

## 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.

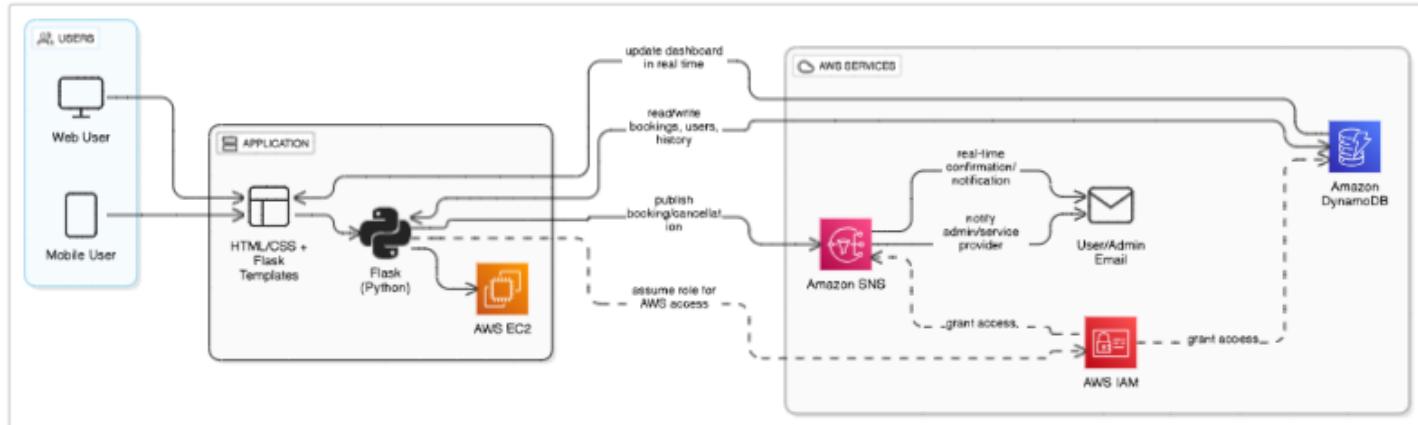
### Scenario 2: Real-Time Booking Confirmation with AWS SNS

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.

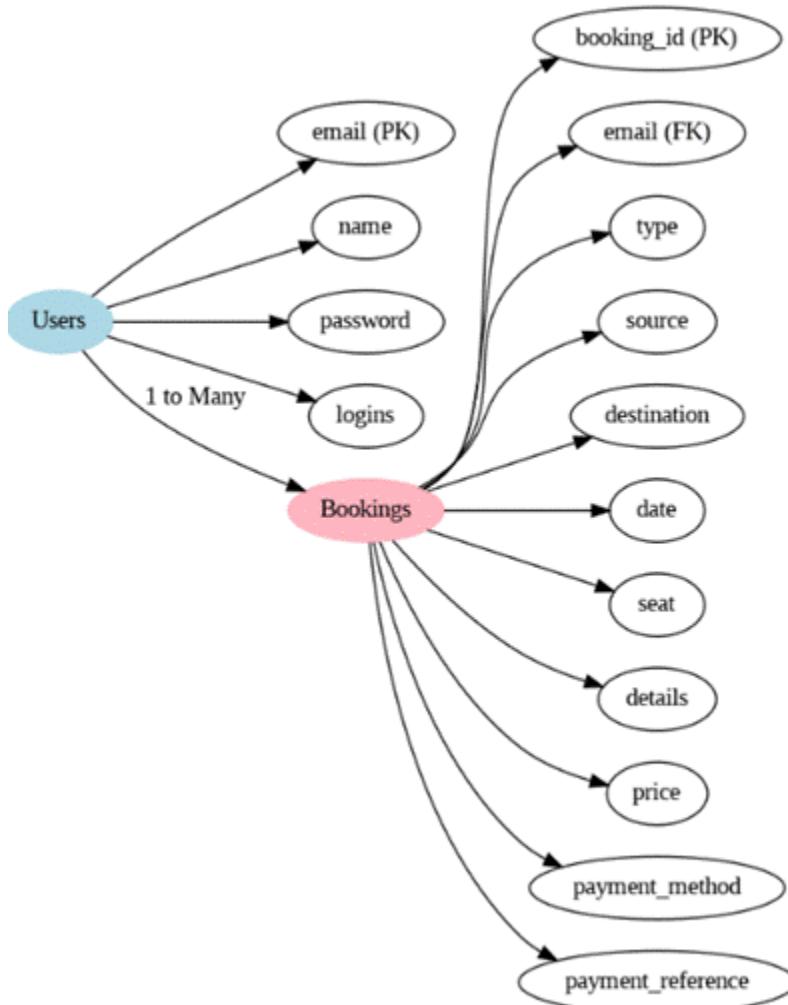
### Scenario 3: Dynamic Dashboard with Personal Travel History

