

# TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using AWS

## Project Description:

**TravelGo** is a full-stack, cloud-based travel booking platform designed to simplify the process of reserving buses, trains, flights, and hotels through a unified interface. Built using Flask as the backend framework, the application is deployed on Amazon EC2 and leverages DynamoDB for efficient storage of user data and bookings. TravelGo allows users to register, log in, search for transportation and accommodation options, and book their travel with ease. Once a booking is confirmed or cancelled, users receive real-time email notifications powered by AWS Simple Notification Service (SNS), keeping them informed throughout their journey.

The platform's user-friendly interface supports dynamic seat selection for buses, hotel filtering based on preferences such as luxury or budget, and provides booking summaries along with centralized cancellation management. By combining cloud scalability, responsive design, and secure session handling, TravelGo delivers a seamless and real-time travel planning experience for users.

## Scenario 1: Hassle-Free Multi-Mode Travel Booking Experience

**TravelGo** offers users a unified platform to search and book buses, trains, flights, and hotels all in one place. For instance, a user planning a trip from Hyderabad to Bangalore can log in, select their preferred mode of transport, choose from available options, and proceed to booking. Flask manages the backend operations such as retrieving travel listings and processing user input in real-time. Hosted on AWS EC2, the platform remains responsive even during high-traffic hours like weekends or holiday seasons, allowing multiple users to browse and book without delay.

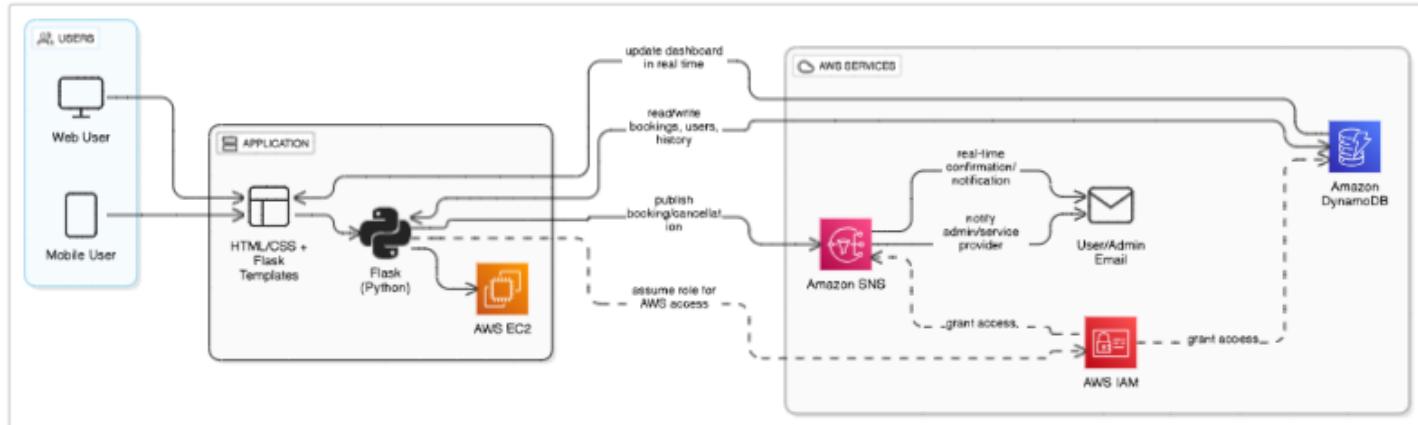
## Scenario 2: Real-Time Booking Confirmation with AWS SNS

Once a booking is made—whether it's a train ticket or a hotel stay—TravelGo uses AWS SNS to instantly notify the user. For example, after a student books a hotel in Chennai, SNS sends a real-time email notification confirming the booking with all the relevant details. This notification is triggered from the Flask backend after the booking is successfully recorded in DynamoDB. Additionally, SNS can alert admin or service providers, ensuring transparency and real-time updates on every transaction.

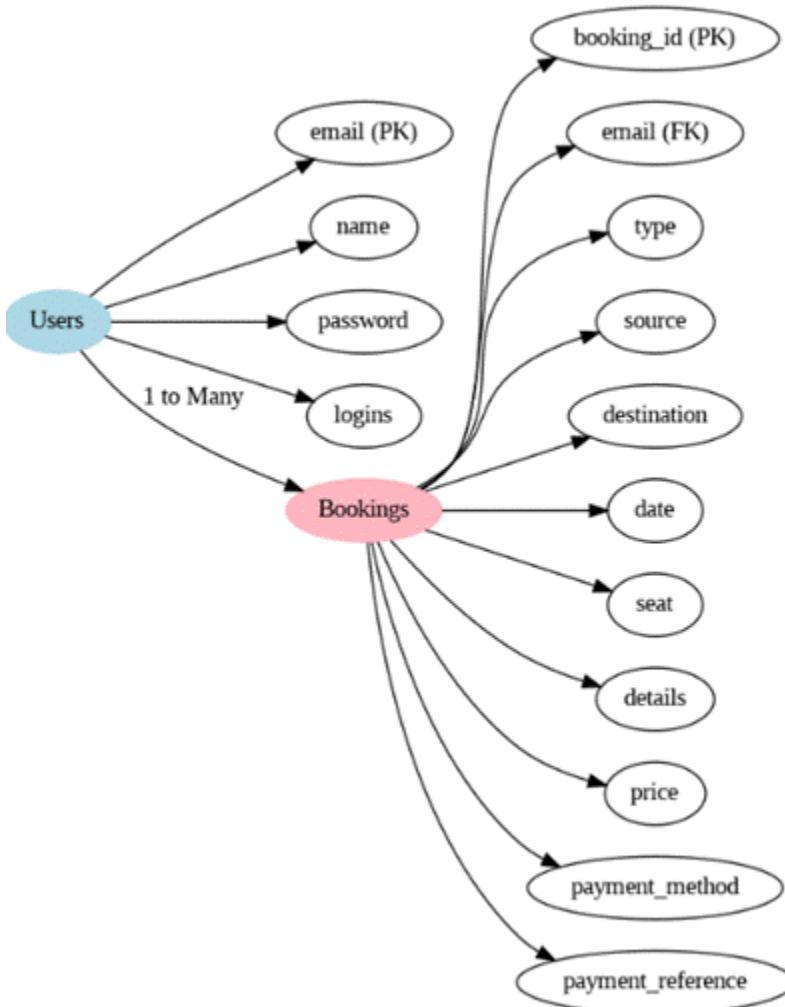
## Scenario 3: Dynamic Dashboard with Personal Travel History

TravelGo features a dynamic user dashboard that displays all past and upcoming bookings for the logged-in user. For example, a user who has booked a flight and a hotel can view these bookings categorized by type, along with dates, price, and cancellation options. Flask fetches this data from AWS DynamoDB, which persistently stores all user bookings. The dashboard UI, powered by responsive HTML/CSS and Flask templates, ensures users can review or manage bookings anytime, from any device, with real-time updates and quick cancellation workflows supported.

## 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

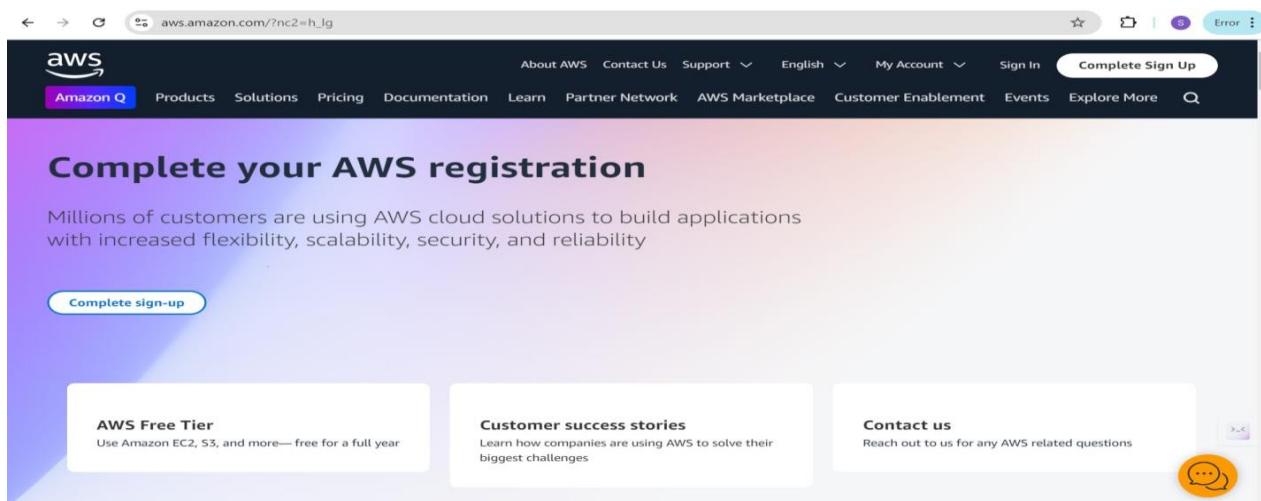
**Activity 7.2:** Run the Flask App

## 8. Testing and Deployment

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

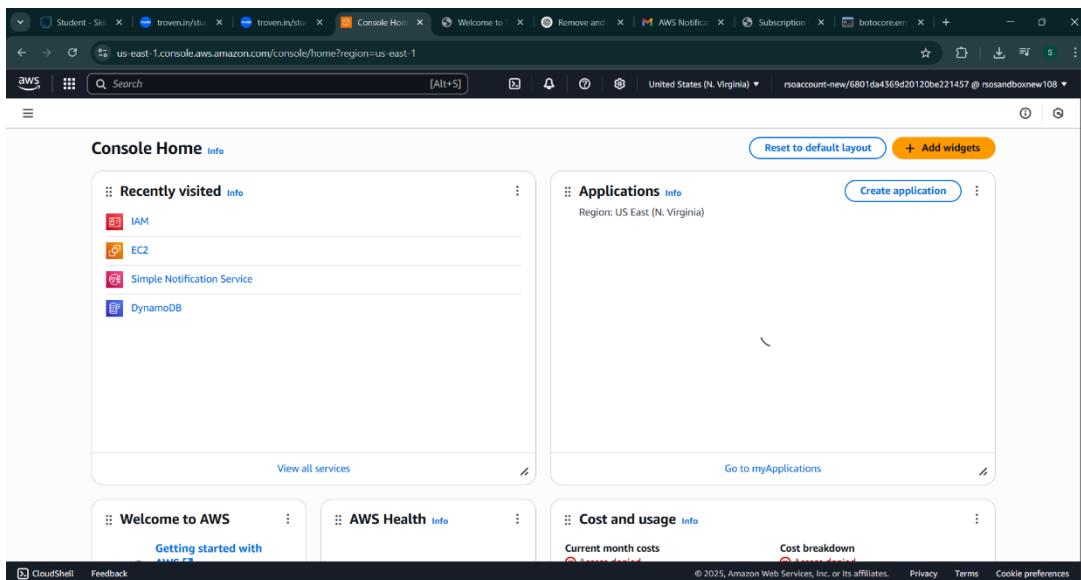
### Milestone 1: AWS Account Setup and Login

