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.

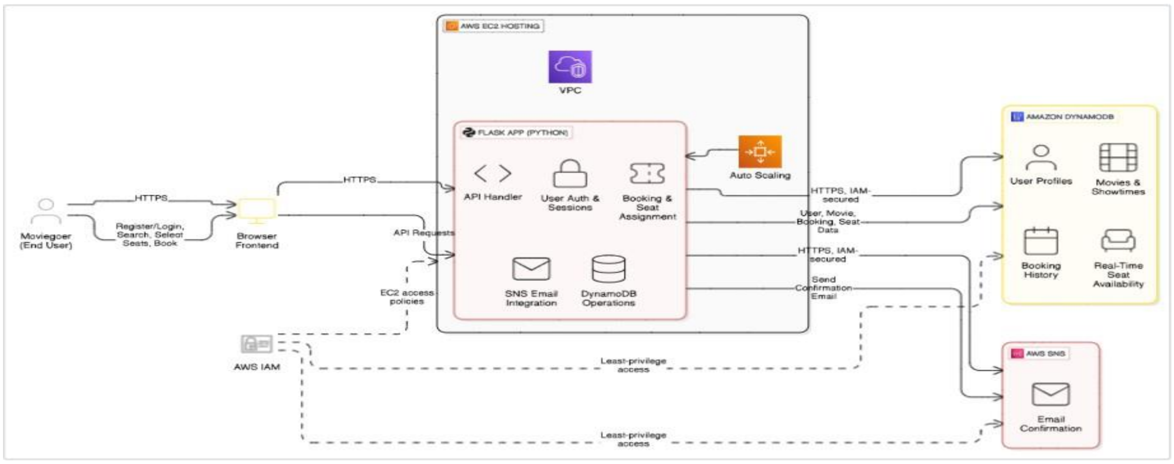
Scenario 3: Admin Views Booking Records (Future Enhancement)

- 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.

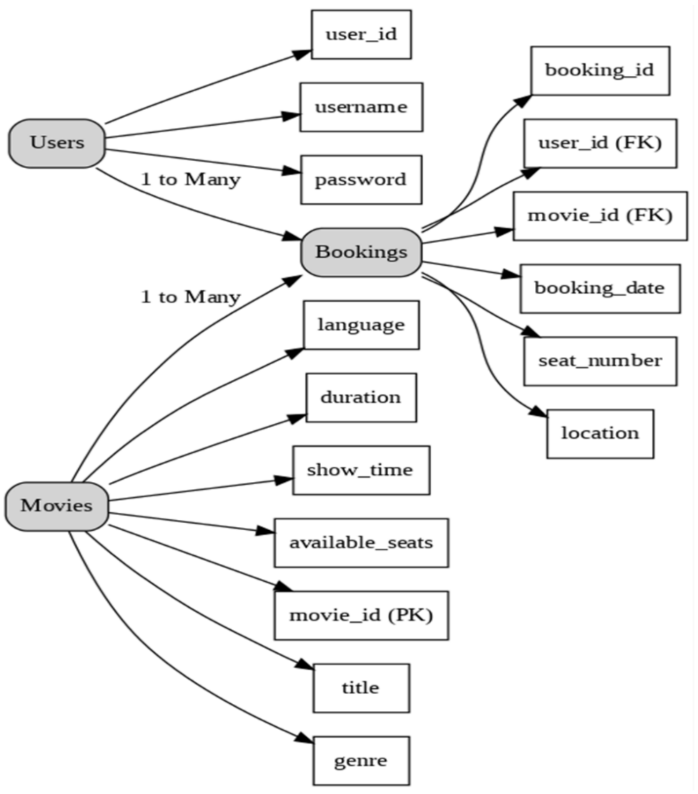
Scenario 4: SMS Booking Confirmation (Optional via SNS)

- 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

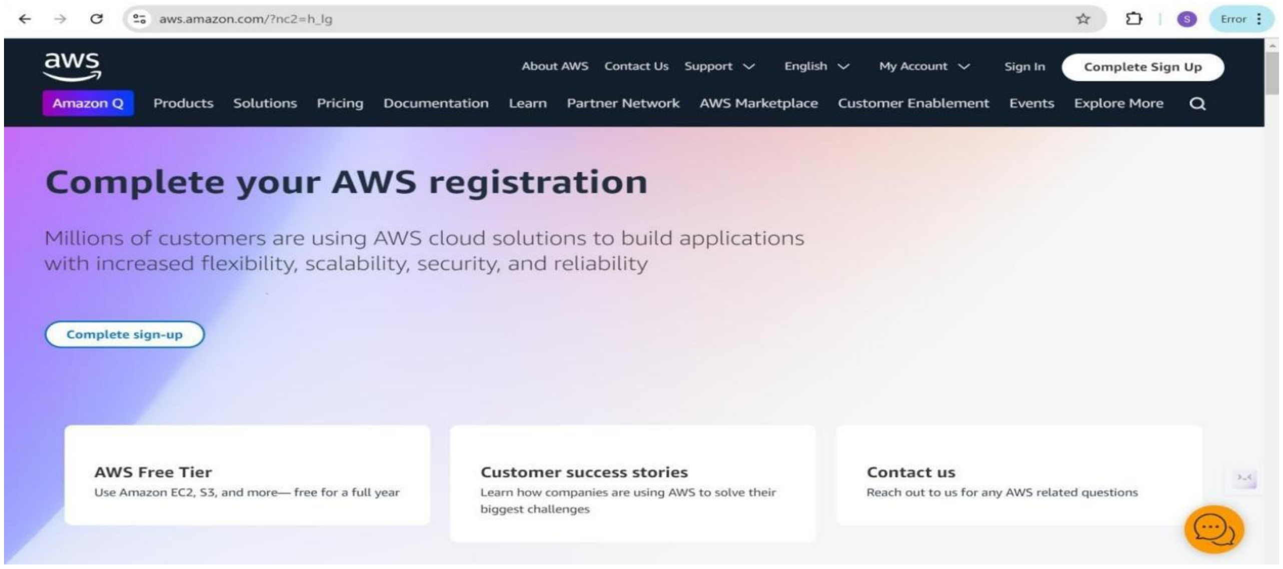
Activity 7.1:Upload Flask Files

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



Complete your AWS registration

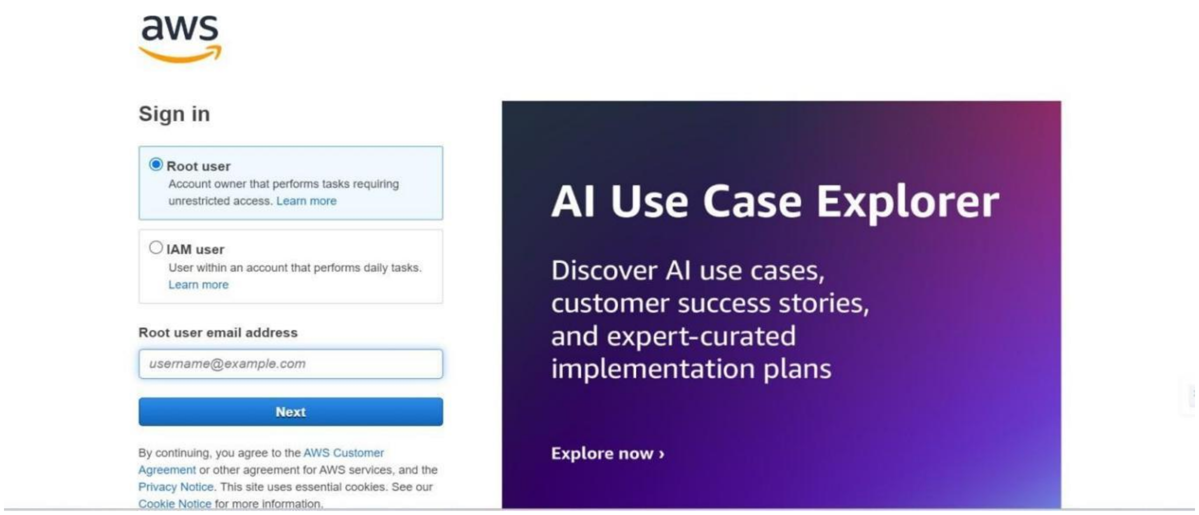
Millions of customers are using AWS cloud solutions to build applications with increased flexibility, scalability, security, and reliability

AWS Free Tier
Use Amazon EC2, S3, and more— free for a full year

Customer success stories
Learn how companies are using AWS to solve their biggest challenges

Contact us
Reach out to us for any AWS related questions

2: Log in to the AWS Management Console.



Sign in

Root user
Account owner that performs tasks requiring unrestricted access. Learn more

IAM user
User within an account that performs daily tasks. Learn more

Root user email address: username@example.com

By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our Cookie Notice for more information.

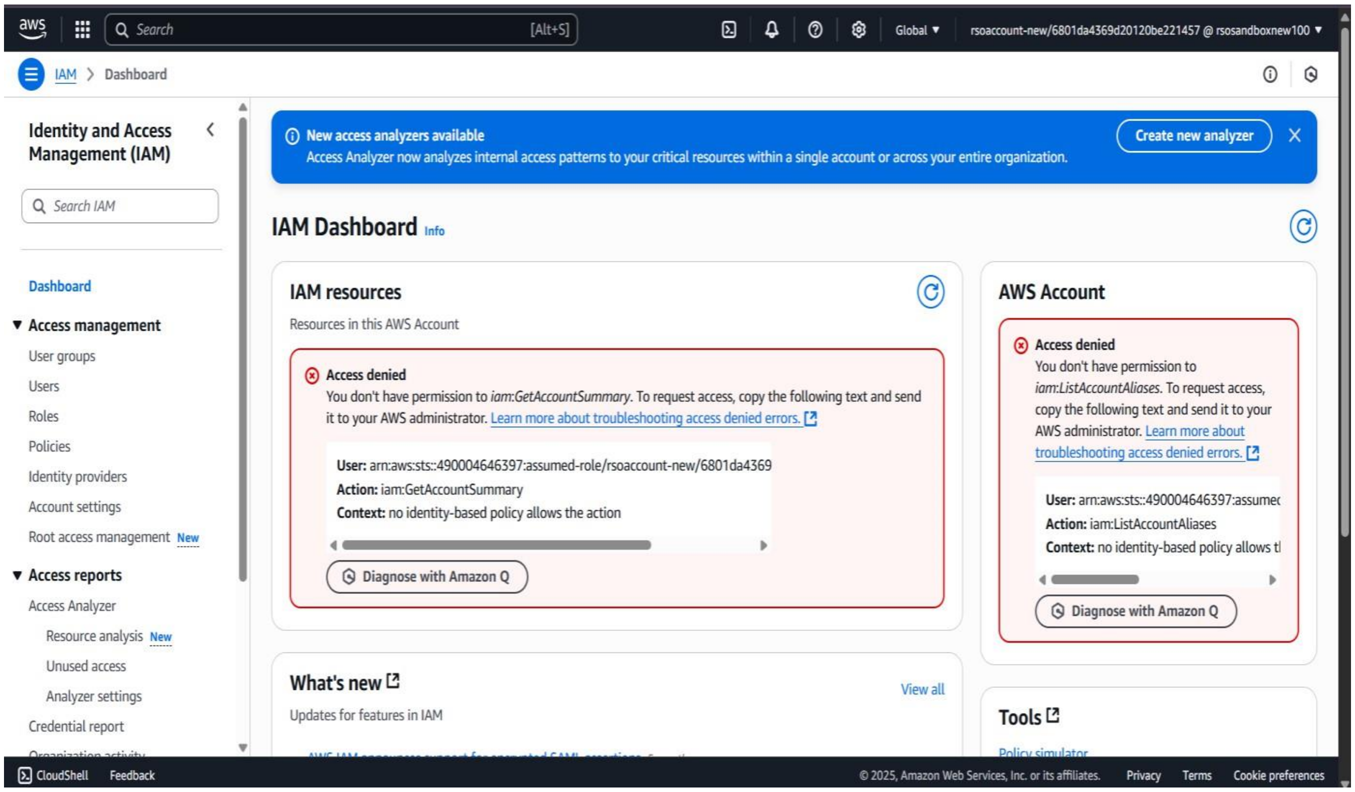
AI Use Case Explorer

Discover AI use cases, customer success stories, and expert-curated implementation plans

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



New access analyzers available
Access Analyzer now analyzes internal access patterns to your critical resources within a single account or across your entire organization.

IAM Dashboard Info

IAM resources
Resources in this AWS Account

Access denied
You don't have permission to *iam:GetAccountSummary*. To request access, copy the following text and send it to your AWS administrator. Learn more about troubleshooting access denied errors.

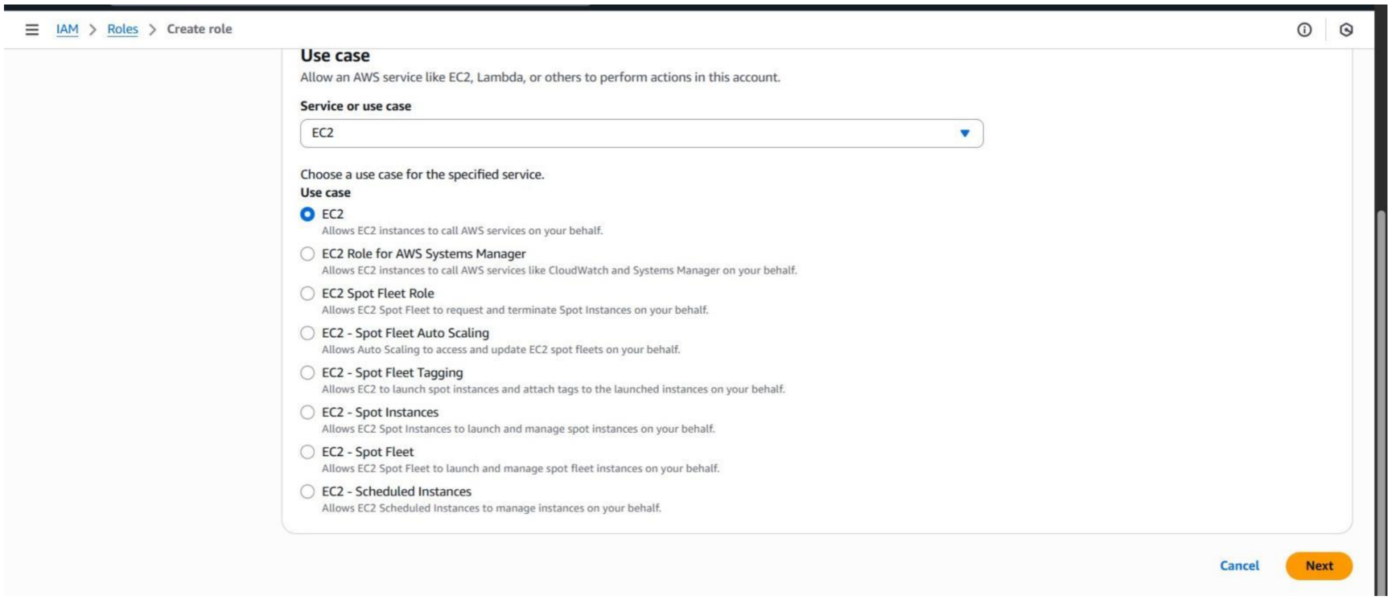
User: arn:aws:sts::490004646397:assumed-role/rsoaccount-new/6801da4369
Action: iam:GetAccountSummary
Context: no identity-based policy allows the action

AWS Account

Access denied
You don't have permission to *iam:ListAccountAliases*. To request access, copy the following text and send it to your AWS administrator. Learn more about troubleshooting access denied errors.

User: arn:aws:sts::490004646397:assumed
Action: iam:ListAccountAliases
Context: no identity-based policy allows t

What's new
Updates for features in IAM



Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
EC2

Choose a use case for the specified service.
Use case

[✓ EC2]
Allows EC2 instances to call AWS services on your behalf.

EC2 Role for AWS Systems Manager
Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.

EC2 Spot Fleet Role
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.

EC2 - Spot Fleet Auto Scaling
Allows Auto Scaling to access and update EC2 spot fleets on your behalf.

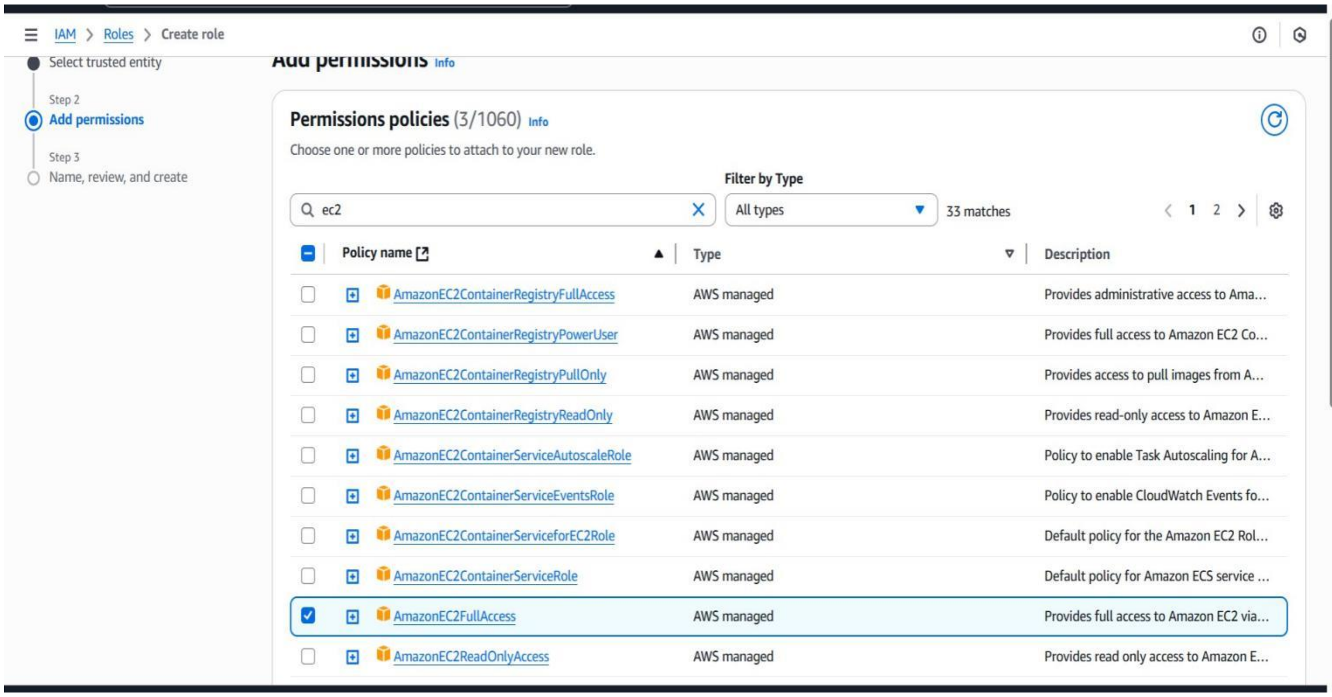
EC2 - Spot Fleet Tagging
Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.