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

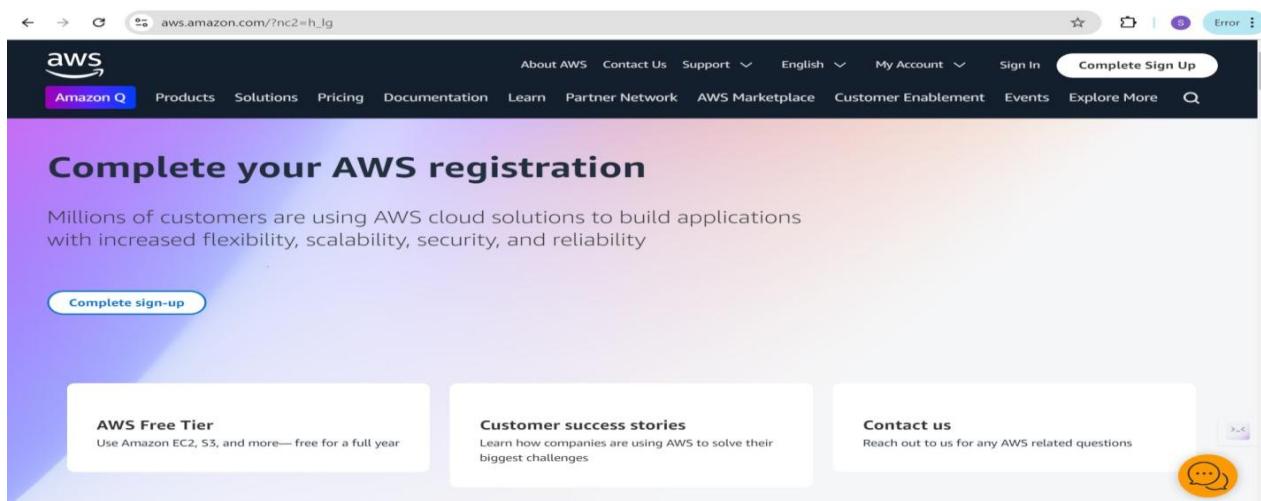
**Activity 7.2:** Run the Flask App

## 8. Testing and Deployment

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

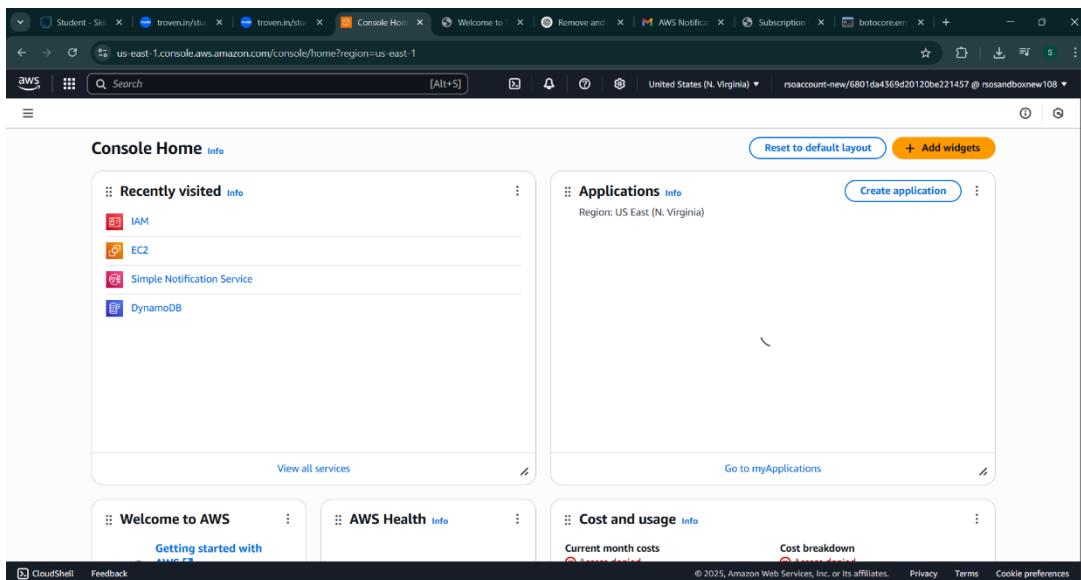
### Milestone 1: AWS Account Setup and Login

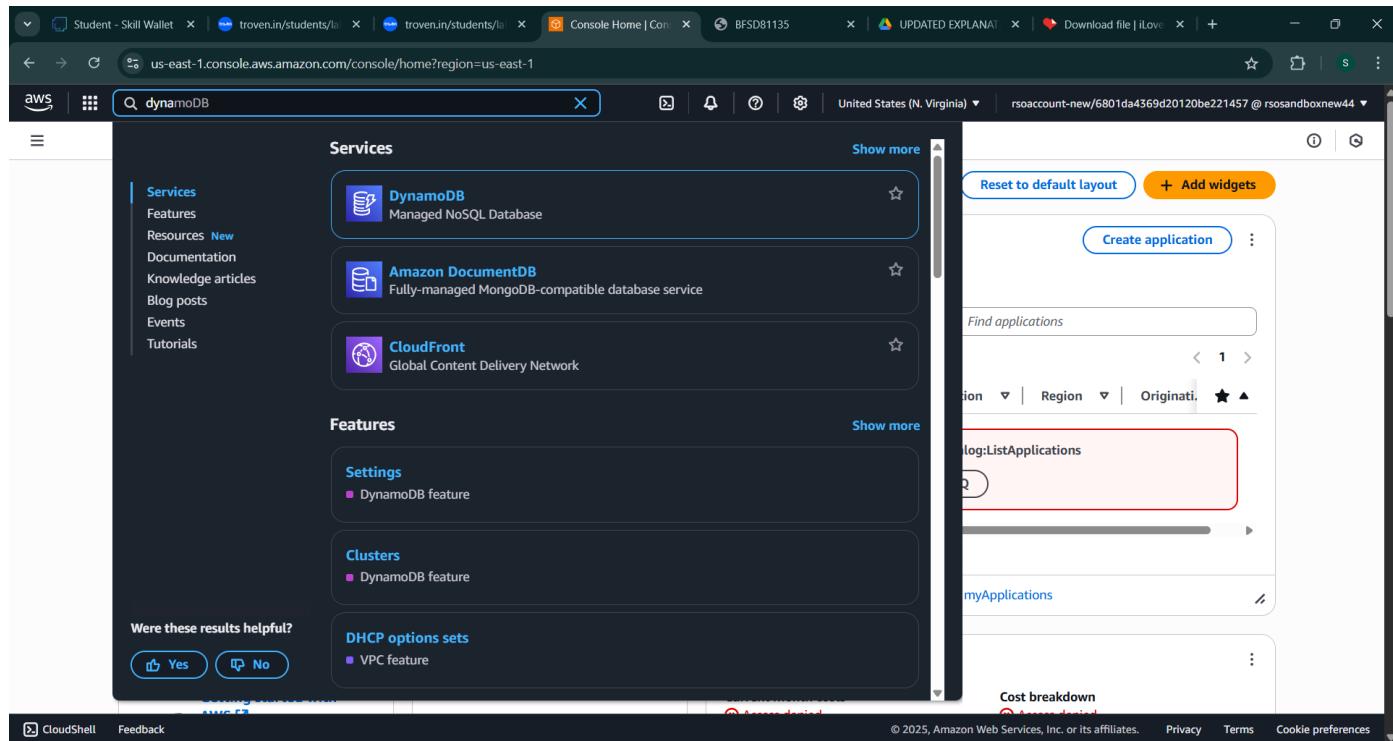
- **Activity 1.1: Set up an AWS account if not already done.**
  - Sign up for an AWS account and configure billing settings.