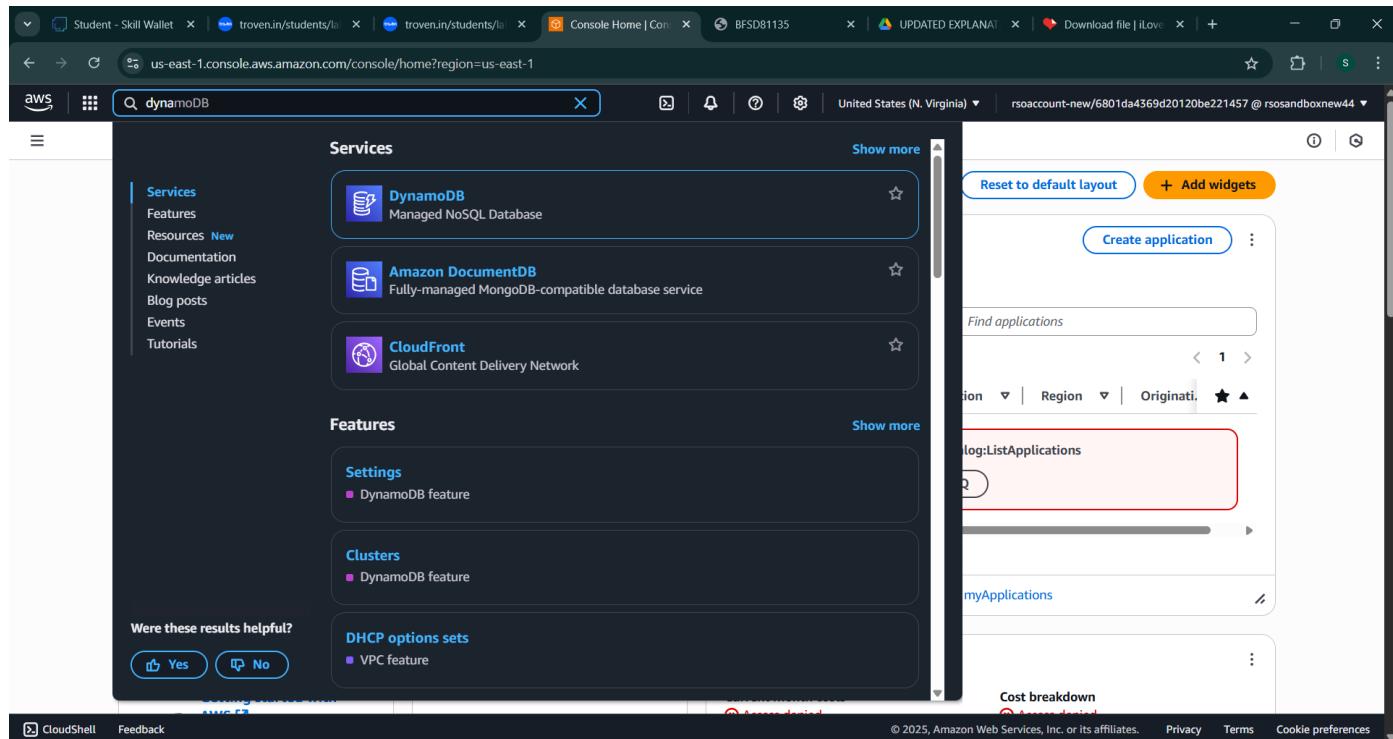
- **Activity 1.1: Set up an AWS account if not already done.**
  - Sign up for an AWS account and configure billing settings.



- **Activity 1.2: Log in to the AWS Management Console**

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



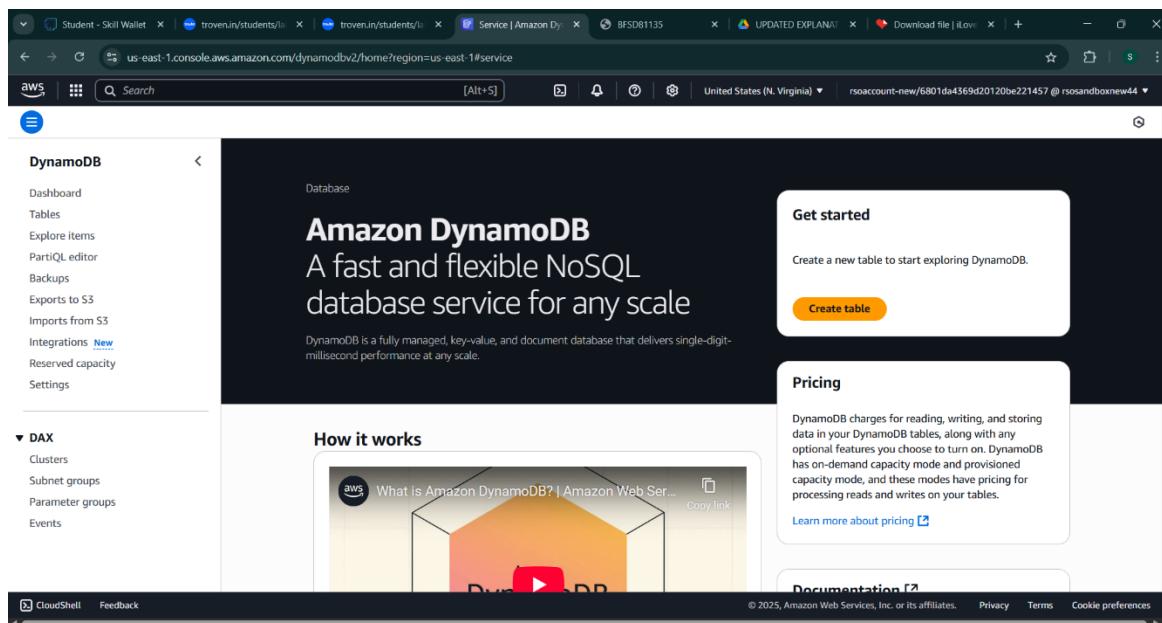


The screenshot shows the AWS Management Console search results for 'dynamoDB'. The top navigation bar includes tabs for 'Student - Skill Wallet', 'troven.in/students/l...', 'troven.in/students/l...', 'Console Home | Con...', 'BFSD81135', 'UPDATED EXPLAIN...', 'Download file | iLove...', and a '+' button. The search bar contains 'dynamoDB'. The left sidebar has a 'Services' section with links to 'Features', 'Resources New', 'Documentation', 'Knowledge articles', 'Blog posts', 'Events', and 'Tutorials'. The main content area displays three services: 'DynamoDB' (Managed NoSQL Database), 'Amazon DocumentDB' (Fully-managed MongoDB-compatible database service), and 'CloudFront' (Global Content Delivery Network). Below these are sections for 'Features' like 'Settings' and 'Clusters', and 'DHCP options sets'. A feedback poll at the bottom asks 'Were these results helpful?' with 'Yes' and 'No' buttons. The right side features a 'Create application' button, a 'Find applications' search bar, and a 'myApplications' section.

## Milestone 2: DynamoDB Database Creation and Setup

- **Activity 2.1: Navigate to the DynamoDB**

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



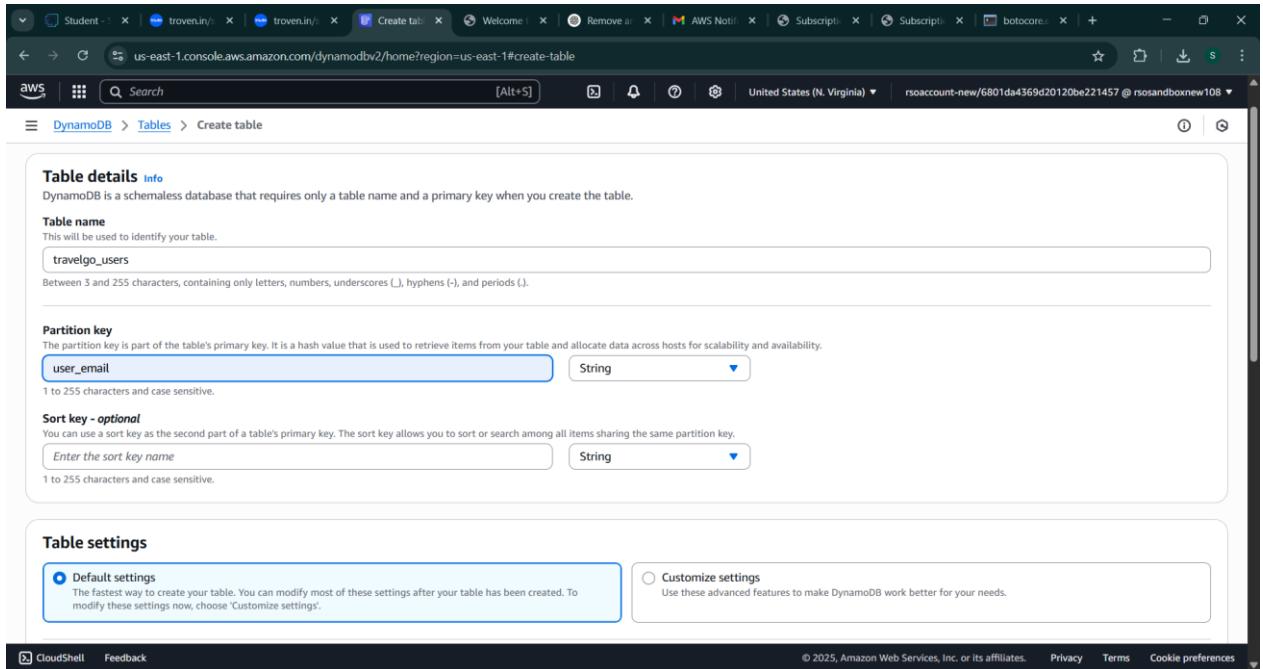
The screenshot shows the 'Amazon DynamoDB' service page. The top navigation bar includes tabs for 'Student - Skill Wallet', 'troven.in/students/l...', 'troven.in/students/l...', 'Service | Amazon Dy...', 'BFSD81135', 'UPDATED EXPLAIN...', 'Download file | iLove...', and a '+' button. The search bar contains 'Search [Alt+S]'. The left sidebar has a 'DynamoDB' section with links to 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations New...', 'Reserved capacity', and 'Settings'. Below this is a 'DAX' section with links to 'Clusters', 'Subnet groups', 'Parameter groups', and 'Events'. The main content area features a 'Database' section with the heading 'Amazon DynamoDB: A fast and flexible NoSQL database service for any scale'. It describes DynamoDB as a fully managed, key-value, and document database. A 'How it works' section includes a video thumbnail titled 'What is Amazon DynamoDB? | Amazon Web Ser...'. To the right, there are 'Get started' (with a 'Create table' button) and 'Pricing' sections. The 'Pricing' section explains that DynamoDB charges for reading, writing, and storing data. The bottom of the page includes a 'Documentation' link, copyright information (© 2025, Amazon Web Services, Inc. or its affiliates.), and links for 'Privacy', 'Terms', and 'Cookie preferences'.

9.  
10.

- **Activity 2.2: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.

**11.**



The screenshot shows the AWS DynamoDB 'Create table' wizard. In the 'Table details' section, the table name is set to 'travelgo\_users'. The 'Partition key' is defined as 'user\_email' of type 'String'. In the 'Table settings' section, the 'Default settings' radio button is selected, indicating the fastest way to create the table. The page includes standard AWS navigation and footer links.



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 - x | troven.in/x | troven.in/x | Create tab | Welcome | Remove | AWS Notif | Subscript | Subscript | botcore/x | + |

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 - x | troven.in/x | troven.in/x | Create tab | Welcome | Remove | AWS Notif | Subscript | Subscript | botcore/x | + |

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 3: SNS Notification Setup

### ● Activity 3.1: Create SNS topics for sending email notifications to users

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

The left sidebar shows the navigation menu for SNS, including Services, Features, Resources, Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials.