EC2 - Spot Instances
Allows EC2 Spot Instances to launch and manage spot instances on your behalf.

EC2 - Spot Fleet
Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.

EC2 - Scheduled Instances
Allows EC2 Scheduled Instances to manage instances on your behalf.

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



IAM > Roles > Create role

Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions Info

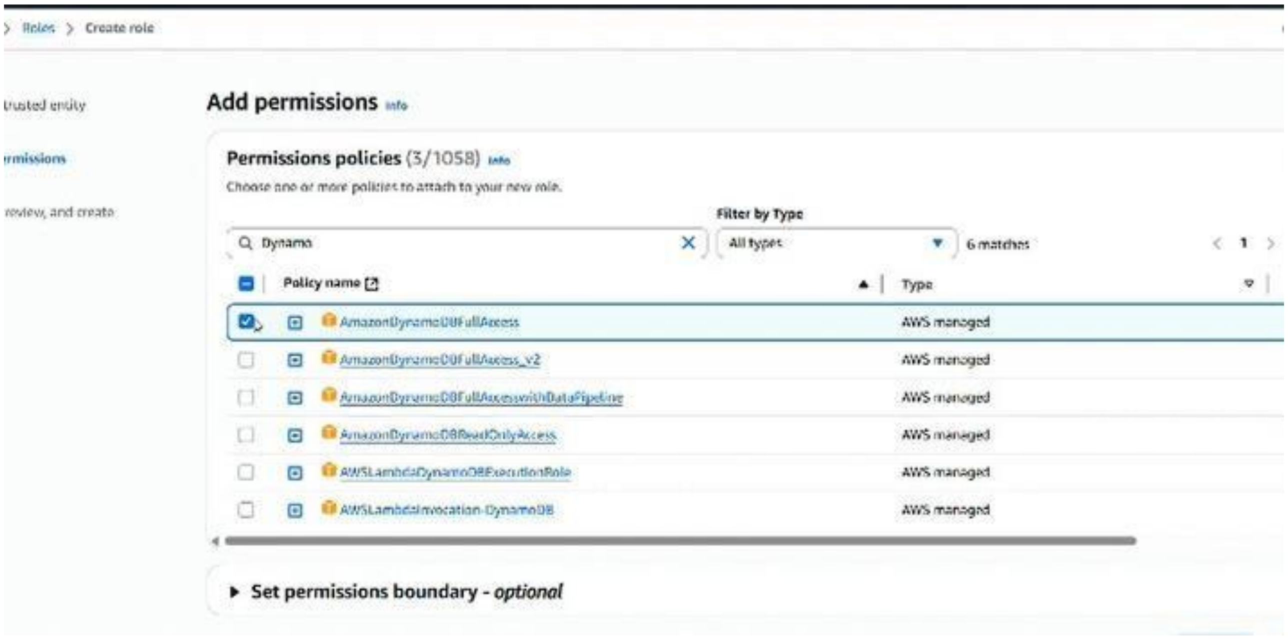
Permissions policies (3/1060) Info

Choose one or more policies to attach to your new role.

Filter by Type

Search: ec2 | All types | 33 matches | < 1 2 >

	Policy name	Type	Description
☐	AmazonEC2ContainerRegistryFullAccess	AWS managed	Provides administrative access to Ama...
☐	AmazonEC2ContainerRegistryPowerUser	AWS managed	Provides full access to Amazon EC2 Co...
☐	AmazonEC2ContainerRegistryPullOnly	AWS managed	Provides access to pull images from A...
☐	AmazonEC2ContainerRegistryReadOnly	AWS managed	Provides read-only access to Amazon E...
☐	AmazonEC2ContainerServiceAutoscaleRole	AWS managed	Policy to enable Task Autoscaling for A...
☐	AmazonEC2ContainerServiceEventsRole	AWS managed	Policy to enable CloudWatch Events fo...
☐	AmazonEC2ContainerServiceforEC2Role	AWS managed	Default policy for the Amazon EC2 Rol...
☐	AmazonEC2ContainerServiceRole	AWS managed	Default policy for Amazon ECS service ...
✓	AmazonEC2FullAccess	AWS managed	Provides full access to Amazon EC2 via...
☐	AmazonEC2ReadOnlyAccess	AWS managed	Provides read only access to Amazon E...



Roles > Create role

trusted entity

rmissions

review, and create

Add permissions Info

Permissions policies (3/1058) Info

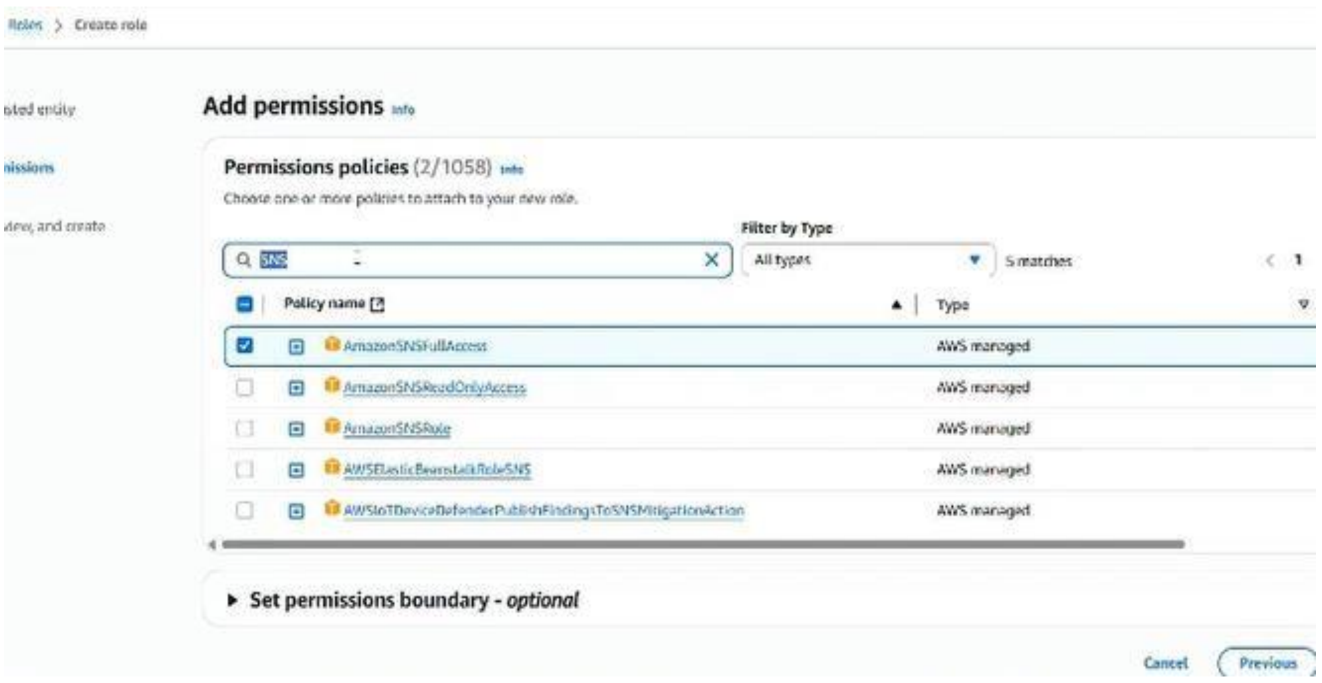
Choose one or more policies to attach to your new role.

Filter by Type

Search: Dynamo | All types | 6 matches | < 1 >

	Policy name	Type
✓	AmazonDynamoDBFullAccess	AWS managed
☐	AmazonDynamoDBFullAccess_v2	AWS managed
☐	AmazonDynamoDBFullAccesswithDataPipeline	AWS managed
☐	AmazonDynamoDBReadOnlyAccess	AWS managed
☐	AWSLambdaDynamoDBExecutionRole	AWS managed
☐	AWSLambdaInvocation-DynamoDB	AWS managed

Set permissions boundary - *optional*



Roles > Create role

Add permissions Info

Permissions policies (2/1058) Info

Choose one or more policies to attach to your new role.

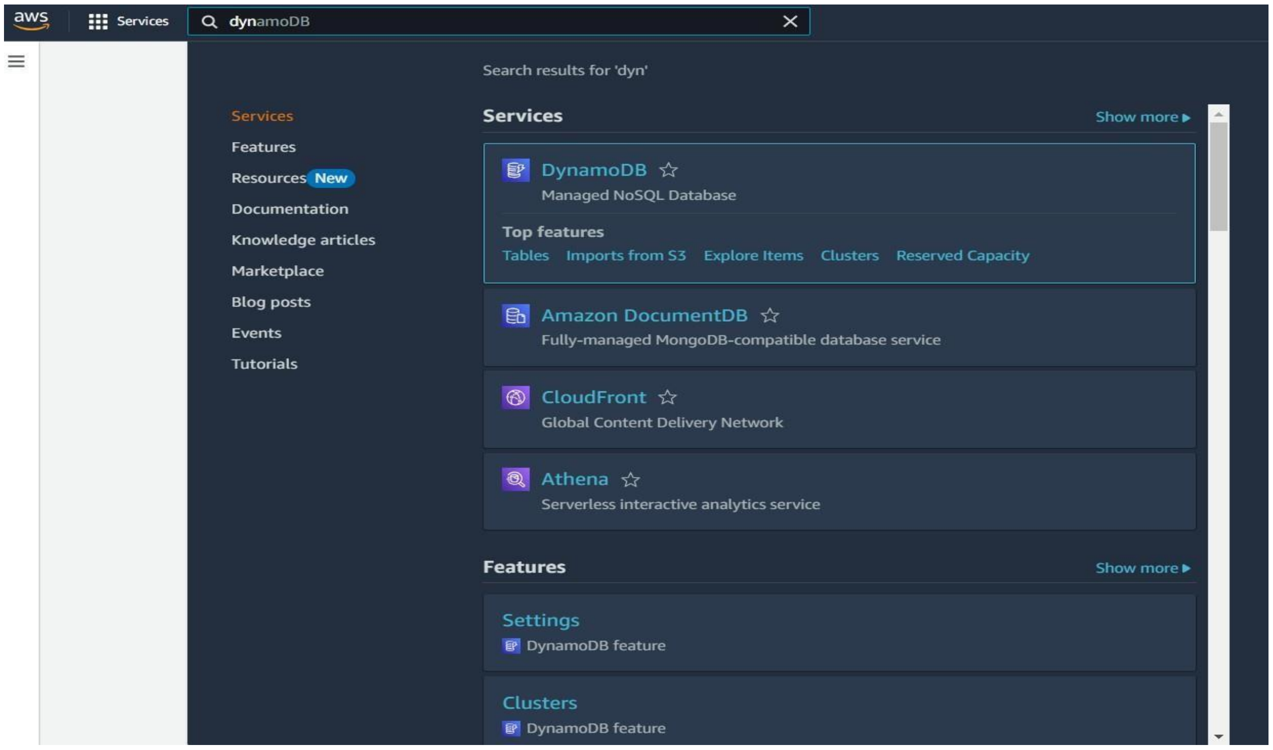
Filter by Type: All types — 5 matches

	Policy name	Type
✓	AmazonSNSFullAccess	AWS managed
	AmazonSNSReadOnlyAccess	AWS managed
	AmazonSNSRole	AWS managed
	AWSElasticBeanstalkRoleSNS	AWS managed
	AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction	AWS managed

▸ Set permissions boundary - *optional*

DynamoDB Database Creation and Setup

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



Search results for 'dyn'

Services

DynamoDB — Managed NoSQL Database
Top features: Tables, Imports from S3, Explore Items, Clusters, Reserved Capacity

Amazon DocumentDB — Fully-managed MongoDB-compatible database service

CloudFront — Global Content Delivery Network

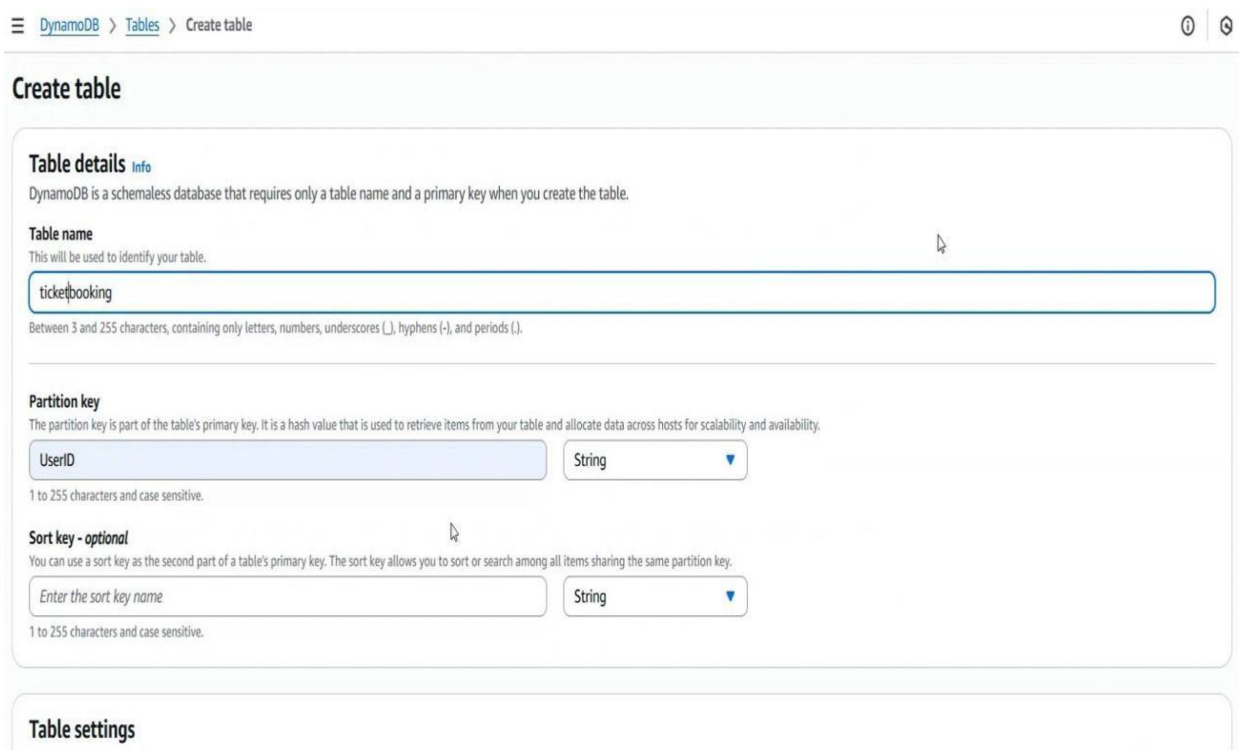
Athena — Serverless interactive analytics service

Features

Settings — DynamoDB feature

Clusters — DynamoDB feature

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.

ticketbooking

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.

UserID | 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.

Enter the sort key name | String

1 to 255 characters and case sensitive.

Table settings

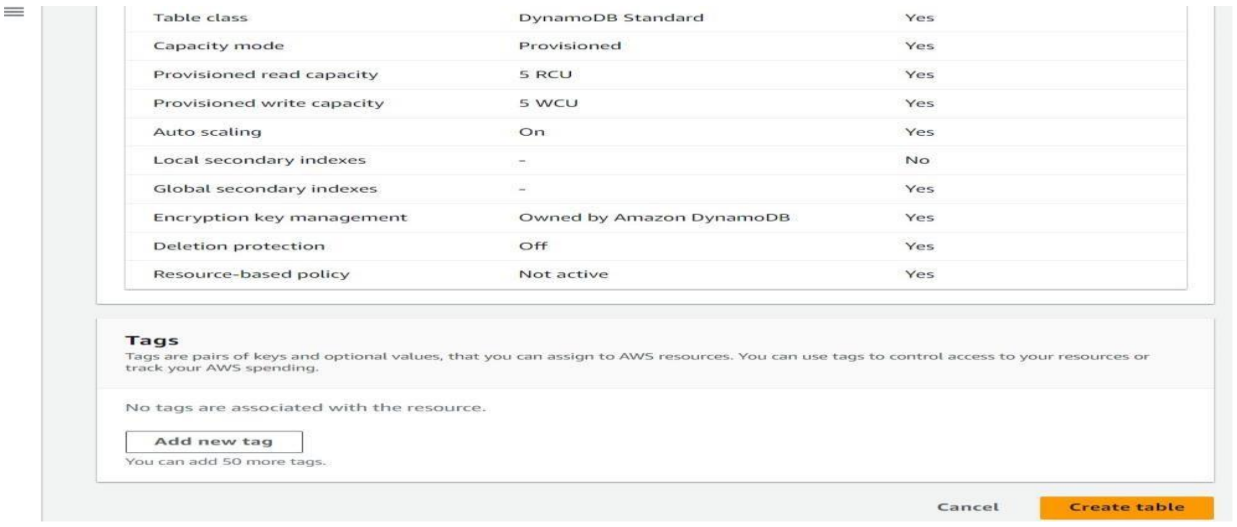


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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

