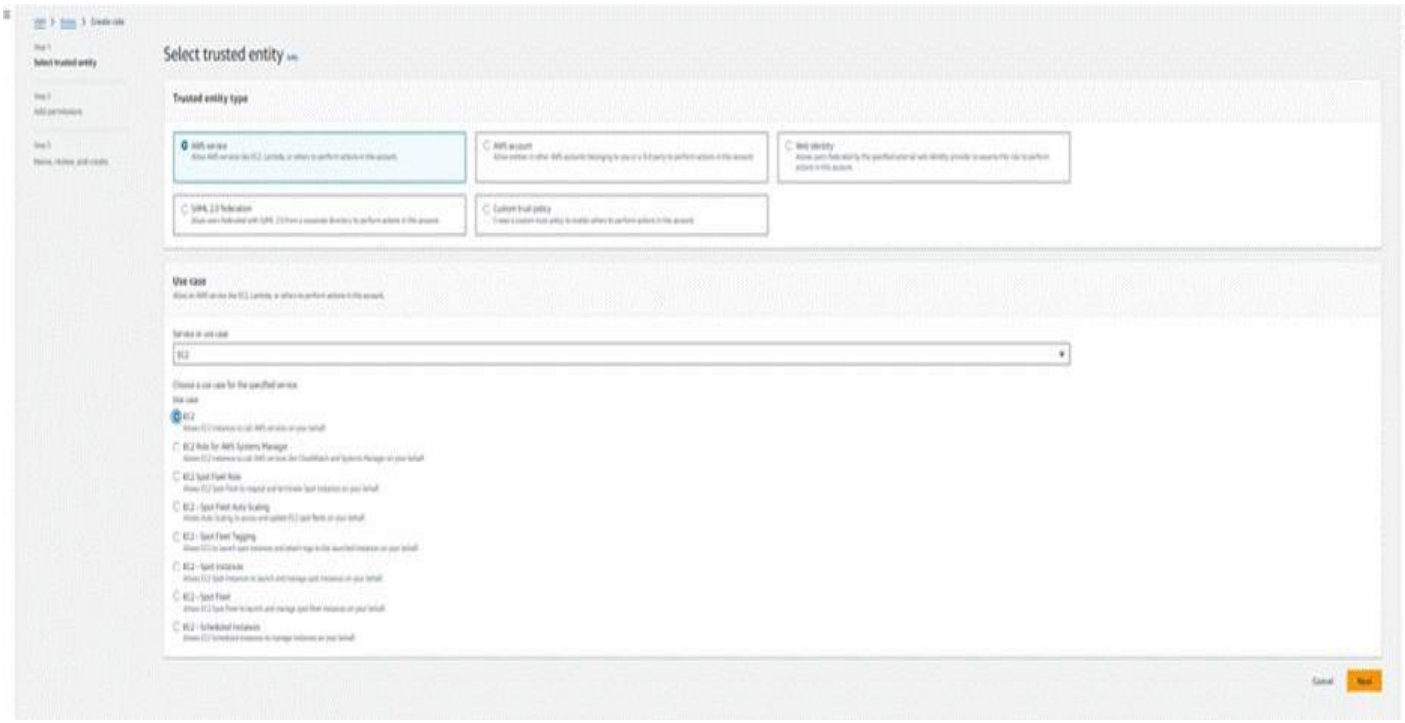
IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows controlled access and ensures security best practices in managing AWS resources.

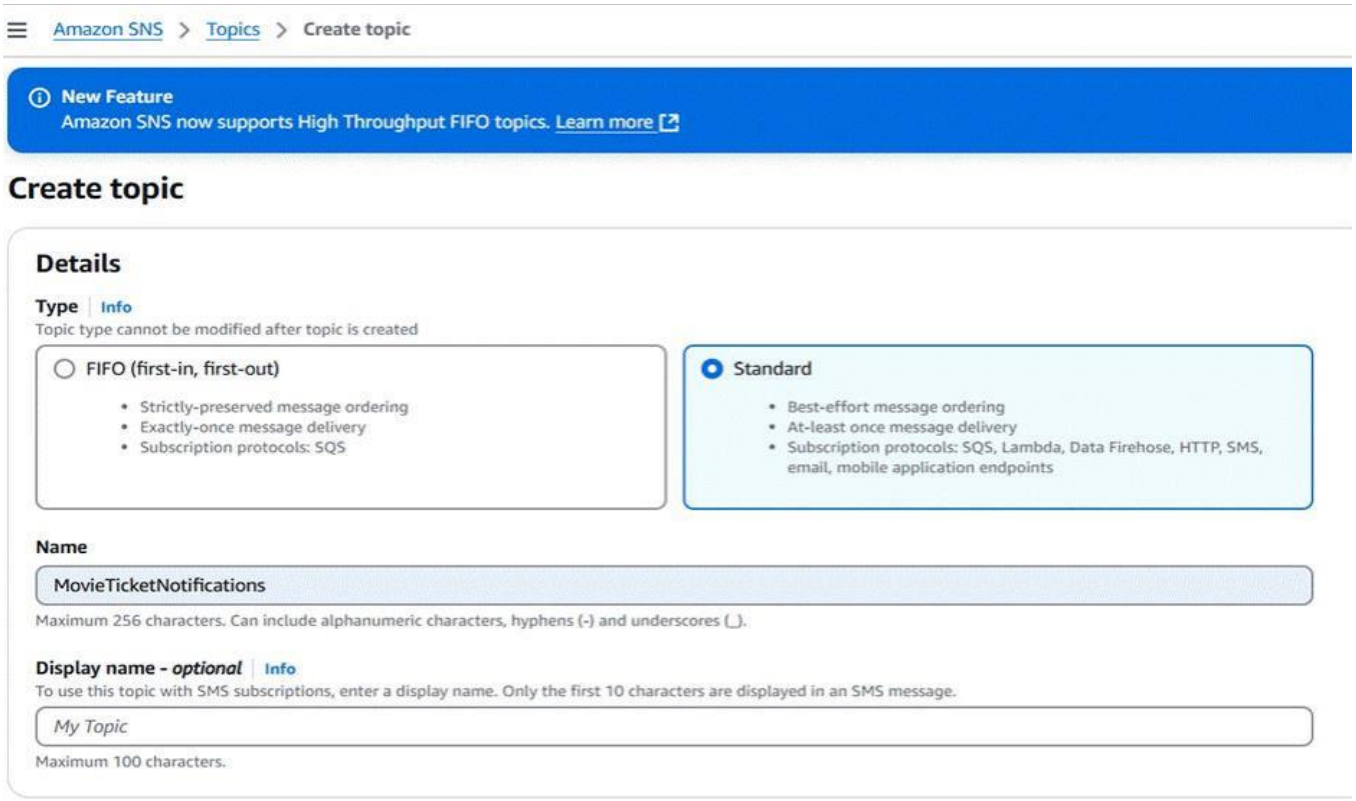
Create IAM Role:

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



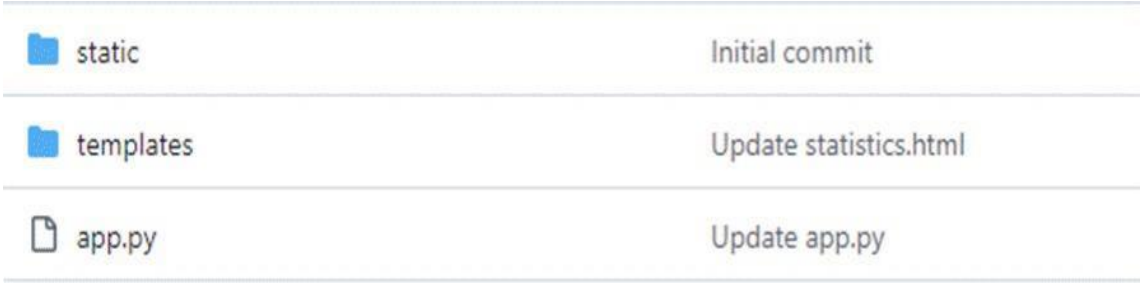

Identity and Access Management (IAM)
IAM > Roles
Roles (6) info
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.
Search IAM
Search
Dashboard
Delete
Create role
Role name
Trusted entities
Last activity