The main content area displays the following information:

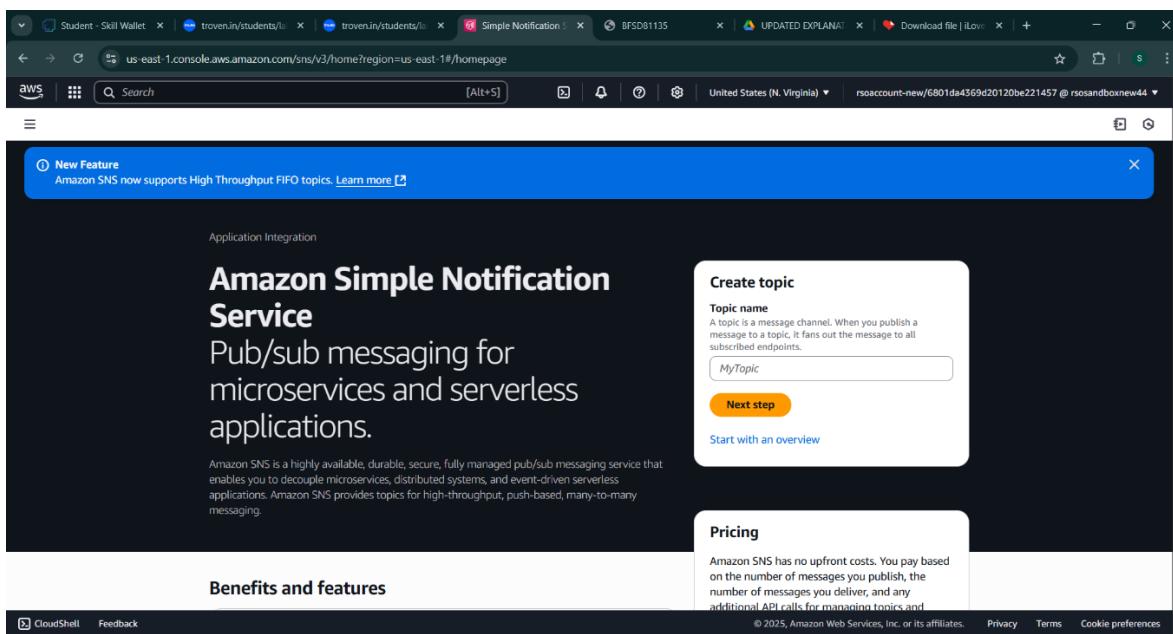
- Services:**
  - Simple Notification Service**: SNS managed message topics for Pub/Sub.
  - Route 53 Resolver**: Resolve DNS queries in your Amazon VPC and on-premises network.
  - Route 53**: Scalable DNS and Domain Name Registration.
- Features:**
  - Events**: ElastiCache feature.
  - SMS**: AWS End User Messaging feature.
  - Hosted zones**: Route 53 feature.

A modal window is open on the right side, showing the configuration for a new SNS topic:

Action	Value	Setting
Region	us-east-1	Region
Topic name	MyNewTopic	Topic name
Subscription Regions	Off	Subscription Regions
Deletion protection	Off	Deletion protection
Favorite	On-demand	Favorite
Read cap	On-demand	Read cap

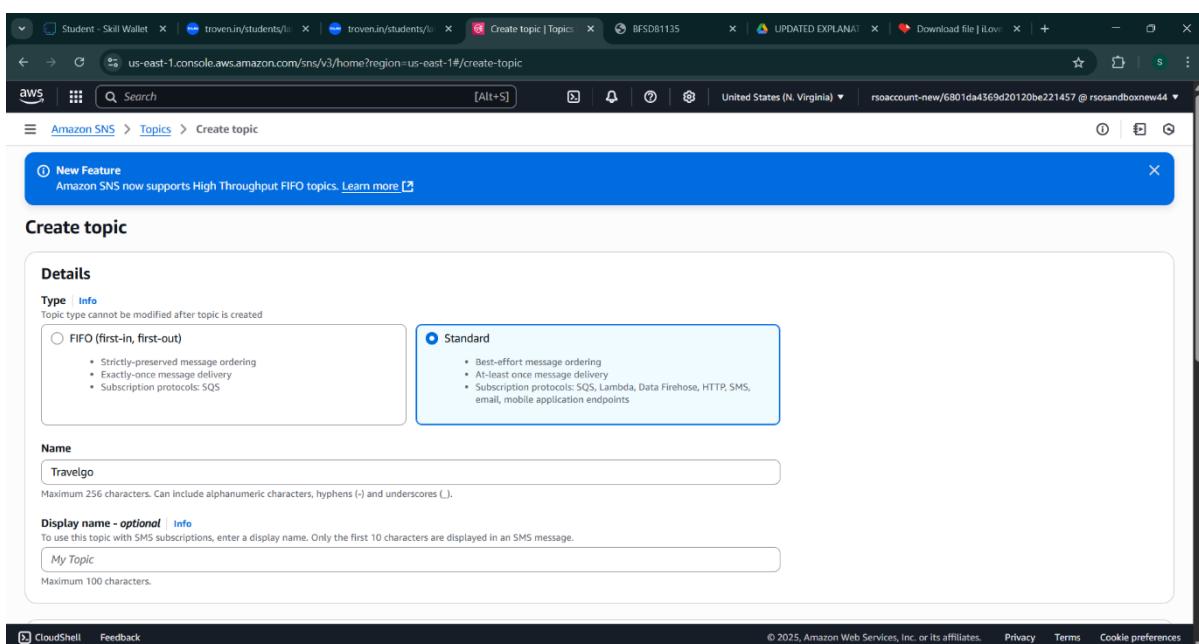
[Actions](#) [Delete](#) [Create topic](#)

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



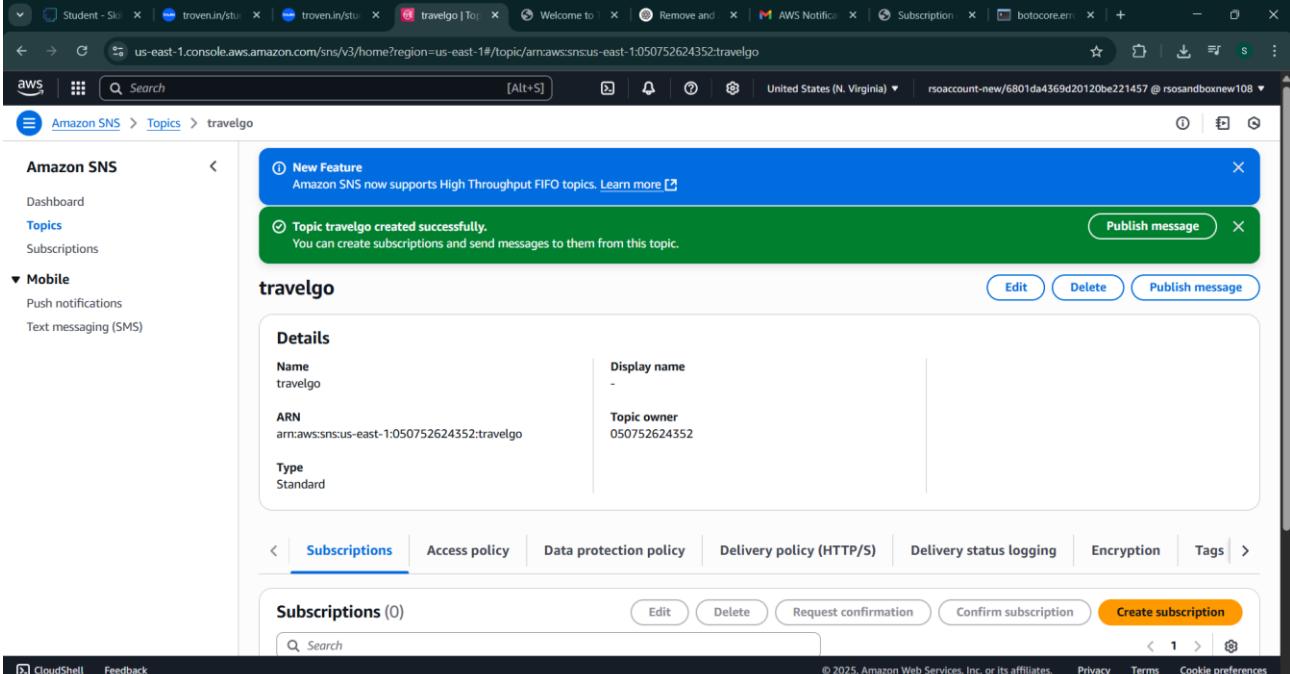
The screenshot shows the AWS SNS dashboard. At the top, there is a blue banner with the text "Amazon SNS now supports High Throughput FIFO topics. Learn more". Below this, the main heading is "Amazon Simple Notification Service" with the subtext "Pub/sub messaging for microservices and serverless applications". A detailed description follows, mentioning its use for decoupling microservices and enabling event-driven serverless applications. To the right, there is a "Create topic" form with a "Topic name" field containing "MyTopic" and a "Next step" button. Another box titled "Pricing" states that there are no upfront costs based on message volume and delivery. At the bottom, there are links for CloudShell, Feedback, and various AWS terms like Privacy, Terms, and Cookie preferences.

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



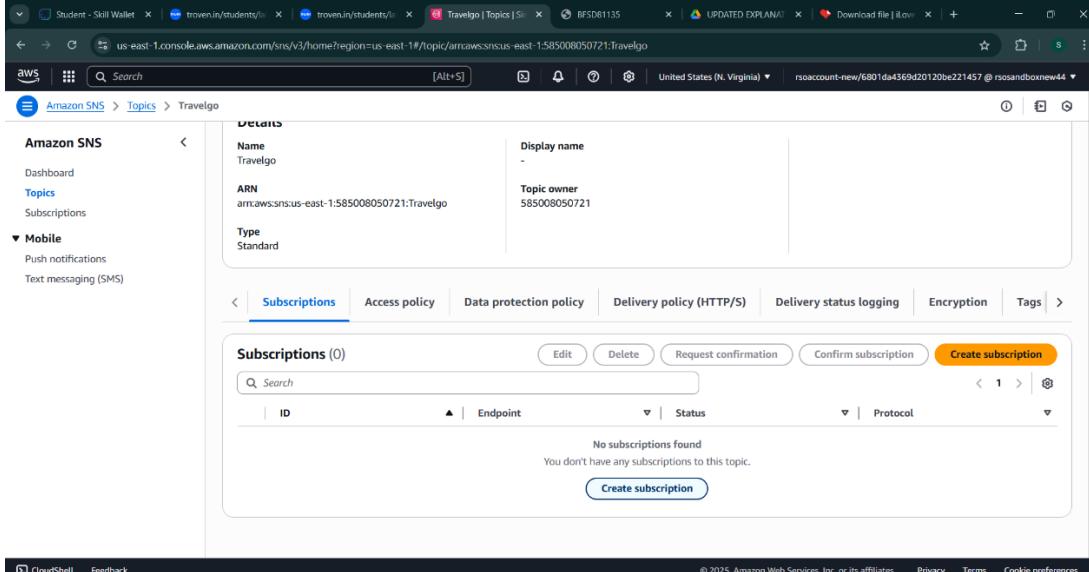
The screenshot shows the "Create topic" wizard on the AWS SNS console. The first step, "Details", is displayed. It allows choosing between "FIFO (first-in, first-out)" and "Standard" message ordering. The "Standard" option is selected, showing bullet points for best-effort ordering, at-least once delivery, and supported protocols (SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints). Below this, there is a "Name" field with "Travelgo" entered, and a note that names must be alphanumeric with hyphens or underscores. There is also a "Display name - optional" field with "My Topic" entered, with a note that only the first 10 characters are displayed in SMS messages. At the bottom, there are links for CloudShell, Feedback, and various AWS terms like Privacy, Terms, and Cookie preferences.

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



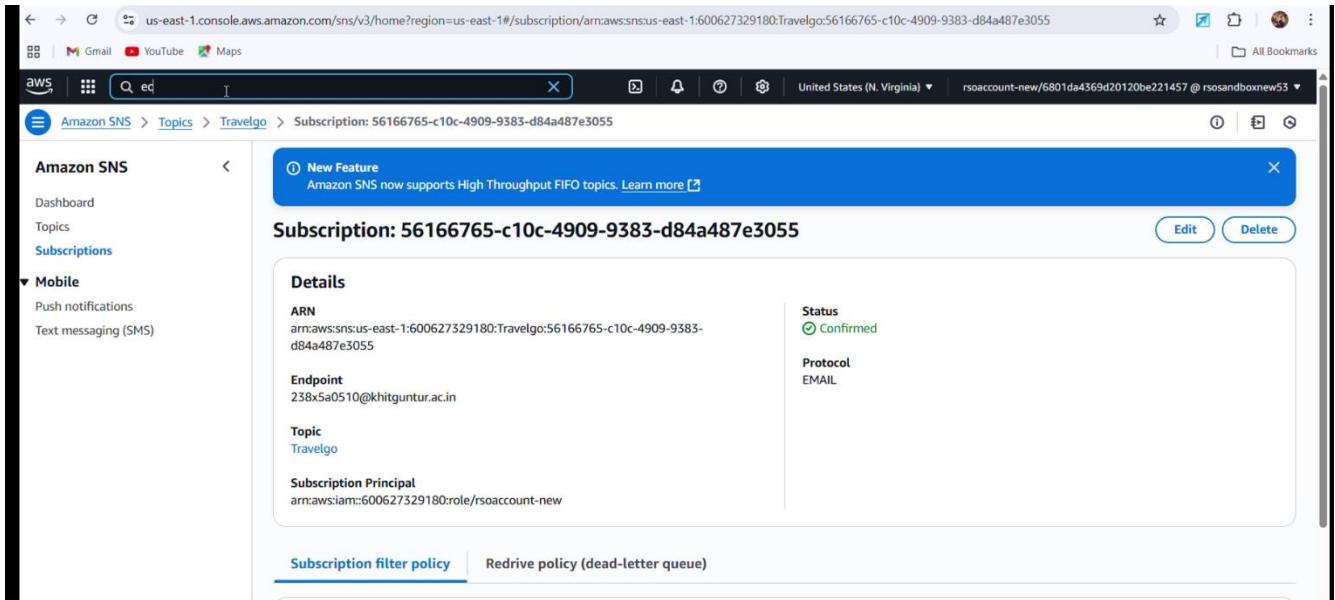
The screenshot shows the AWS SNS Topics page. A green success message states "Topic travelgo created successfully. You can create subscriptions and send messages to them from this topic." Below this, the "Details" section shows the topic name "travelgo", ARN "arn:aws:sns:us-east-1:050752624352:travelgo", and type "Standard". The "Subscriptions" tab is selected, showing 0 subscriptions. There are buttons for "Edit", "Delete", "Publish message", and "Create subscription".