- **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 console search results for 'dynamoDB'. The left sidebar has a 'Services' section with links to Features, Resources, Documentation, Knowledge articles, Blog posts, Events, and Tutorials. The main search results are:

- DynamoDB**: Managed NoSQL Database
- Amazon DocumentDB**: Fully-managed MongoDB-compatible database service
- CloudFront**: Global Content Delivery Network

Under 'Features':

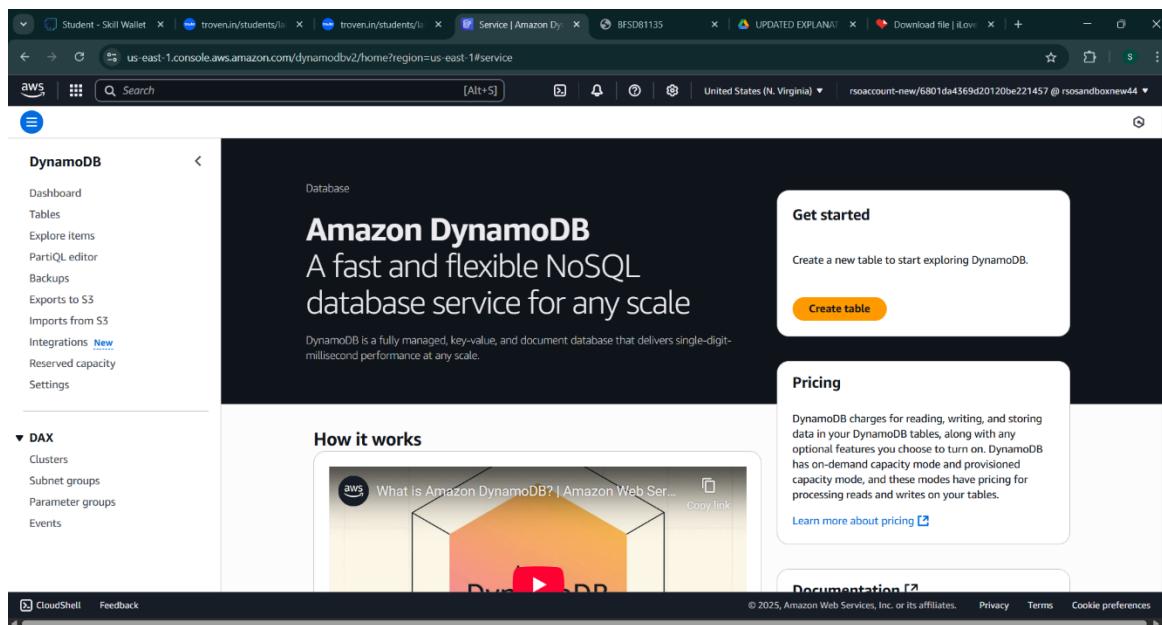
- Settings**: □ DynamoDB feature
- Clusters**: □ DynamoDB feature
- DHCP options sets**: □ VPC feature

A sidebar on the right shows 'Create application' and 'Find applications' sections, along with a 'Cost breakdown' 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 AWS DynamoDB service home page. The left sidebar has links for Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main content area includes:

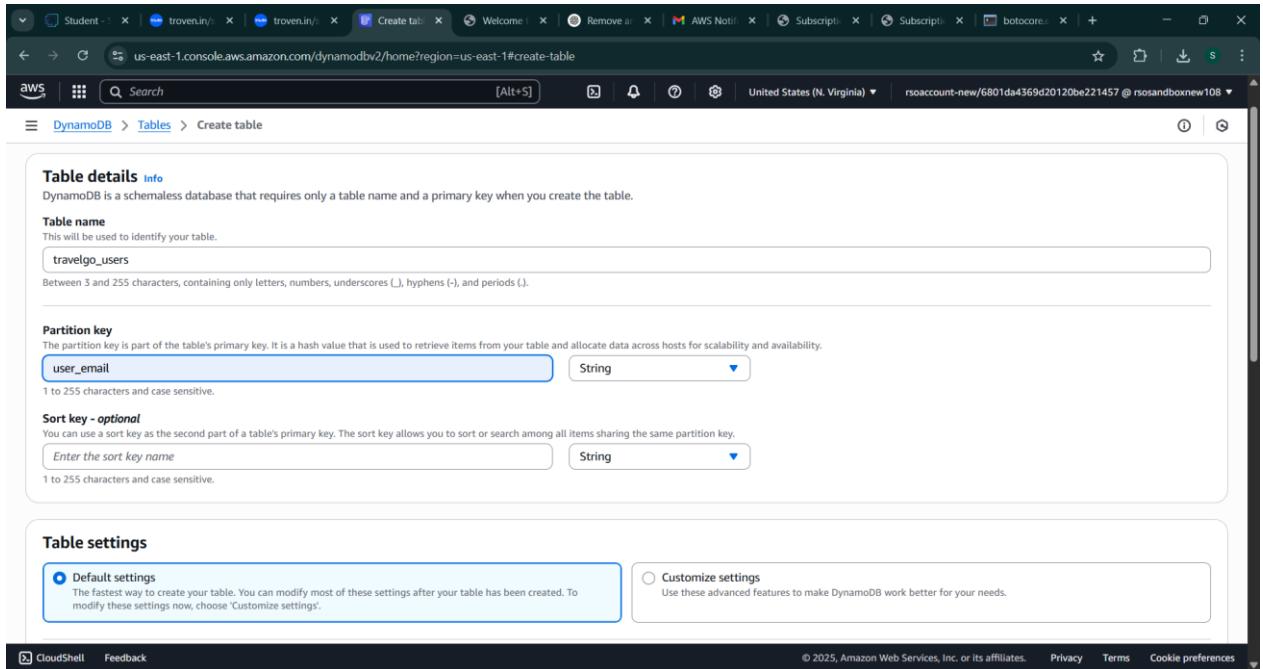
- Amazon DynamoDB**: A fast and flexible NoSQL database service for any scale.
- How it works**: A video player showing a presentation about DynamoDB.
- Get started**: Create a new table to start exploring DynamoDB.
- Pricing**: DynamoDB charges for reading, writing, and storing data in your DynamoDB tables, along with any optional features you choose to turn on. DynamoDB has on-demand capacity mode and provisioned capacity mode, and these modes have pricing for processing reads and writes on your tables.
- Documentation**: Learn more about pricing.

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 'Create table' wizard in the AWS DynamoDB console. 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 it's the fastest way to create the table. The 'Customize settings' option is also available for advanced features.



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 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.

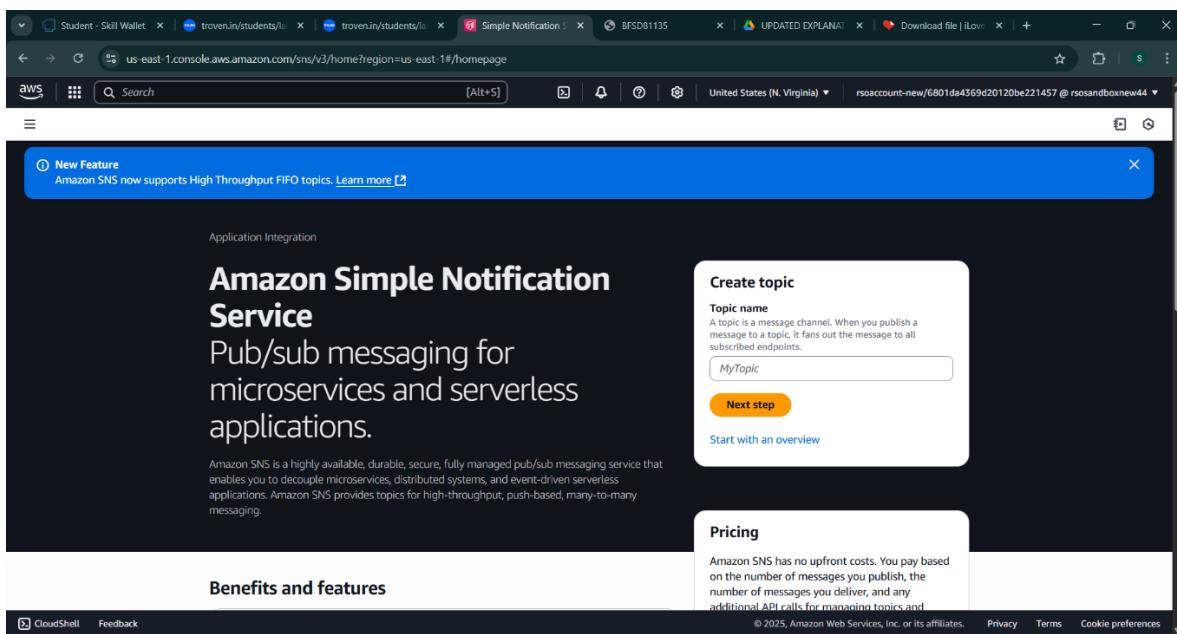
The left sidebar shows the navigation menu for SNS, including Services, Features, Resources, Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials.

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.