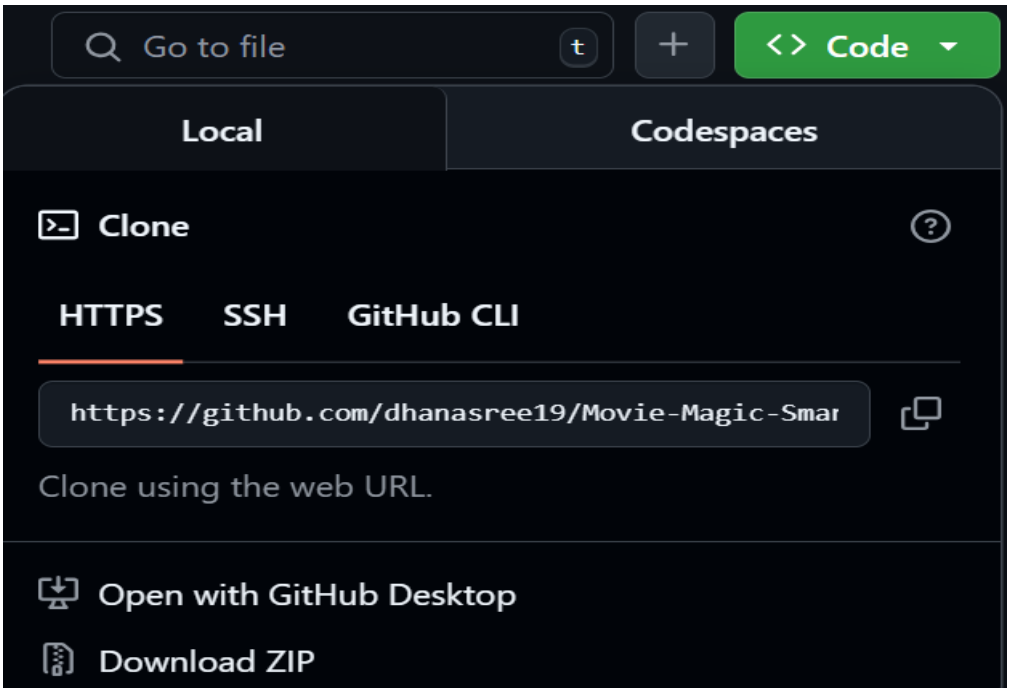

Select trusted entity
Trusted entity type
Use case



Load your Project Files to GitHub

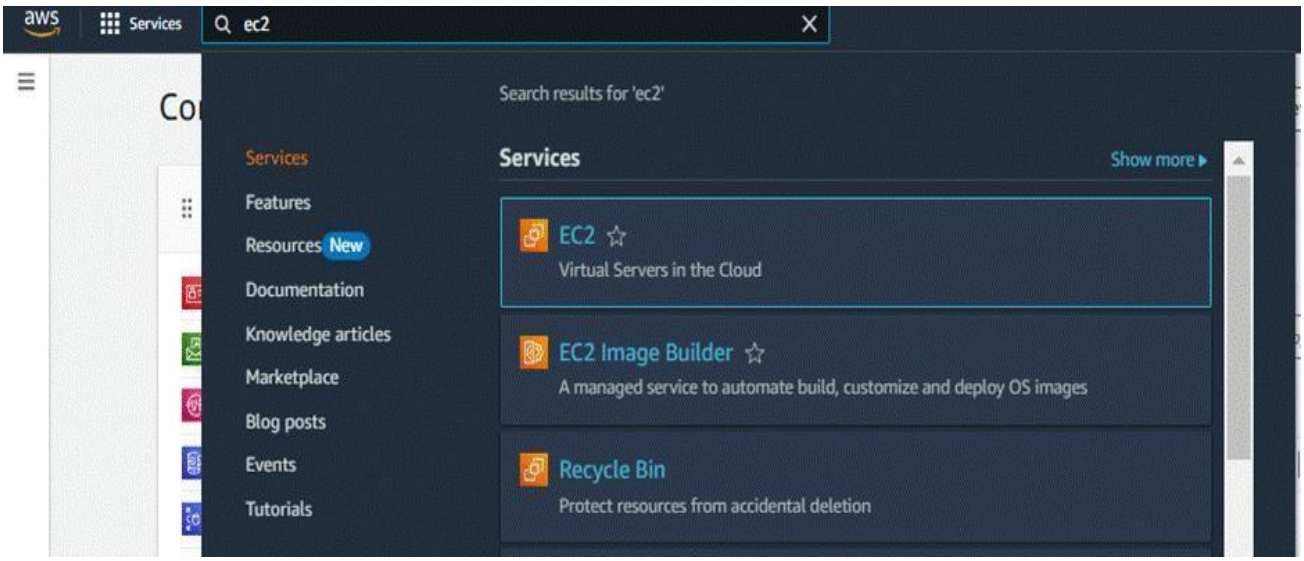
- Load your Flask app and Html files into GitHub repository.



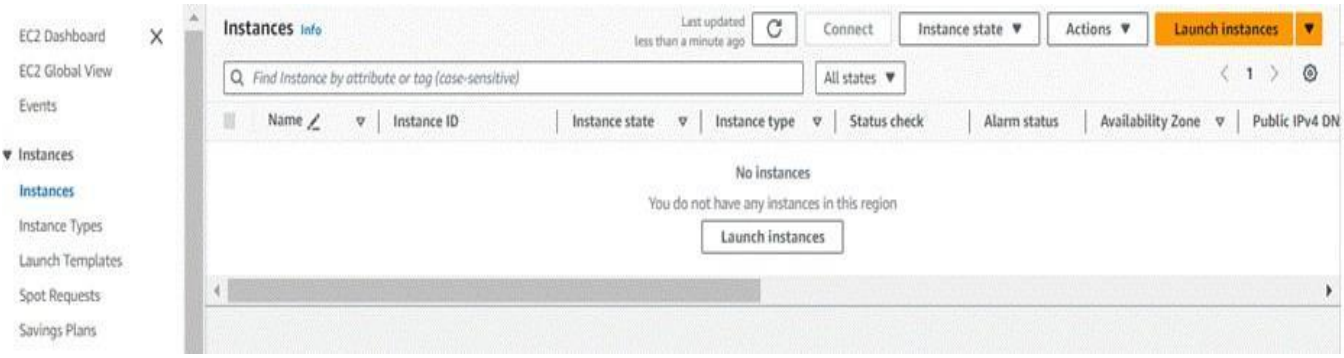


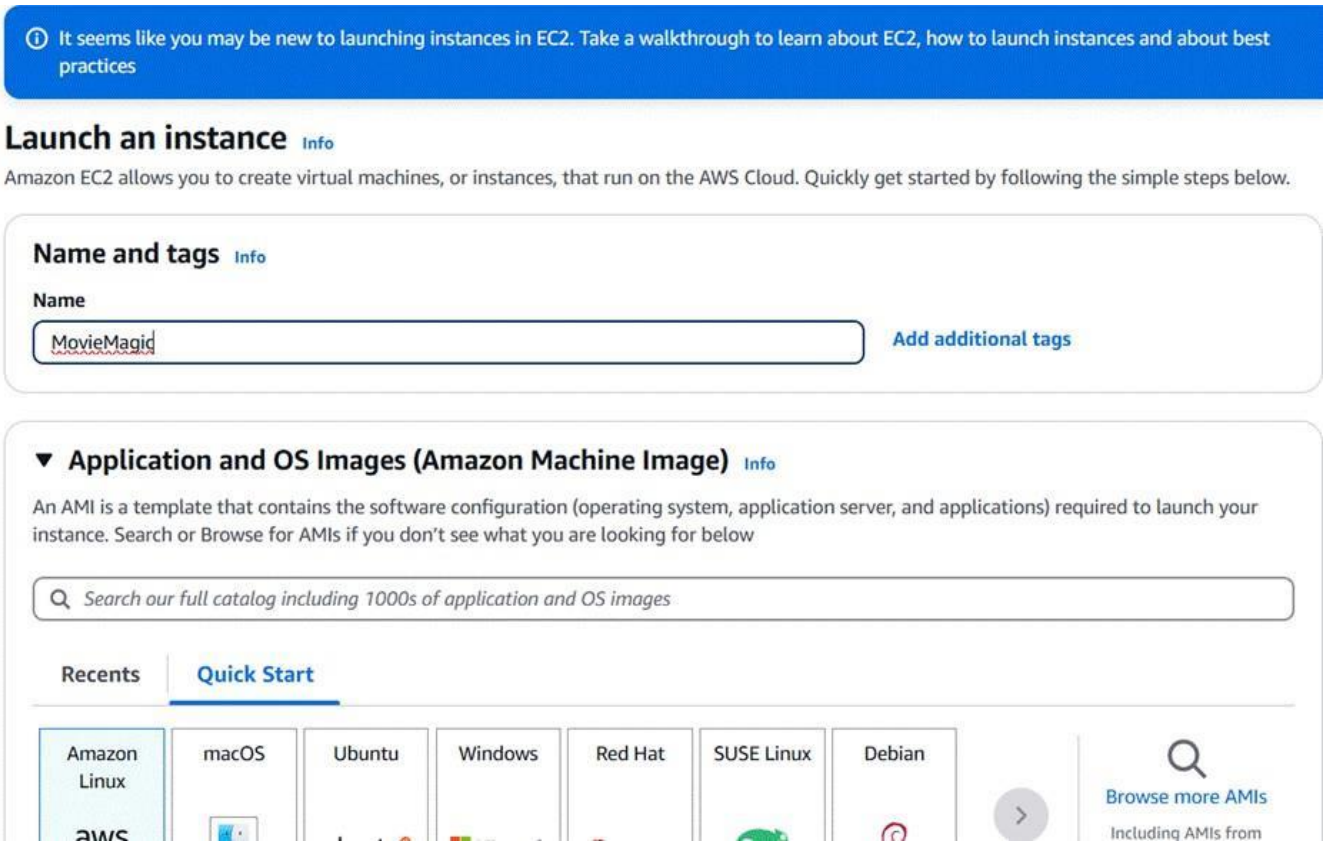
Launch an EC2 instance to host the Flask