- Configure the SNS topic and note down the **Topic ARN**.
- **Activity 3.2: Subscribe users relevant SNS topics to receive real-time notifications when a book request is made.**



The screenshot shows the AWS SNS Topics page with the "Subscriptions" tab selected. It displays a table with columns for ID, Endpoint, Status, and Protocol. A message at the bottom says "No subscriptions found. You don't have any subscriptions to this topic." There is a "Create subscription" button at the bottom of the table.

- 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". The left sidebar shows "Amazon SNS" with "Subscriptions" selected. The main content area displays the following details:

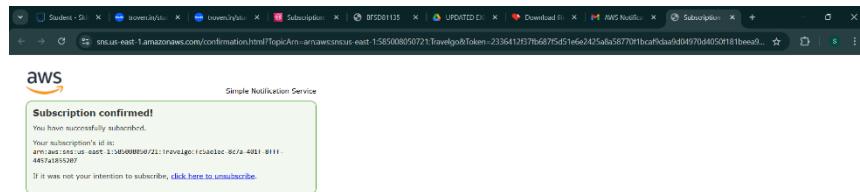
Details	Value
ARN	arn:aws:sns:us-east-1:600627329180:Travelgo:56166765-c10c-4909-9383-d84a487e3055
Endpoint	238x5a0510@khitguntur.ac.in
Topic	Travelgo
Subscription Principal	arn:aws:iam::600627329180:role/rsoaccount-new

Status: Confirmed (Green)

Protocol: EMAIL

Buttons at the bottom: "Edit" and "Delete".

After subscription request for the mail confirmation

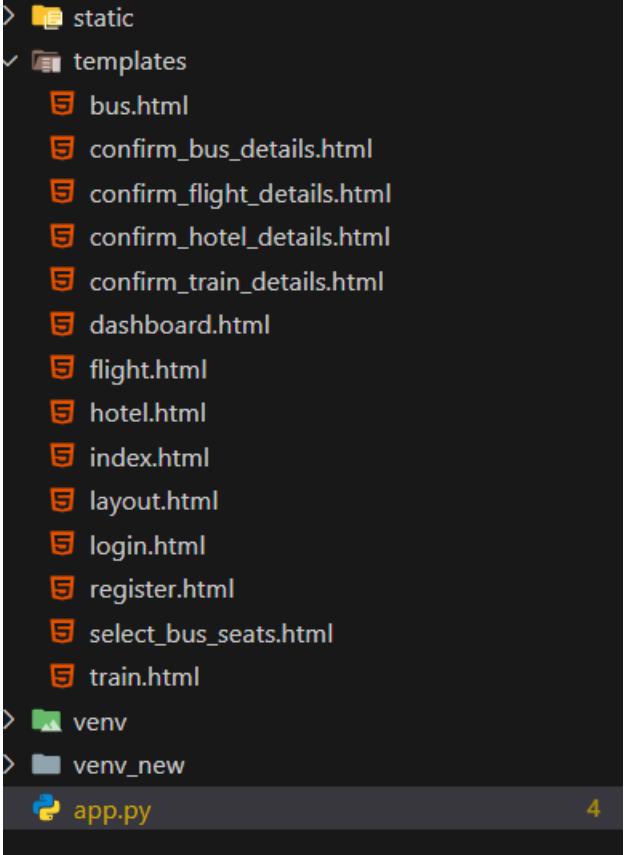


- Successfully done with the SNS mail subscription and setup, now store the ARN link.

## Milestone 4:Backend Development and Application Setup

- **Activity 4.1: Develop the backend using Flask**

- File Explorer Structure



The screenshot shows a file explorer window with the following structure:

- static
- templates
  - bus.html
  - confirm\_bus\_details.html
  - confirm\_flight\_details.html
  - confirm\_hotel\_details.html
  - confirm\_train\_details.html
  - dashboard.html
  - flight.html
  - hotel.html
  - index.html
  - layout.html
  - login.html
  - register.html
  - select\_bus\_seats.html
  - train.html
- venv
- venv\_new
- app.py

A small number '4' is visible in the bottom right corner of the file explorer window.

**Description:** set up the **TravelGO** project with an app.py file, a static/ folder for assets, and a templates/ directory containing all required HTML pages like home, login, register, booking-specific pages (e.g., bus.html, train.html)

### Description of the code :

- **Flask App Initialization**

```
from flask import Flask, render_template, request, redirect, url_for, session, jsonify, flash
import boto3
from boto3.dynamodb.conditions import Key, Attr
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
from decimal import Decimal
import uuid
import random
```

**Description:** import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations,

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

**Description:** initialize the Flask application instance using Flask(\_\_name\_\_) to start building the web app.

- **Dynamodb Setup:**

```
3 # AWS Setup using IAM Role
4 REGION = 'us-east-1' # Replace with your actual AWS region
5 dynamodb = boto3.resource('dynamodb', region_name=REGION)
6 sns_client = boto3.client('sns', region_name=REGION)
7
8 users_table = dynamodb.Table('travelgo_users')
9 trains_table = dynamodb.Table('trains') # Note: This table is declared by
0 bookings_table = dynamodb.Table('bookings')
```

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

- **SNS Connection**

```
SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:418272775181:TravelGo:b7d6f5bc-7710-49de-97bc-ddecff5fd023'
topic ARN

# Function to send SNS notifications

def send_sns_notification(subject, message):
    try:
        sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Subject=subject,
            Message=message
        )
    except Exception as e:
        print(f"SNS Error: Could not send notification - {e}")
        # Optionally, flash an error message to the user or log it more robustly.
    # Function
```

**Description:** Configure **SNS** to send notifications when a book request is submitted. Paste your stored ARN link in the sns\_topic\_arn space, along with the region\_name where the SNS topic is created.

- **Routes for Web Pages**

- **Home Route:**

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

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

- Register Route:

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        # existing = users_collection.find_one({'email': email}) # MongoDB
        existing = users_table.get_item(Key={'email': email}) # DynamoDB
        if 'Item' in existing:
            flash('Email already exists!', 'error')
            return render_template('register.html')
        hashed_password = generate_password_hash(password)
        # users_collection.insert_one({'email': email, 'password': hashed_password}) # MongoDB
        users_table.put_item(Item={'email': email, 'password': hashed_password}) # DynamoDB
        flash('Registration successful! Please log in.', 'success')
        return redirect(url_for('login'))
    return render_template('register.html')
```

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

- login Route (GET/POST):

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if 'username' in session:
        session.pop('username', None)
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        # user = users_collection.find_one({'email': email}) # MongoDB
        user = users_table.get_item(Key={'email': email}) # DynamoDB
        if 'Item' in user and check_password_hash(user['Item']['password'], password):
            session['email'] = email
            flash('Logged in successfully!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid email or password!', 'error')
            return render_template('login.html')
    return render_template('login.html')
```

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

- Home,Bus,Train,Flight,Hotel Routes:

```

@app.route('/dashboard')
def dashboard():
    if 'email' not in session:
        return redirect(url_for('login'))

    user_email = session['email']
    # user_bookings = list(bookings_collection.find({'user_email': user_email}).sort('booking_date', -1)) # MongoDB
    response = bookings_table.query(
        KeyConditionExpression=Key('user_email').eq(user_email),
        ScanIndexForward=False
    )
    bookings = response.get('Items', [])
    for booking in bookings:
        if 'total_price' in booking:
            try:
                booking['total_price'] = float(booking['total_price'])
            except Exception:
                booking['total_price'] = 0.0

    for booking in bookings:
        if '_id' in booking and isinstance(booking['_id'], ObjectId):
            booking['_id'] = str(booking['_id'])

        # Determine vehicle_type for display based on booking_type
        if booking.get('booking_type') == 'bus':
            booking['vehicle_type'] = booking.get('type', 'Bus')
            # Add seat information for bus bookings
            if booking.get('selected_seats'):
                booking['seats_display'] = ', '.join(booking['selected_seats'])
            else:
                booking['seats_display'] = 'N/A'
        elif booking.get('booking_type') == 'train':
            booking['vehicle_type'] = f"Train {booking.get('train_number', '')}" if booking.get('train_number') else "Train"
            # Add seat information for train bookings
            if booking.get('selected_seats'):
                booking['seats_display'] = ', '.join(booking['selected_seats'])
            else:
                booking['seats_display'] = 'N/A'
        elif booking.get('booking_type') == 'flight':
            booking['vehicle_type'] = f"Flight {booking.get('flight_number', '')} ({booking.get('airline', '')})"
            # Add seat information for flights
            if booking.get('selected_seats'):
                booking['seats_display'] = ', '.join(booking['selected_seats'])
            else:
                booking['seats_display'] = 'N/A'
        else:
            booking['vehicle_type'] = booking.get('booking_type', 'N/A')

    return render_template('dashboard.html', username=user_email, bookings=bookings)

```

**Description:** define /dashboard-page to render the main homepage, to handle booking selection and redirection.

- **confirmbooking Routes:**

```

@app.route('/train')
def train():
    if 'email' not in session:
        return redirect(url_for('login'))
    return render_template('train.html')

@app.route('/confirm_train_details')
def confirm_train_details():
    if 'email' not in session:
        return redirect(url_for('login'))

    name = request.args.get('name')
    train_number = request.args.get('trainNumber')
    source = request.args.get('source')
    destination = request.args.get('destination')
    departure_time = request.args.get('departureTime')
    arrival_time = request.args.get('arrivalTime')
    price_per_person = float(request.args.get('price'))
    travel_date = request.args.get('date')
    num_persons = int(request.args.get('persons'))
    train_id = request.args.get('trainId')

    total_price = price_per_person * num_persons

    booking_details = {
        'name': name,
        'train_number': train_number,
        'source': source,
        'destination': destination,
        'departure_time': departure_time,
        'arrival_time': arrival_time,
        'price_per_person': Decimal(price_per_person),
        'travel_date': travel_date,
        'num_persons': num_persons,
        'total_price': Decimal(total_price),
        'item_id': train_id,
        'booking_type': 'train',
        'user_email': session['email']
    }
    session['pending_booking'] = booking_details
    return render_template('confirm_train_details.html', booking=booking_details)
    
```

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

#### Exit Route:

```

@app.route('/logout')
def logout():
    session.pop('email', None)
    flash('You have been logged out.', 'info')
    return redirect(url_for('index'))
    
```

**Description:** define /logout route the index.html page to render when the user chooses to leave or close the application.