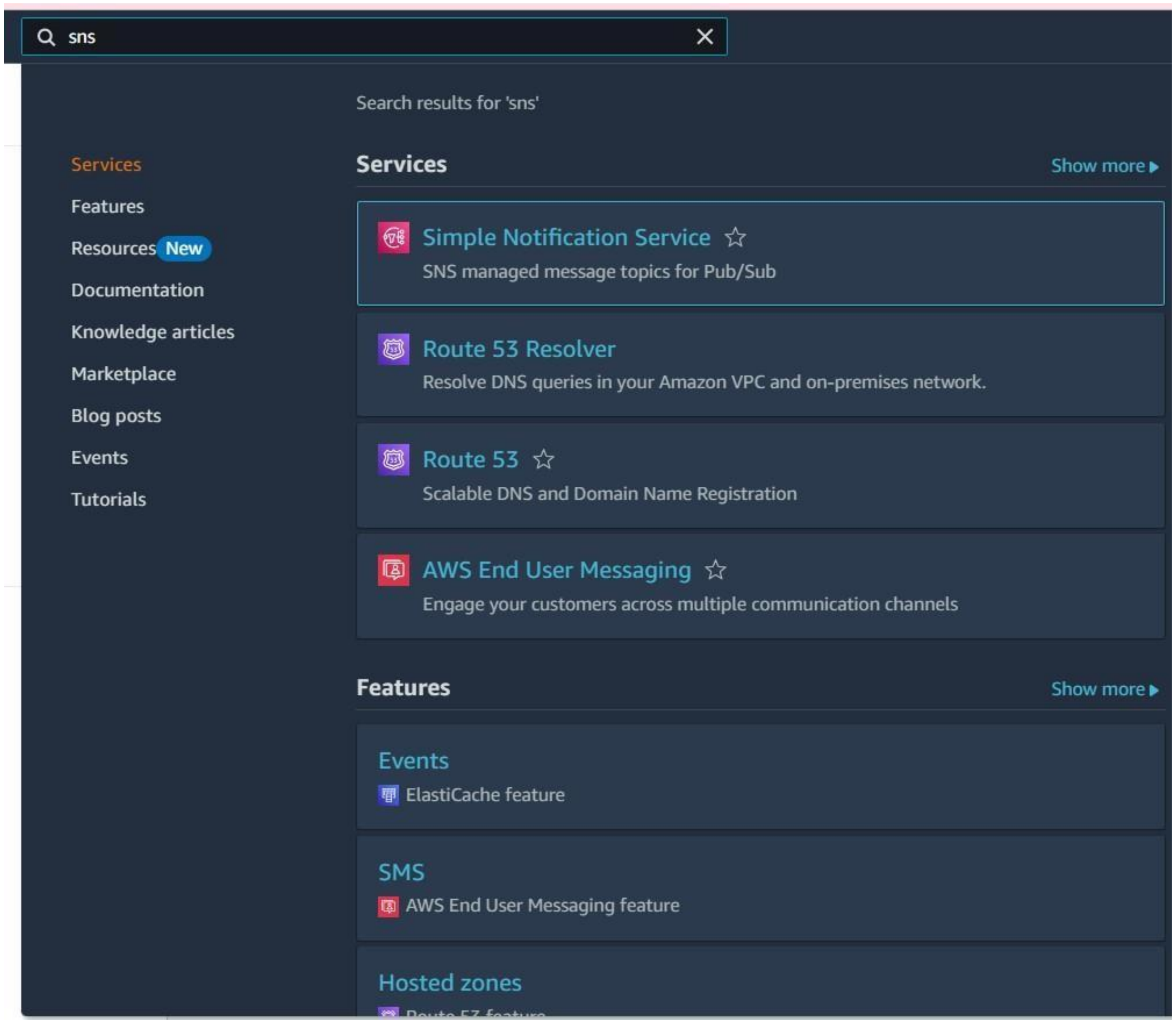
Add new tag

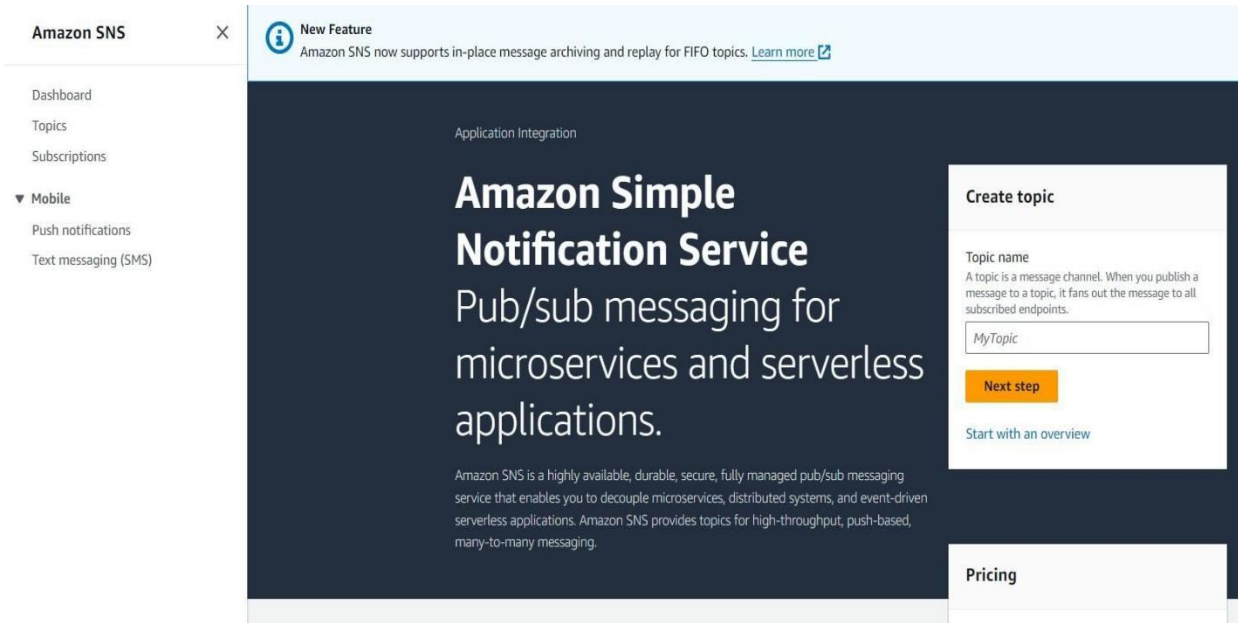
You can add 50 more tags.

Cancel Create table

SNS Notification Setup

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


sns
Search results for 'sns'
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Services
Show more
Simple Notification Service
SNS managed message topics for Pub/Sub
Route 53 Resolver
Resolve DNS queries in your Amazon VPC and on-premises network.
Route 53
Scalable DNS and Domain Name Registration
AWS End User Messaging
Engage your customers across multiple communication channels
Features
Show more
Events
ElastiCache feature
SMS
AWS End User Messaging feature
Hosted zones



New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more

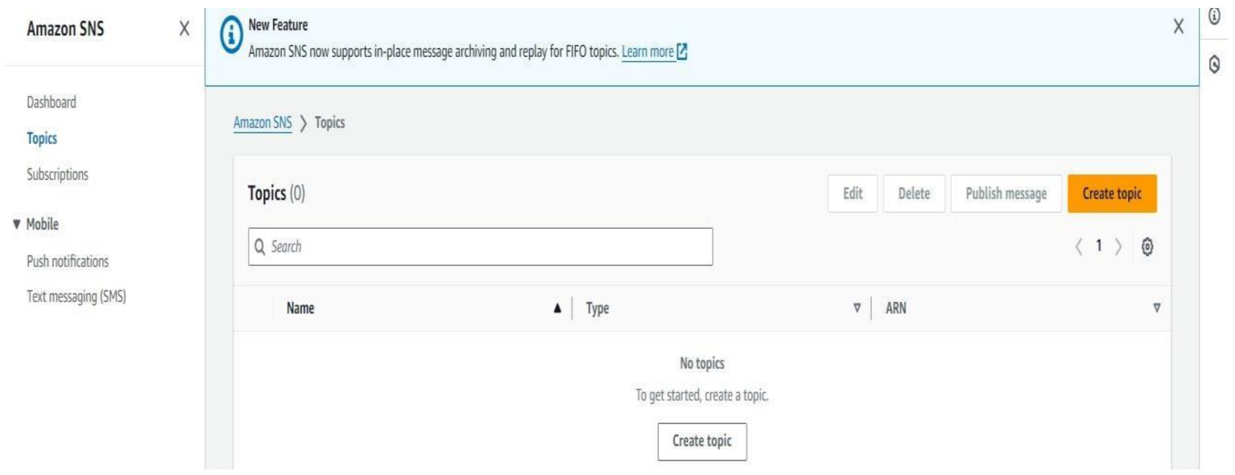
Application Integration

Amazon Simple Notification Service
Pub/sub messaging for microservices and serverless applications.

Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.

Topic name
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.
MyTopic

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



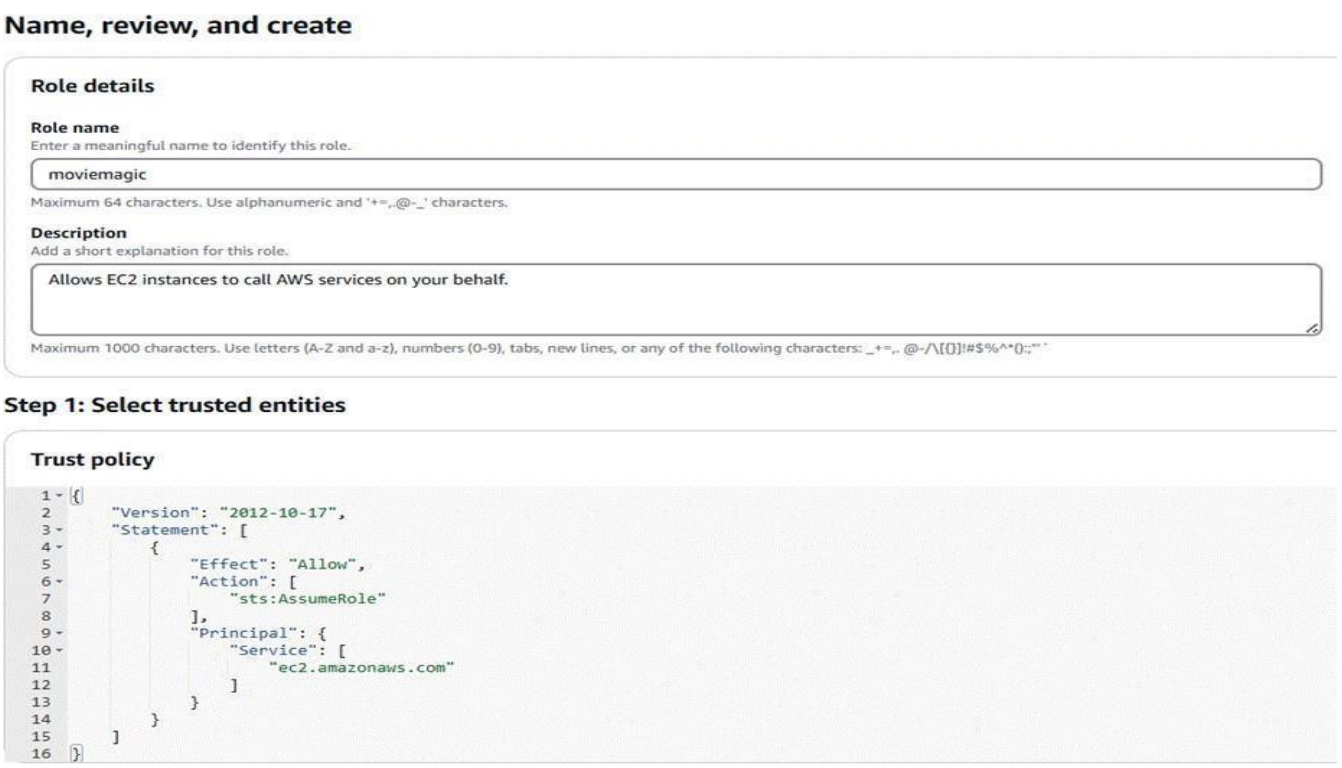
New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more

Topics (0)

Name	Type	ARN

No topics
To get started, create a topic.

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



Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
moviemagic
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*();"'`

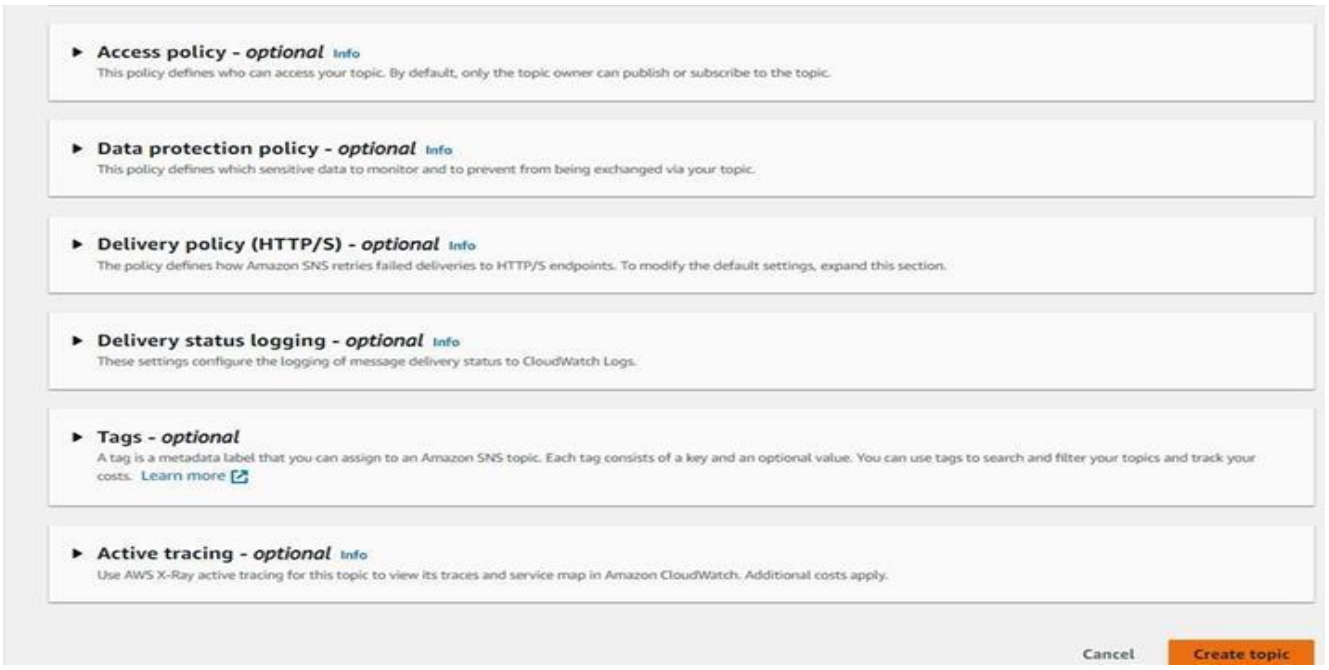
Step 1: Select trusted entities

Trust policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

➢ 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



▸ Access policy - *optional* Info
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

▸ Data protection policy - *optional* Info
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

▸ Delivery policy (HTTP/S) - *optional* Info
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

▸ Delivery status logging - *optional* Info
These settings configure the logging of message delivery status to CloudWatch Logs.

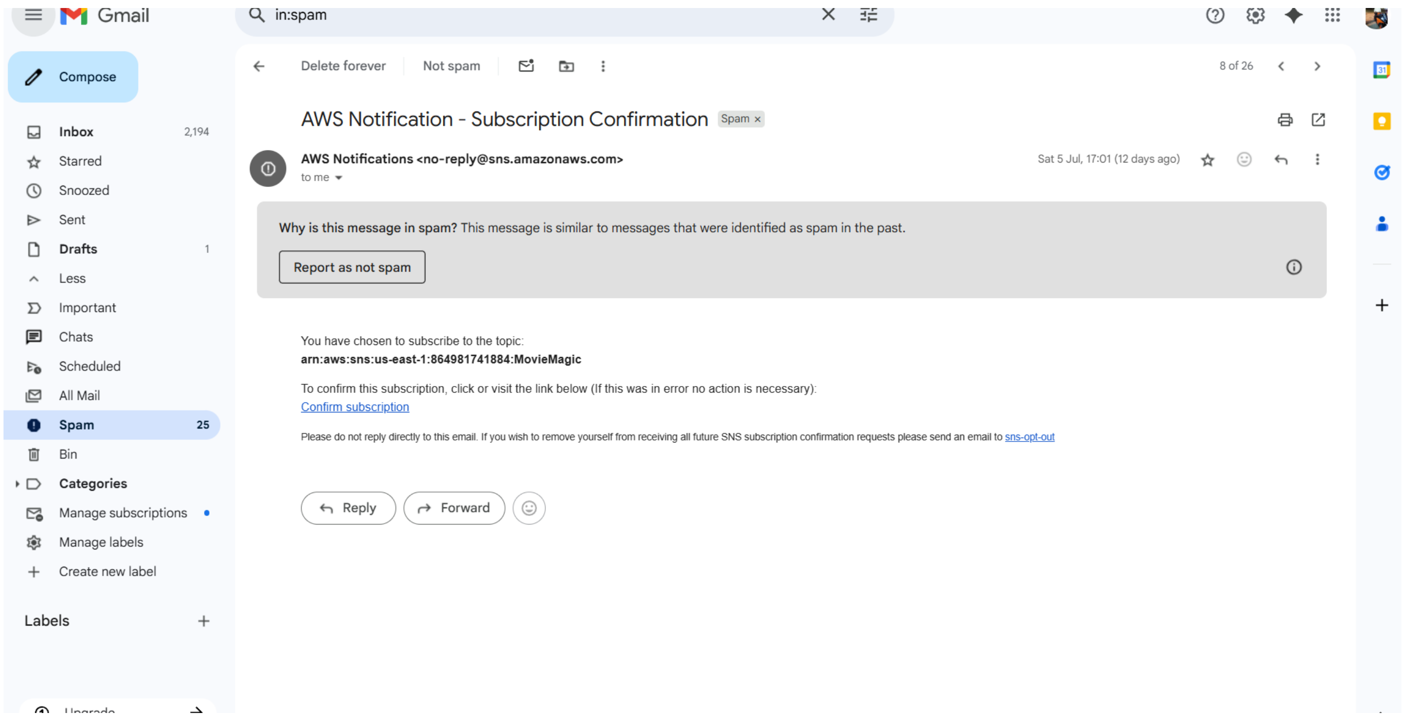
▸ Tags - *optional*
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. Learn more

▸ Active tracing - *optional* Info
Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

Cancel Create topic

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.



AWS Notification - Subscription Confirmation

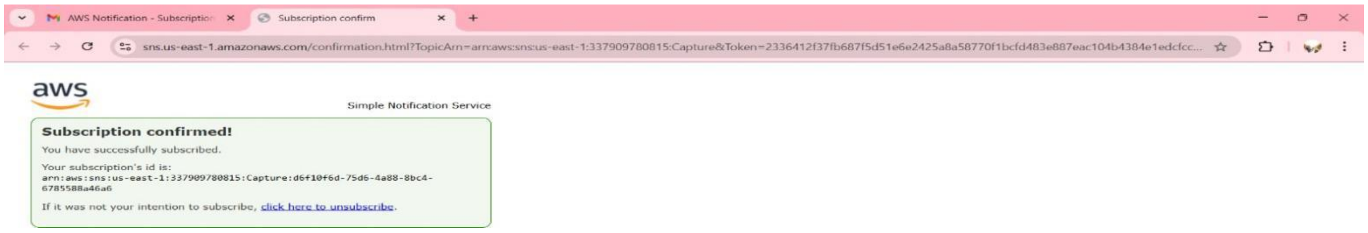
AWS Notifications <no-reply@sns.amazonaws.com>
to me
Sat 5 Jul, 17:01 (12 days ago)

Why is this message in spam? This message is similar to messages that were identified as spam in the past.

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:864981741884:MovieMagic

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to sns-opt-out



sns.us-east-1.amazonaws.com/confirmation.html?TopicArn=arn:aws:sns:us-east-1:337909780815:Capture&Token=2336412f37fb687f5d51e6e2425a8a58770f1bcfd483e887eac104b4384e1edcfcc...

