

Movie Magic—Smart Movie Ticket Booking System

Project Description:

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

Scenarios :

Scenario 1: Efficient Ticket Booking System for Users

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

Scenario 2: Seamless Booking Confirmation Notifications

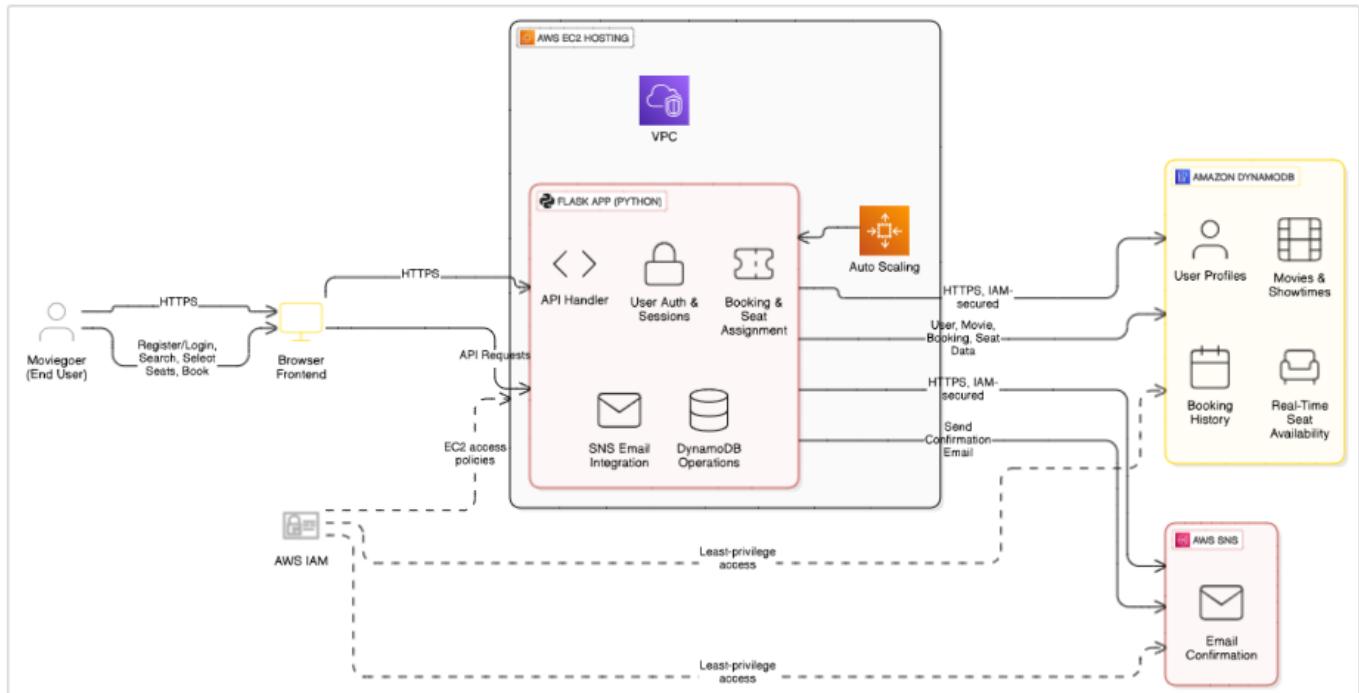
When a user completes a ticket booking, the Movie Magic System leverages AWS SNS to send instant email notifications to confirm the booking. For instance, once the booking is submitted, Flask processes the transaction, and SNS sends a customized email to the user with all ticket details, including movie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

Scenario 3: Easy Access to Movies and Events

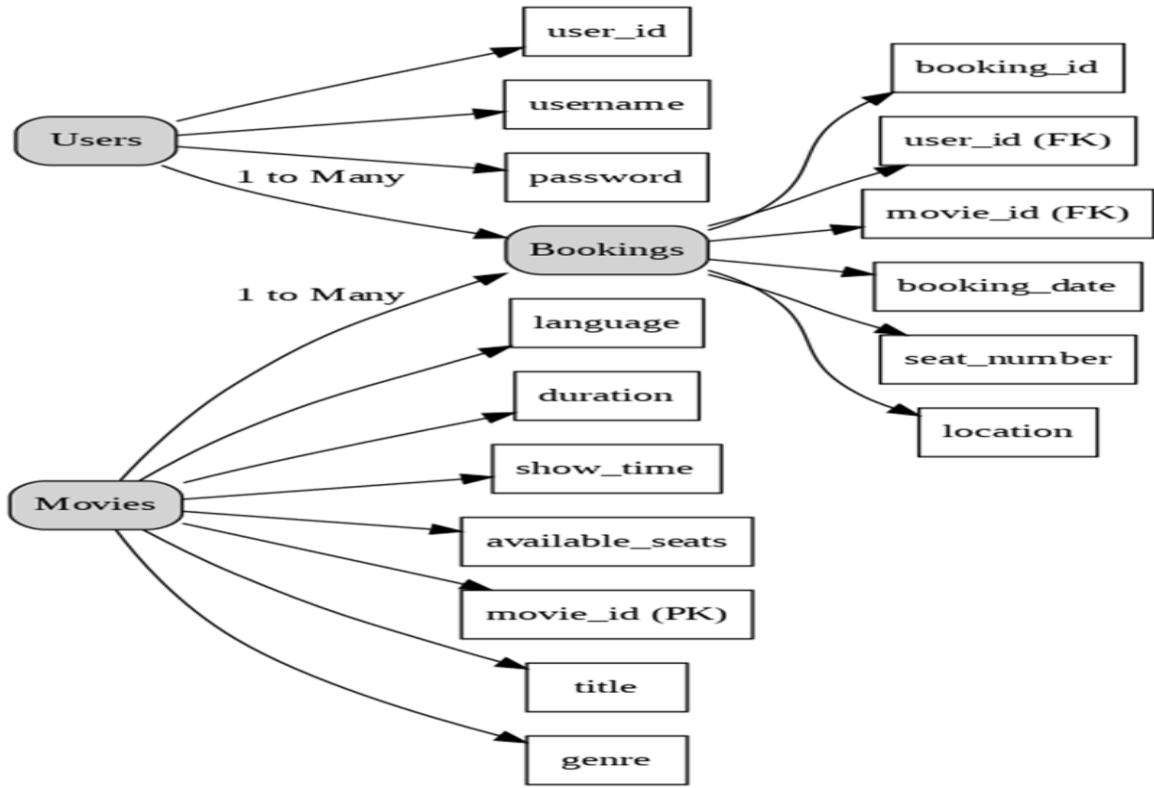
The Movie Magic platform offers users a seamless interface to explore currently running movies and upcoming live events. After logging in, a user can search by location or genre and instantly view listings with showtimes, ratings, and available seats. Flask dynamically fetches this data from DynamoDB,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

AWS ARCHITECTURE

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, enhancing communication and user engagement.



Entity Relationship (ER)Diagram:



Pre-requisites

- AWS Account Setup :
<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
- AWS IAM (Identity and Access Management) :
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
- AWS EC2 (Elastic Compute Cloud) :
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
- AWS DynamoDB :
<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>
- Amazon SNS :
<https://docs.aws.amazon.com/sns/latest/dg/welcome.html>
- Git Documentation :
<https://git-scm.com/doc>
- VS Code Installation : (download the VS Code using the below link or you can get that in Microsoft store)
<https://code.visualstudio.com/download>

Project Flow

Milestone 1. Backend Development and Application Setup

- **Develop the Backend Using Flask.**
- **Integrate AWS Services Using boto3.**

Milestone 2. AWS Account Setup and Login

- **Set up an AWS account if not already done.**
- **Log in to the AWS Management Console**

Milestone 3. DynamoDB Database Creation and Setup

- **Create a DynamoDB Table.**
- **Configure Attributes for User Data and Book Requests.**

Milestone 4. SNS Notification Setup

- **Create SNS topics for book request notifications.**
- **Subscribe users and library staff to SNS email notifications.**

Milestone 5. IAM Role Setup

- **Create IAM Role**
- **Attach Policies**

Milestone 6. EC2 Instance Setup

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

Milestone 7. Deployment on EC2

- **Upload Flask Files**
- **Run the Flask App**

Milestone 8. Testing and Deployment

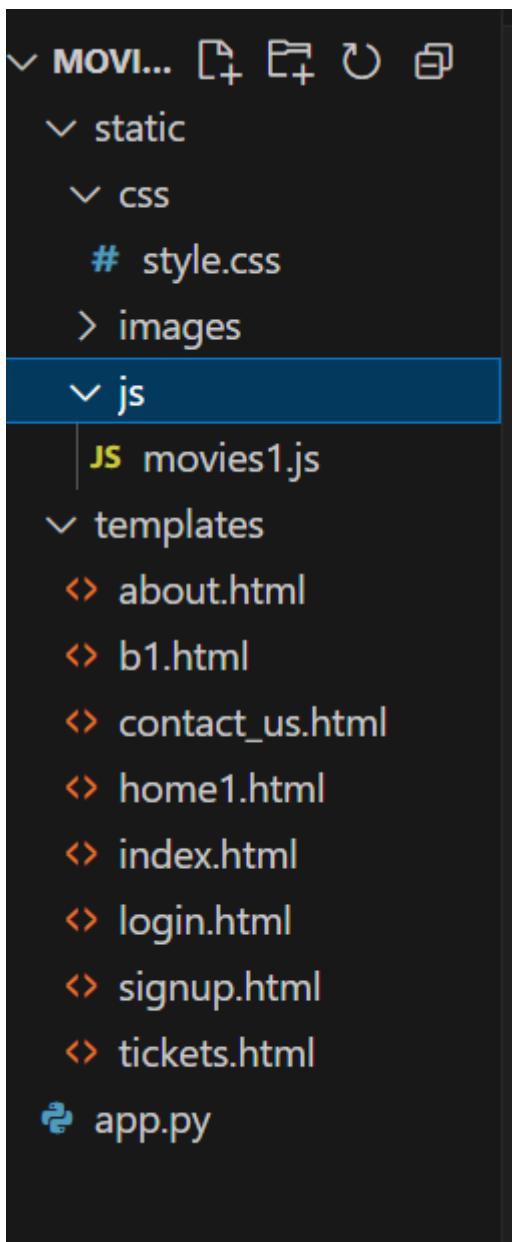
- **Conduct functional testing to verify user registration, login, book requests, and notifications.**
-

Milestone 1 : Web Application Development And Setup

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

- File Explorer Structure



```
✓ MOVIE... 📄 📝 ⏪ ⏴ 🗃
  ↘ static
    ↘ css
      # style.css
    > images
  ↘ js
    JS movies1.js
  ↘ templates
    <> about.html
    <> b1.html
    <> contact_us.html
    <> home1.html
    <> index.html
    <> login.html
    <> signup.html
    <> tickets.html
  🐍 app.py
```

Description of the code :

- **Flask App Initialization**

Imports and Configuration:

```
1  from flask import Flask, render_template, request, redirect, url_for, session, flash
2  from werkzeug.security import generate_password_hash, check_password_hash
3  from datetime import datetime
4  import boto3
5  import uuid
6  import json
7  import os
8  from botocore.exceptions import ClientError
```