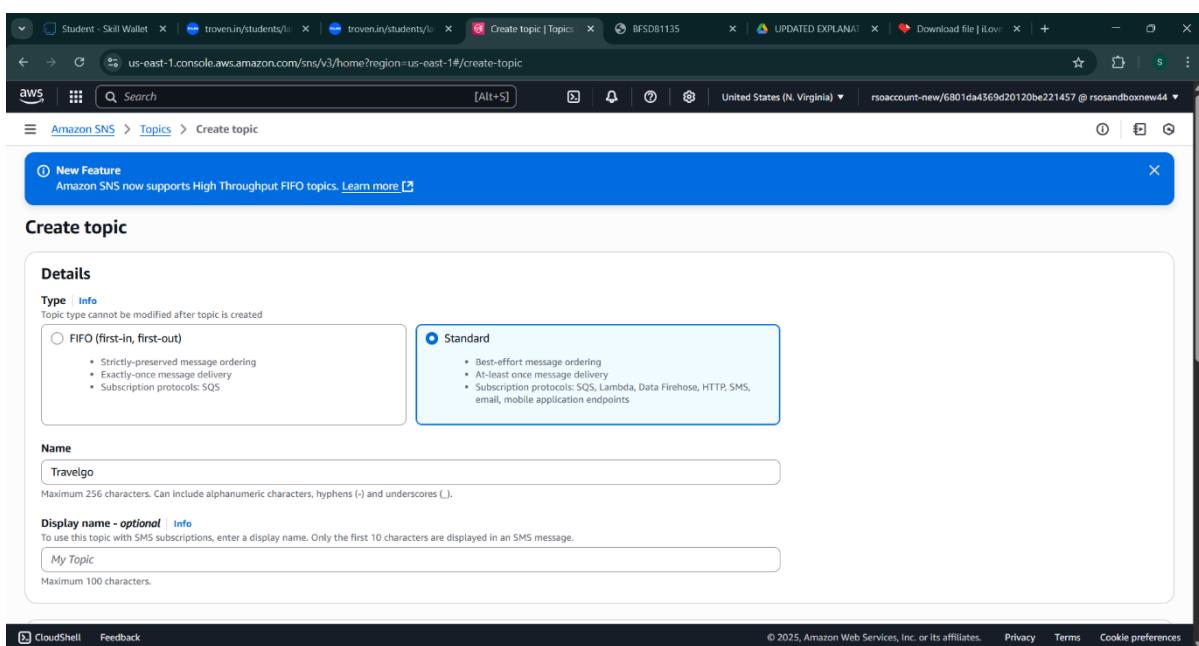
A modal window titled 'Create table' is open on the right side of the screen, showing fields for Region, Deletion protection, and Read capacity units.

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



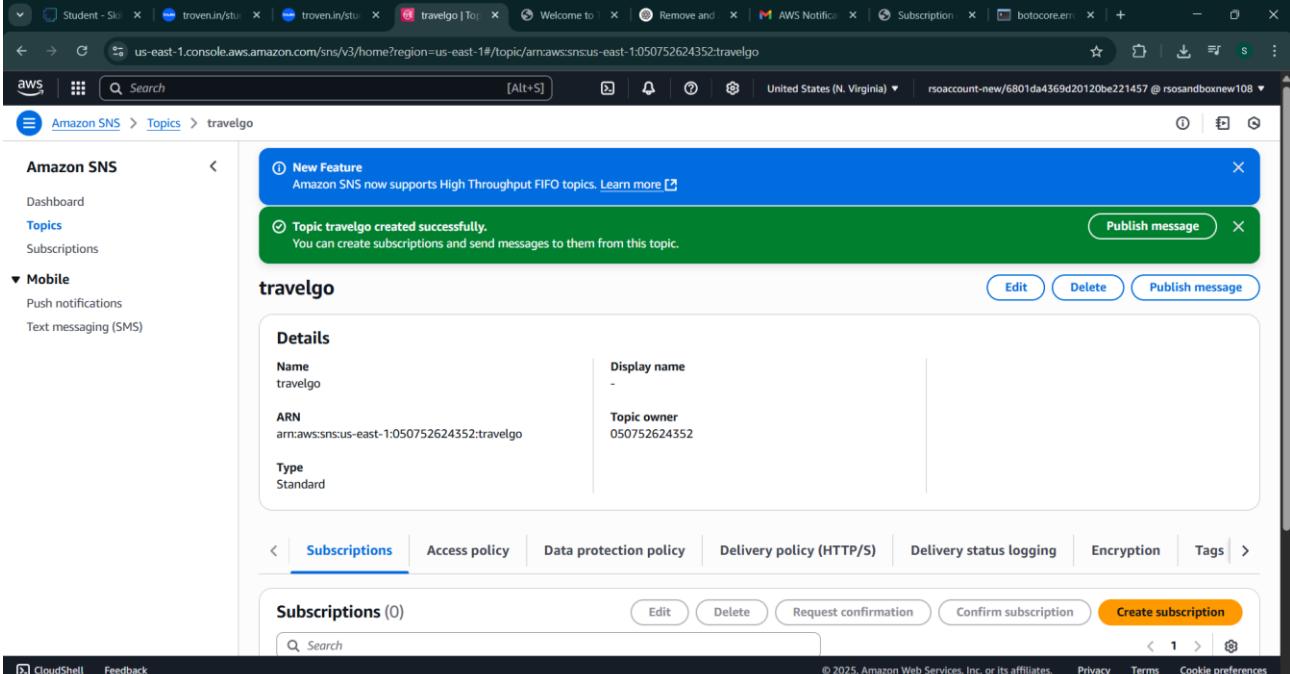
The screenshot shows the AWS Simple Notification Service (SNS) dashboard. At the top, there is a blue banner with the text "Amazon SNS now supports High Throughput FIFO topics. Learn more". Below the banner, the page title is "Amazon Simple Notification Service" with the subtitle "Pub/sub messaging for 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.