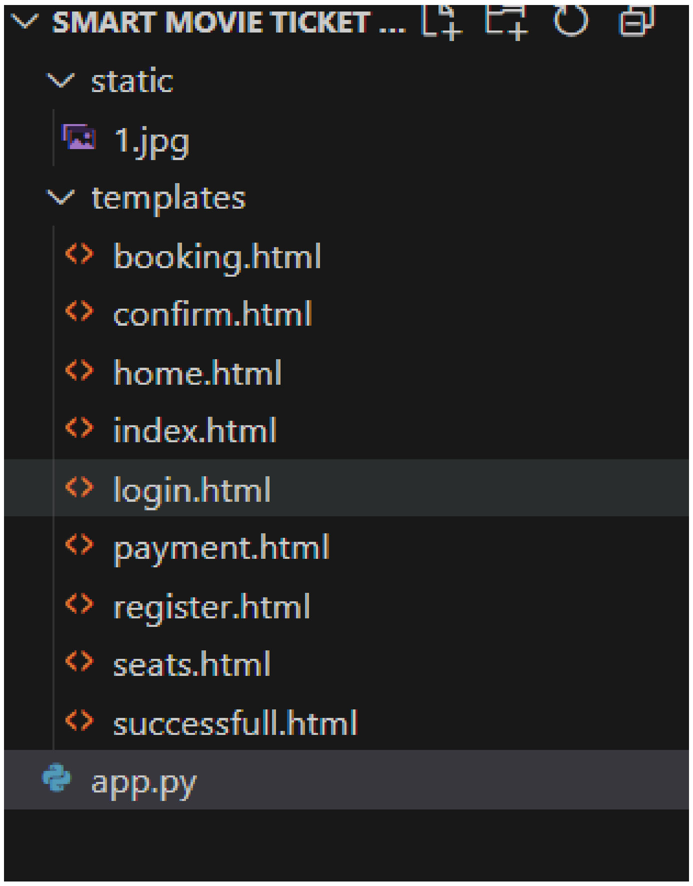
Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:
arn:aws:sns:us-east-1:337909780815:Capture:d6f10f6d-75d6-4a88-8bc4-6785588a46a6

If it was not your intention to subscribe, click here to unsubscribe.

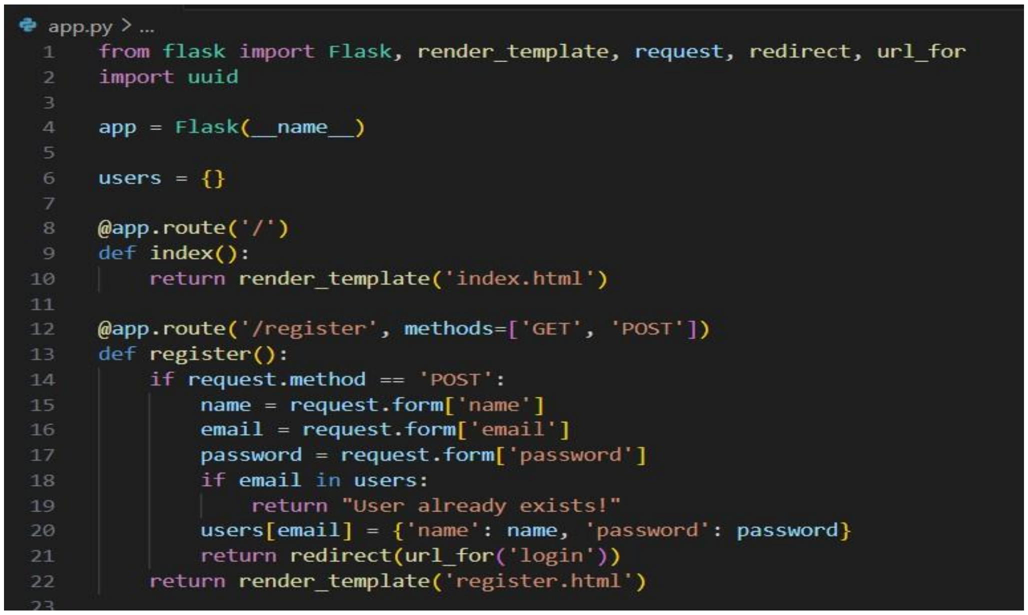
Backend Development and Application Setup


SMART MOVIE TICKET ...

```
static
    1.jpg
templates
    booking.html
    confirm.html
    home.html
    index.html
    login.html
    payment.html
    register.html
    seats.html
    successfull.html
app.py
```

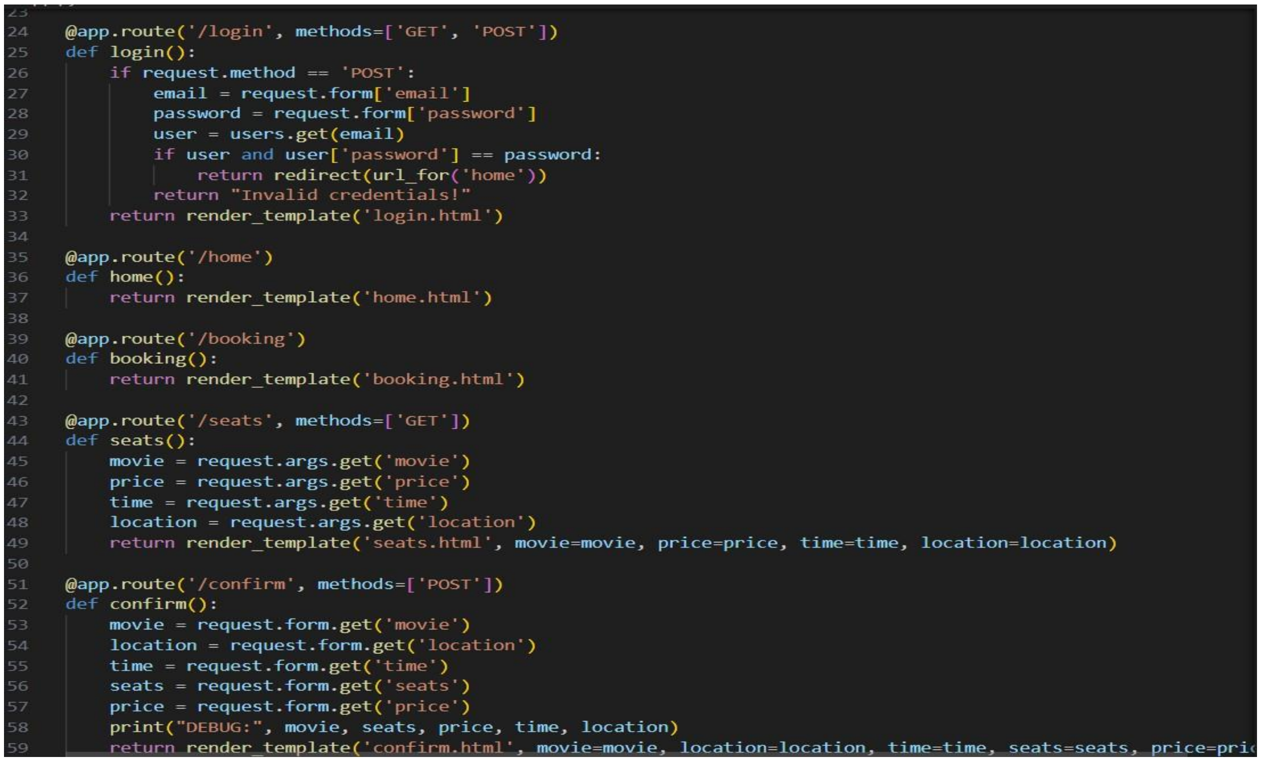
Develop the backend using Flask

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

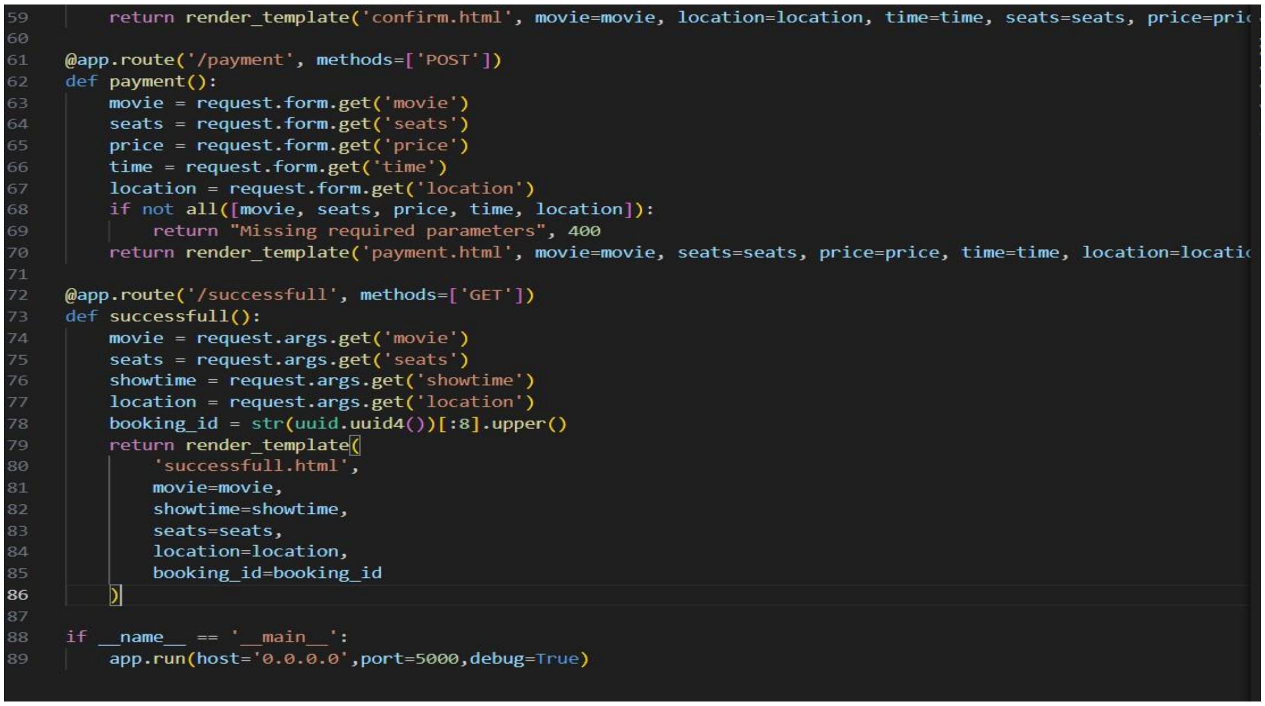

app.py > ...

```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

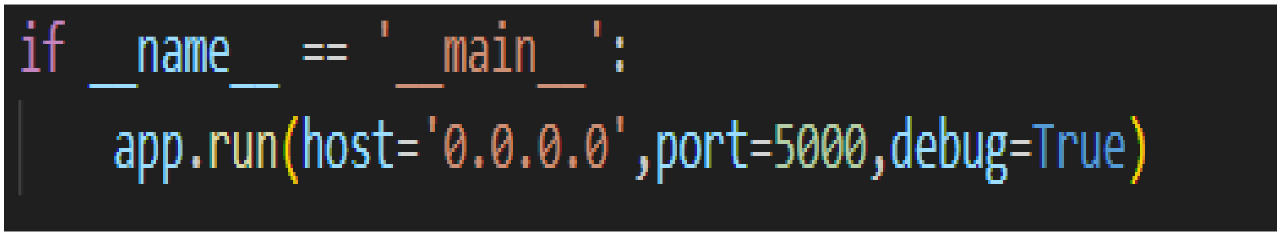


```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = users.get(email)
        if user and user['password'] == password:
            return redirect(url_for('home'))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



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

@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=locatic

@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:



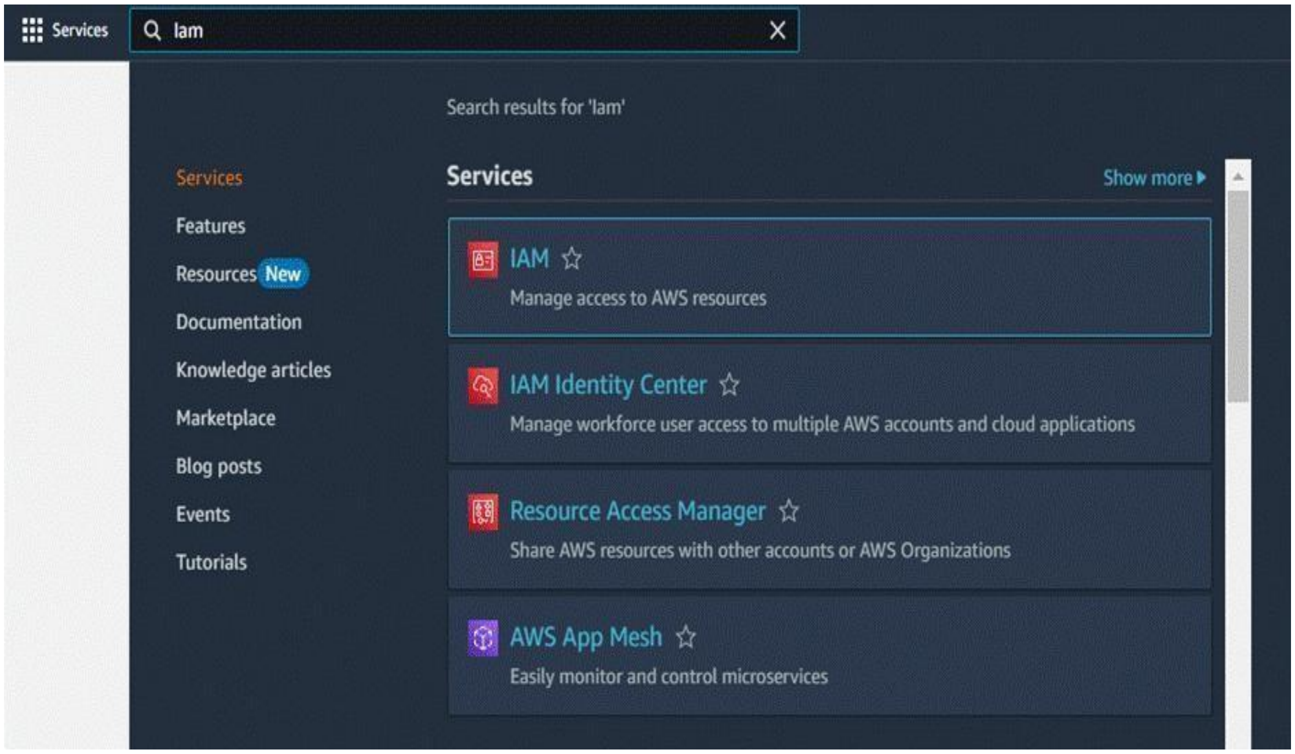
```
if __name__ == '__main__':
    app.run(host='0.0.0.0',port=5000,debug=True)
```

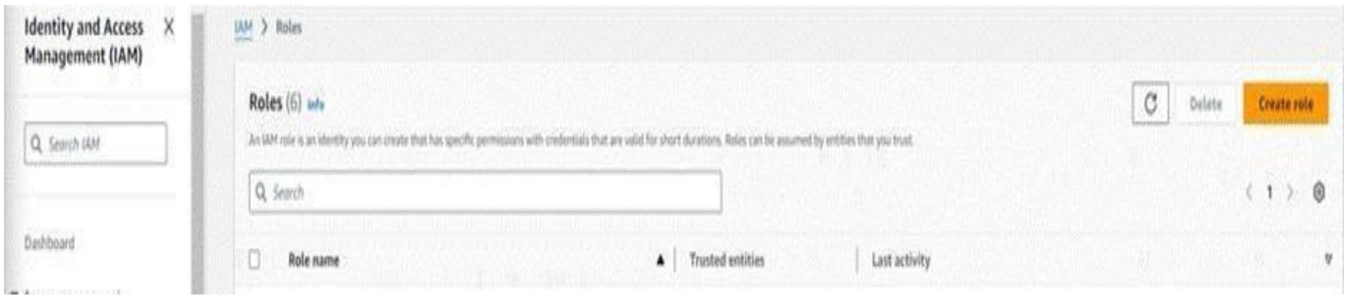
IAM Role Setup:

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.


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



Identity and Access Management (IAM)

Search IAM

Dashboard

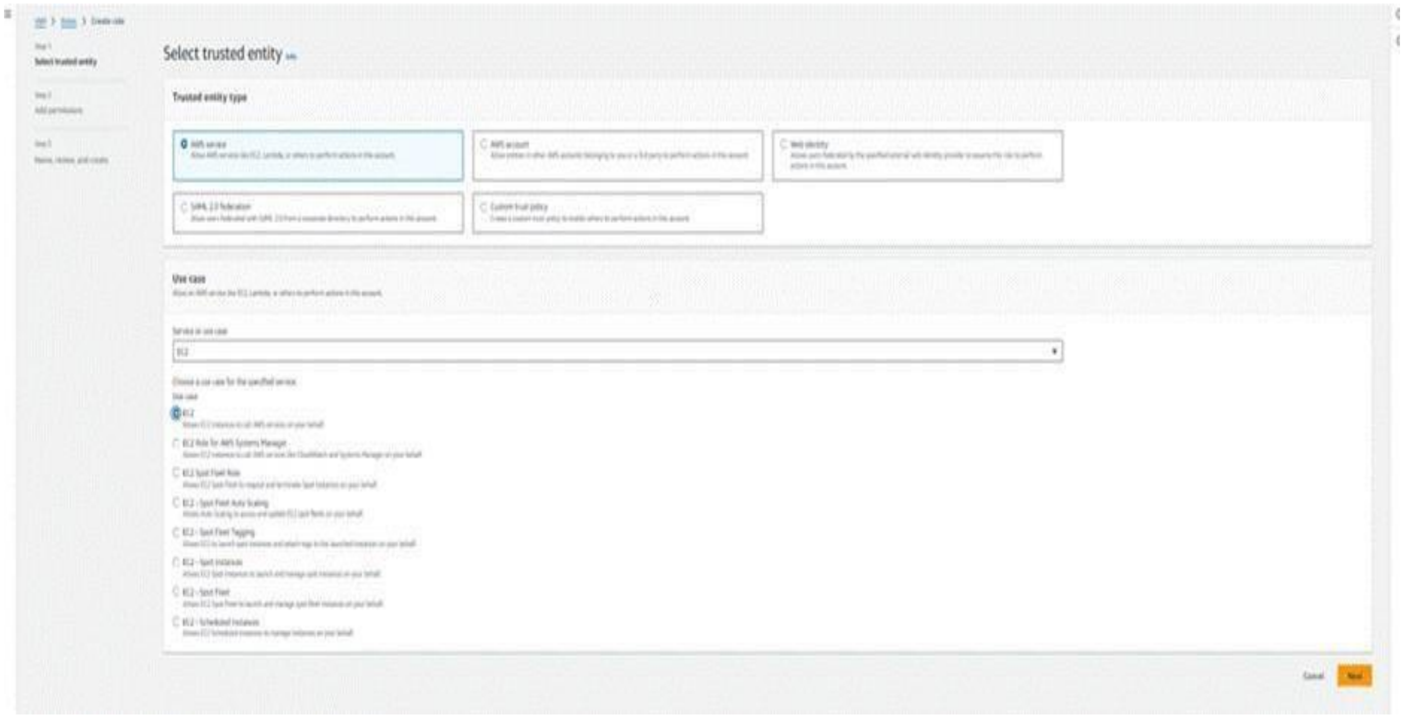
IAM > Roles

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

Role name	Trusted entities	Last activity



IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity

Trusted entity type

[✓ AWS service] Allow AWS services like EC2, Lambda, or others to perform actions in this account.