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:**

```
11 # Use a static secret key
12 app.secret_key = 'your_static_secret_key_here' # Replace with your own secret string
13
14 # AWS Configuration - read from environment variables for security
15 AWS_REGION = os.environ.get('AWS_REGION', 'ap-south-1')
16
17 # Fix the SNS_TOPIC_ARN assignment - this was the main issue
18 # Instead of using os.environ.get with the ARN as the key, set it directly
19 SNS_TOPIC_ARN = 'arn:aws:sns:ap-south-1:605134430972:MovieTicketNotifications'
20
21 # Initialize AWS services with proper credentials handling
22 # On EC2, this will use the instance profile/role automatically
23 dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION)
24 sns = boto3.client('sns', region_name=AWS_REGION)
25
26 # DynamoDB tables
27 USERS_TABLE_NAME = os.environ.get('USERS_TABLE_NAME', 'MovieMagic_Users')
28 BOOKINGS_TABLE_NAME = os.environ.get('BOOKINGS_TABLE_NAME', 'MovieMagic_Bookings')
29
30 users_table = dynamodb.Table(USERS_TABLE_NAME)
31 bookings_table = dynamodb.Table(BOOKINGS_TABLE_NAME)
32
```

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**

- **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.

```
def send_booking_confirmation(booking):
    """Send booking confirmation email using SNS"""
    # Check if SNS_TOPIC_ARN is set
    if not SNS_TOPIC_ARN:
        print("SNS_TOPIC_ARN is not set. Unable to send notification.")
        return False

    try:
        # Format email content
        email_subject = f"MovieMagic Booking Confirmation - {booking['booking_id']}"

        email_message = f"""
Hello {booking['user_name']},

Your movie ticket booking is confirmed!

Booking Details:
-----
Booking ID: {booking['booking_id']}
Movie: {booking['movie_name']}
Date: {booking['date']}
Time: {booking['time']}
Theater: {booking['theater']}
Location: {booking['address']}
Seats: {booking['seats']}
Amount Paid: ₹{booking['amount_paid']}

Please show this confirmation at the theater to collect your tickets.
    
```

```
Thank you for choosing MovieMagic!
"""

# User email
user_email = booking['booked_by']

print(f"Attempting to send notification to {user_email} via SNS topic {SNS_TOPIC_ARN}")

# Send directly to the email using SNS
response = sns.publish(
    TopicArn=SNS_TOPIC_ARN,
    Subject=email_subject,
    Message=email_message,
    MessageAttributes={
        'email': {
            'DataType': 'String',
            'StringValue': user_email
        }
    }
)

print(f"SNS publish response: {response}")
print(f"Booking confirmation sent to {user_email}")
return True

except Exception as e:
    print(f"Error sending booking confirmation: {str(e)}")
    return False
```

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

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

        try:
            # Check if user already exists
            response = users_table.get_item(Key={'email': email})
            if 'Item' in response:
                flash('Email already registered!', 'danger')
                return redirect(url_for('signup'))

            # Create new user in DynamoDB
            user_id = str(uuid.uuid4())
            users_table.put_item(
                Item={
                    'id': user_id,
                    'name': name,
                    'email': email,
                    'password': password,
                    'created_at': datetime.now().isoformat()
                }
            )

            flash('Registration successful! Please login.', 'success')
            return redirect(url_for('login'))
        except ClientError as e:
            print(f"Error accessing DynamoDB: {e.response['Error']['Message']}")
            flash('An error occurred during registration. Please try again.', 'danger')

    return render_template('signup.html')
```

- **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']

    try:
        # Get user from DynamoDB
        response = users_table.get_item(Key={'email': email})

        if 'Item' in response:
            user = response['Item']
            if check_password_hash(user['password'], password):
                session['user'] = {
                    'id': user['id'],
                    'name': user['name'],
                    'email': user['email']
                }
                return redirect(url_for('home1'))

            flash('Invalid email or password', 'danger')
        except ClientError as e:
            print(f"Error accessing DynamoDB: {e.response['Error']['Message']} ")
            flash('An error occurred. Please try again later.', 'danger')

    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.

```
0  @app.route('/logout')
1  def logout():
2      session.pop('user', None)
3      flash('You have been logged out!', 'info')
4      return redirect(url_for('index'))
5
6  # Application Routes
7  @app.route('/home1')
8  def home1():
9      if 'user' not in session:
10          return redirect(url_for('login'))
11      return render_template('home1.html')
12
13  @app.route('/about')
14  def about():
15      return render_template('about.html')
16
17  @app.route('/contact_us')
18  def contact():
19      return render_template('contact_us.html')
20
```

- **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.

```
# Booking page route
@app.route('/b1', methods=['GET'], endpoint='b1') # Add explicit endpoint
def booking_page():
    if 'user' not in session:
        return redirect(url_for('login'))

    return render_template('b1.html',
        movie=request.args.get('movie'),
        theater=request.args.get('theater'),
        address=request.args.get('address'),
        price=request.args.get('price')
    )
```

- **Tickets Page Route:**

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.

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

    try:
        # Extract booking details from form
        movie_name = request.form.get('movie')
        booking_date = request.form.get('date')
        show_time = request.form.get('time')
        theater_name = request.form.get('theater')
        theater_address = request.form.get('address')
        selected_seats = request.form.get('seats') # Changed from selected_seats
        amount_paid = request.form.get('amount') # Changed from total_price

        # Generate a unique booking ID
        booking_id = f'MVM-{datetime.now().strftime("%Y%m%d")}-{str(uuid.uuid4())[:8]}'

        # Store booking in DynamoDB
        booking_item = {
            'booking_id': booking_id,
            'movie_name': movie_name,
            'date': booking_date,
            'time': show_time,
            'theater': theater_name,
            'address': theater_address,
            'booked_by': session['user']['email'],
            'user_name': session['user']['name'],
            'seats': selected_seats,
```

```

        'user_name': session['user']['name'],
        'seats': selected_seats,
        'amount_paid': amount_paid,
        'booking_time': datetime.now().isoformat()
    }

bookings_table.put_item(Item=booking_item)

# Send email notification via SNS
notification_sent = send_booking_confirmation(booking_item)
if notification_sent:
    flash('Booking confirmation has been sent to your email!', 'success')

# Pass the booking details to the tickets template
return render_template('tickets.html', booking=booking_item)

except Exception as e:
    print(f"Error processing booking: {str(e)}")
    flash('Error processing booking', 'danger')
    return redirect(url_for('home1'))

```

Application Entry point:

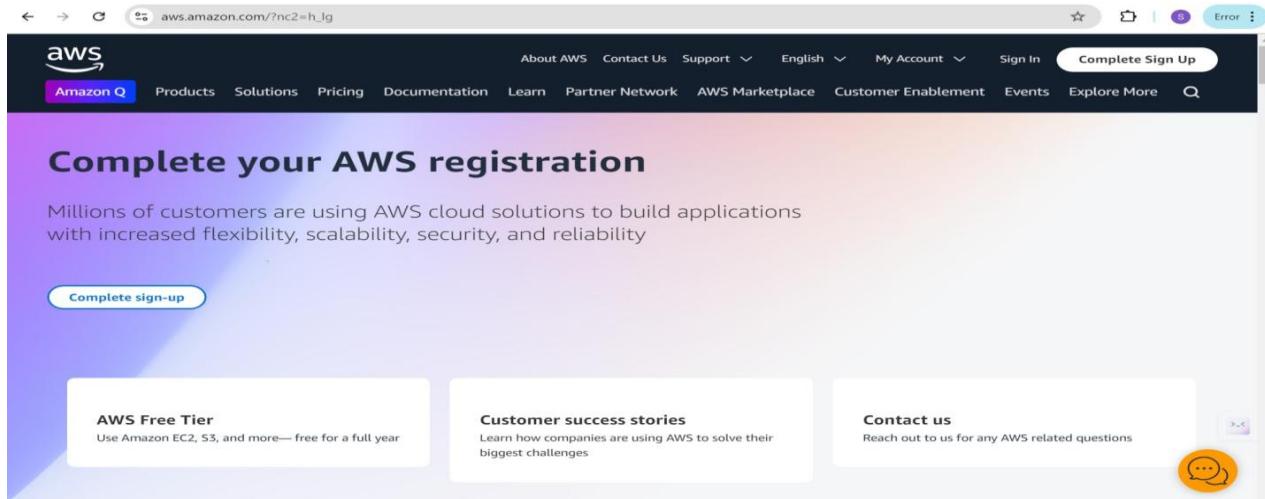
```

if __name__ == '__main__':
    # Using Flask's built-in server as requested
    port = int(os.environ.get('PORT', 5000))
    # You can set debug=False in production
    app.run(host='0.0.0.0', port=port, debug=True)

```

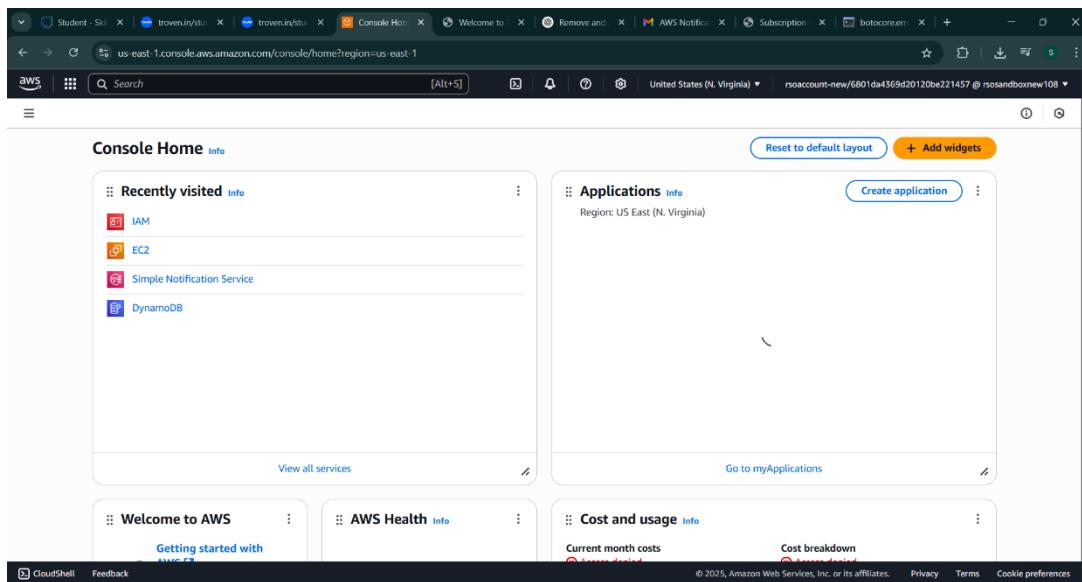
- **Description:** This block starts the Flask application using the built-in development server, setting the host, port, and enabling debug mode for easier development and testing.