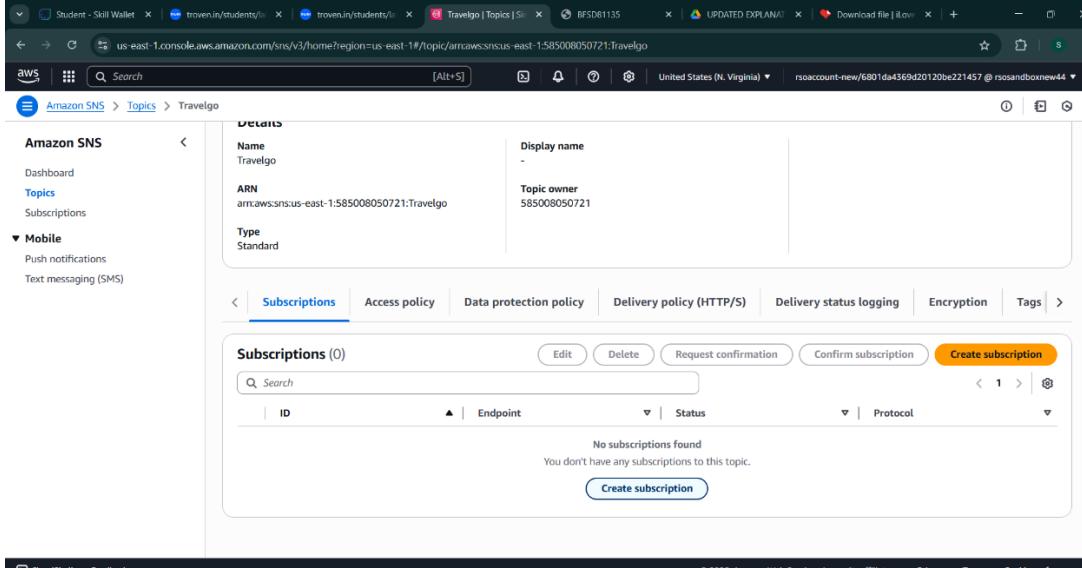
The screenshot shows the "Create topic" wizard in the AWS SNS service. The first step, "Details", is displayed. It includes sections for "Type" (with "Info" link), "Name" (with "Travelgo" entered), and "Display name - optional" (with "My Topic" entered). The "Type" section has two options: "FIFO (first-in, first-out)" and "Standard". The "Standard" type is selected, and 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, there is a "Create topic" button.

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



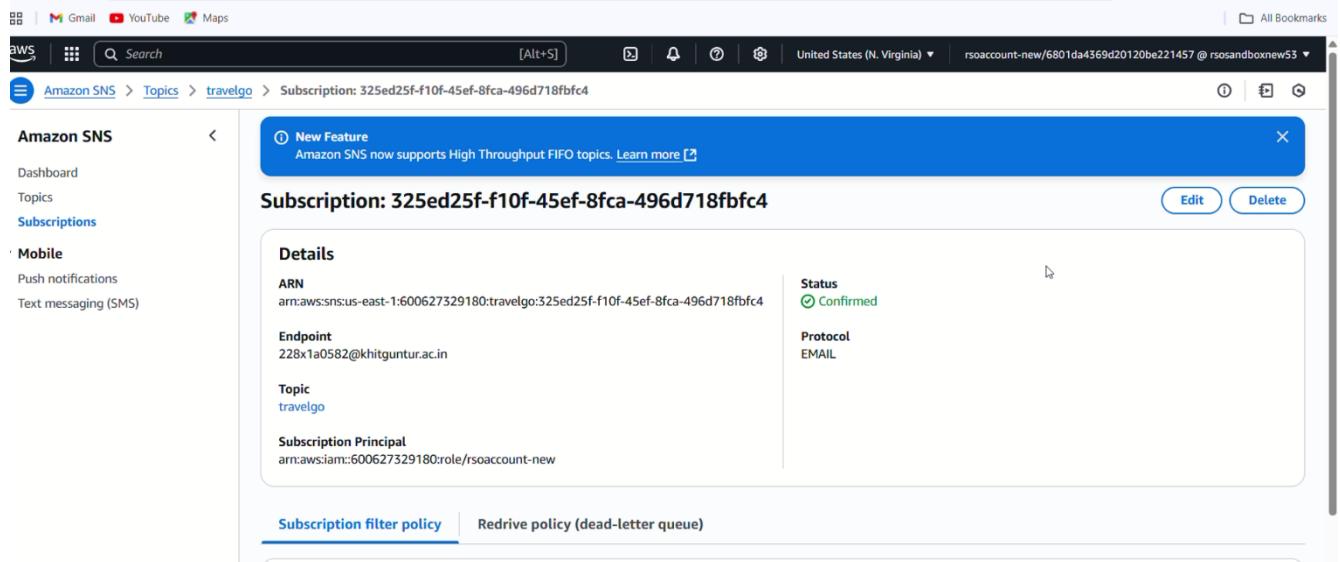
The screenshot shows the AWS SNS Topics page. A success message at the top right states: "Topic travelgo created successfully. You can create subscriptions and send messages to them from this topic." Below this, the "travelgo" topic details are shown, including its ARN (arn:aws:sns:us-east-1:050752624352:travelgo) and type (Standard). The "Subscriptions" tab is selected, showing 0 subscriptions. Other tabs include Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags.