AWS account — Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

Web identity — [illegible]

SAML 2.0 federation — Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

Custom trust policy — Create a custom trust policy to enable others to perform actions in this account.

Use case

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case: EC2

Choose a use case for the specified service.

Use case

[✓ EC2] Allows EC2 instances to call AWS services on your behalf.

EC2 Role for AWS Systems Manager — Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.

EC2 Spot Fleet Role — Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.

EC2 - Spot Fleet Auto Scaling — Allows Auto Scaling to access and update EC2 spot fleets on your behalf.

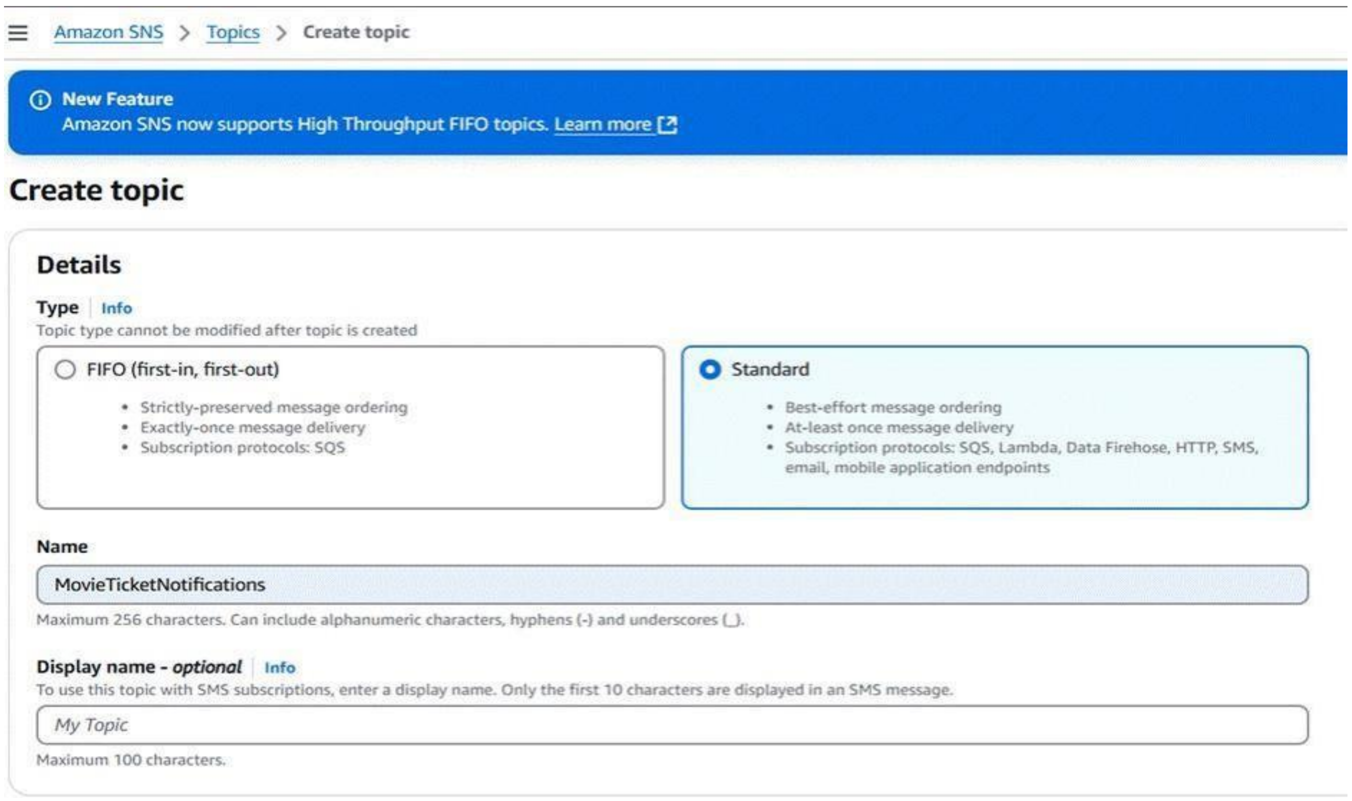
EC2 - Spot Fleet Tagging — Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.

EC2 - Spot Instances — Allows EC2 Spot Instances to launch and manage spot instances on your behalf.

EC2 - Spot Fleet — Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.

EC2 - Scheduled Instances — Allows EC2 Scheduled Instances to manage instances on your behalf.

Cancel Next


Amazon SNS > Topics > Create topic

New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more

Create topic

Details

Type | Info
Topic type cannot be modified after topic is created

FIFO (first-in, first-out)
- Strictly-preserved message ordering
- Exactly-once message delivery
- Subscription protocols: SQS

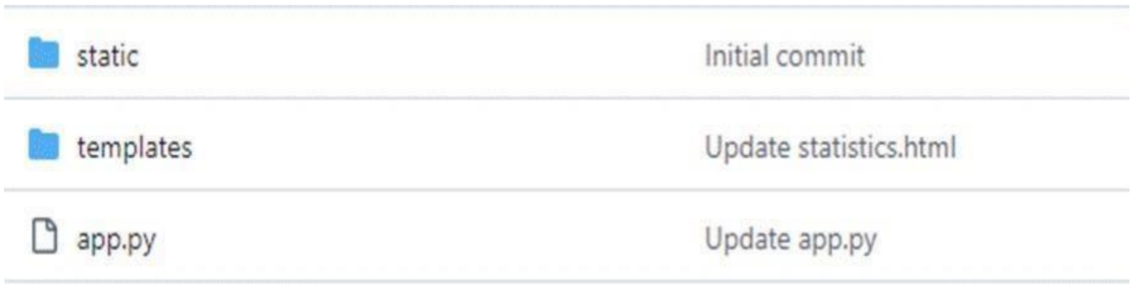
[✓ Standard]
- Best-effort message ordering
- At-least once message delivery
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Name
MovieTicketNotifications
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

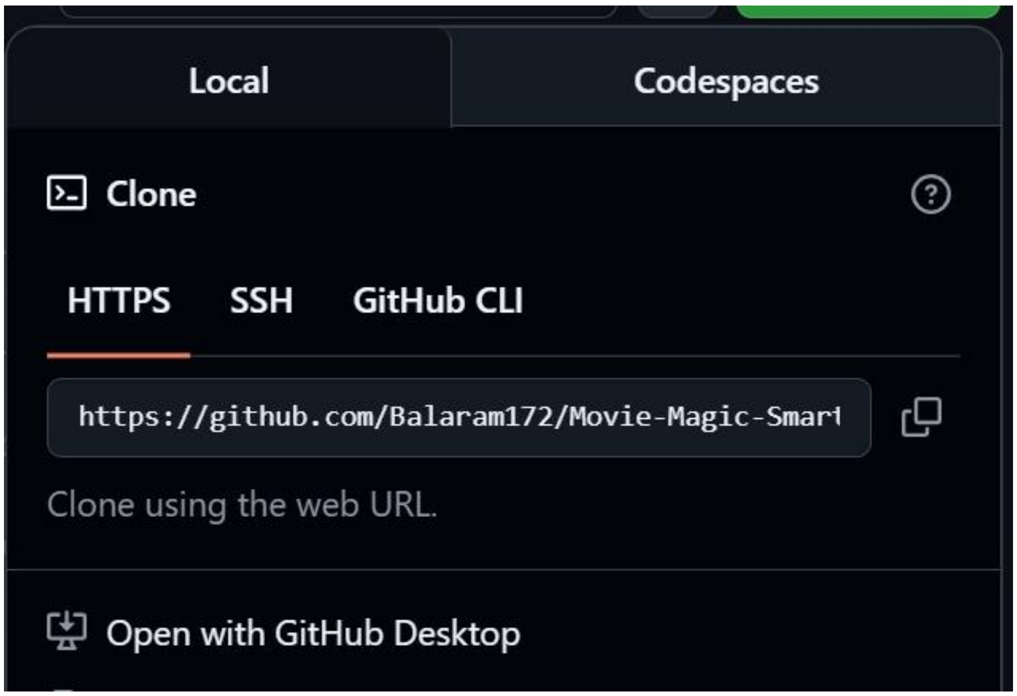
Display name - *optional* | Info
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.
My Topic
Maximum 100 characters.

Load your Project Files to GitHub

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



static	Initial commit
templates	Update statistics.html
app.py	Update app.py

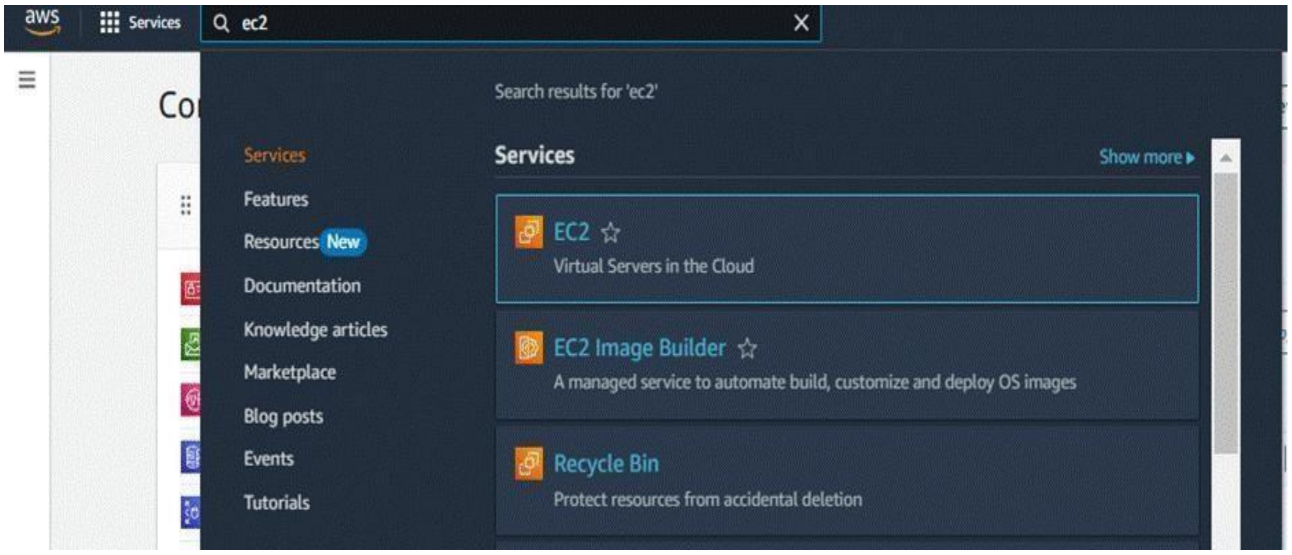


```
https://github.com/Balaram172/Movie-Magic-Smart
```

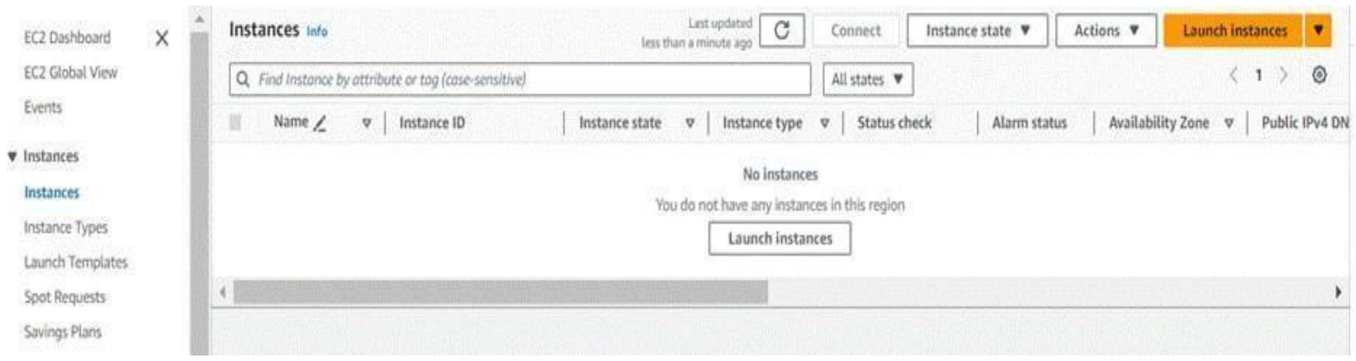
Clone using the web URL.