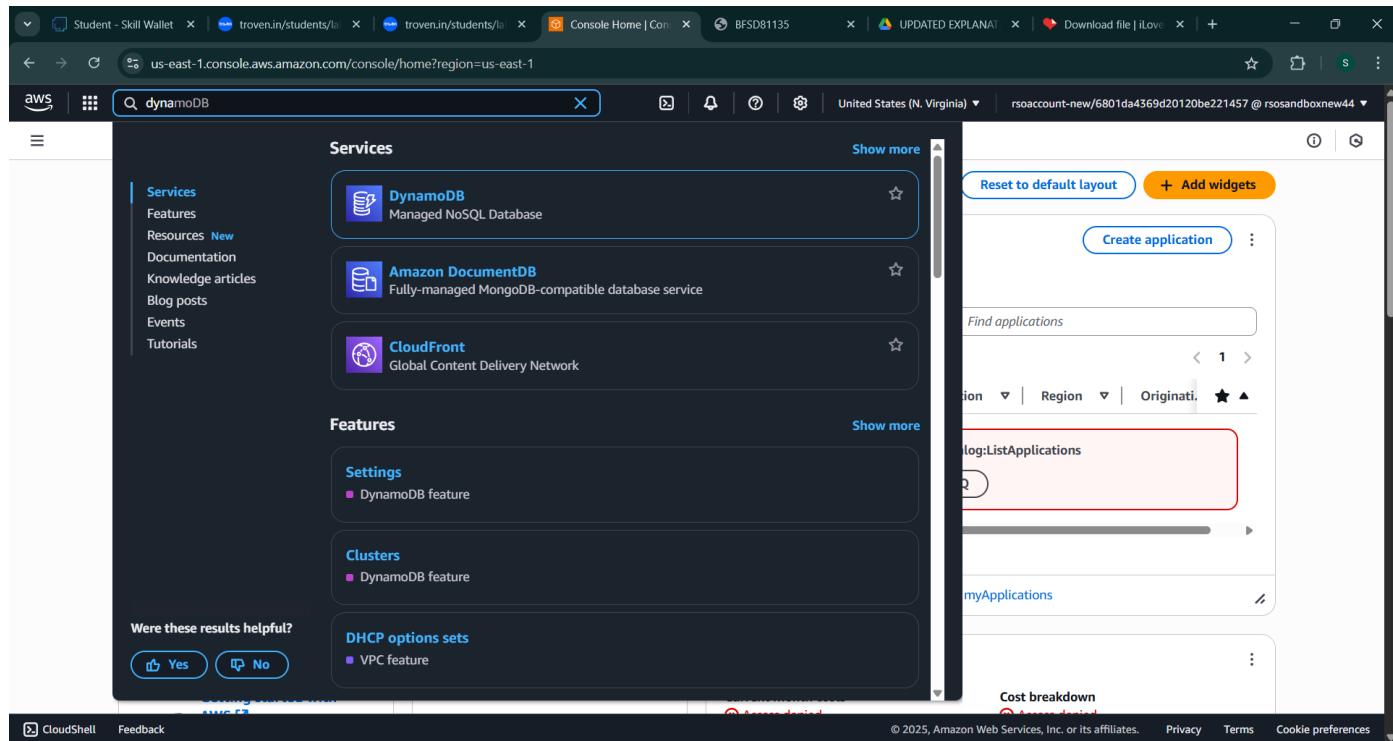
Milestone 2 : AWS Account Setup



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

- After setting up your account, log in to the [AWS Management Console](#).

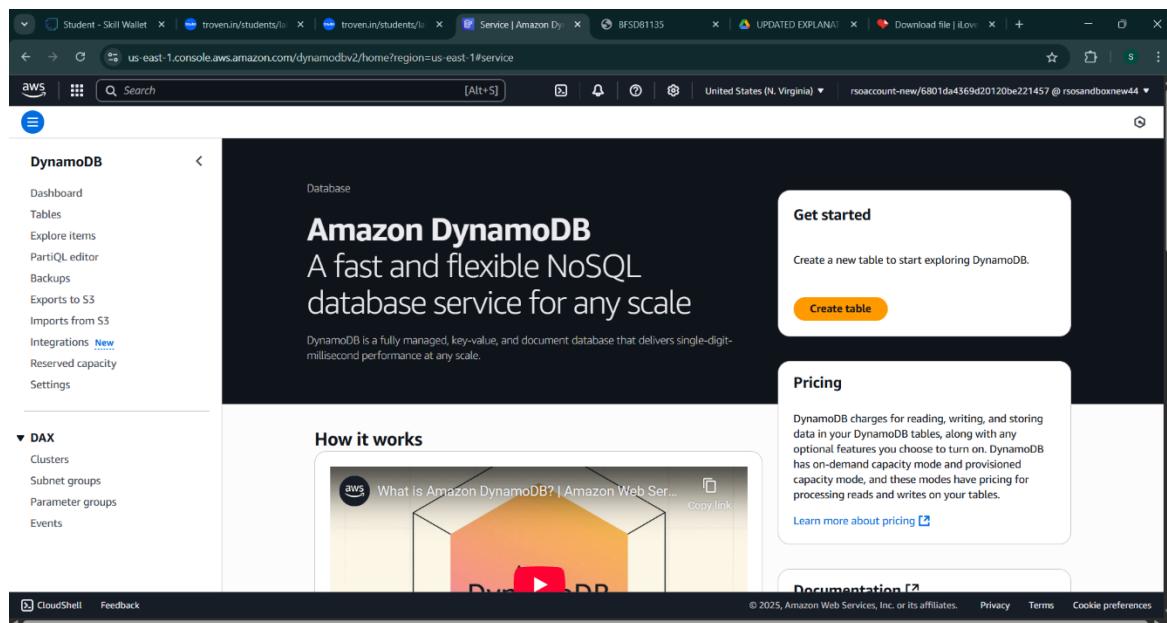




Milestone 3: DynamoDB Database Creation and Setup

- **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.

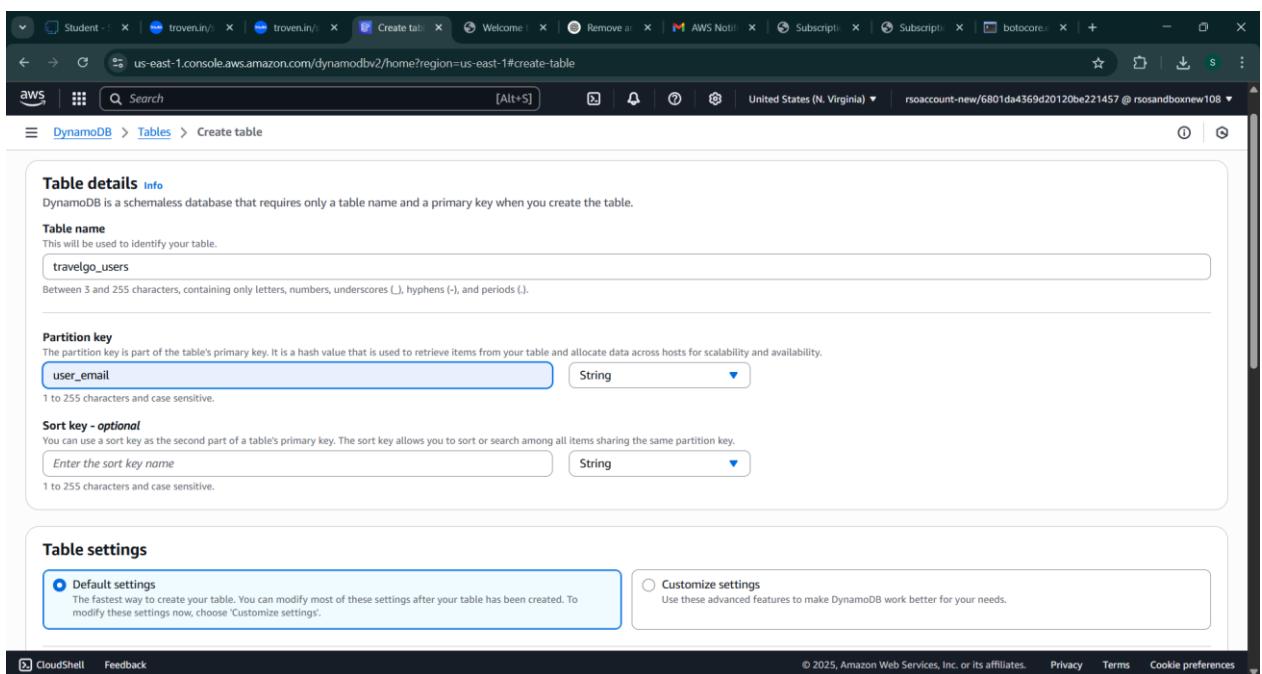




Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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

Student - troven.in/ | troven.in/ | Create tab | Welcome | Remove | AWS Notif | Subscript | Subscript | botcore | + | AWS | Search [Alt+S] | United States (N. Virginia) | rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew108 |

DynamoDB > Tables > 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.

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.
 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.
 String
1 to 255 characters and case sensitive.

Table settings

Default settings
 The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

Customize settings
 Use these advanced features to make DynamoDB work better for your needs.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Student - troven.in/ | troven.in/ | Create tab | Welcome | Remove | AWS Notif | Subscript | Subscript | botcore | + | AWS | Search [Alt+S] | United States (N. Virginia) | rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew108 |

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.

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.
 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.
 String
1 to 255 characters and case sensitive.

Table settings

Default settings
 The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

Customize settings
 Use these advanced features to make DynamoDB work better for your needs.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS DynamoDB 'Create table' wizard.

Table Configuration:

Maximum write capacity units	-	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	AWS owned key	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.

Note: This table will be created with auto scaling deactivated. You do not have permissions to turn on auto scaling.

[Cancel](#) [Create table](#)

Milestone 4: SNS Notification Setup

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

Screenshot of the AWS Simple Notification Service (SNS) console.

The search bar shows 'sns'.

Services:

- 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)

Features:

- Events (ElastiCache feature)
- SMS (AWS End User Messaging feature)
- Hosted zones (Route 53 feature)

Create Topic:

Any tag value: []

Region: []

Action: []

Deletion protection: []

Favorite: []

Read capacity units: []

Write capacity units: []

On-demand: []

Off: []

On-demand: []

Off: []

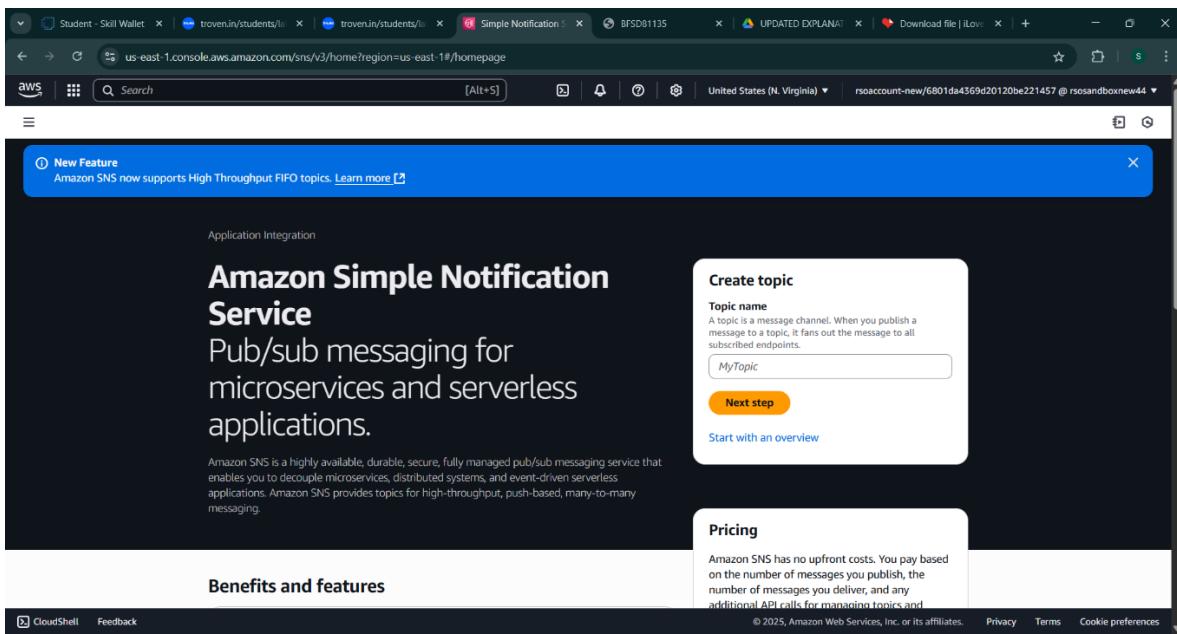
On-demand: []

Off: []

On-demand: []