### Deployment Code:

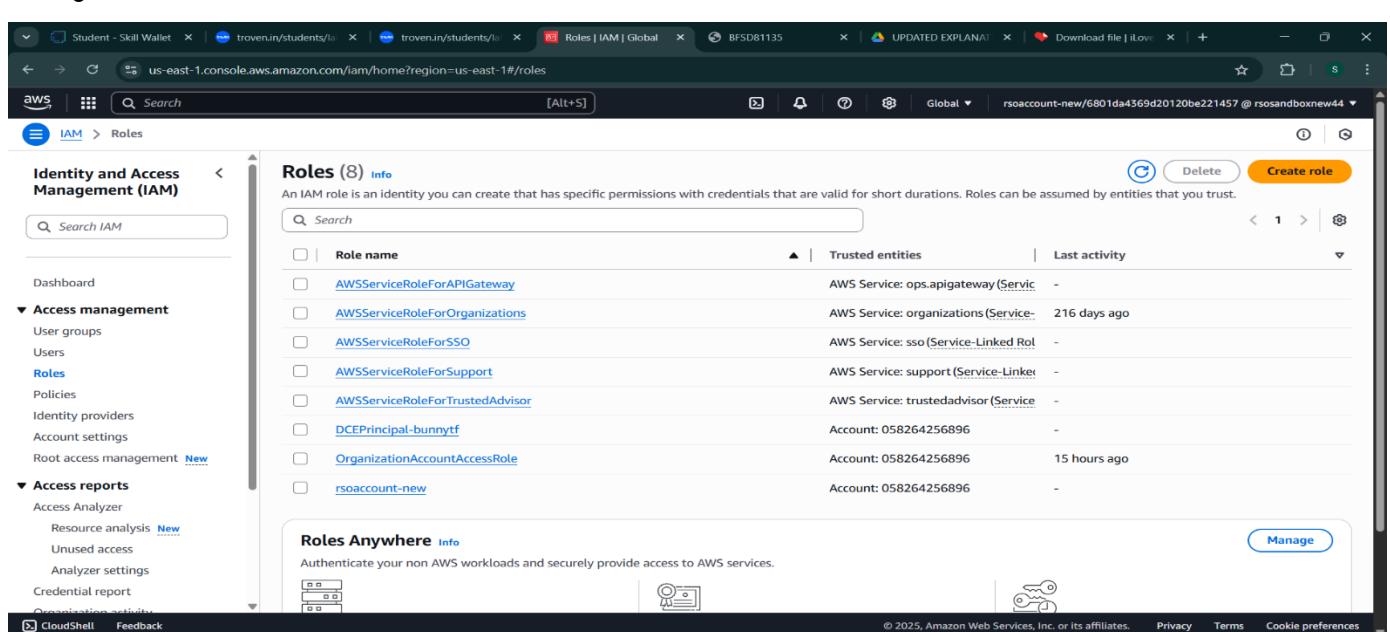
```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

**Description:** start the Flask server to listen on all network interfaces (0 . 0 . 0 . 0) with debug mode enabled for development and testing.

## Milestone 5: IAM Role Setup

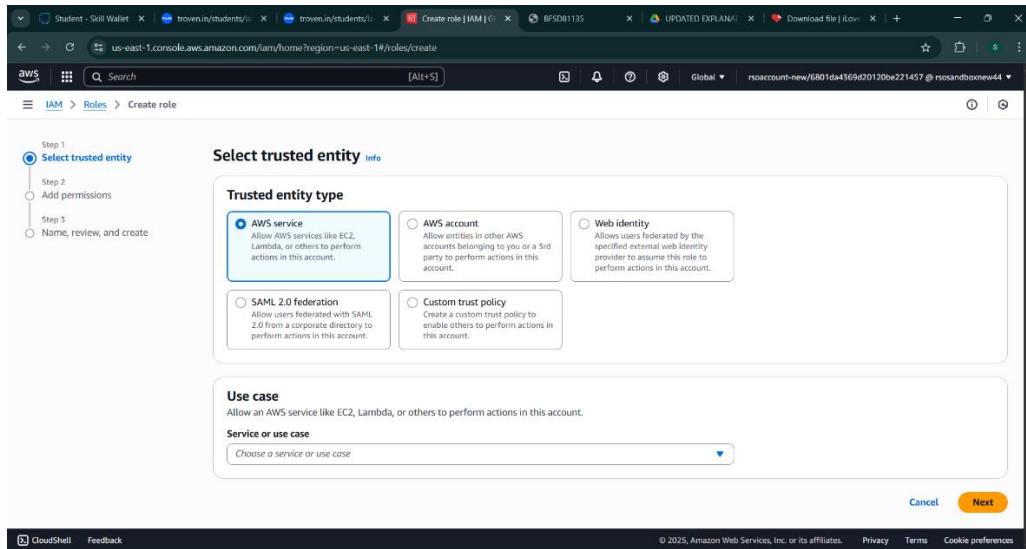
- **Activity 5.1:Create IAM Role.**

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



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	-



Step 1  
 **Select trusted entity** Info

Step 2  
 Add permissions

Step 3  
 Name, review, and create

**Select trusted entity** Info

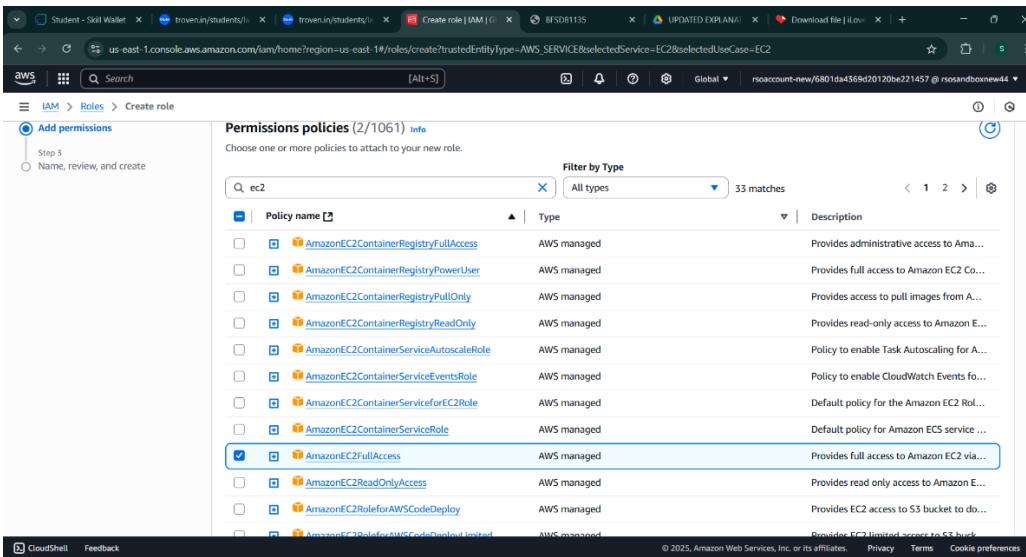
**Trusted entity type**

- 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 its 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**

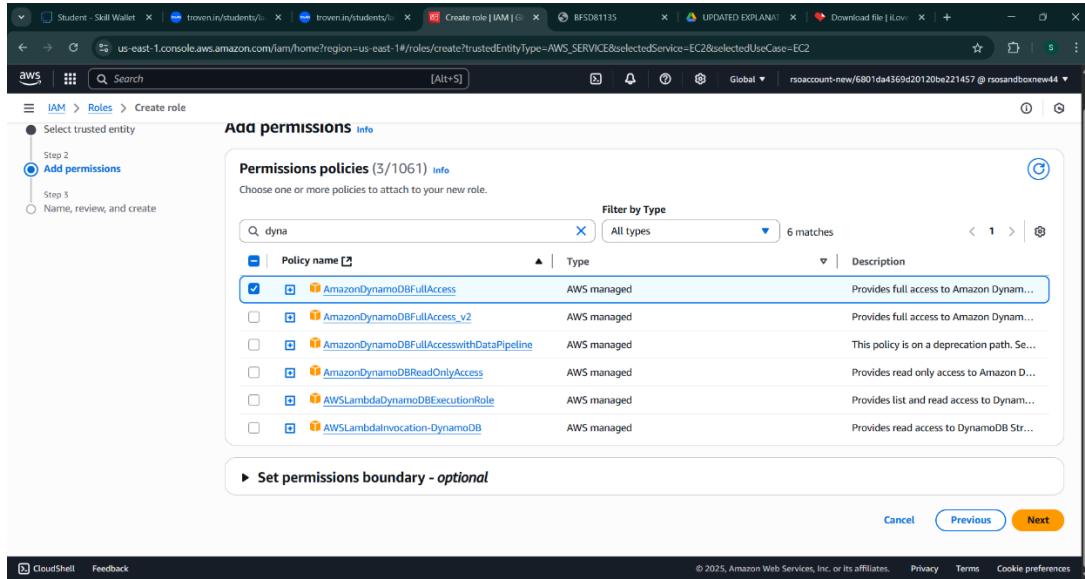
Policy name	Type	Description
AmazonEC2ContainerRegistryFullAccess	AWS managed	Provides administrative access to Amazon E...
AmazonEC2ContainerRegistryPowerUser	AWS managed	Provides full access to Amazon EC2 Co...
AmazonEC2ContainerRegistryPullOnly	AWS managed	Provides access to pull images from A...
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 ...
<input checked="" type="checkbox"/> <b>AmazonEC2FullAccess</b>	AWS managed	Provides full access to Amazon EC2 via...
AmazonEC2ReadonlyAccess	AWS managed	Provides read only access to Amazon E...
AmazonEC2RoleforAWSCodeDeploy	AWS managed	Provides EC2 access to S3 bucket to do...
AmazonEC2RoleforAWSCodeDeployWithLambda	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.



aws | Search [Alt+S]

IAM > Roles > Create role

Step 1 Select trusted entity

Step 2 **Add permissions**

Step 3 Name, review, and create

### Add permissions Info

Permissions policies (3/1061) Info

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

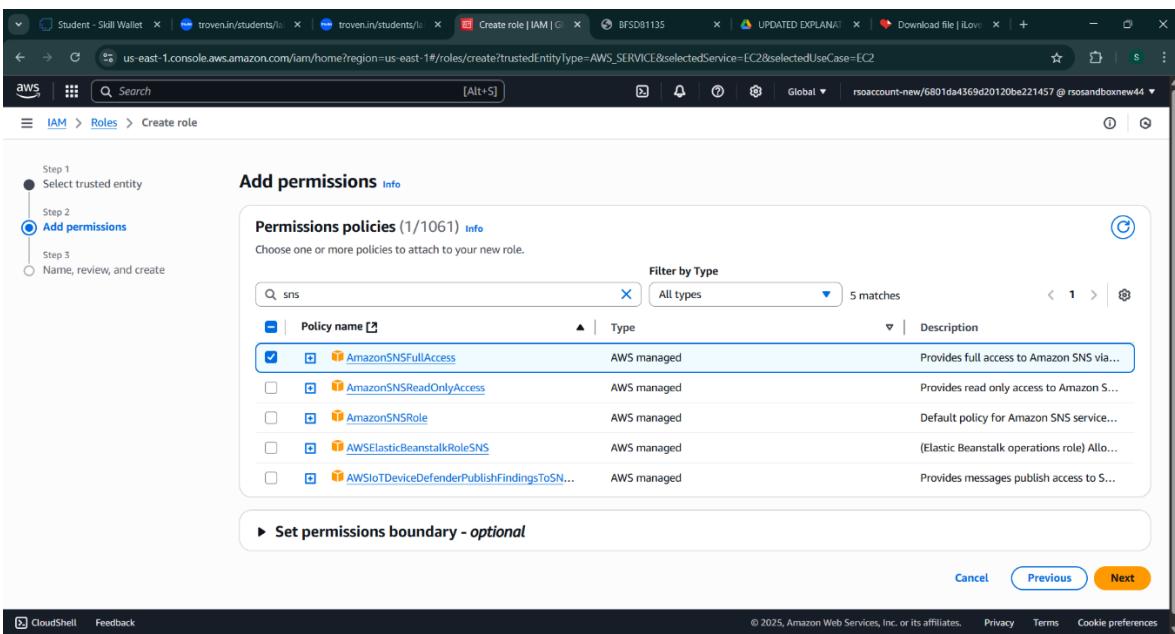
Filter by Type  All types 6 matches

Policy name	Type	Description
<input checked="" type="checkbox"/>  AmazonDynamoDBFullAccess	AWS managed	Provides full access to Amazon DynamoDB.
<input type="checkbox"/>  AmazonDynamoDBFullAccess_v2	AWS managed	Provides full access to Amazon DynamoDB.
<input type="checkbox"/>  AmazonDynamoDBFullAccesswithDataPipeline	AWS managed	This policy is on a deprecation path. See <a href="#">AmazonDynamoDBFullAccess</a> .
<input type="checkbox"/>  AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only access to Amazon DynamoDB.
<input type="checkbox"/>  AWSLambdaDynamoDBExecutionRole	AWS managed	Provides list and read access to DynamoDB.
<input type="checkbox"/>  AWSLambdaInvocation-DynamoDB	AWS managed	Provides read access to DynamoDB Stream.

▶ Set permissions boundary - optional

Cancel Previous Next

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



aws | Search [Alt+S]

IAM > Roles > Create role

Step 1 Select trusted entity

Step 2 **Add permissions**

Step 3 Name, review, and create

### Add permissions Info

Permissions policies (1/1061) Info

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

Filter by Type  All types 5 matches