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

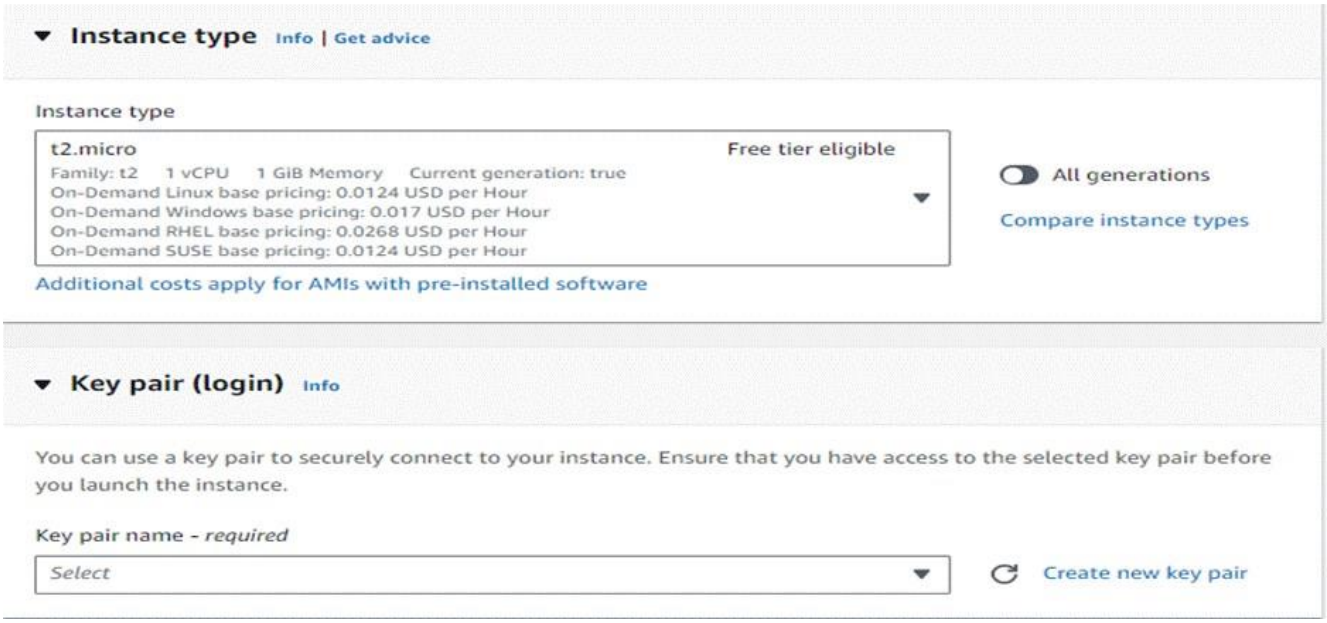


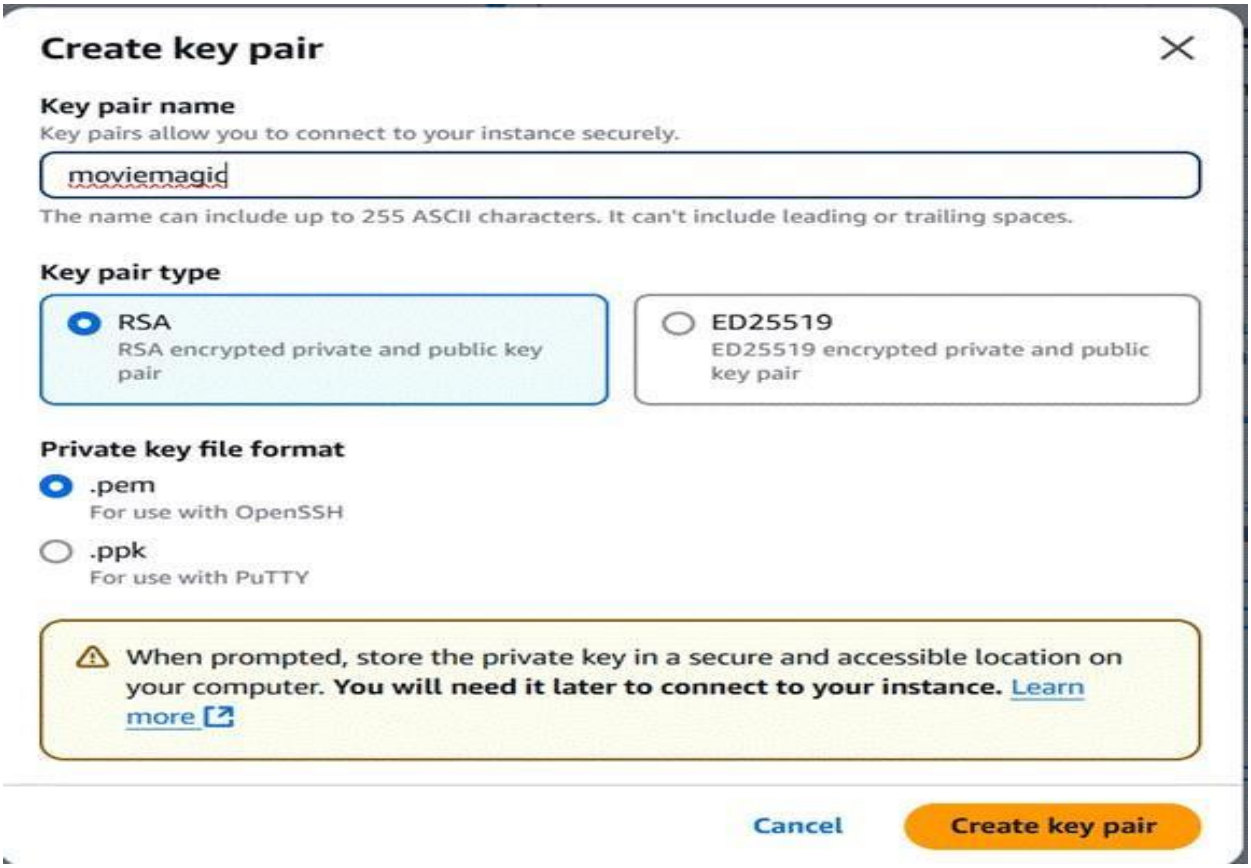
- Click on Launch instance to launch EC2 instance





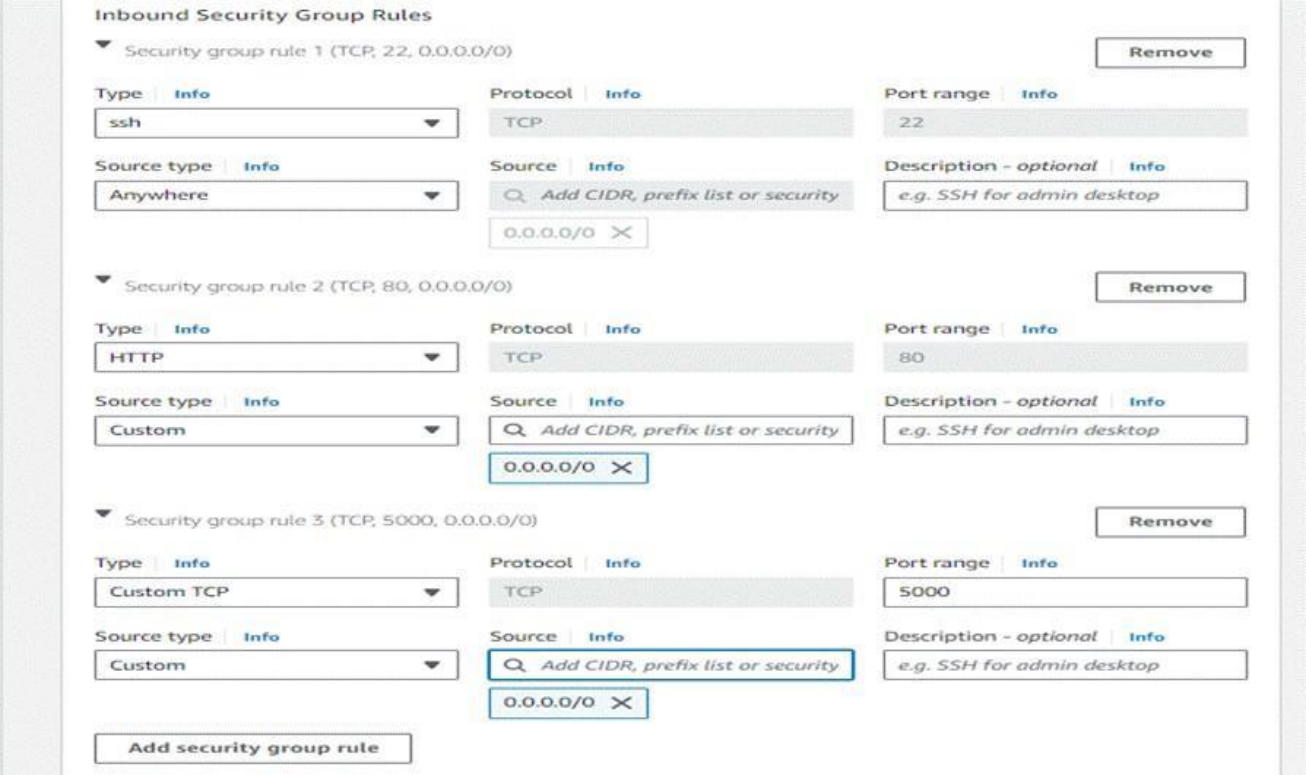
- Create and download the key pair for Server access.