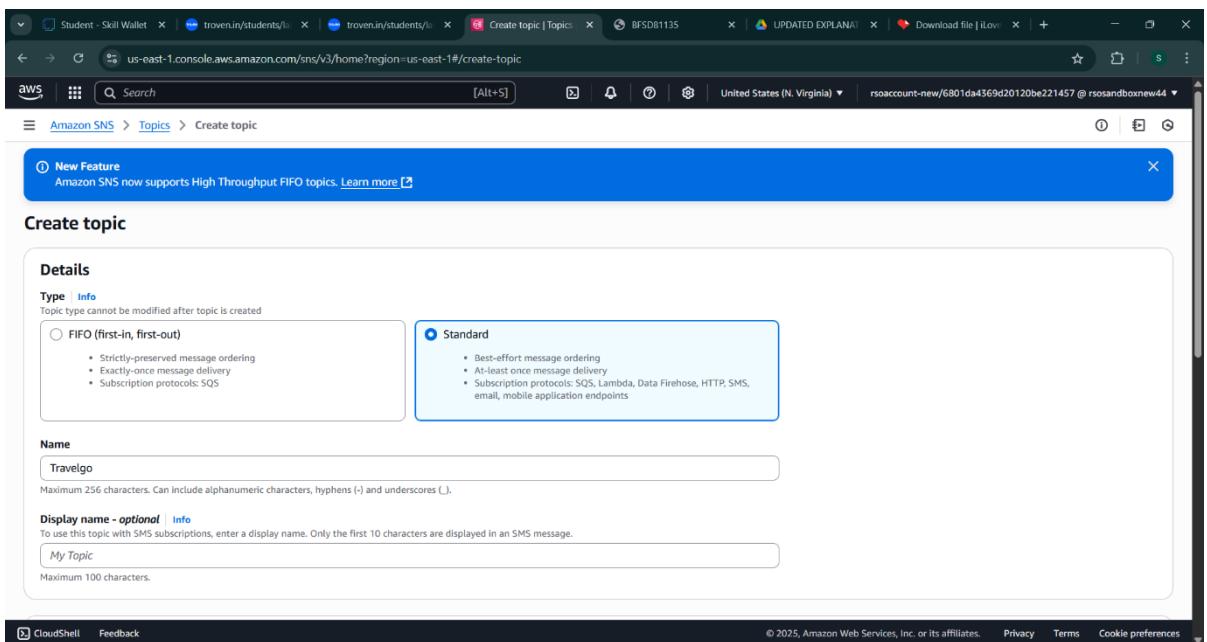
[Actions](#) [Delete](#) [Create topic](#)

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



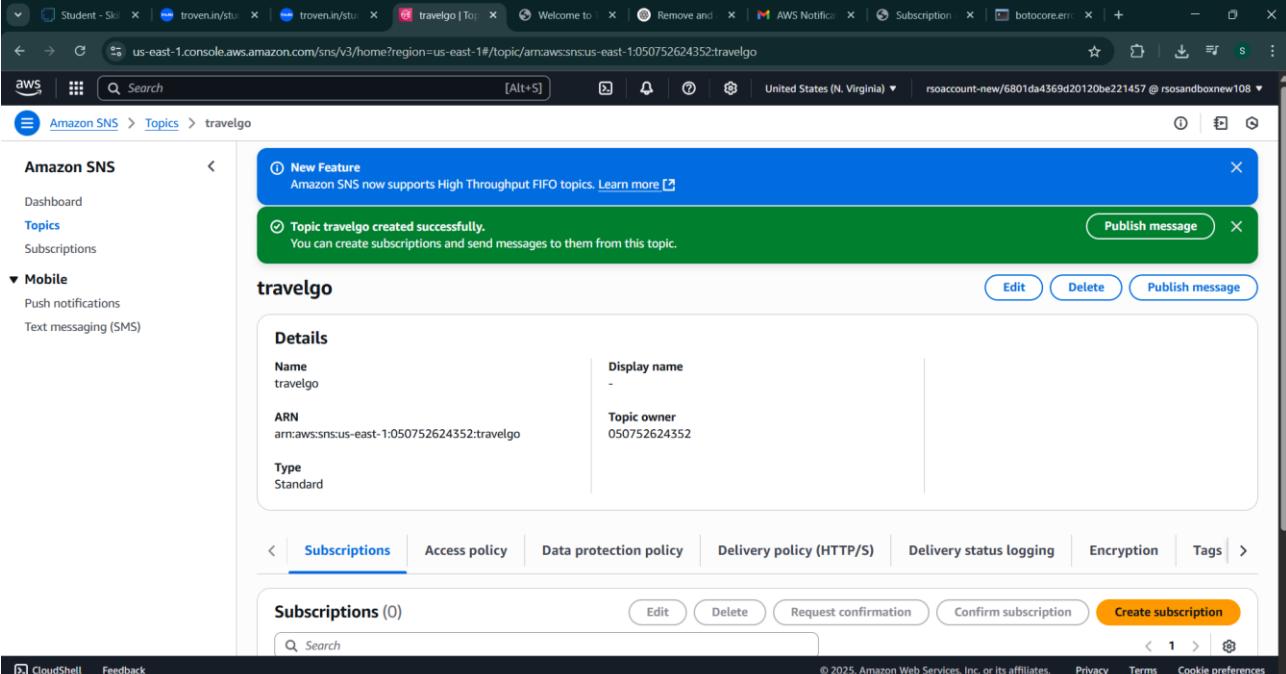
The screenshot shows the AWS Simple Notification Service (SNS) dashboard. At the top, there is a blue banner with the text "Amazon SNS now supports High Throughput FIFO topics. Learn more". Below the banner, the page title is "Amazon Simple Notification Service" with the subtitle "Pub/sub messaging for microservices and serverless applications". To the right, there is a "Create topic" button. A tooltip for "Topic name" explains that it is a message channel where published messages are sent to all subscribed endpoints. The "Next step" button is highlighted in orange. Below the "Create topic" section, there is a "Pricing" section stating that there are no upfront costs based on message volume and delivery. At the bottom of the dashboard, there is a "Benefits and features" section.

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



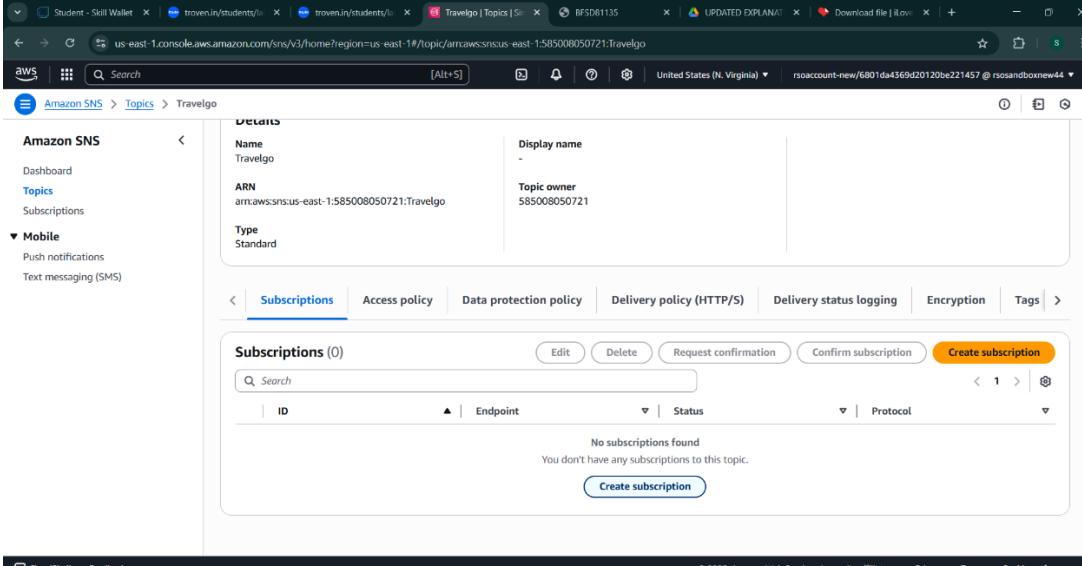
The screenshot shows the "Create topic" wizard in the AWS SNS service. The first step, "Details", is displayed. It includes sections for "Type" (with "Info" link), "Name" (with "Travelgo" entered), and "Display name - optional" (with "My Topic" entered). The "Type" section has two options: "FIFO (first-in, first-out)" and "Standard". The "Standard" type is selected and highlighted with a blue border. Its description includes "Best-effort message ordering", "At-least once message delivery", and "Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints". At the bottom of the "Details" step, there is a "Next Step" button.

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



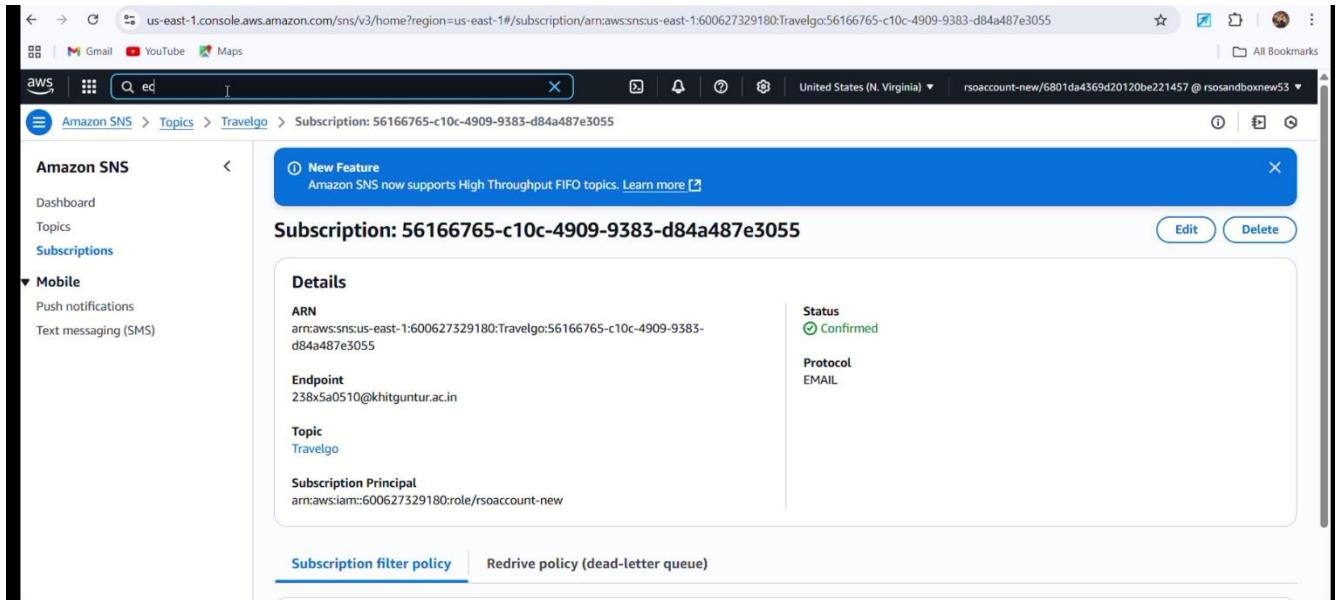
The screenshot shows the AWS SNS Topics page. A success message indicates that the topic 'travelgo' was created successfully. The ARN is listed as arn:aws:sns:us-east-1:050752624352:travelgo. The Subscriptions tab shows 0 subscriptions.

- 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.**



The screenshot shows the AWS SNS Topics page for the 'travelgo' topic. The ARN is listed as arn:aws:sns:us-east-1:585008050721:Travelgo. The Subscriptions tab shows 0 subscriptions.