Launch an EC2 instance to host the Flask

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

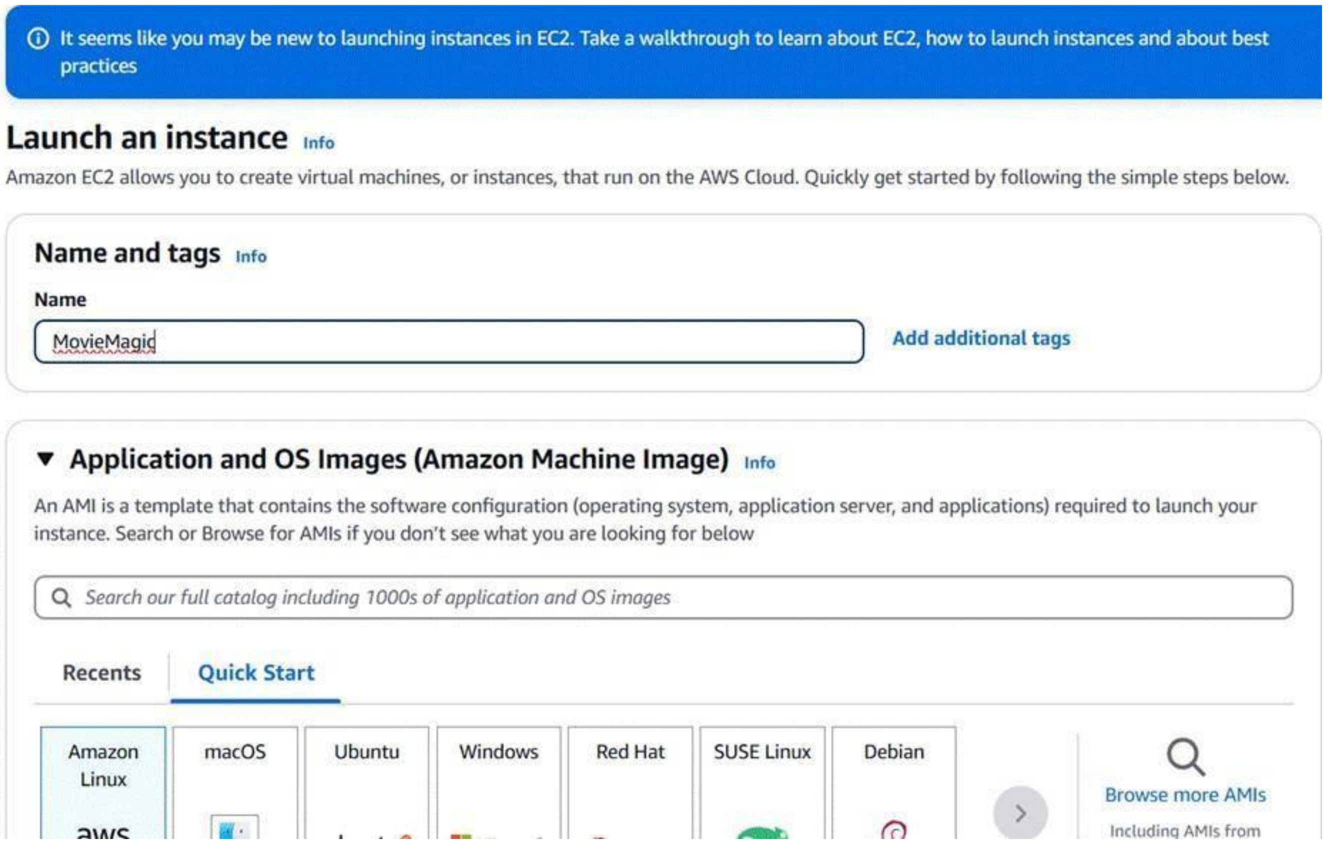


➢ Click on Launch instance to launch EC2 instance



No instances

You do not have any instances in this region



It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices

Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

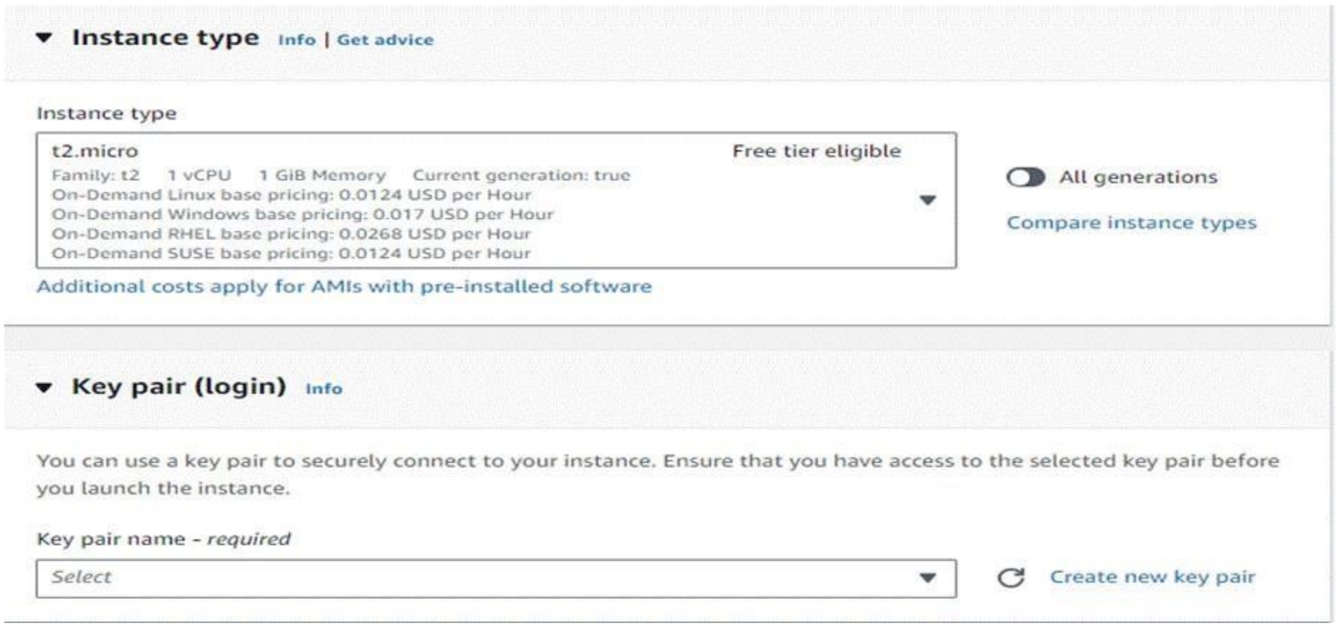
Name and tags Info

Name: MovieMagic

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

➢ 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

Additional costs apply for AMIs with pre-installed software

All generations

Compare instance types

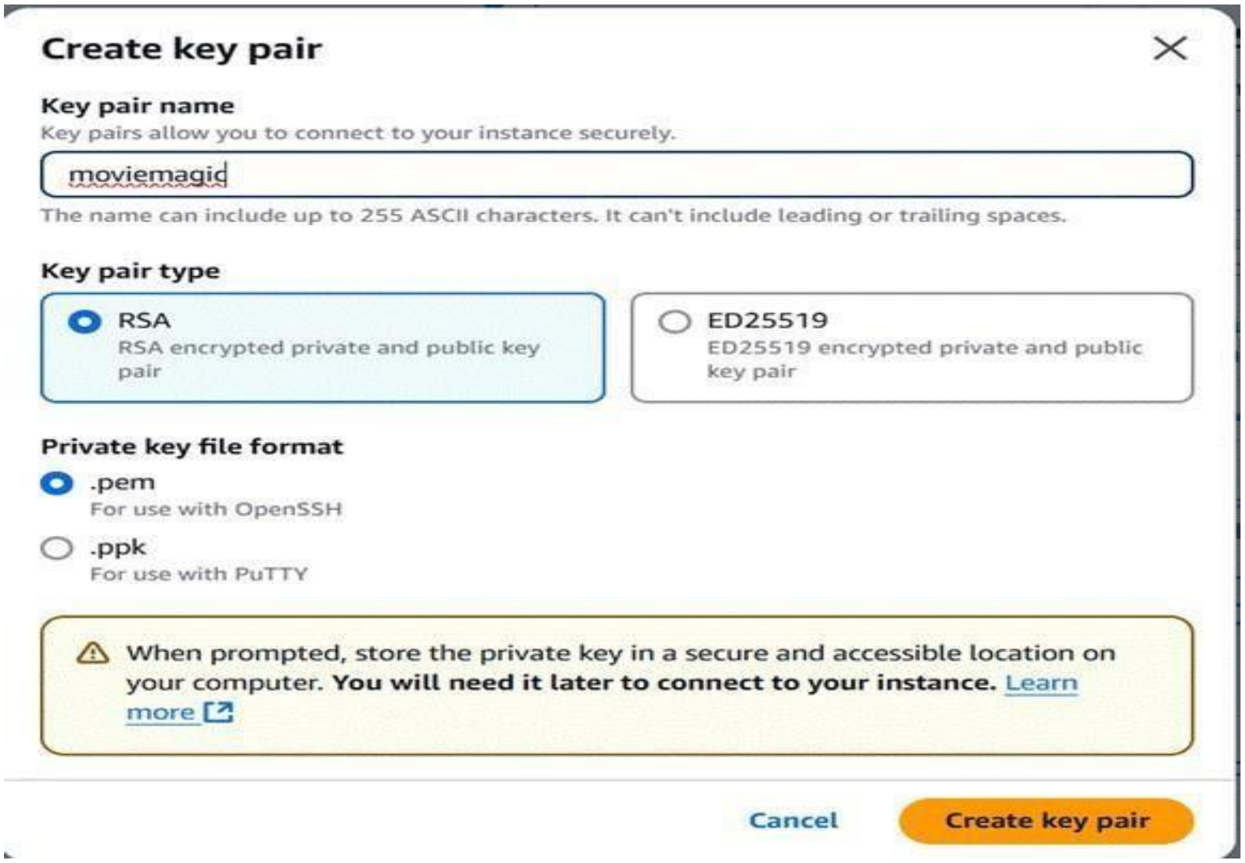
▼ Key pair (login) Info

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

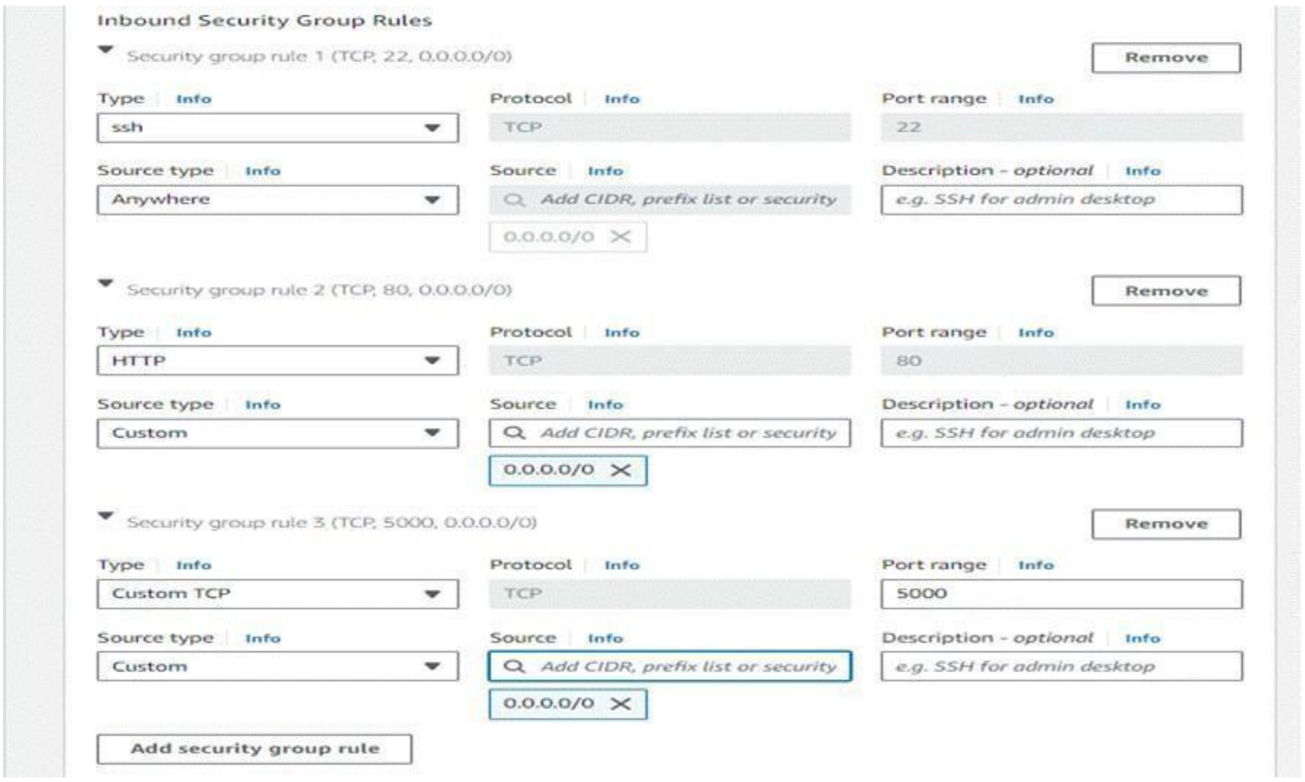
.ppk For use with PuTTY

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.



Inbound Security Group Rules

Security group rule 1 (TCP, 22, 0.0.0.0/0)

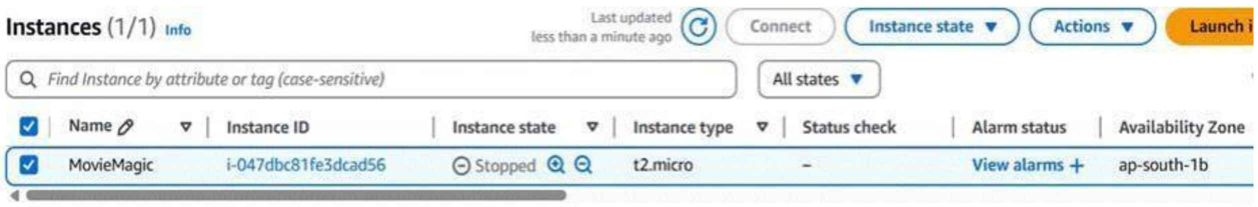
Type: ssh	Protocol: TCP	Port range: 22
Source type: Anywhere	Source: 0.0.0.0/0	Description - optional: e.g. SSH for admin desktop

Security group rule 2 (TCP, 80, 0.0.0.0/0)

Type: HTTP	Protocol: TCP	Port range: 80
Source type: Custom	Source: 0.0.0.0/0	Description - optional: e.g. SSH for admin desktop

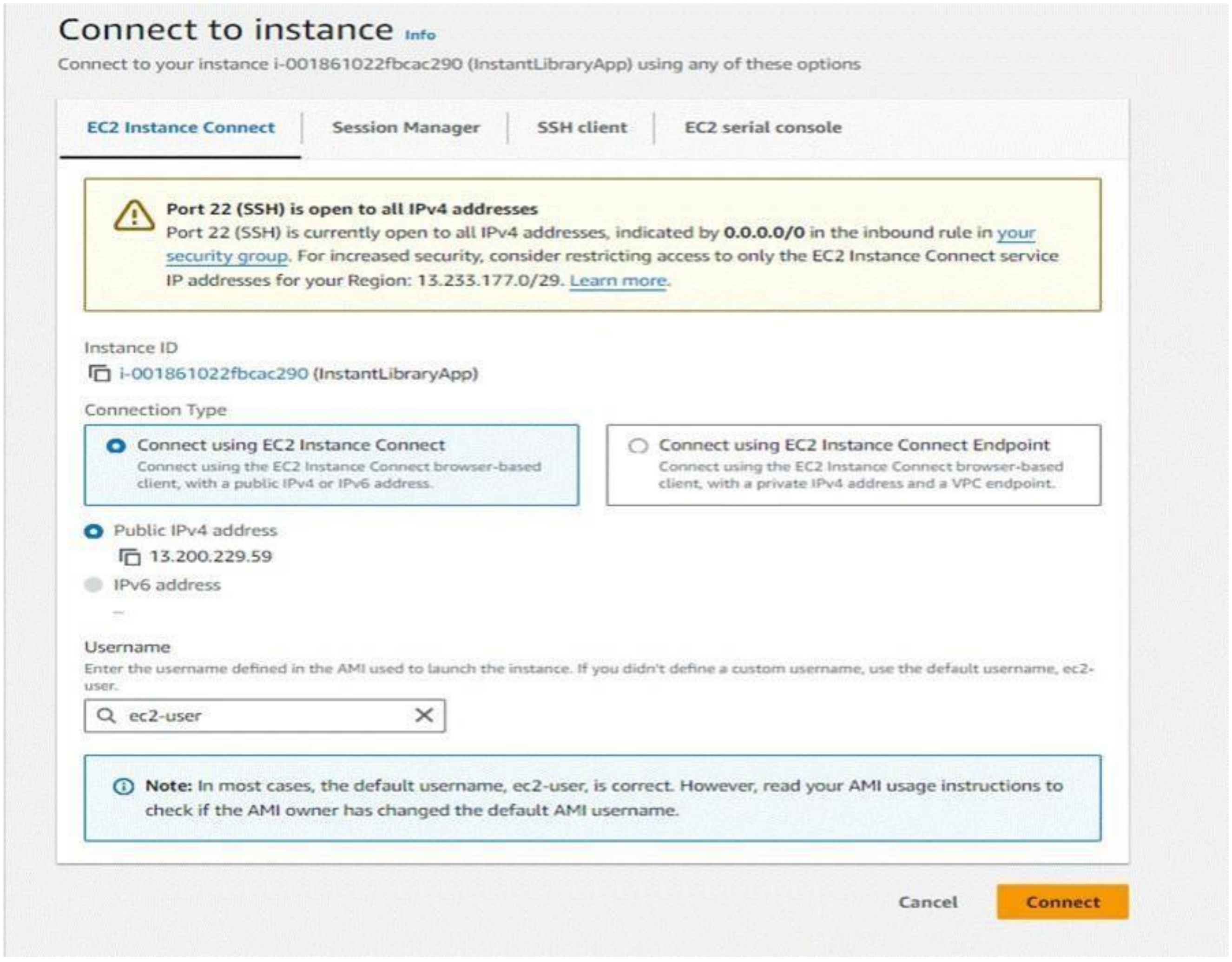
Security group rule 3 (TCP, 5000, 0.0.0.0/0)

Type: Custom TCP	Protocol: TCP	Port range: 5000
Source type: Custom	Source: 0.0.0.0/0	Description - optional: e.g. SSH for admin desktop


Instances (1/1) Info
Last updated less than a minute ago
Connect
Instance state ▼
Actions ▼
Launch i
Find Instance by attribute or tag (case-sensitive)
All states ▼

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
MovieMagic	i-047dbc81fe3dcad56	Stopped	t2.micro	–	View alarms +	ap-south-1b

➢ Now connect the EC2 with the files



Connect to your instance i-001861022fbcac290 (InstantLibraryApp) using any of these options

Port 22 (SSH) is open to all IPv4 addresses
Port 22 (SSH) is currently open to all IPv4 addresses, indicated by **0.0.0.0/0** in the inbound rule in your security group. For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 13.233.177.0/29. Learn more.

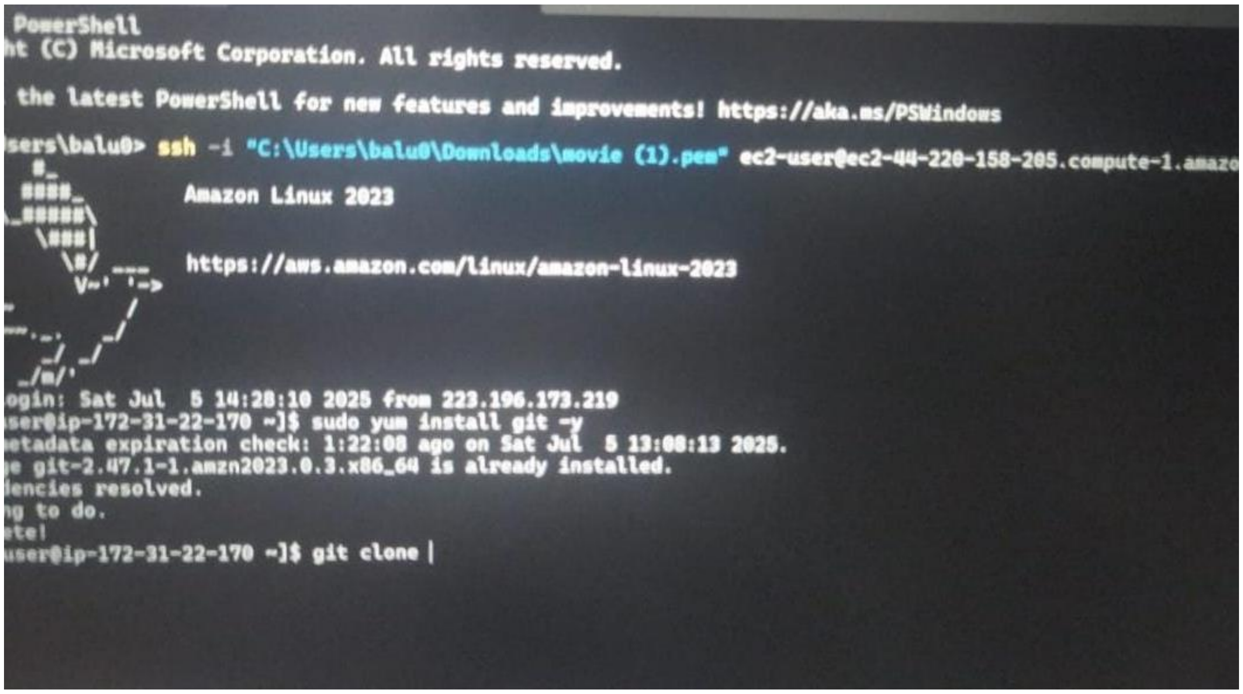
Instance ID
i-001861022fbcac290 (InstantLibraryApp)

Connection Type
[✓ Connect using EC2 Instance Connect] Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.
Connect using EC2 Instance Connect Endpoint: Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

[✓ Public IPv4 address] 13.200.229.59
IPv6 address –

Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.
ec2-user

Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.



```
PowerShell
ht (C) Microsoft Corporation. All rights reserved.

 the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

sers\balu0> ssh -i "C:\Users\balu0\Downloads\movie (1).pem" ec2-user@ec2-44-220-158-205.compute-1.amazo
   #_
  ####_          Amazon Linux 2023
 _#####\
   \###|
     \#/  ___    https://aws.amazon.com/linux/amazon-linux-2023
      V~' '->
         /
-.._.   _/
   _/ _/
  _/m/'
ogin: Sat Jul  5 14:28:10 2025 from 223.196.173.219
ser@ip-172-31-22-170 ~]$ sudo yum install git -y
etadata expiration check: 1:22:08 ago on Sat Jul  5 13:08:13 2025.
e git-2.47.1-1.amzn2023.0.3.x86_64 is already installed.
lencies resolved.
ng to do.
ete!
user@ip-172-31-22-170 ~]$ git clone |
```

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/Balaram172/Movie-Magic-Smart-Movie-Ticket-Booking-System.git

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

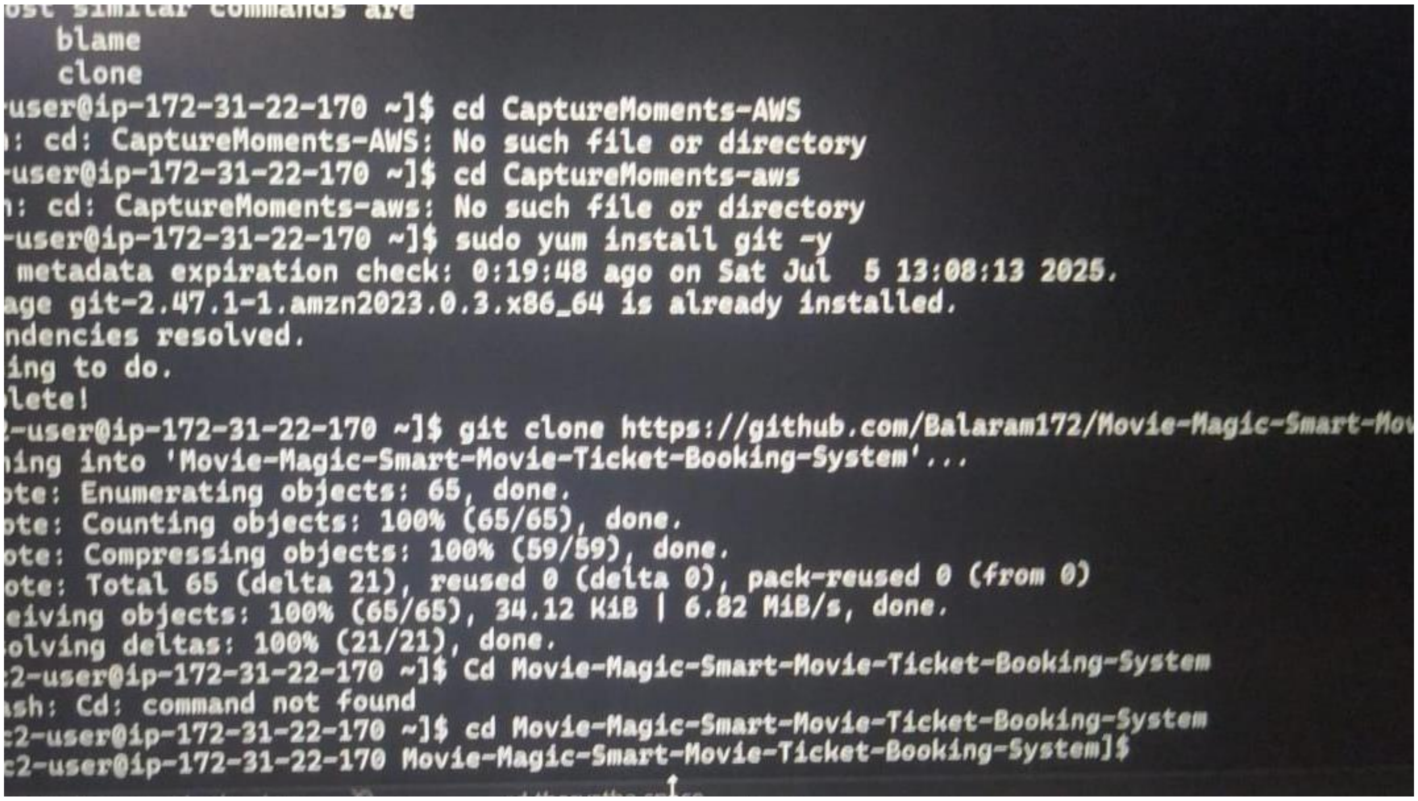
This will download your project to the EC2 instance.