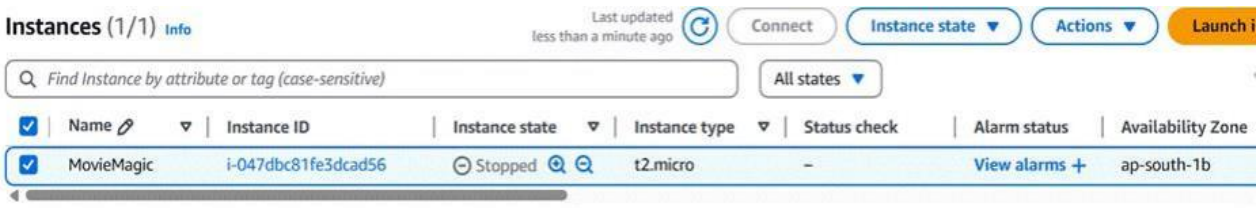

Instance type Info | Get advice
Instance type
t2.micro
Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0124 USD per Hour
On-Demand Windows base pricing: 0.017 USD per Hour
On-Demand RHEL base pricing: 0.0268 USD per Hour
On-Demand SUSE base pricing: 0.0124 USD per Hour
All generations
Compare instance types
Additional costs apply for AMIs with pre-installed software
Key pair (login) Info
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
Key pair name - required
Select
Create new key pair


Create key pair
Key pair name
Key pairs allow you to connect to your instance securely.
moviemagic
The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.
Key pair type
RSA
RSA encrypted private and public key pair
ED25519
ED25519 encrypted private and public key pair
Private key file format
.pem
For use with OpenSSH
.ppk
For use with PuTTY
When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. Learn more
Cancel
Create key pair

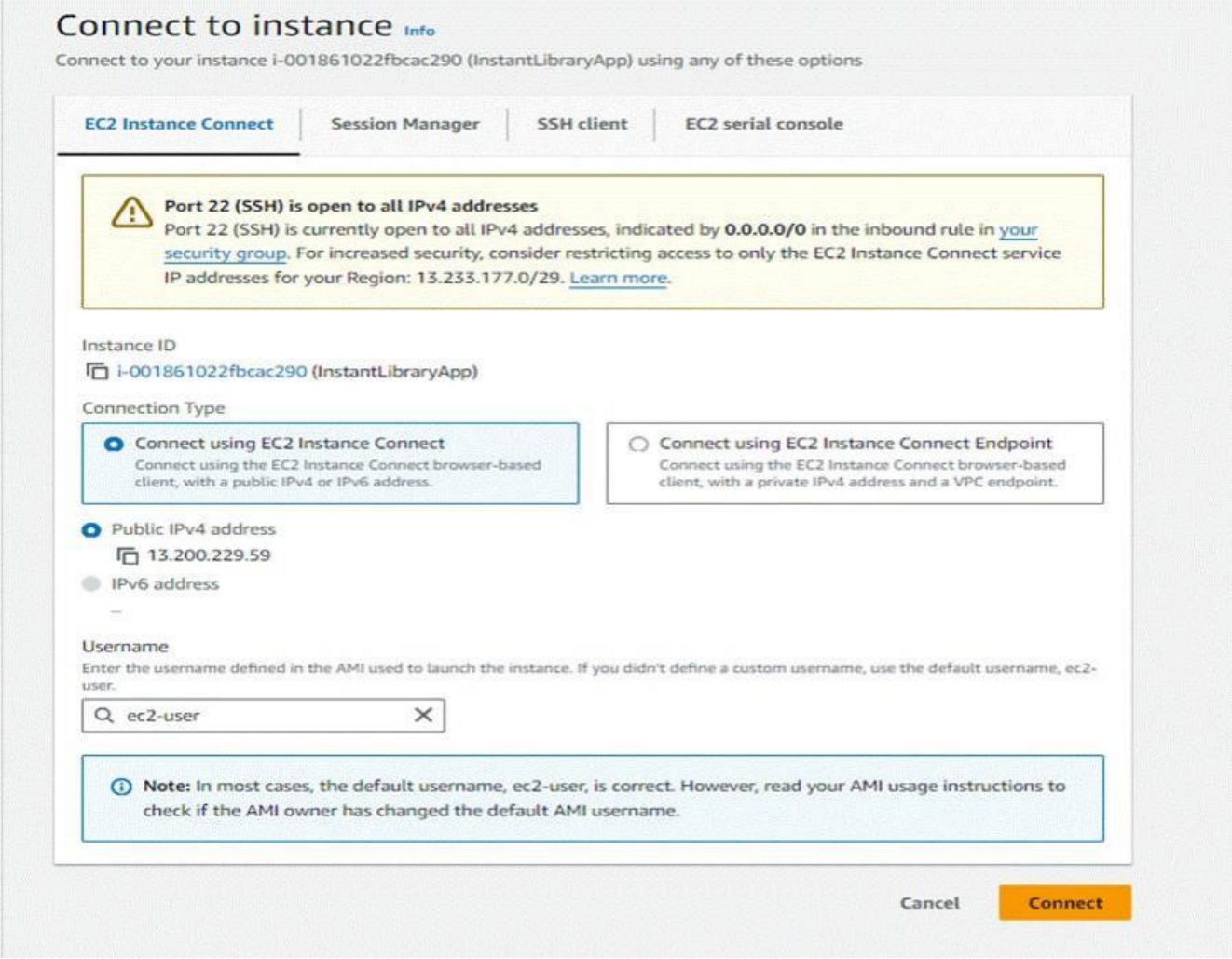
Edit Inbound Rules:

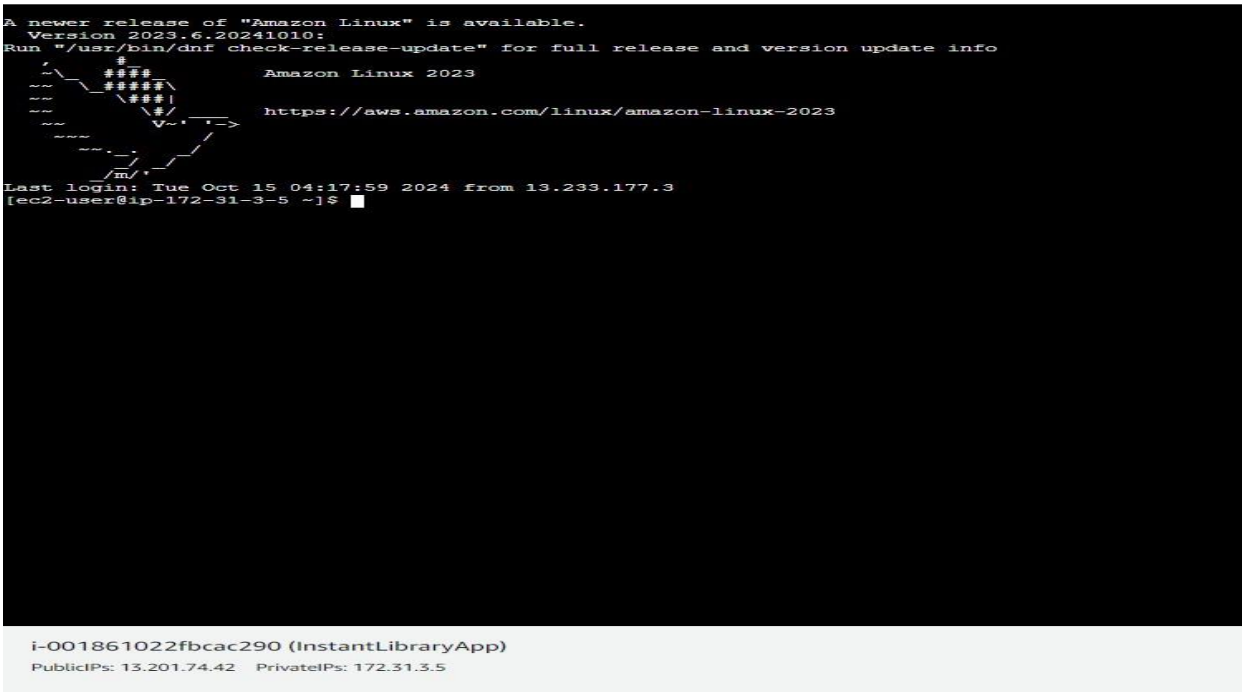
Select the EC2 instance you just launched and ensure it's in the "running" state. Navigate to the "Security" tab, then click "Edit inbound rules." Add a new rule with the following settings: Type – Custom TCP, Protocol – TCP, Port Range – 5000, Source – Anywhere (IPv4) 0.0.0.0/0. This allows external access to your Flask application running on port 5000.