Policy name	Type	Description
<input checked="" type="checkbox"/>  AmazonSNSFullAccess	AWS managed	Provides full access to Amazon SNS via the AWS Lambda permission model.
<input type="checkbox"/>  AmazonSNSReadOnlyAccess	AWS managed	Provides read only access to Amazon SNS.
<input type="checkbox"/>  AmazonSNSRole	AWS managed	Default policy for Amazon SNS service role.
<input type="checkbox"/>  AWSElasticBeanstalkRoleSNS	AWS managed	(Elastic Beanstalk operations role) Allows the Elastic Beanstalk service to publish messages to SNS.
<input type="checkbox"/>  AWSIoTDeviceDefenderPublishFindingsToSN...	AWS managed	Provides messages publish access to SNS.

▶ Set permissions boundary - optional

Cancel Previous Next

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

## Milestone 6: EC2 Instance Setup

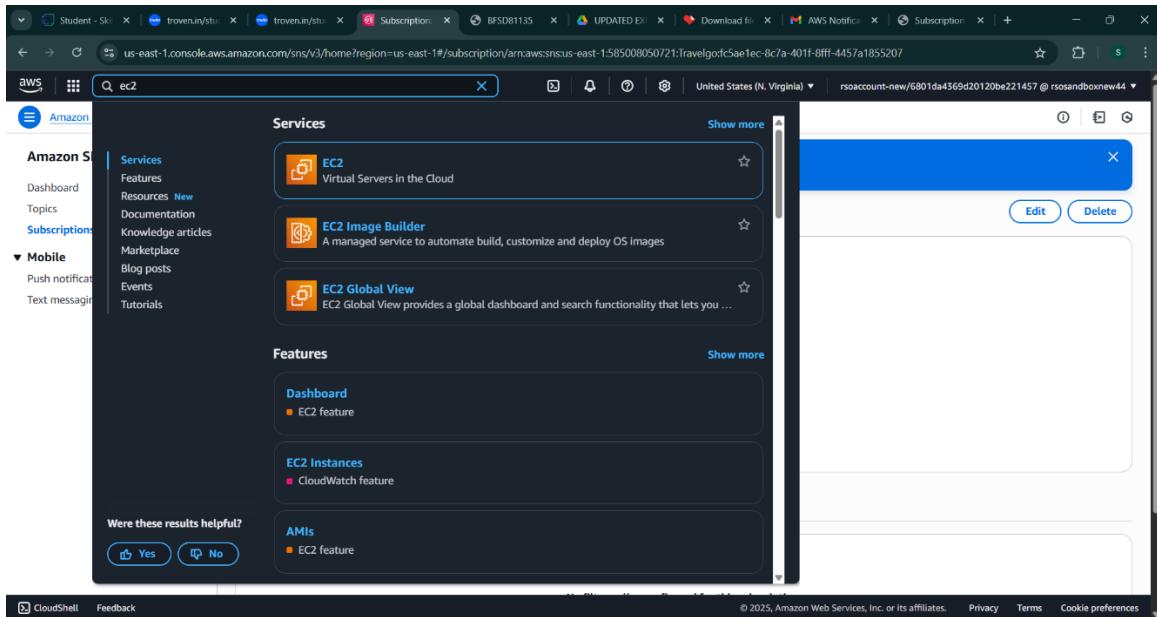
- Note: Load your Flask app and Html files into GitHub repository.

 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

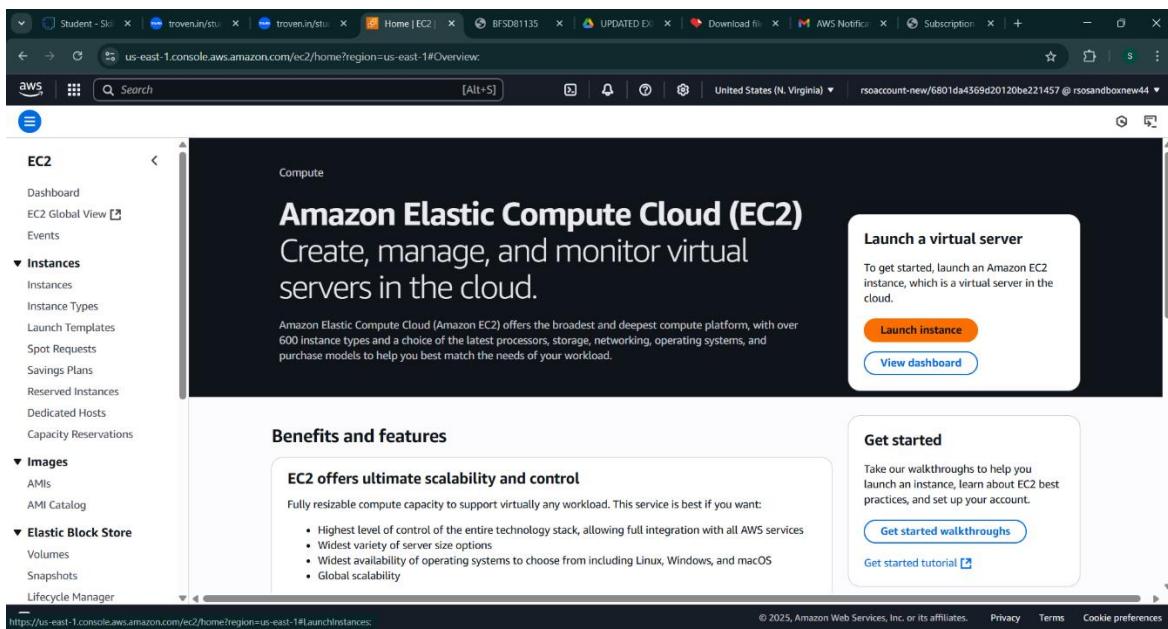
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- **Launch EC2 Instance**

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



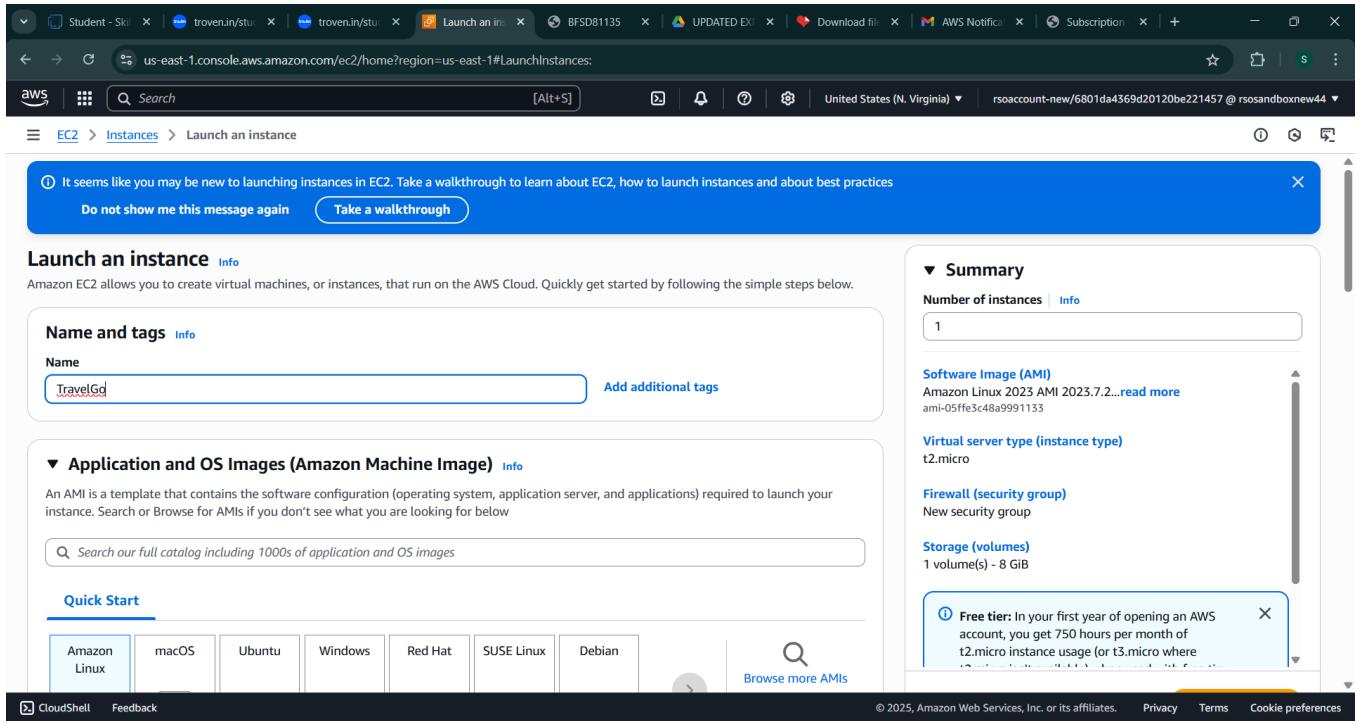
The screenshot shows the AWS search results for the query "ec2". The results are categorized under "Services" and "Features". The "EC2" service card is highlighted, showing its icon (a server), name, and description: "Virtual Servers in the Cloud". Other cards include "EC2 Image Builder" and "EC2 Global View". Below these, under "Features", there are cards for "Dashboard", "EC2 Instances", and "AMIs". At the bottom left, there are "Yes" and "No" buttons for a helpfulness poll. On the right, a modal window titled "EC2" is open, showing "Edit" and "Delete" buttons.



The screenshot shows the AWS EC2 home page. The left sidebar includes links for EC2 Dashboard, EC2 Global View, Events, Instances (with sub-links for Instances 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 features a large heading "Amazon Elastic Compute Cloud (EC2)" with the sub-headline "Create, manage, and monitor virtual servers in the cloud.". A call-to-action box contains "Launch a virtual server" with "Launch instance" and "View dashboard" buttons. Another box titled "Get started" contains "Get started walkthroughs" and "Get started tutorial". The footer includes standard AWS links for privacy, terms, and cookie preferences.

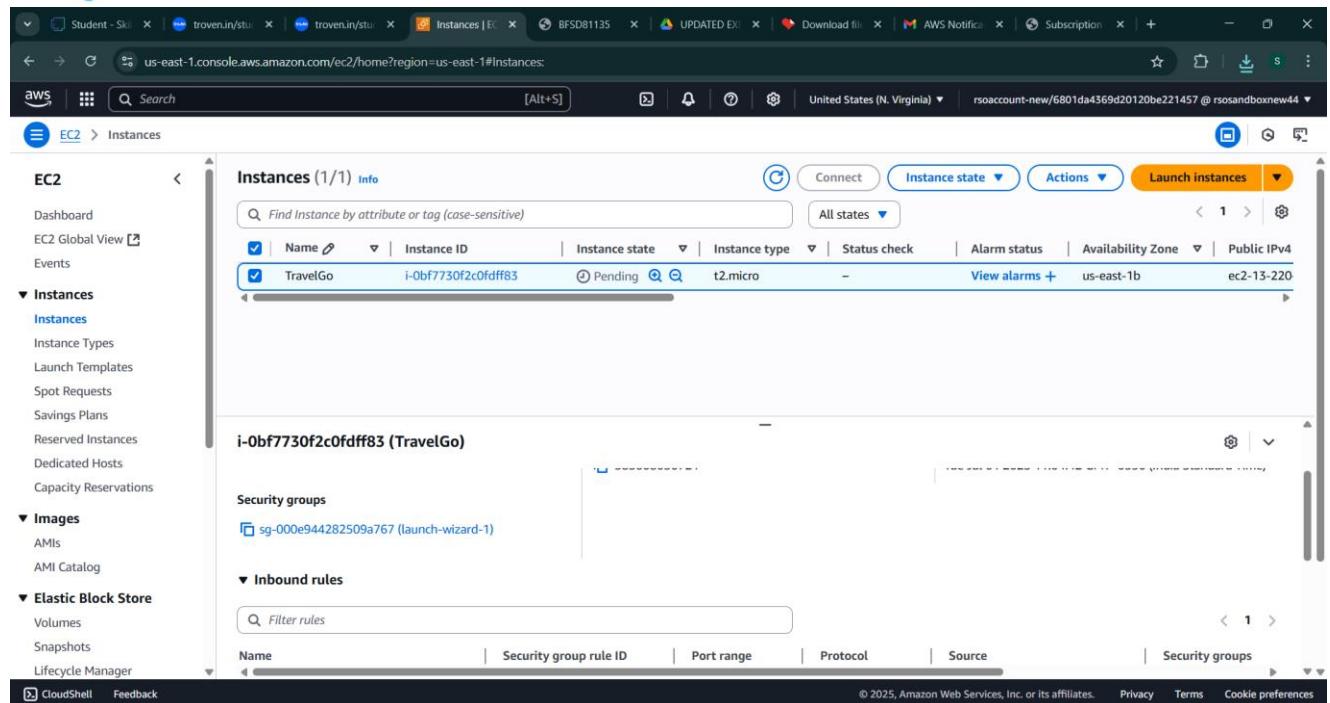
- Click on Launch instance to launch EC2 instance

- 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.



Instances (1/1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
TravelGo	i-0bf7730f2c0fdff83	Pending	t2.micro	-	View alarms +	us-east-1b	ec2-13-220

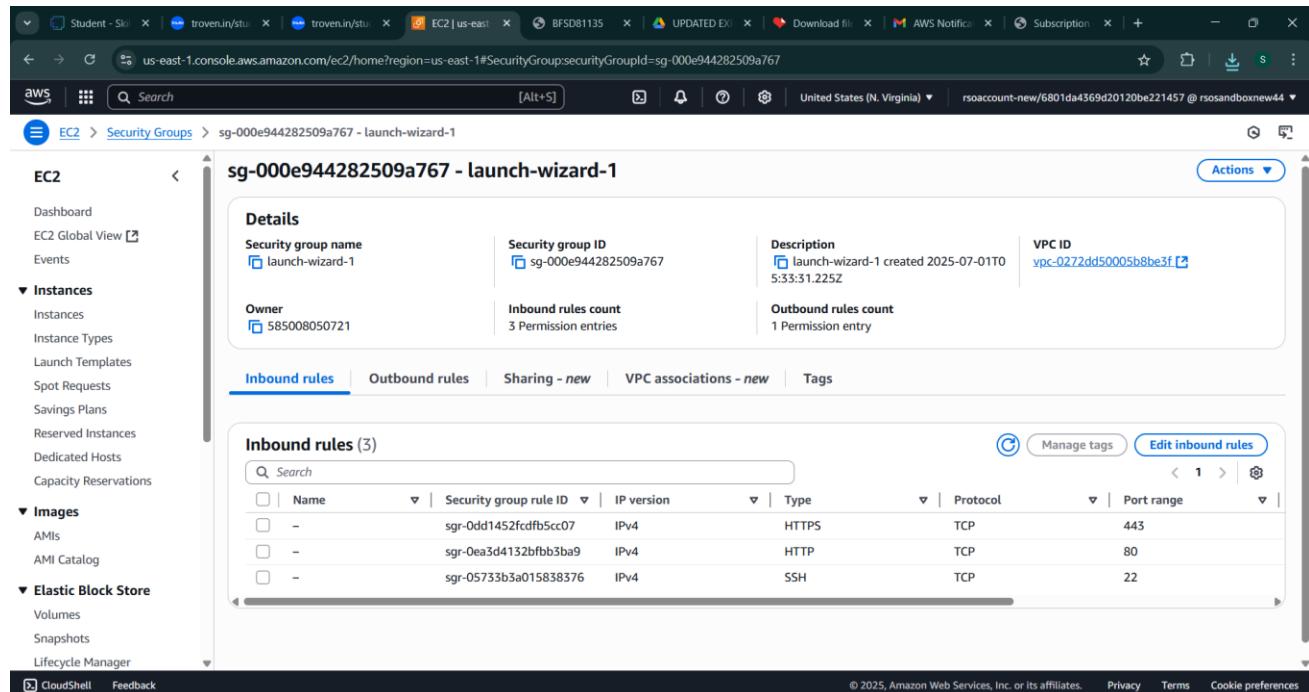
**i-0bf7730f2c0fdff83 (TravelGo)**

Security groups: sg-000e944282509a767 (launch-wizard-1)

Inbound rules:

Name	Security group rule ID	Port range	Protocol	Source

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



sg-000e944282509a767 - launch-wizard-1

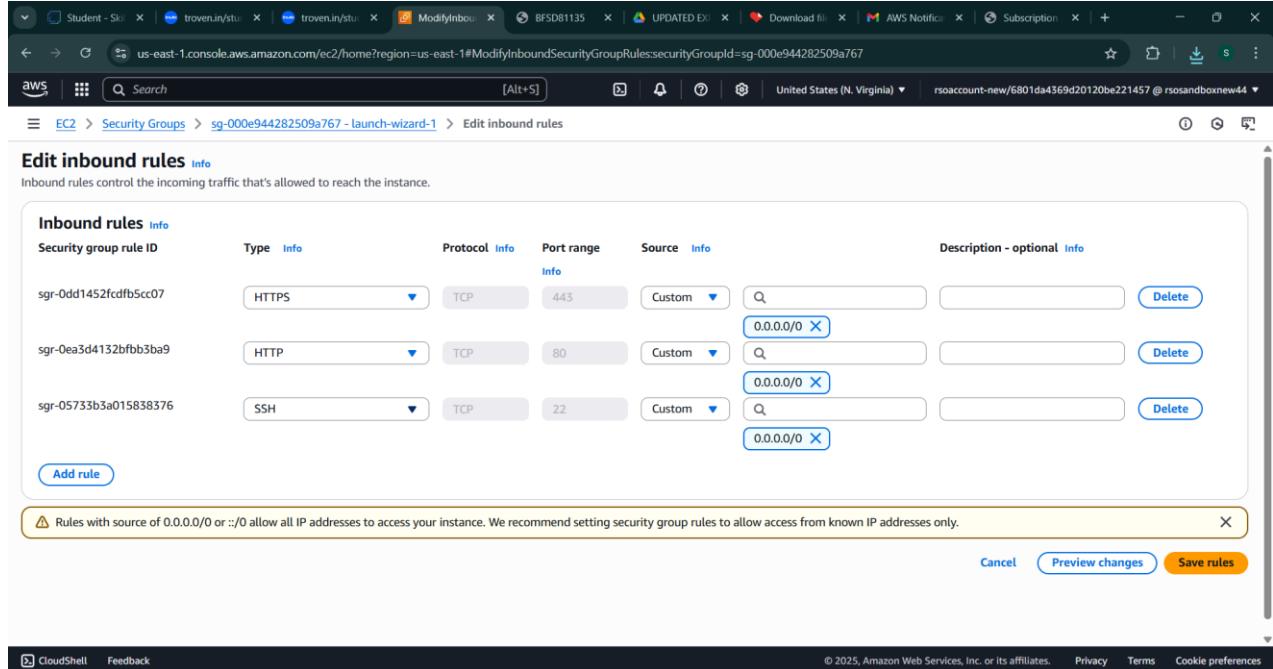