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

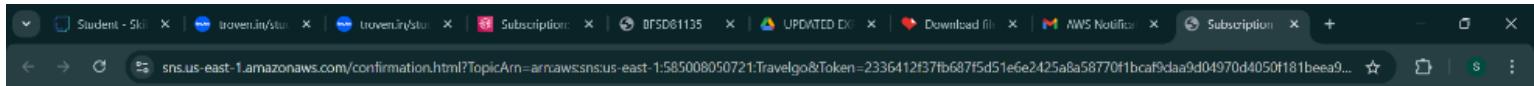


The screenshot shows the AWS SNS console with a subscription details page. The URL in the browser is `us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/subscription/arm:aws:sns:us-east-1:600627329180:Travelgo:56166765-c10c-4909-9383-d84a487e3055`. The page title is "Subscription: 56166765-c10c-4909-9383-d84a487e3055". On the left, there's a sidebar with "Amazon SNS" and "Subscriptions" selected. The main content area shows the following details:

Details	Status
ARN <code>arn:aws:sns:us-east-1:600627329180:Travelgo:56166765-c10c-4909-9383-d84a487e3055</code>	Confirmed
Endpoint <code>238x5a0510@khitguntur.ac.in</code>	Protocol EMAIL
Topic <code>Travelgo</code>	
Subscription Principal <code>arn:aws:iam::600627329180:role/rsoaccount-new</code>	

At the bottom, there are tabs for "Subscription filter policy" (which is selected) and "Redrive policy (dead-letter queue)".

After subscription request for the mail confirmation



Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:
arn:aws:sns:us-east-1:585008050721:travelgo:fcbac1ec-8c7a-401f-8fff-4457a1855207

If it was not your intention to subscribe, [click here to unsubscribe](#).

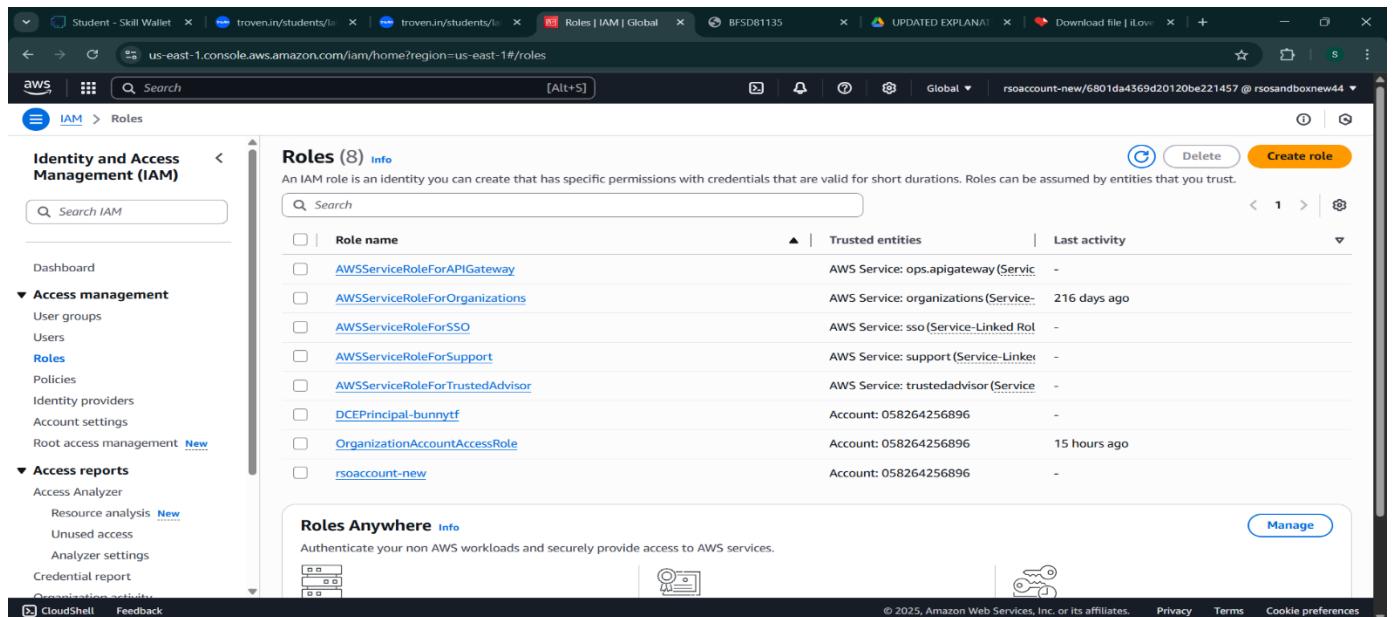
Successfully done with the SNS mail subscription and setup, now store the ARN link

Milestone 5: IAM Role Setup

- **Activity 5.1: Create IAM Role.**

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

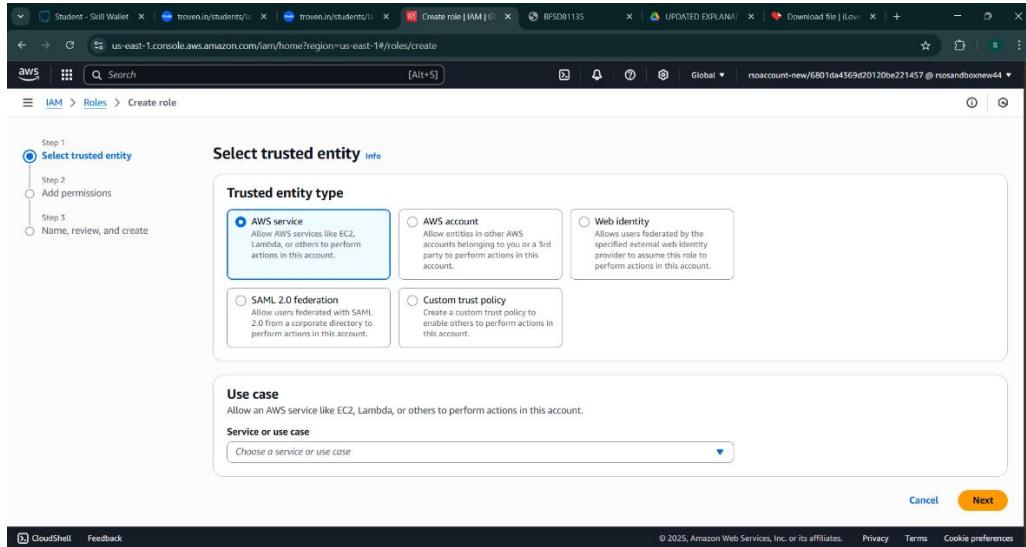
-



The screenshot shows the AWS IAM Roles page with 8 roles listed:

Role name	Trusted entities	Last activity
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service)	-
AWSServiceRoleForOrganizations	AWS Service: organizations (Service)	216 days ago
AWSServiceRoleForSSO	AWS Service: sso (Service-Linked Role)	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service)	-
DCEPrincipal-bunnytf	Account: 058264256896	-
OrganizationAccountAccessRole	Account: 058264256896	15 hours ago
rsoaccount-new	Account: 058264256896	-

Below the table, there is a section titled "Roles Anywhere" with a "Manage" button.



Step 1
 Select trusted entity Info

Step 2
 Add permissions

Step 3
 Name, review, and create

Select trusted entity Info

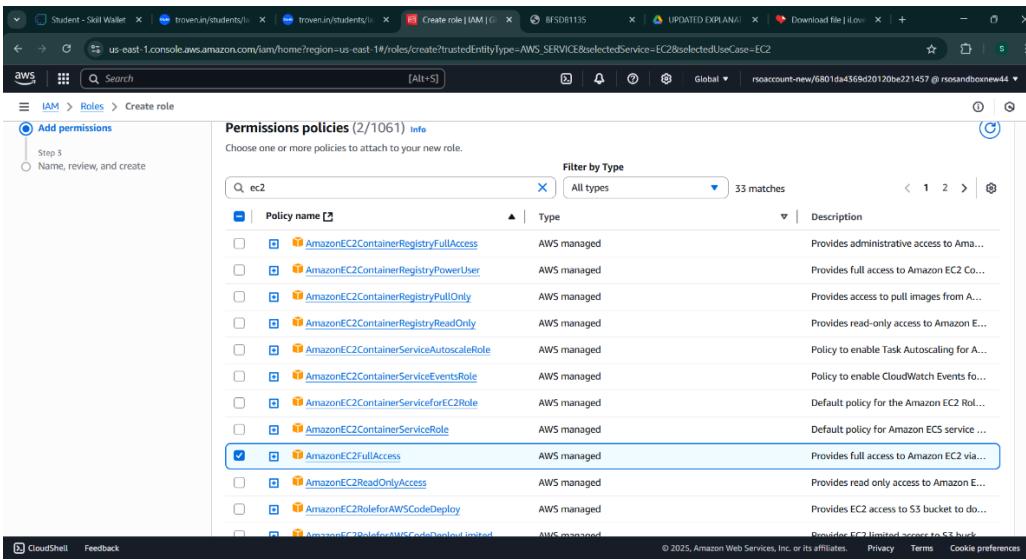
Trusted entity type

- 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**
Allow users authenticated by the specified external web identity provider to assume this role to perform actions in this account.
- 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

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences



Step 1
 Select trusted entity Info

Step 2
 Add permissions

Step 3
 Name, review, and create

Permissions policies (2/1061) Info

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

Filter by Type

Policy name	Type	Description
<input checked="" type="checkbox"/> AmazonEC2ContainerRegistryFullAccess	AWS managed	Provides administrative access to Amazon E...
<input type="checkbox"/> AmazonEC2ContainerRegistryPowerUser	AWS managed	Provides full access to Amazon EC2 Co...
<input type="checkbox"/> AmazonEC2ContainerRegistryPullOnly	AWS managed	Provides access to pull images from A...
<input type="checkbox"/> AmazonEC2ContainerRegistryReadOnly	AWS managed	Provides read-only access to Amazon E...
<input type="checkbox"/> AmazonEC2ContainerServiceAutoscaleRole	AWS managed	Policy to enable Task Autoscaling for A...
<input type="checkbox"/> AmazonEC2ContainerServiceEventsRole	AWS managed	Policy to enable CloudWatch Events fo...
<input type="checkbox"/> AmazonEC2ContainerServiceforEC2Role	AWS managed	Default policy for the Amazon EC2 Rol...
<input type="checkbox"/> AmazonEC2ContainerServiceRole	AWS managed	Default policy for Amazon ECS service ...
<input checked="" type="checkbox"/> AmazonEC2FullAccess	AWS managed	Provides full access to Amazon EC2 via...
<input type="checkbox"/> AmazonEC2ReadOnlyAccess	AWS managed	Provides read only access to Amazon E...
<input type="checkbox"/> AmazonEC2RoleforAWSCodeDeploy	AWS managed	Provides EC2 access to S3 bucket to do...
<input type="checkbox"/> AmazonEC2RoleforAWSCodeDeployInitiated	AWS managed	Provides EC2 limited access to S3 buck...

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

● 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.

Screenshot of the AWS IAM 'Create role' wizard Step 2: Add permissions.

The search bar shows 'dyna'. The results list several AWS managed policies:

- AmazonDynamoDBFullAccess**: Provides full access to Amazon DynamoDB.
- AmazonDynamoDBFullAccess_v2**: Provides full access to Amazon DynamoDB.
- AmazonDynamoDBFullAccesswithDataPipeline**: This policy is on a deprecation path. See...
- AmazonDynamoDBReadOnlyAccess**: Provides read only access to Amazon DynamoDB.
- AWSLambdaDynamoDBExecutionRole**: Provides list and read access to DynamoDB.
- AWSLambdaInvocation-DynamoDB**: Provides read access to DynamoDB Stream.

Buttons at the bottom: Cancel, Previous, Next.

Screenshot of the AWS IAM 'Create role' wizard Step 2: Add permissions.