- Now connect the EC2 with the files





Deployment Using EC2 :

Deployment on an EC2 instance involves launching a server, configuring security groups for public access, and uploading your application files. After setting up necessary dependencies and environment variables, start your application and ensure it's running on the correct port. Finally, bind your domain or use the public IP to make the application accessible online.

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

Clone Your Flask Project from GitHub

git clone: https://github.com/dhanasree19/Movie-Magic-Smart-Movie-Ticket-Booking-System.git

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

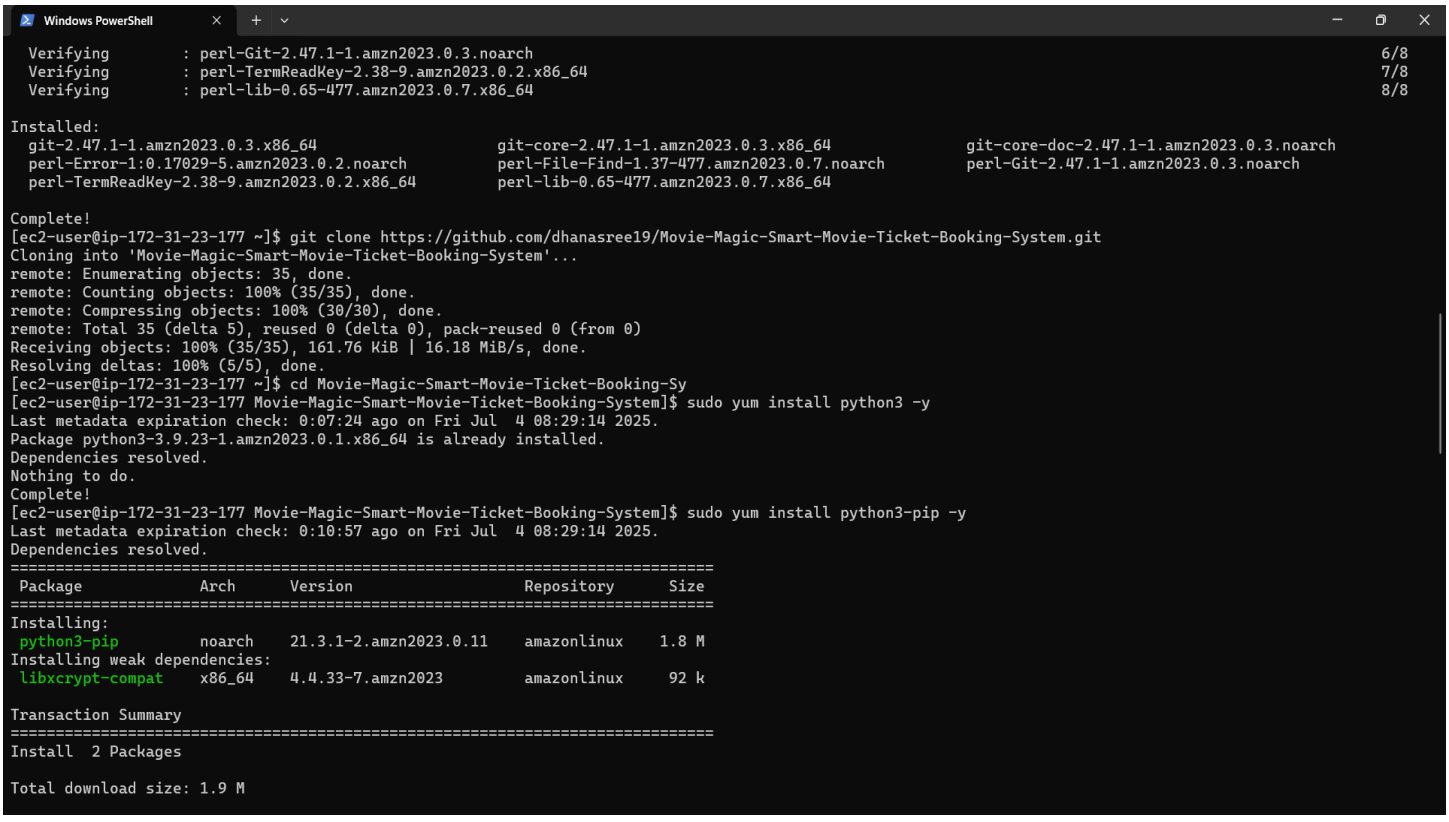
This will download your project to the EC2 instance.

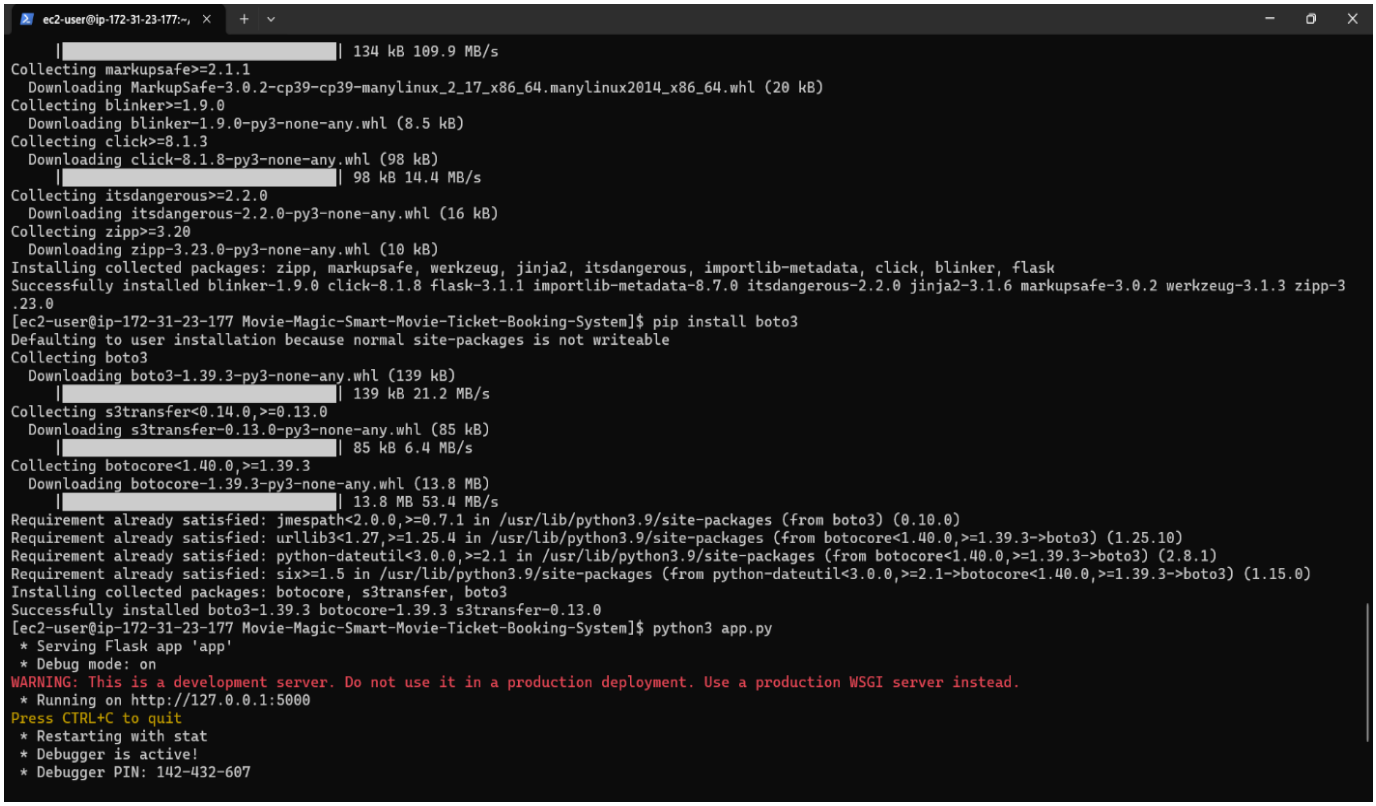
- To navigate to the project directory, run the following command: cd MovieMagic
- Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application: sudo flask run --host=0.0.0.0 --port=5000