- 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.**



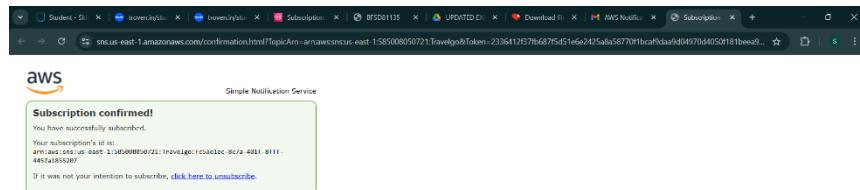
The screenshot shows the AWS SNS Topics page for the 'travelgo' topic. The "Subscriptions" tab is selected, showing 0 subscriptions. The table header includes columns for ID, Endpoint, Status, and Protocol. A message at the bottom of the table area says: "No subscriptions found. You don't have any subscriptions to this topic." The "Create subscription" button is visible 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 address bar is `rsoaccount-new/6801da4369d20120be221457 @ rsoandboxnew53`. The page displays a confirmation message: "Amazon SNS now supports High Throughput FIFO topics. Learn more". Below this, the subscription ID is listed as **Subscription: 325ed25f-f10f-45ef-8fca-496d718fbfc4**. The "Details" section shows the ARN (arn:aws:sns:us-east-1:600627329180:travelgo:325ed25f-f10f-45ef-8fca-496d718fbfc4), Endpoint (228x1a0582@khitguntur.ac.in), Topic (travelgo), and Subscription Principal (arn:aws:iam::600627329180:role/rsoaccount-new). The status is shown as **Status Confirmed** and **Protocol EMAIL**. There are "Edit" and "Delete" buttons at the top right of the main card.

After subscription request for the mail confirmation

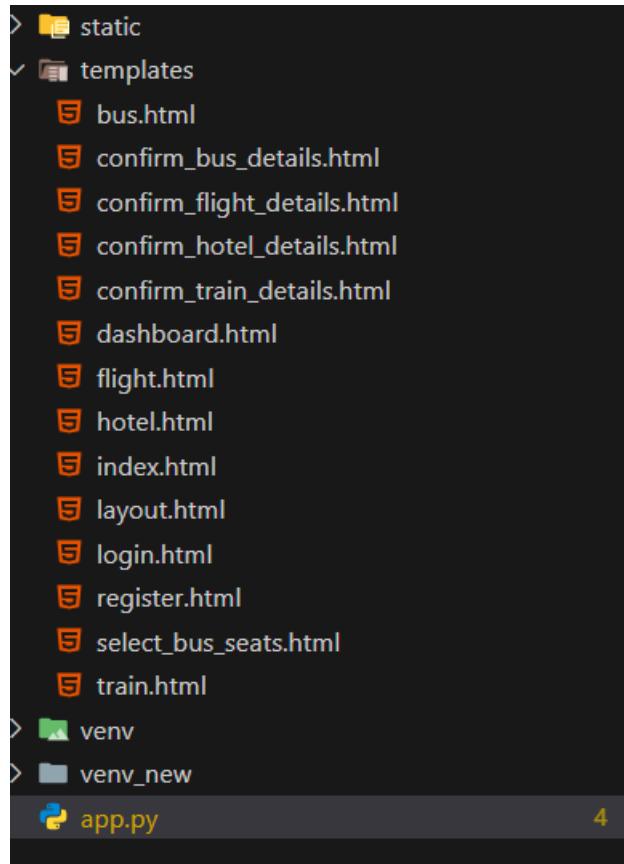


- Successfully done with the SNS mail subscription and setup, now store the ARN link.