- 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:

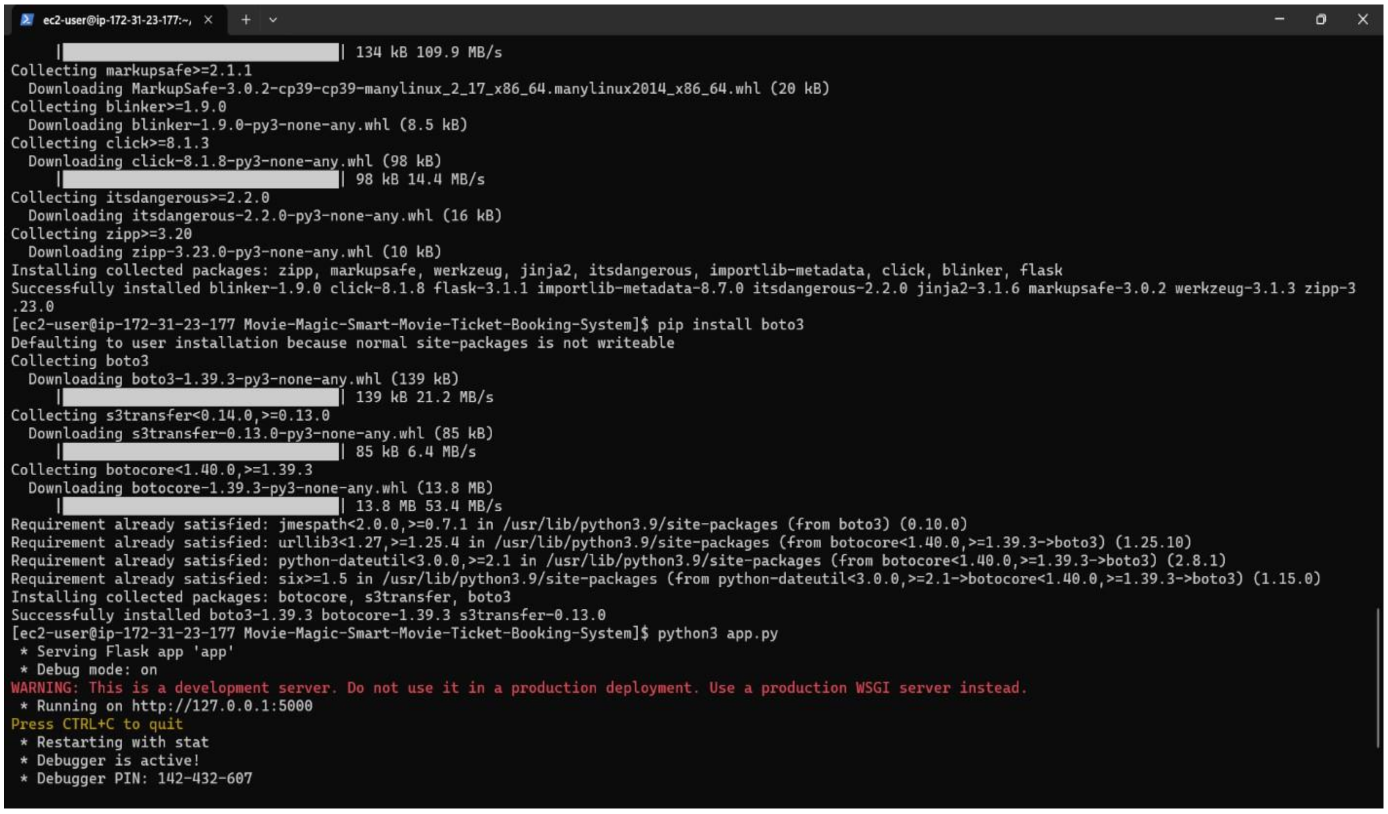
Run the Flask Application: sudo flask run --host=0.0.0.0 --port=5000

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

- Run the Flask app on the EC2 instance



```
ost similar commands are
    blame
    clone
user@ip-172-31-22-170 ~]$ cd CaptureMoments-AWS
: cd: CaptureMoments-AWS: No such file or directory
user@ip-172-31-22-170 ~]$ cd CaptureMoments-aws
: cd: CaptureMoments-aws: No such file or directory
user@ip-172-31-22-170 ~]$ sudo yum install git -y
 metadata expiration check: 0:19:48 ago on Sat Jul  5 13:08:13 2025.
age git-2.47.1-1.amzn2023.0.3.x86_64 is already installed.
ndencies resolved.
ing to do.
lete!
-user@ip-172-31-22-170 ~]$ git clone https://github.com/Balaram172/Movie-Magic-Smart-Mov
ing into 'Movie-Magic-Smart-Movie-Ticket-Booking-System'...
ote: Enumerating objects: 65, done.
ote: Counting objects: 100% (65/65), done.
ote: Compressing objects: 100% (59/59), done.
ote: Total 65 (delta 21), reused 0 (delta 0), pack-reused 0 (from 0)
eiving objects: 100% (65/65), 34.12 KiB | 6.82 MiB/s, done.
olving deltas: 100% (21/21), done.
2-user@ip-172-31-22-170 ~]$ Cd Movie-Magic-Smart-Movie-Ticket-Booking-System
sh: Cd: command not found
2-user@ip-172-31-22-170 ~]$ cd Movie-Magic-Smart-Movie-Ticket-Booking-System
2-user@ip-172-31-22-170 Movie-Magic-Smart-Movie-Ticket-Booking-System]$
```



```
ec2-user@ip-172-31-23-177:~,
     |████████████████████████████████| 134 kB 109.9 MB/s
Collecting markupsafe>=2.1.1
  Downloading MarkupSafe-3.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting blinker>=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting click>=8.1.3
  Downloading click-8.1.8-py3-none-any.whl (98 kB)
     |████████████████████████████████| 98 kB 14.4 MB/s
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting zipp>=3.20
  Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: zipp, markupsafe, werkzeug, jinja2, itsdangerous, importlib-metadata, click, blinker, flask
Successfully installed blinker-1.9.0 click-8.1.8 flask-3.1.1 importlib-metadata-8.7.0 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3 zipp-3
.23.0
[ec2-user@ip-172-31-23-177 Movie-Magic-Smart-Movie-Ticket-Booking-System]$ pip install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.39.3-py3-none-any.whl (139 kB)
     |████████████████████████████████| 139 kB 21.2 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
     |████████████████████████████████| 85 kB 6.4 MB/s
Collecting botocore<1.40.0,>=1.39.3
  Downloading botocore-1.39.3-py3-none-any.whl (13.8 MB)
     |████████████████████████████████| 13.8 MB 53.4 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-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/python3.9/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-172-31-23-177 Movie-Magic-Smart-Movie-Ticket-Booking-System]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 142-432-607
```

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

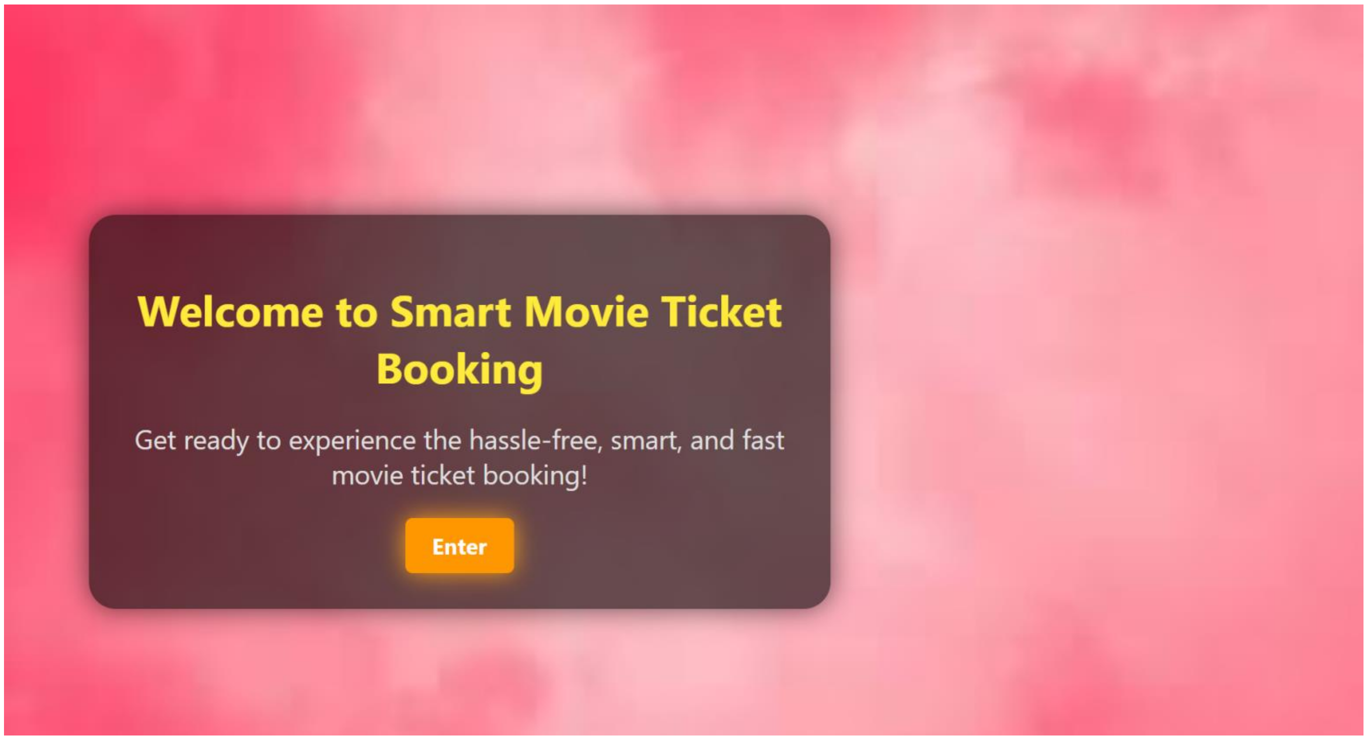
Public IP’s: http://54.197.161.11:5000

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

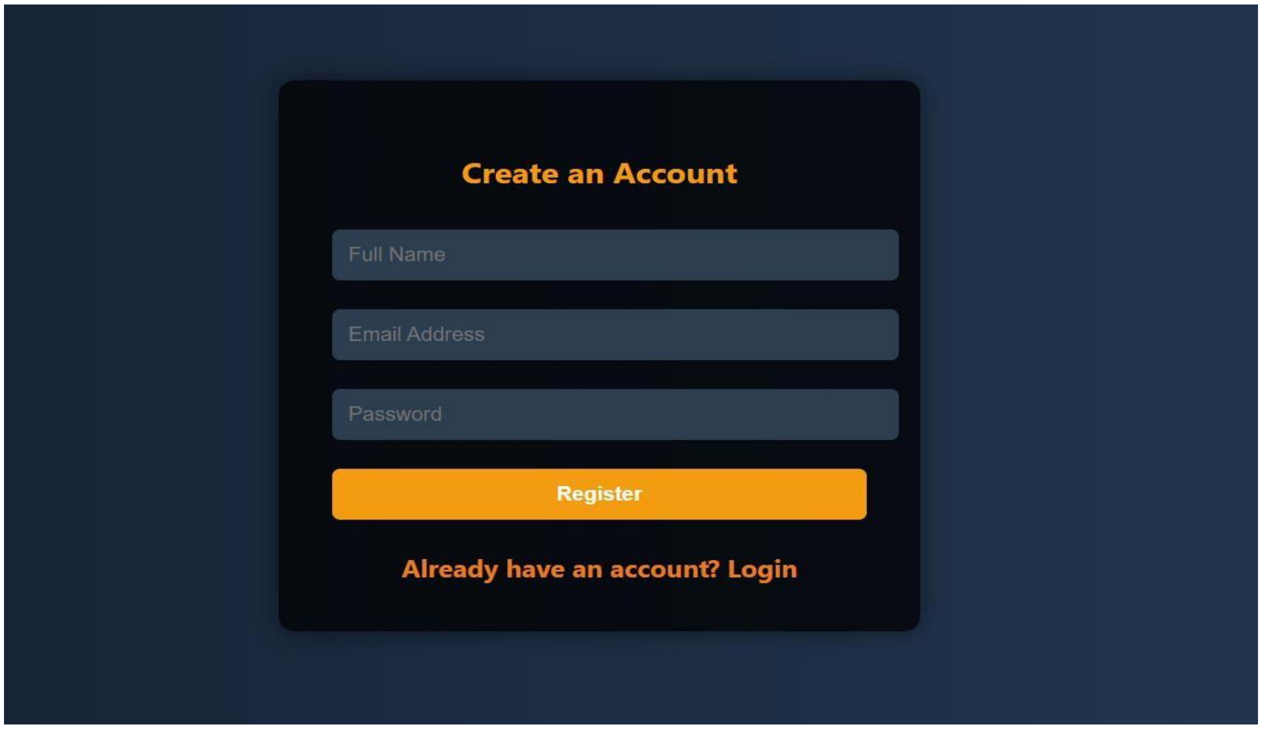
Index Page:



Welcome to Smart Movie Ticket Booking

Get ready to experience the hassle-free, smart, and fast movie ticket booking!

Register Page:



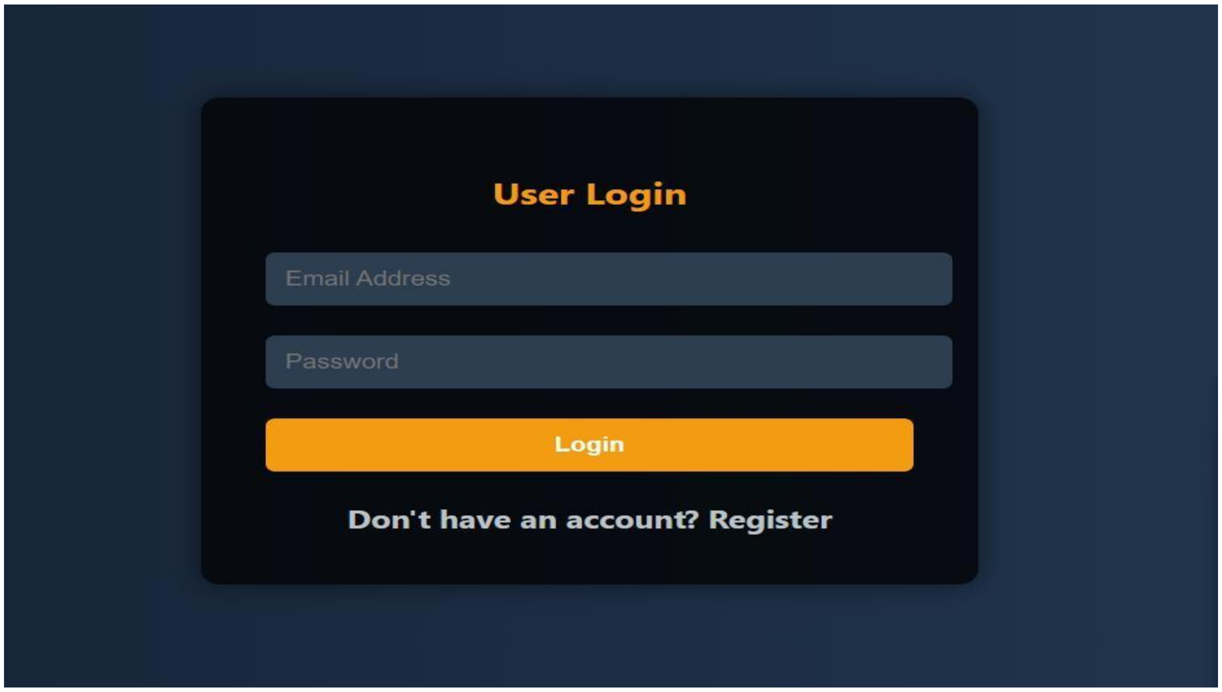
Create an Account

Full Name

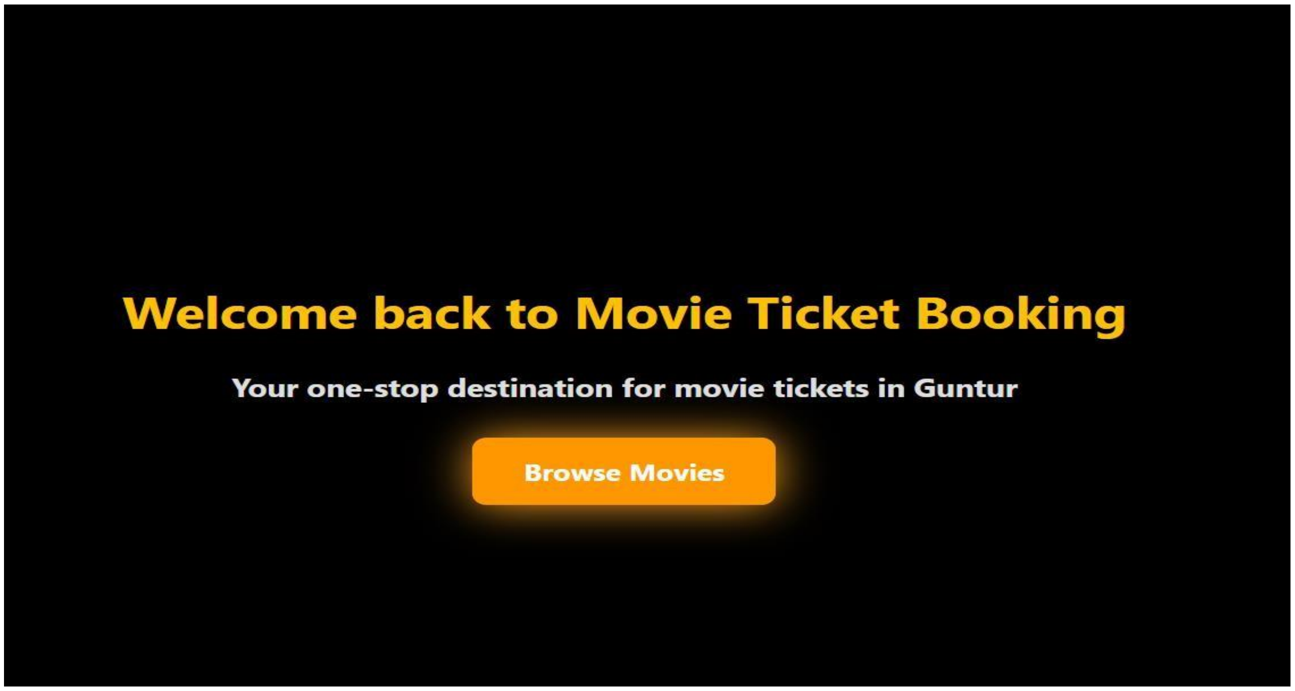
Email Address

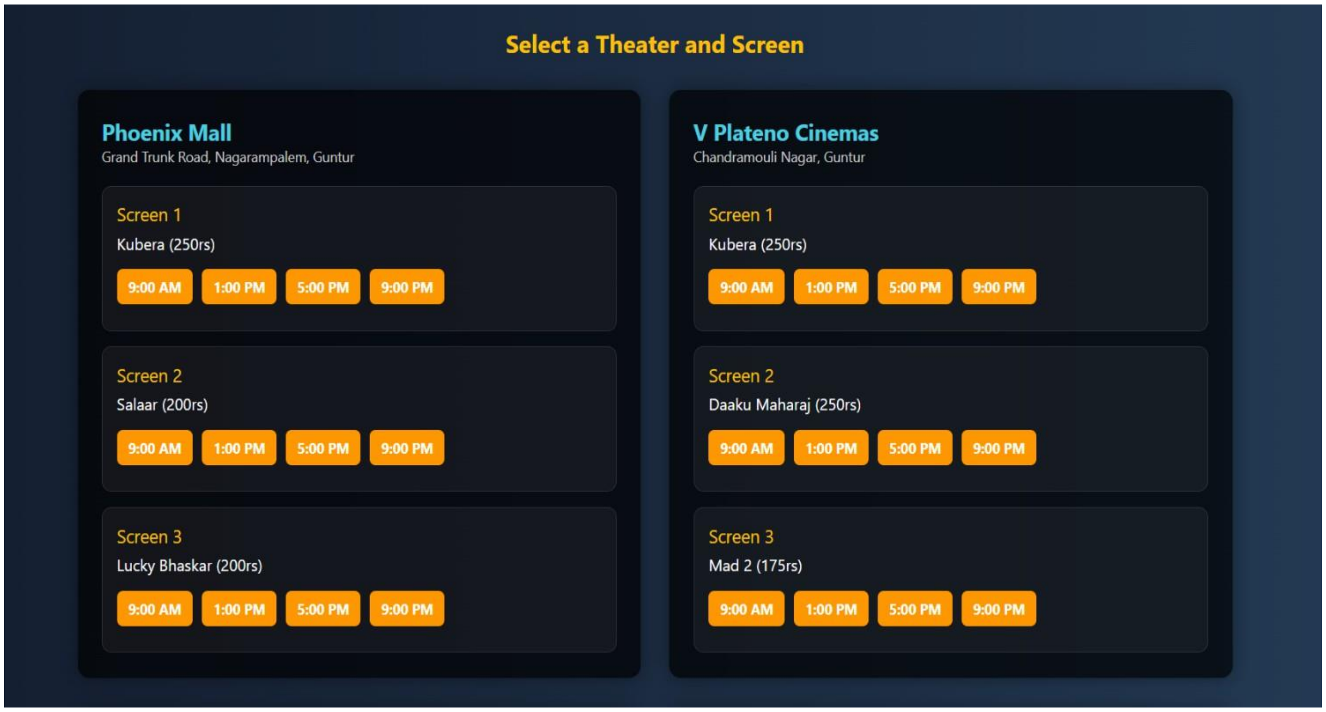
Password

Login Page:



Home page:


Welcome back to Movie Ticket Booking
Your one-stop destination for movie tickets in Guntur
Browse Movies



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

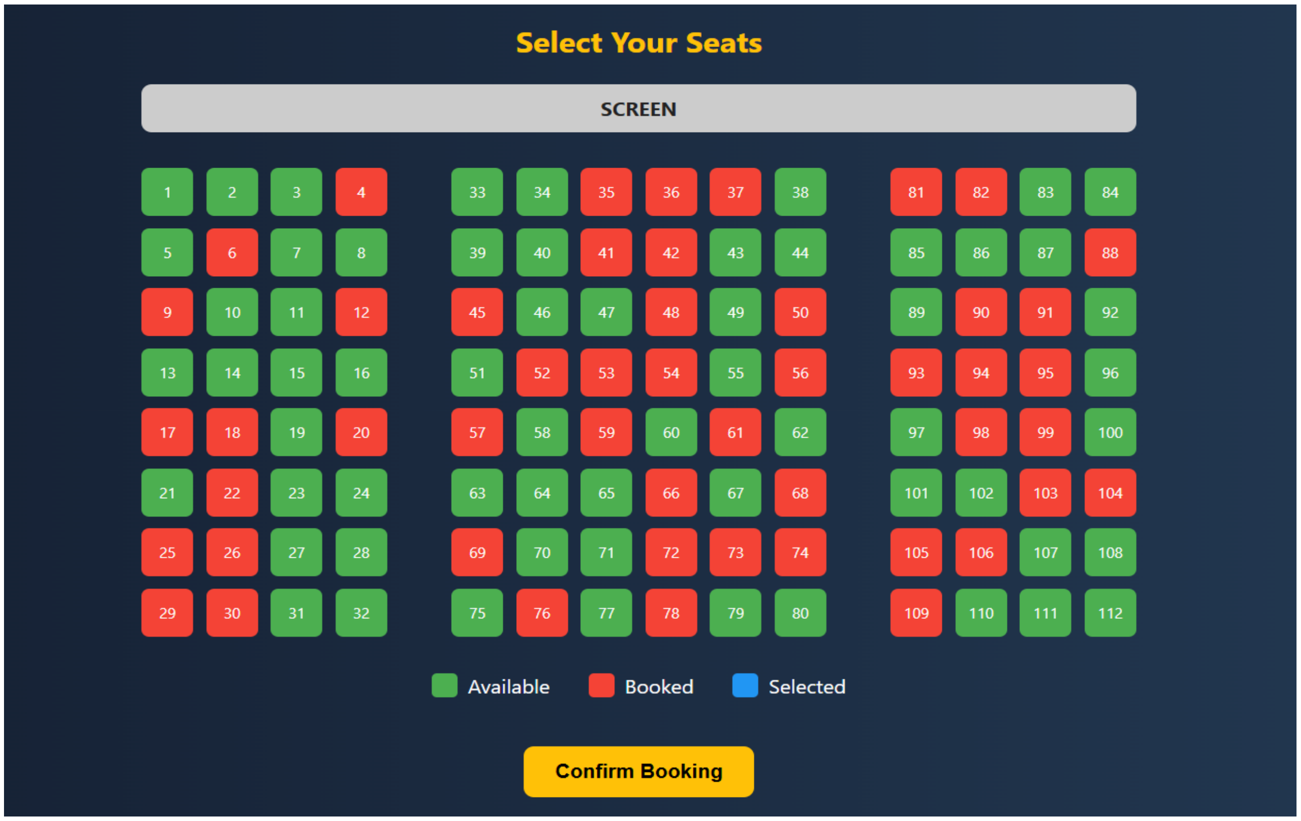
Screen 3
Lucky Bhaskar (200rs)
9:00 AM | 1:00 PM | 5:00 PM | 9:00 PM

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

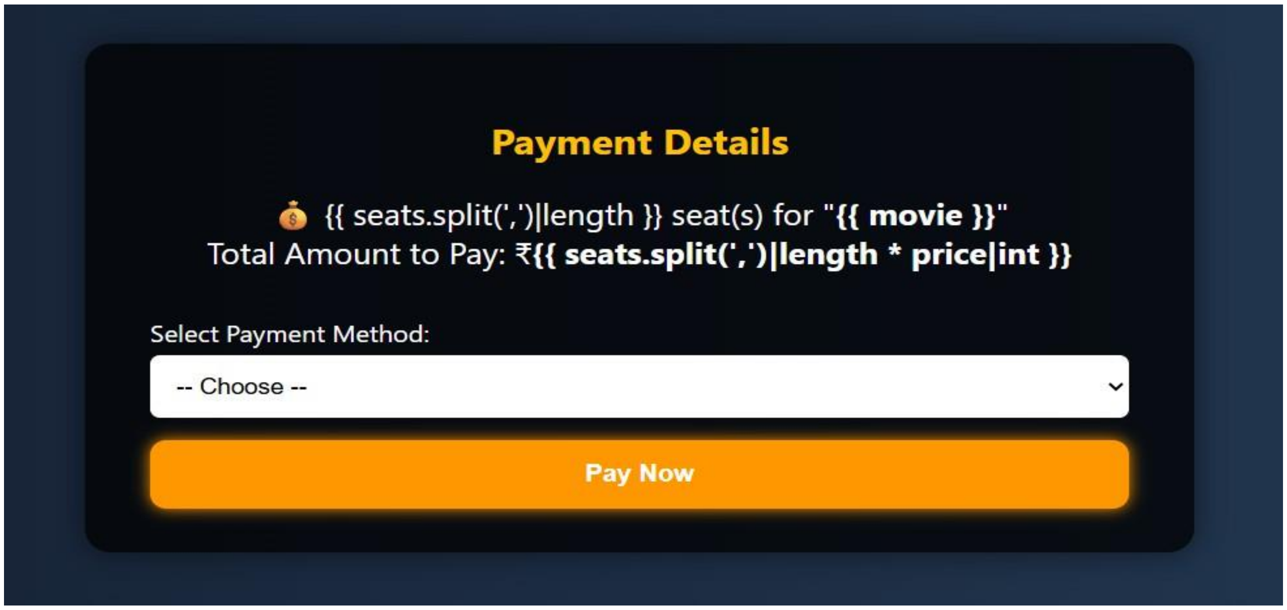
SCREEN

🟩 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

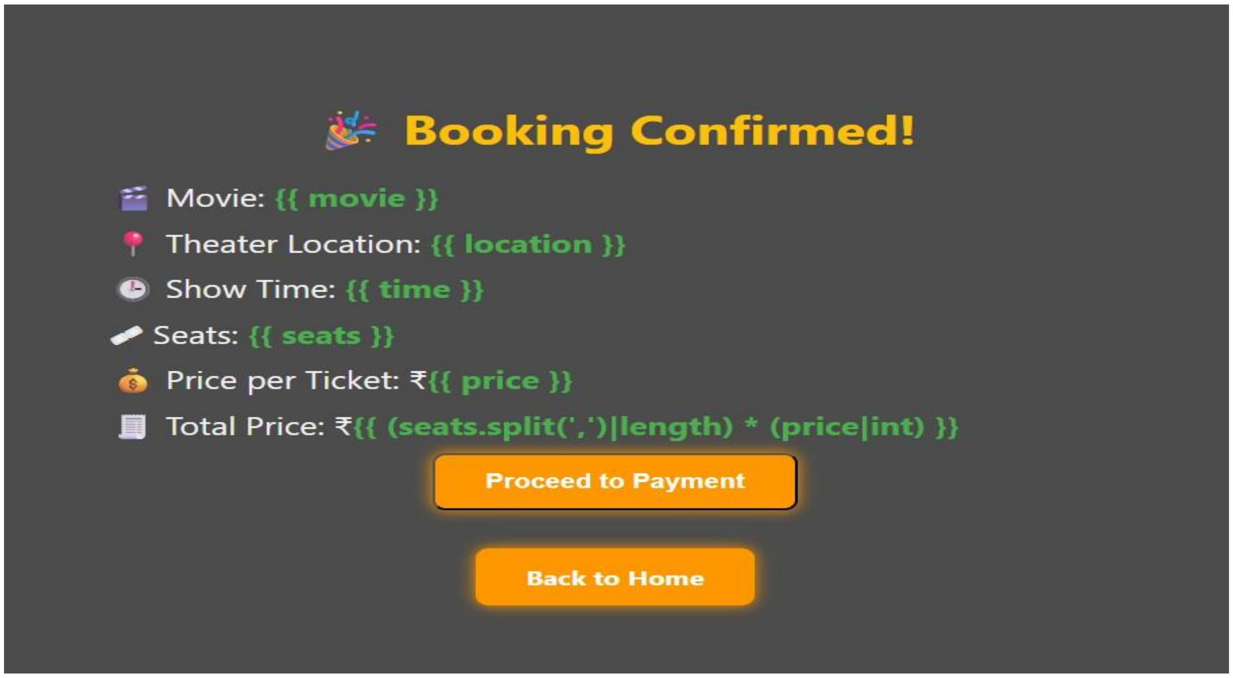


Payment Details

💰 {{ seats.split(',')|length }} seat(s) for "**{{ movie }}**"
Total Amount to Pay: ₹**{{ seats.split(',')|length * price|int }}**

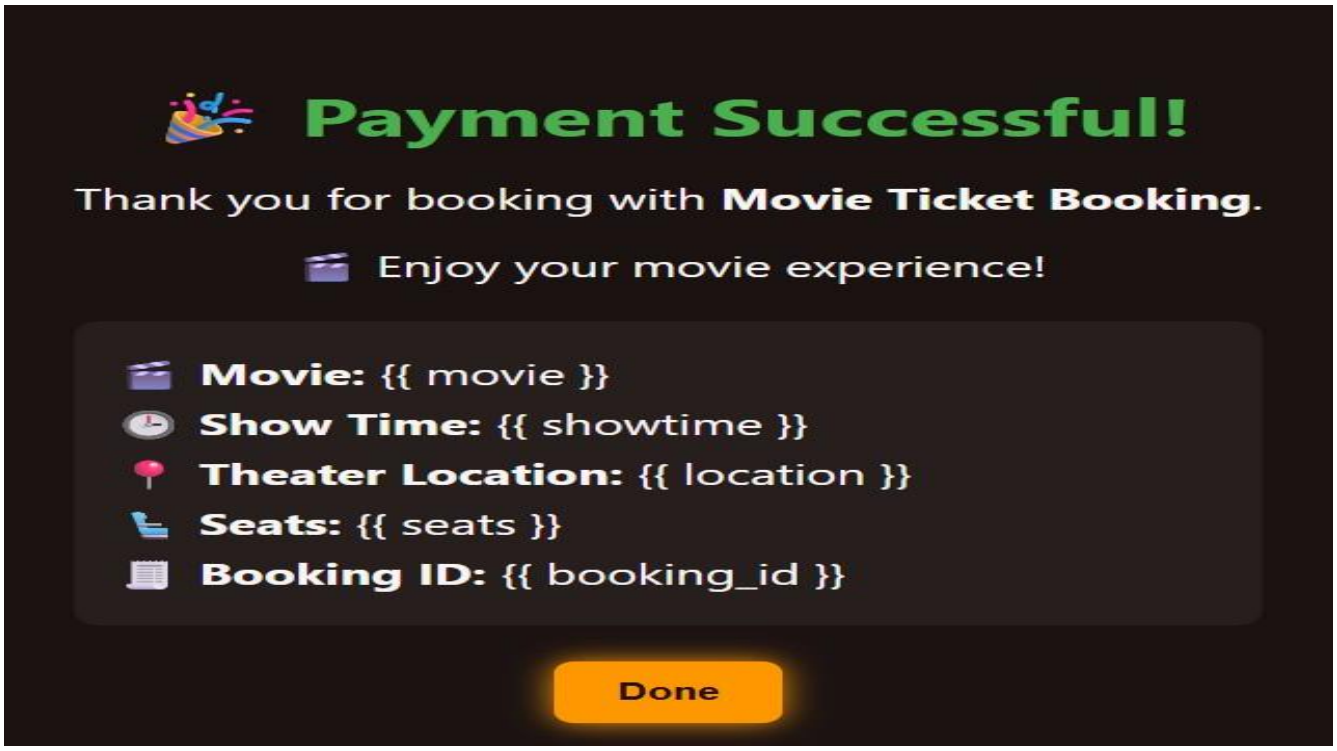
Select Payment Method:

-- Choose --



🎉 Booking Confirmed!

🎬 Movie: {{ movie }}
📍 Theater Location: {{ location }}
🕒 Show Time: {{ time }}
🎟 Seats: {{ seats }}
💰 Price per Ticket: ₹{{ price }}
🧾 Total Price: ₹{{ (seats.split(',')|length) * (price|int) }}



🎉 Payment Successful!

Thank you for booking with **Movie Ticket Booking**.

🎬 Enjoy your movie experience!

🎬 **Movie:** {{ movie }}
🕒 **Show Time:** {{ showtime }}
📍 **Theater Location:** {{ location }}
💺 **Seats:** {{ seats }}
🧾 **Booking ID:** {{ booking_id }}

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