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

- Run the Flask app on the EC2 instance





- Access the website through: **your-ec2-public-ip**

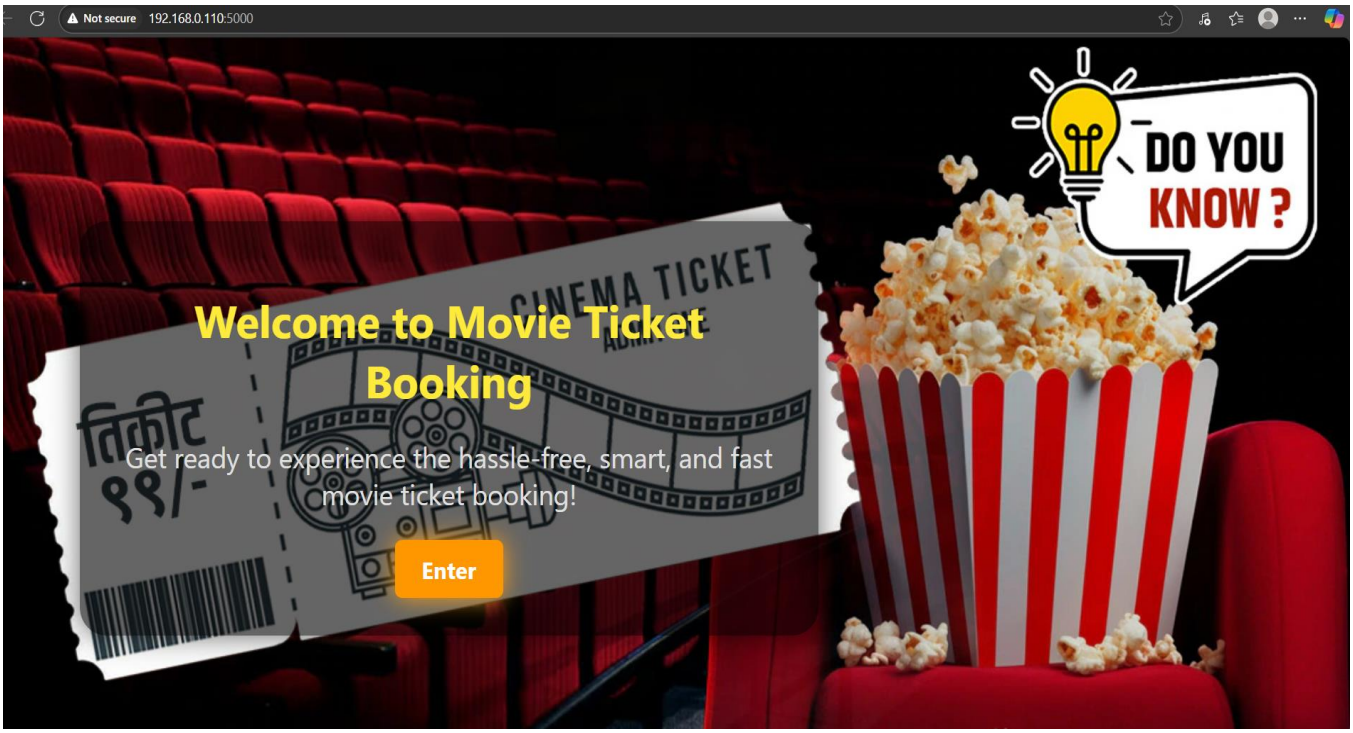
Public IP's: http://54.197.161.11:5000

Testing and Deployment

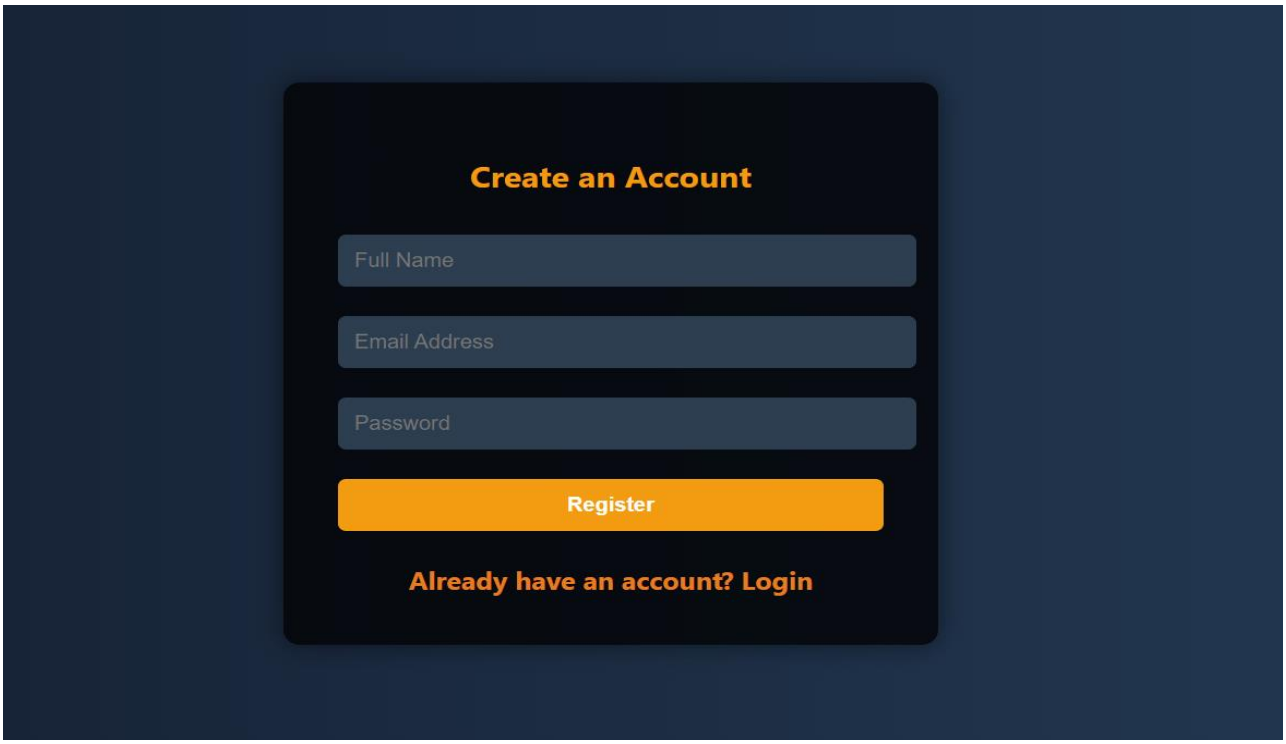
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment

Functional testing to verify the Project

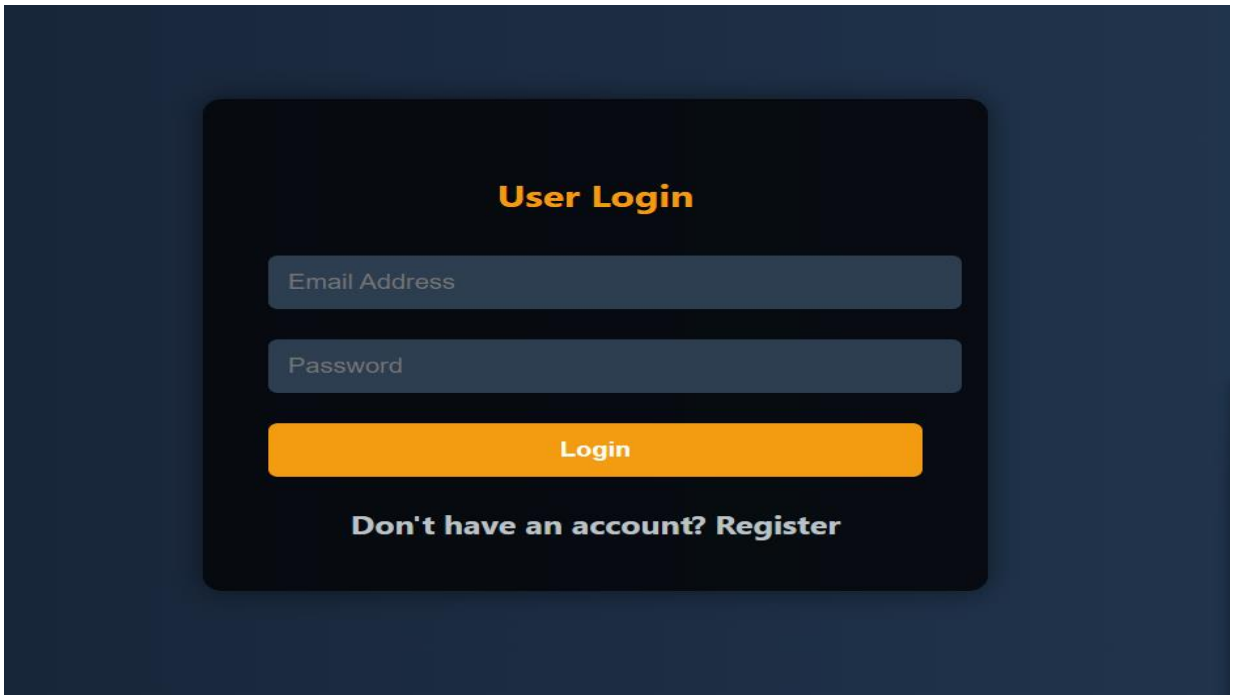
Index Page:



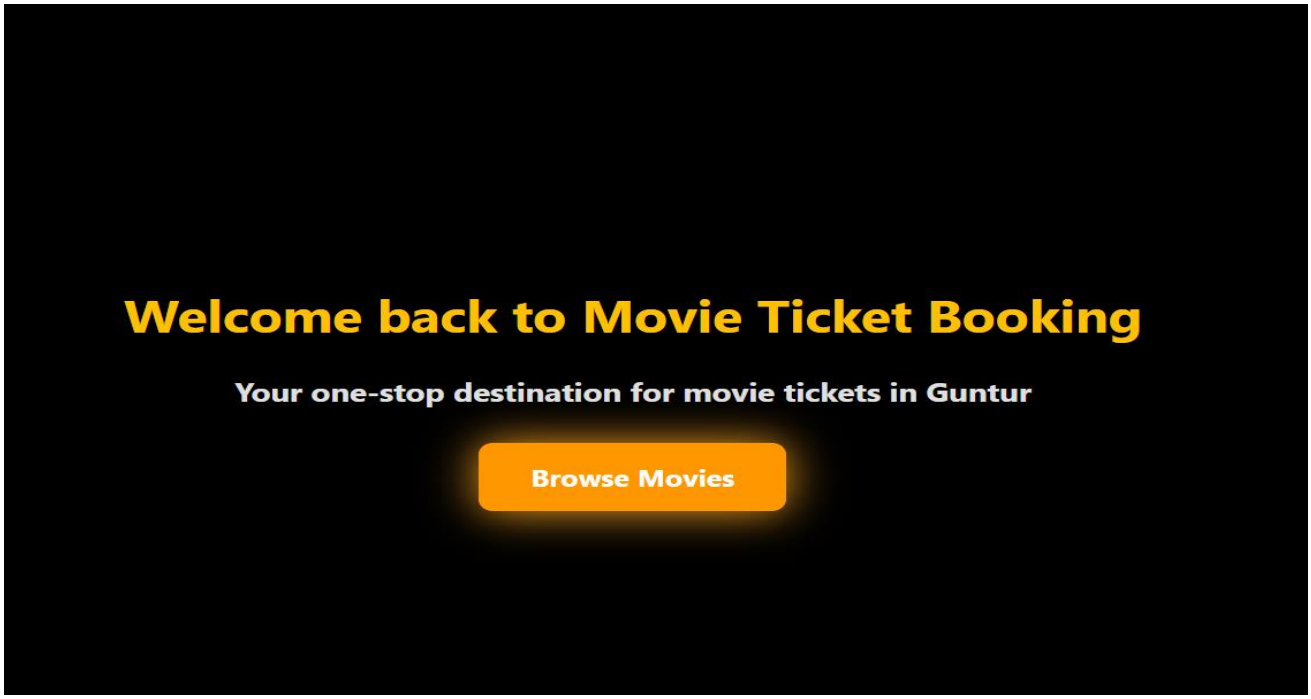
Register Page:

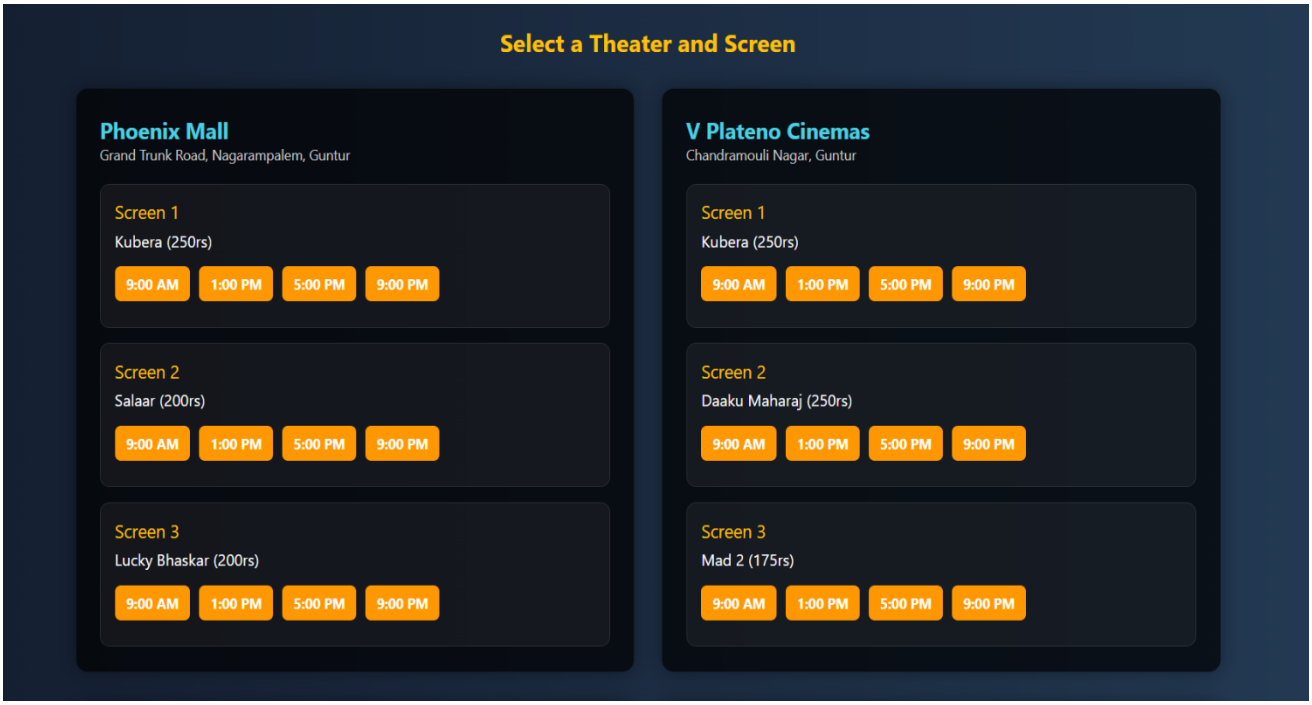


Login Page:



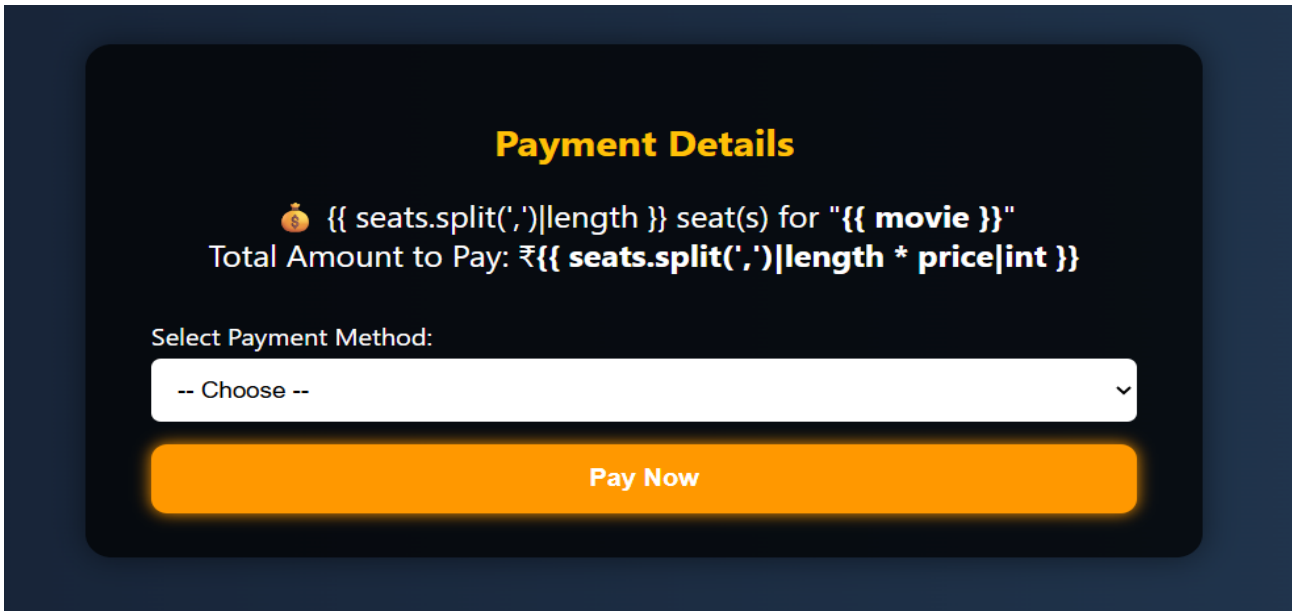
Home page:

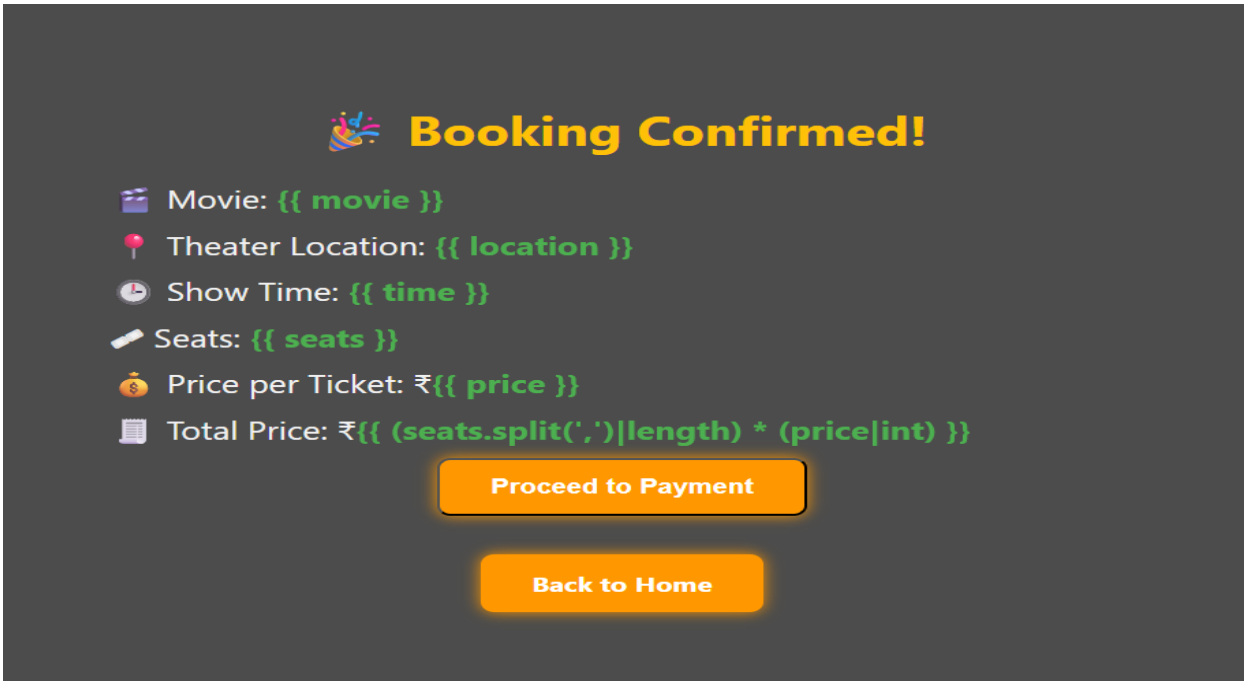


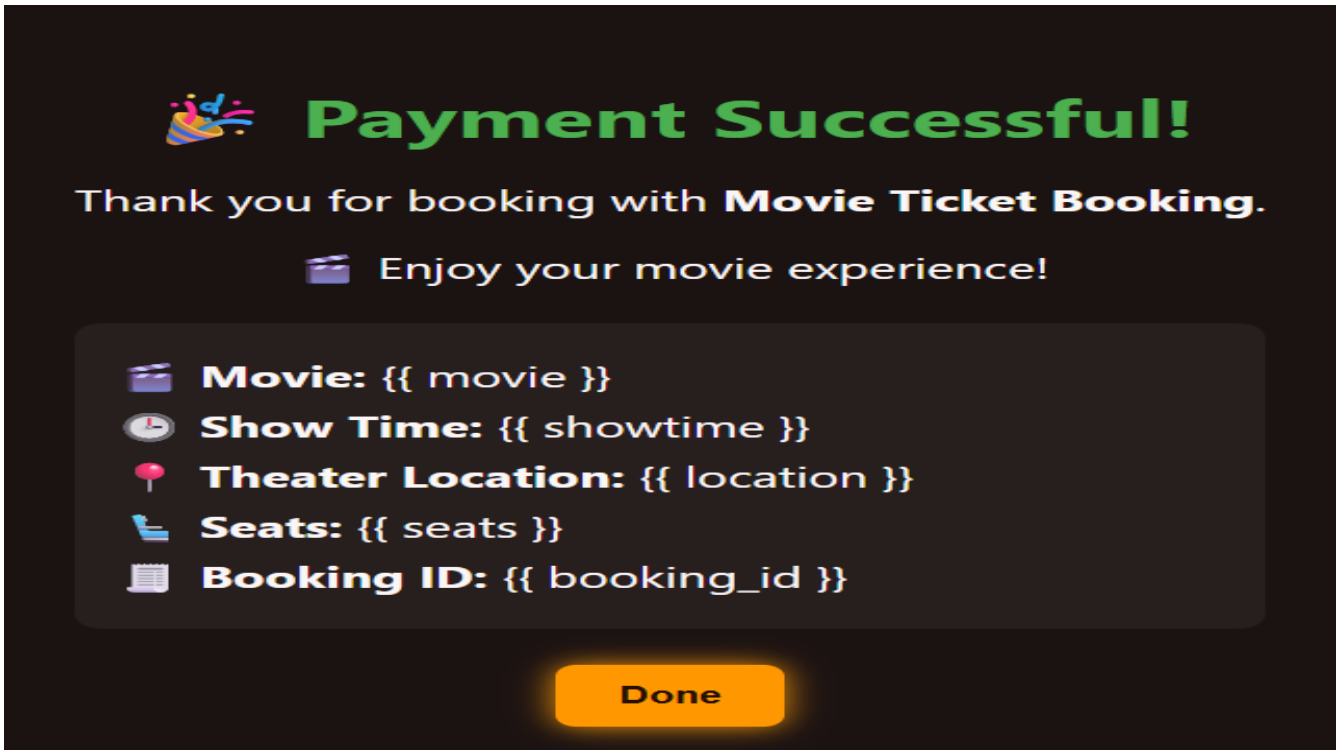

Select a Theater and Screen
Phoenix Mall
Grand Trunk Road, Nagarampalem, Guntur
Screen 1
Kubera (250rs)
9:00 AM
1:00 PM
5:00 PM
9:00 PM
Screen 2
Salaar (200rs)
9:00 AM
1:00 PM
5:00 PM
9:00 PM
Screen 3
Lucky Bhaskar (200rs)
9:00 AM
1:00 PM
5:00 PM
9:00 PM
V Plateno Cinemas
Chandramouli Nagar, Guntur
Screen 1
Kubera (250rs)
9:00 AM
1:00 PM
5:00 PM
9:00 PM
Screen 2
Daaku Maharaj (250rs)
9:00 AM
1:00 PM
5:00 PM
9:00 PM
Screen 3
Mad 2 (175rs)
9:00 AM
1:00 PM
5:00 PM
9:00 PM


Select Your Seats
SCREEN
1 2 3 4 33 34 35 36 37 38 81 82 83 84
5 6 7 8 39 40 41 42 43 44 85 86 87 88
9 10 11 12 45 46 47 48 49 50 89 90 91 92
13 14 15 16 51 52 53 54 55 56 93 94 95 96
17 18 19 20 57 58 59 60 61 62 97 98 99 100
21 22 23 24 63 64 65 66 67 68 101 102 103 104
25 26 27 28 69 70 71 72 73 74 105 106 107 108
29 30 31 32 75 76 77 78 79 80 109 110 111 112
Available
Booked
Selected
Confirm Booking

Payment


Payment Details
💰 {{ seats.split(',')|length }} seat(s) for "{{ movie }}"
Total Amount to Pay: ₹{{ seats.split(',')|length * price|int }}
Select Payment Method:
-- Choose --
Pay Now


🎉 Booking Confirmed!
🎬 Movie: {{ movie }}
📍 Theater Location: {{ location }}
🕑 Show Time: {{ time }}
💺 Seats: {{ seats }}
💰 Price per Ticket: ₹{{ price }}
🧾 Total Price: ₹{{ (seats.split(',')|length) * (price|int) }}
Proceed to Payment
Back to Home



Final Conclusion:

Your Flask-based backend system is **successfully set up** and ready to manage the full movie ticket booking flow. Here's a summary of what you've built and achieved:

What the Project Does

- **User Management**:
  - Users can register and log in.
  - User sessions are handled in memory (basic dictionary for now).
- **Movie Browsing & Booking**:
  - Theaters and movies are listed for users.
  - Movie details are passed through routes to show available seats.
- **Seat Selection & Payment**:
  - Users can select seats, view booking summary, and proceed to a mock payment form.
  - Input validation ensures correct card information (as a demo).
- **Booking Confirmation**:
  - A unique booking ID is generated using uuid.
  - The booking confirmation (with ticket details) is shown.
- **Modular Code Structure**:
  - Cleanly separated routes and templates.
  - HTML files are stored in templates/ and assets like images in static