Details

Security group name	Security group ID	Description	VPC ID
launch-wizard-1	sg-000e944282509a767	launch-wizard-1 created 2025-07-01T05:33:31.225Z	vpc-0272dd50005b8be3f
Owner	585008050721	Inbound rules count 3 Permission entries	Outbound rules count 1 Permission entry

Inbound rules (3)

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-0dd1452fcfb5cc07	IPv4	HTTPS	TCP	443
-	sgr-0ea3d4132bfbb3ba9	IPv4	HTTP	TCP	80
-	sgr-05733b3a015838376	IPv4	SSH	TCP	22

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



**Edit inbound rules** Info

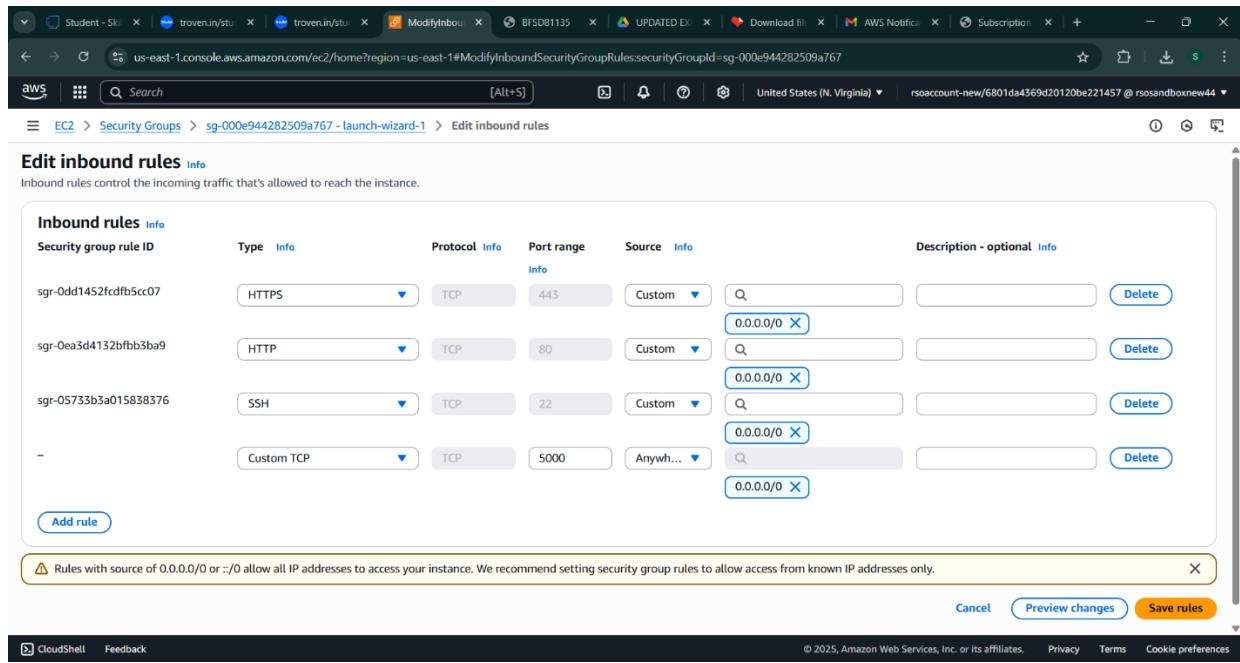
Inbound rules control the incoming traffic that's allowed to reach the instance.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdfb5cc07	HTTPS	TCP	443	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: blue;">X</span> <span style="color: red;">Delete</span>
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: blue;">X</span> <span style="color: red;">Delete</span>
sgr-05733b3a015838376	SSH	TCP	22	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: blue;">X</span> <span style="color: red;">Delete</span>

[Add rule](#)

⚠ Rules with source of 0.0.0.0/0 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.

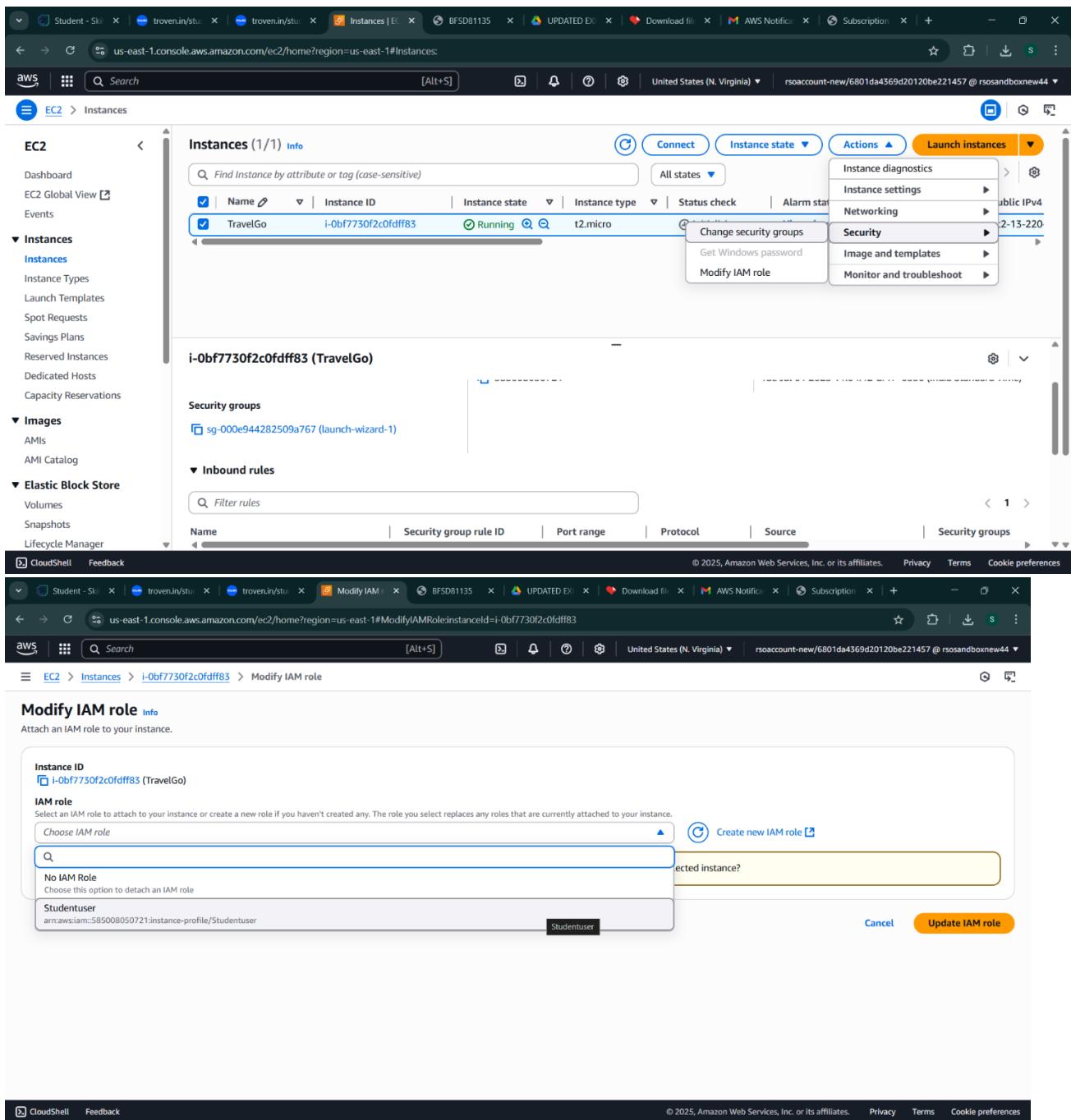
Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdfb5cc07	HTTPS	TCP	443	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: blue;">X</span> <span style="color: red;">Delete</span>
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: blue;">X</span> <span style="color: red;">Delete</span>
sgr-05733b3a015838376	SSH	TCP	22	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: blue;">X</span> <span style="color: red;">Delete</span>
-	Custom TCP	TCP	5000	Anywhere	<input type="text"/> 0.0.0.0/0 <span style="color: blue;">X</span> <span style="color: red;">Delete</span>

[Add rule](#)

⚠ Rules with source of 0.0.0.0/0 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

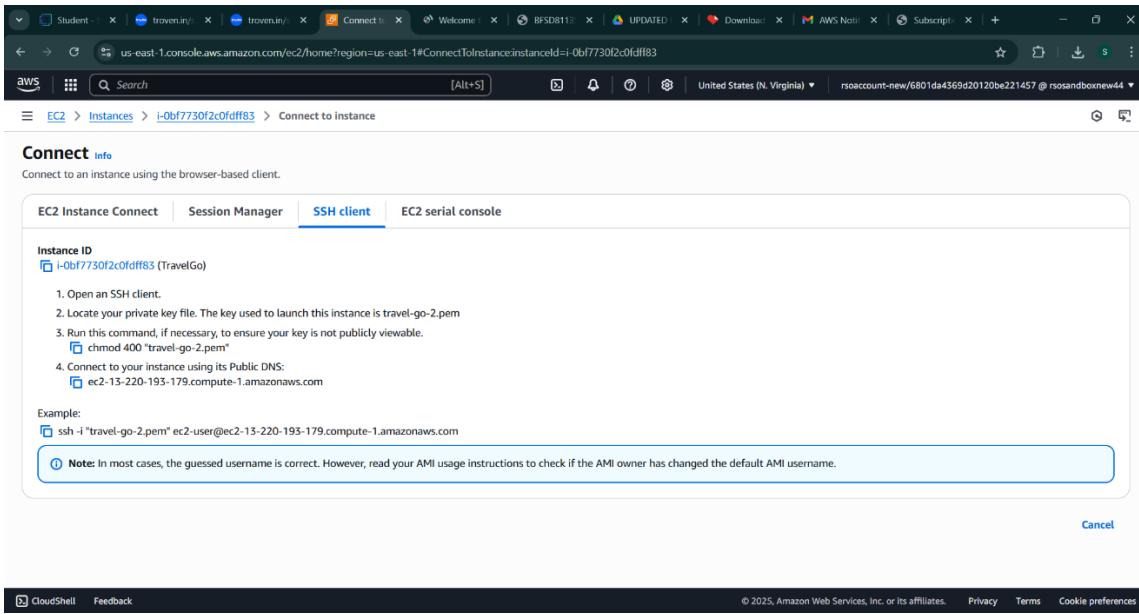
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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



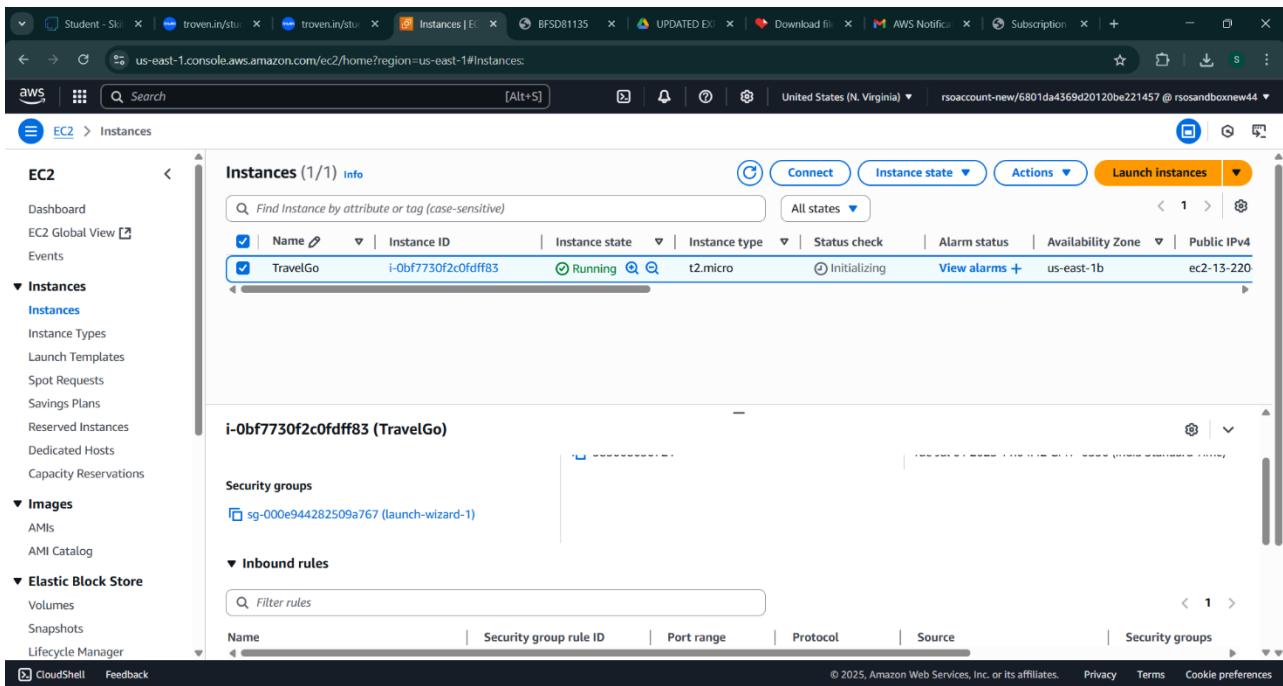
The screenshot shows two windows of the AWS Management Console:

- Top Window (Instances Page):** The URL is `us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#instances`. The left sidebar shows the EC2 navigation path. The main area displays one instance named "TravelGo" (i-0bf7730f2c0fdf83) which is "Running". A context menu is open over this instance, with the "Security" option highlighted. Other options include "Instance diagnostics", "Instance settings", "Networking", "Get Windows password", "Modify IAM role", and "Monitor and troubleshoot".