The search bar shows 'sns'. The results list several AWS managed policies:

- AmazonSNSFullAccess**: Provides full access to Amazon SNS via...
- AmazonSNSReadOnlyAccess**: Provides read only access to Amazon SNS.
- AmazonSNSRole**: Default policy for Amazon SNS service...
- AWSElasticBeanstalkRoleSNS**: (Elastic Beanstalk operations role) Allo...
- AWSIoTDeviceDefenderPublishFindingsToSN...**: Provides messages publish access to S...

Buttons at the bottom: Cancel, Previous, Next.

Milestone 6: EC2 Instance Setup

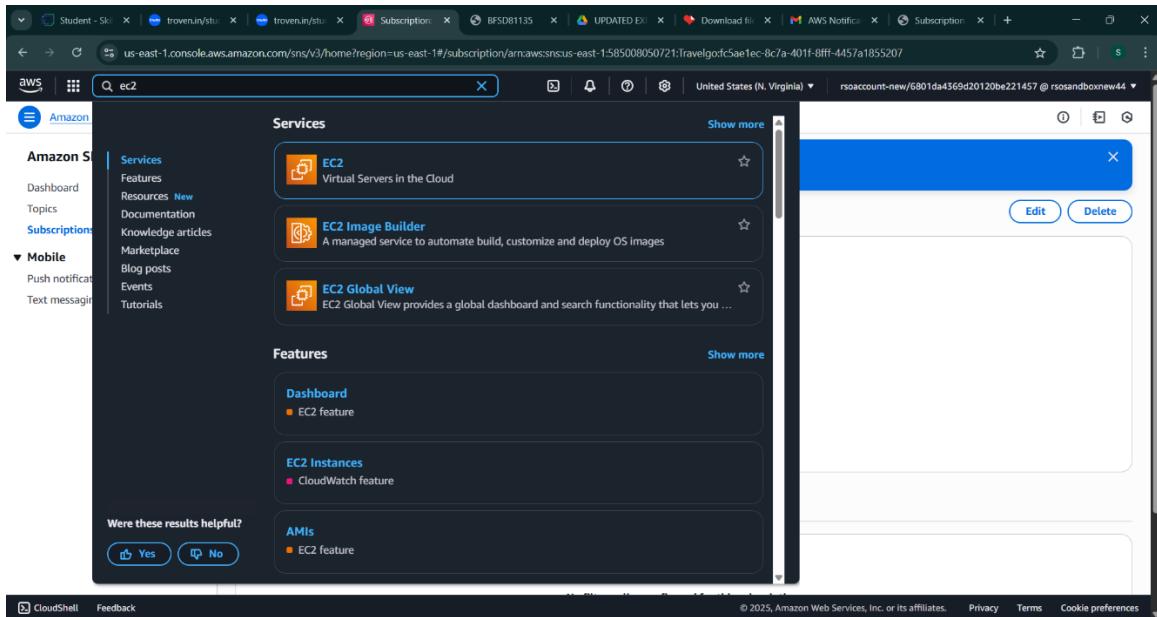
- Note: Load your Flask app and Html files into GitHub repository.

 static	Added full project files
 templates	app.py changed
 venv	Added full project files
 venv_new	Added full project files
 README.md	first commit
 app.py	Update app.py

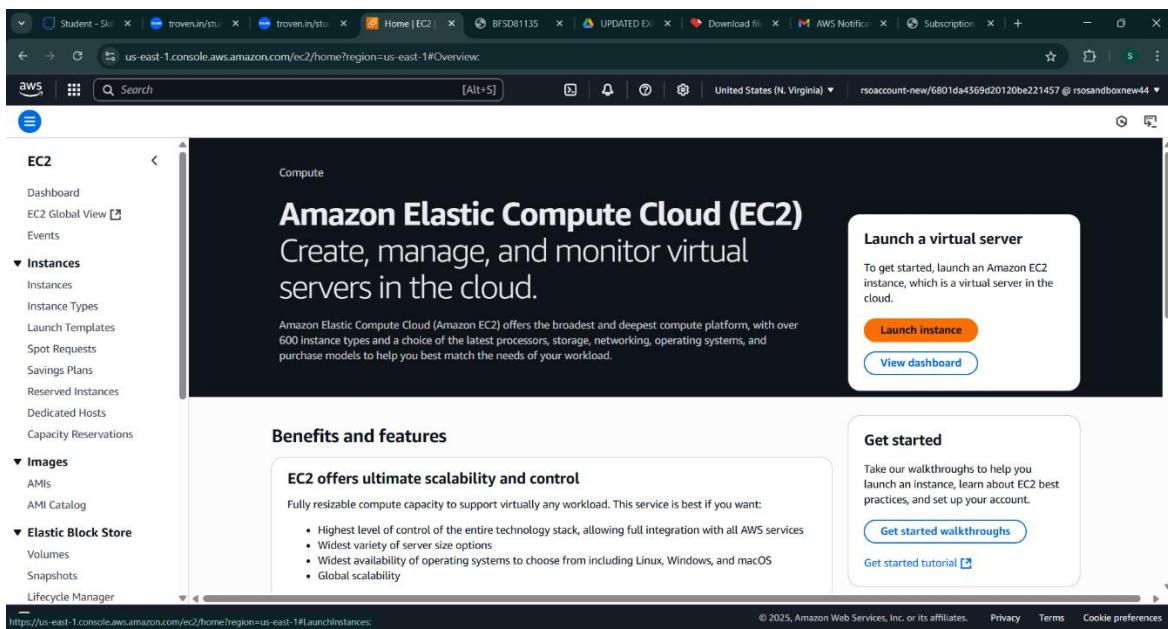
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- **Launch EC2 Instance**

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



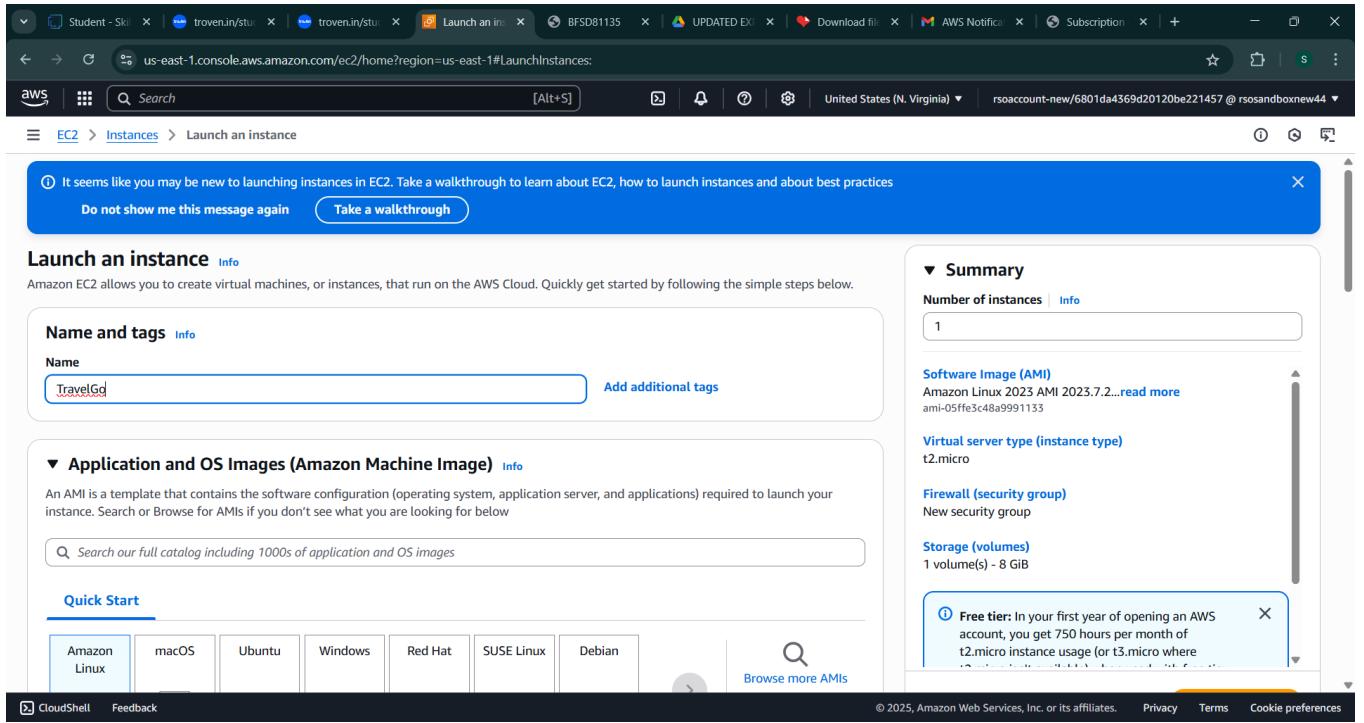
The screenshot shows the AWS search results for the query "ec2". The results are categorized under "Services" and "Features". The "Services" section includes cards for EC2, EC2 Image Builder, and EC2 Global View. The "Features" section includes cards for Dashboard, EC2 Instances, and AMIs. A modal window is open on the right side, showing a progress bar and some text, likely related to launching an instance. The left sidebar shows the navigation menu for the AWS console.



The screenshot shows the AWS EC2 Overview page. The left sidebar lists various EC2 services: Dashboard, EC2 Global View, Events, Instances, Images, Elastic Block Store, and Lifecycle Manager. The main content area features a large heading "Amazon Elastic Compute Cloud (EC2)" with the subtext "Create, manage, and monitor virtual servers in the cloud." Below this, there is a callout box with "Launch a virtual server" and "Launch instance" and "View dashboard" buttons. Another callout box on the right is titled "Get started" with "Get started walkthroughs" and "Get started tutorial" buttons. The footer of the page includes standard AWS links like "Privacy", "Terms", and "Cookie preferences".

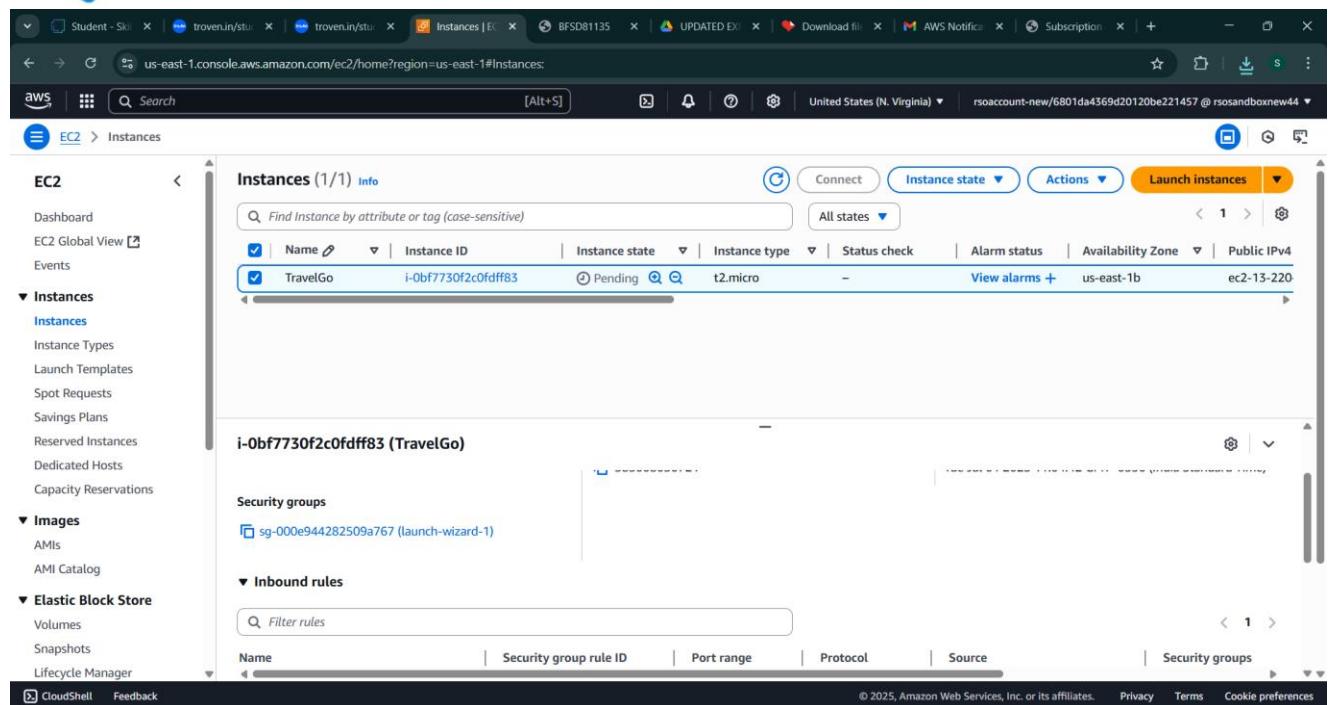
- Click on Launch instance to launch EC2 instance