- Bottom Window (Modify IAM Role Dialog):** The URL is `us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ModifyIAMRoleInstanceId=i-0bf7730f2c0fdf83`. It shows the "Modify IAM role" dialog for the "TravelGo" instance. The "Instance ID" field is pre-filled with "i-0bf7730f2c0fdf83 (TravelGo)". The "IAM role" section contains a dropdown menu with "Choose IAM role" and "Create new IAM role" options. Below this is a search bar and a list of IAM roles, with "Studentuser" selected. A note says "Choose this option to detach an IAM role". At the bottom right 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 displays the URL `us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ConnectToInstance$instanceId=i-0bf7730f2c0fdff83`. Below this, the breadcrumb navigation shows `EC2 > Instances > i-0bf7730f2c0fdff83 > Connect to instance`. The main content area is titled "Connect Info" and contains instructions for connecting via an SSH client. It includes a list of steps: 1. Open an SSH client, 2. Locate your private key file, 3. Run this command if necessary, and 4. Connect to your instance using its Public DNS. A note at the bottom states: "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 located at the bottom right.



The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed, showing the "Instances" section is selected. The main content area displays a table titled "Instances (1/1) Info". The table lists one instance: "TravelGo" (Instance ID: i-0bf7730f2c0fdff83, Instance state: Running, Instance type: t2.micro, Status check: Initializing). Below the table, the instance details for "i-0bf7730f2c0fdff83 (TravelGo)" are shown, including "Security groups" (sg-000e944282509a767 (launch-wizard-1)) and "Inbound rules". A "CloudShell" and "Feedback" link are at the bottom left, and a copyright notice for 2025 Amazon Web Services, Inc. or its affiliates is at the bottom right.

## 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
```

## Milestone 8: Testing and Deployment

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

**Welcome Page:**

**Register Page:**

**Login Page:**

**Bus Booking Page:**

**Train Booking page:**

**Flight Booking Page:**

**Hotel Booking Page:**

## Conclusion:

The **TravelGo** Website has been successfully developed and deployed using a scalable and cloud-native architecture. Leveraging AWS services such as EC2 for hosting, DynamoDB for real-time data management, and SNS for instant booking and cancellation notifications, the platform provides a seamless travel booking experience for users. TravelGo enables registered users to search and book buses, trains, flights, and hotels in a centralized, intuitive interface, eliminating the complexities of navigating multiple travel services.

The cloud infrastructure ensures high availability and smooth performance even during peak usage, while the Flask backend ensures efficient handling of user authentication, dynamic booking flows, and data transactions. Real-time notification integration via AWS SNS allows users to receive booking confirmations and cancellations immediately via email, improving communication and user engagement.

In summary, the **TravelGo** Website offers a modern, reliable, and user-friendly solution for managing travel and accommodation needs. It highlights the potential of cloud-based platforms in building unified travel systems, simplifying operations, and enhancing the overall user experience.