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



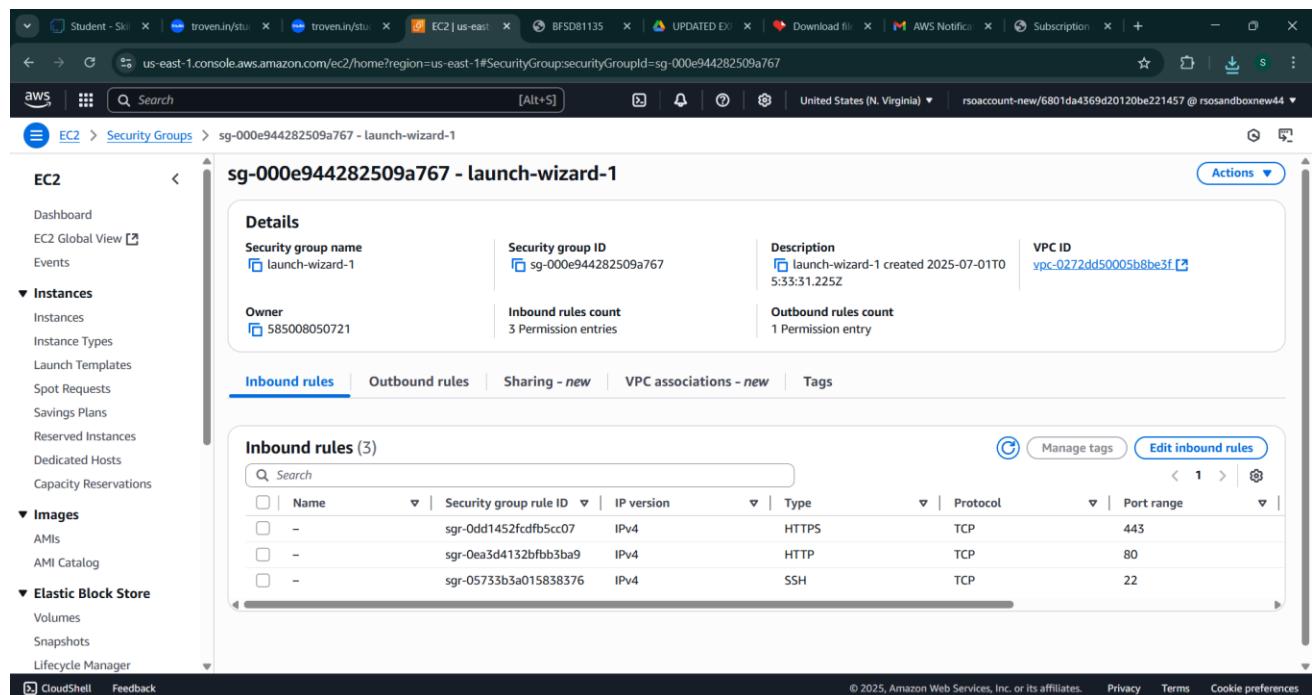
The screenshot shows the AWS EC2 'Launch an instance' wizard. The first step, 'Name and tags', has 'TravelGd' entered in the 'Name' field. The second step, 'Application and OS Images (Amazon Machine Image)', shows 'Amazon Linux' selected from a list. The third step, 'Summary', shows the configuration: 1 instance, Amazon Linux 2023 AMI 2023.7.2..., t2.micro instance type, New security group, and 1 volume(s) - 8 GiB. A note about the free tier is visible on the right.

- Create and download the key pair for Server access.



The screenshot shows the AWS EC2 Instances page. On the left, there is a navigation sidebar for EC2, including sections for Dashboard, EC2 Global View, Events, Instances (with sub-options like Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes, Snapshots, Lifecycle Manager). The main content area displays the 'Instances (1/1) Info' section. A table lists one instance: TravelGo (i-0bf7730f2c0fdff83). The instance is in a Pending state, t2.micro type, and is associated with the sg-000e944282509a767 security group. Below the table, there are tabs for Security groups and Inbound rules.

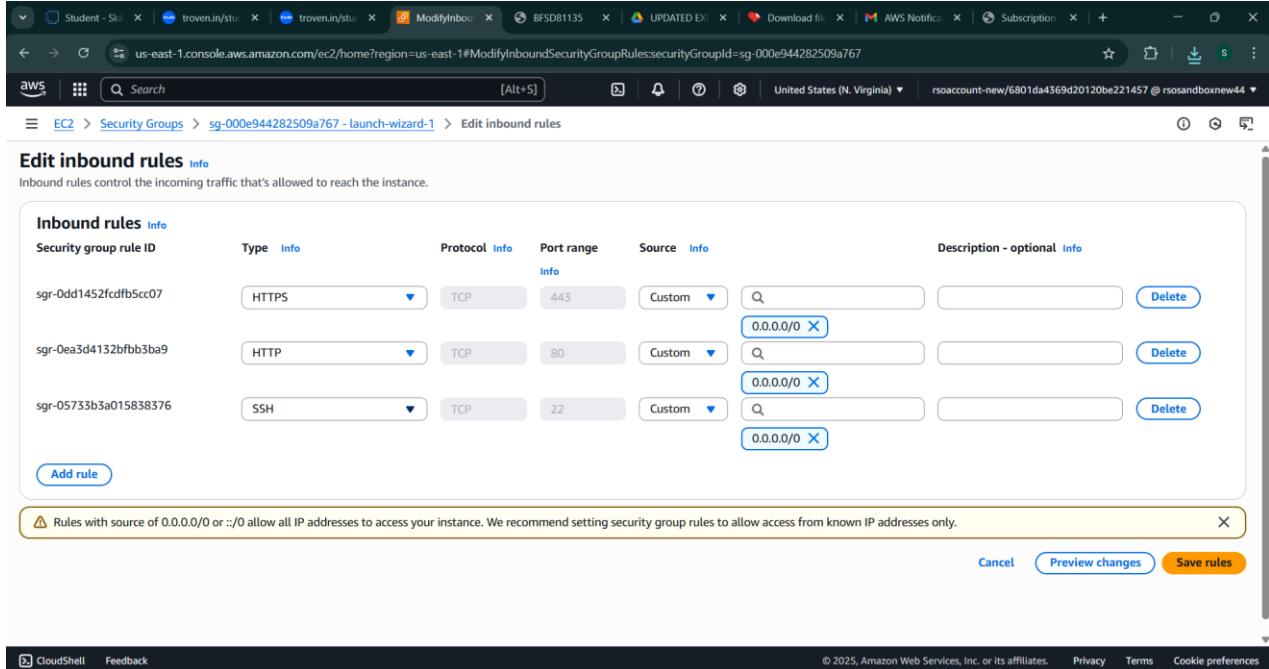
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**



The screenshot shows the AWS Security Groups page. The left sidebar is identical to the one in the previous screenshot. The main content area shows the details for the security group sg-000e944282509a767, which is associated with the launch-wizard-1 instance. The 'Inbound rules' tab is selected, displaying three rules:

Name	Security group rule ID	Type	Protocol	Port range
-	sgr-0dd1452fcfb5cc07	HTTPS	TCP	443
-	sgr-0ea3d4132bfbb3ba9	HTTP	TCP	80
-	sgr-05733b3a015838376	SSH	TCP	22

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



Edit inbound rules Info

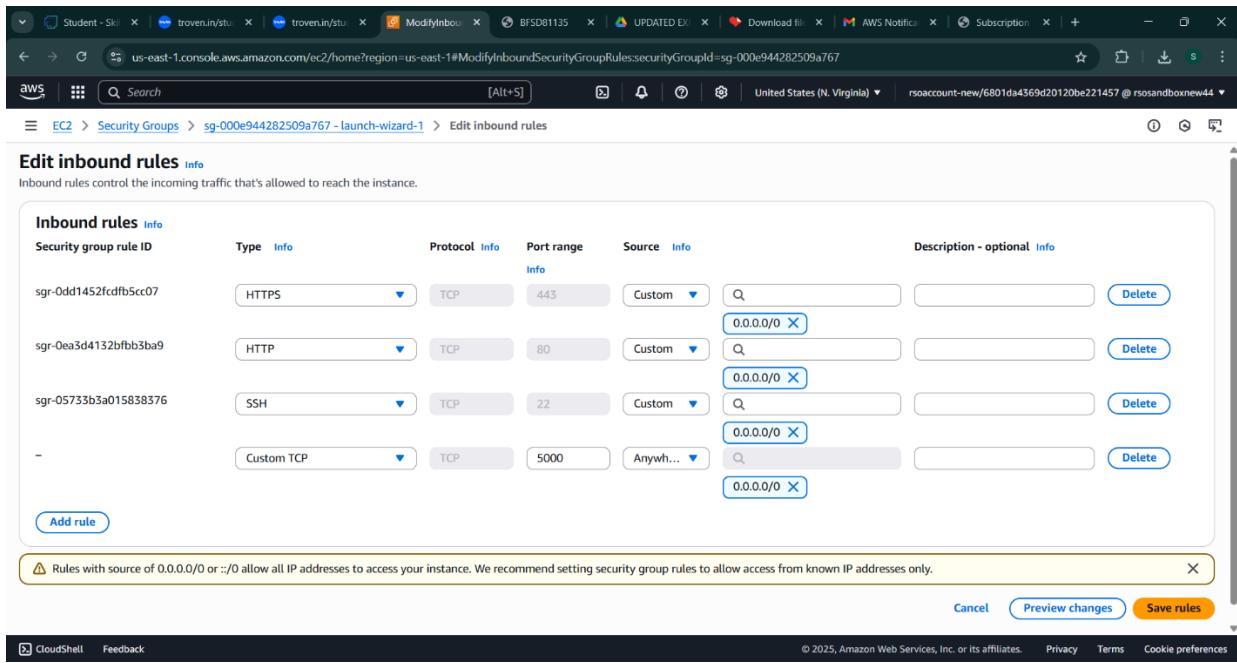
Inbound rules control the incoming traffic that's allowed to reach the instance.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdfb5cc07	HTTPS	TCP	443	Custom	0.0.0.0/0
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	0.0.0.0/0
sgr-05733b3a015838376	SSH	TCP	22	Custom	0.0.0.0/0

Add rule

⚠ Rules with source of 0.0.0.0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel Preview changes Save rules



Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

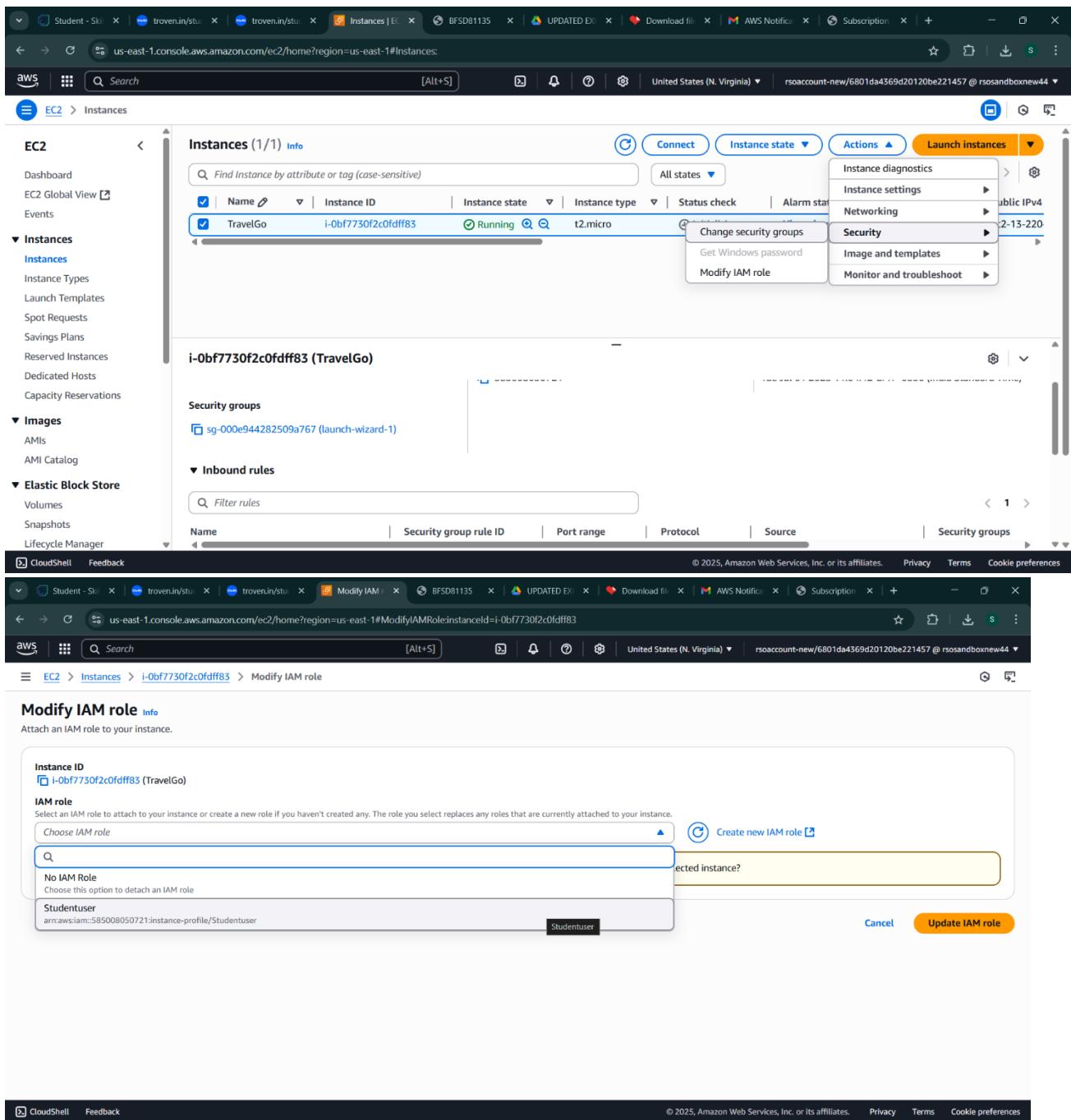
Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdfb5cc07	HTTPS	TCP	443	Custom	0.0.0.0/0
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	0.0.0.0/0
sgr-05733b3a015838376	SSH	TCP	22	Custom	0.0.0.0/0
-	Custom TCP	TCP	5000	Anywh...	0.0.0.0/0

Add rule

⚠ Rules with source of 0.0.0.0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

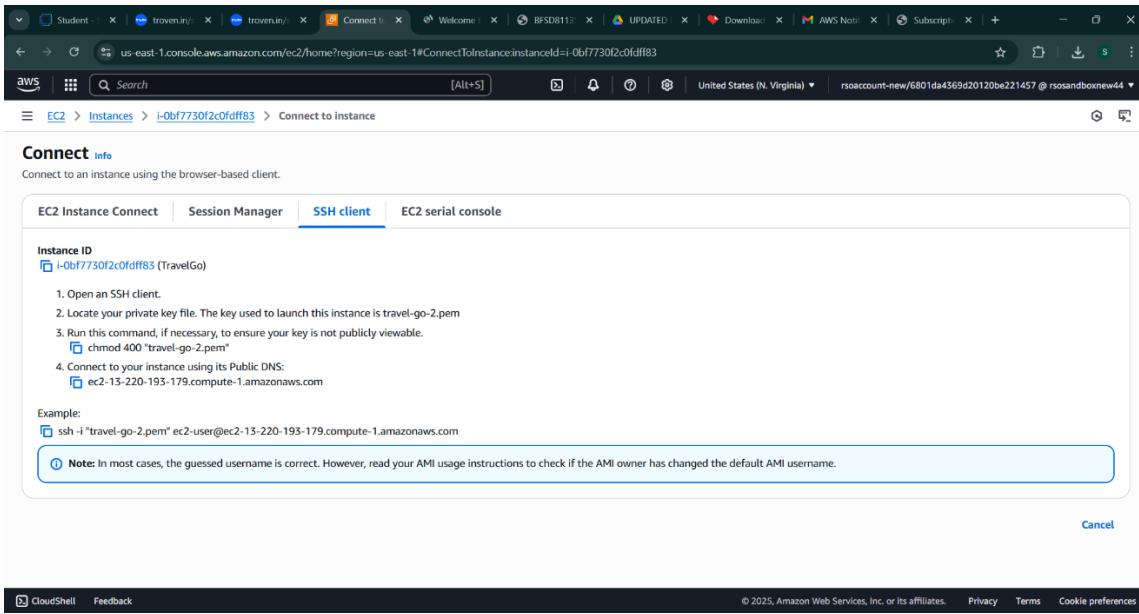
Cancel Preview changes Save rules

- 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.

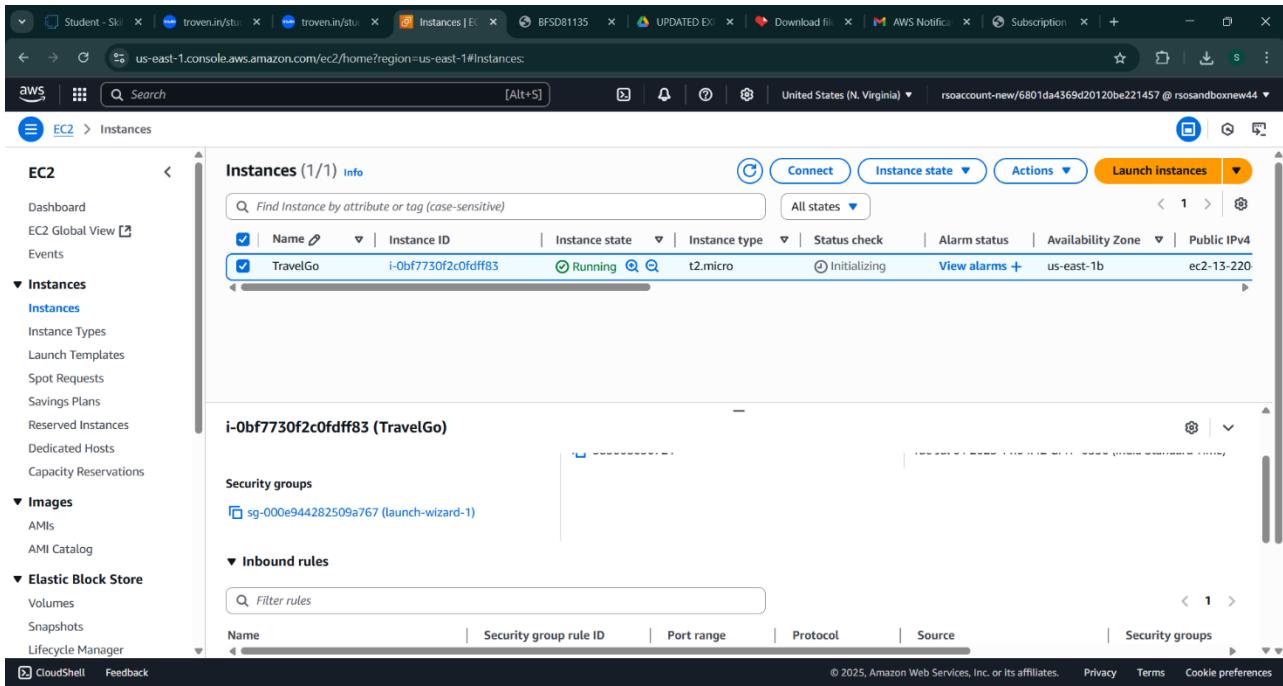


The screenshot shows two overlapping browser windows. The top window is the AWS Management Console - EC2 Instances page, displaying a single instance named 'TravelGo' (i-0bf7730f2c0fdff83) which is 'Running'. The bottom window is a 'Modify IAM role' dialog for the same instance. In the Instances page, the 'Actions' menu for the instance has been opened, showing options like 'Change security groups', 'Get Windows password', and 'Modify IAM role'. The 'Modify IAM role' dialog shows the 'Instance ID' as 'i-0bf7730f2c0fdff83 (TravelGo)'. Under the 'IAM role' section, there is a dropdown menu with 'Choose IAM role' and a search bar, and a link to 'Create new IAM role'. A note says 'Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.' Below this, there is a 'No IAM Role' section with a link to 'Choose this option to detach an IAM role'. A dropdown menu shows 'Studentuser' selected. At the bottom right of the dialog are 'Cancel' and 'Update IAM role' buttons.

- Now connect the EC2 with the files



The screenshot shows the AWS EC2 Connect interface. At the top, it says "EC2 > Instances > i-0bf7730f2c0fdff83 > Connect to instance". Below this, there are tabs for "EC2 Instance Connect", "Session Manager", and "SSH client" (which is selected). A note says "Connect to an instance using the browser-based client." Under "Instance ID", it shows "i-0bf7730f2c0fdff83 (TravelGo)". It lists steps: 1. Open an SSH client, 2. Locate your private key file, 3. Run this command if necessary, 4. Connect to your instance using its Public DNS. It also shows an example command: "ssh -i "travel-go-2.pem" ec2-user@ec2-13-220-193-179.compute-1.amazonaws.com". A note at the bottom says "Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username." A "Cancel" button is at the bottom right.



The screenshot shows the AWS EC2 Instances page. The left sidebar shows navigation options like Dashboard, EC2 Global View, Events, Instances, Images, and Elastic Block Store. The main area displays "Instances (1/1) Info" for instance "i-0bf7730f2c0fdff83 (TravelGo)". The instance details table includes columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4. The instance is listed as "Running" (t2.micro). Below the table, sections for "Security groups" (showing "sg-000e944282509a767 (launch-wizard-1)") and "Inbound rules" are visible. A "CloudShell" and "Feedback" link are at the bottom.

Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

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

Verify Installations:

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

Activity 7.2: Clone Your Flask Project from GitHub

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

Run: 'git clone <https://github.com/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/AlekhyaPenubakula/InstantLibrary.git'

- 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, 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-172-31-26-55 Travel-go]$ 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 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/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-172-31-26-55 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 deployment. 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
```

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.

THANK YOU