## Milestone 4:Backend Development and Application Setup

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

- File Explorer Structure



**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:

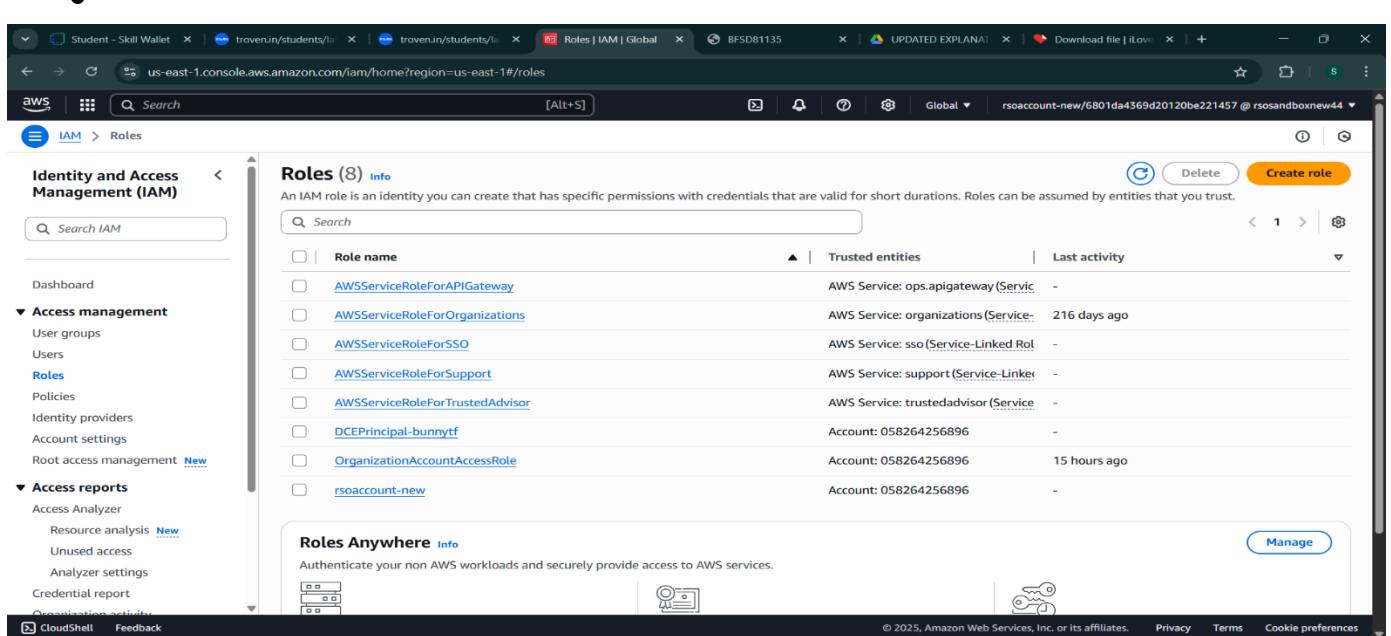
```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

**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

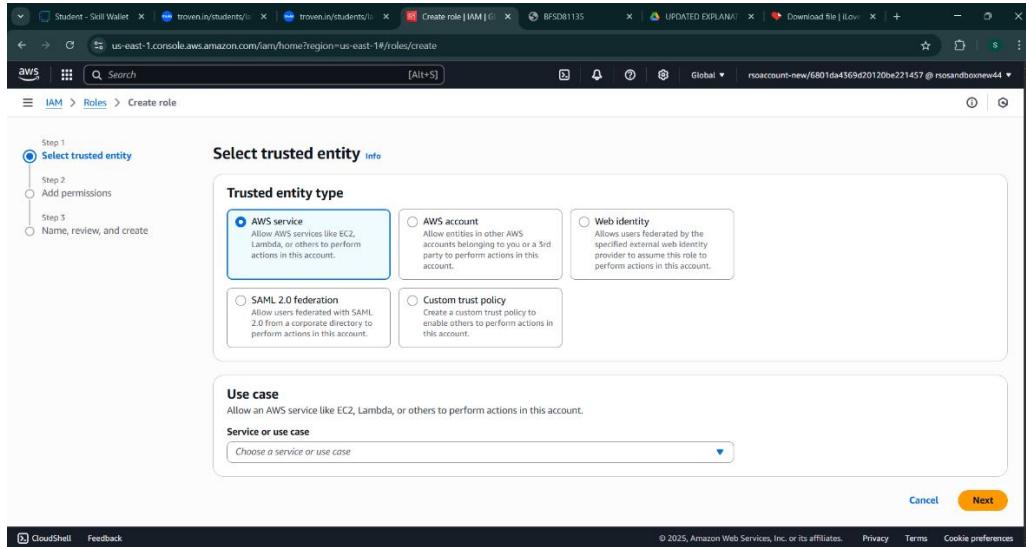
- **Activity 5.1:Create IAM Role.**

- 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

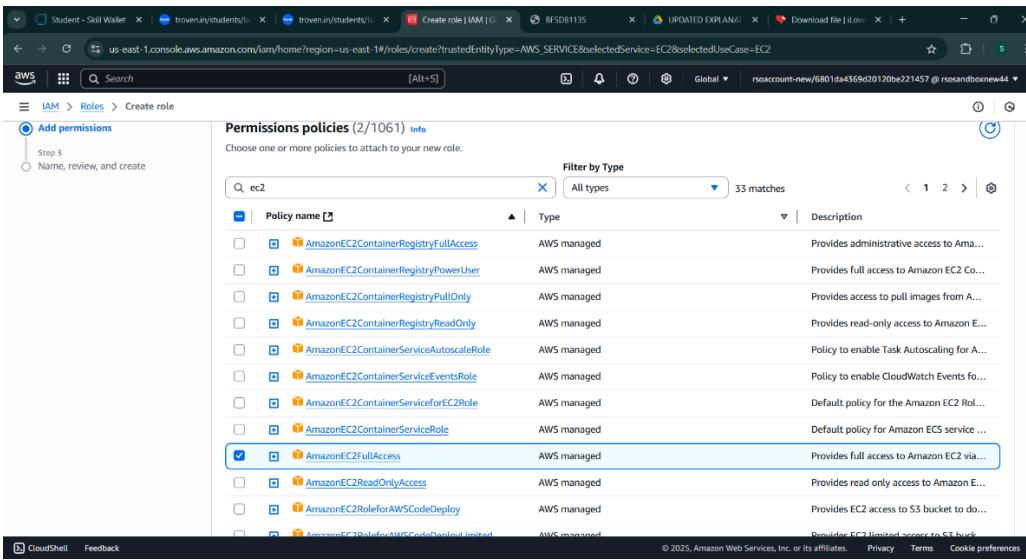
**Select trusted entity** Info

**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 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**



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...

## ● 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 includes:

- AmazonDynamoDBFullAccess**: AWS managed. Provides full access to Amazon DynamoDB.
- AmazonDynamoDBFullAccess\_v2**: AWS managed. Provides full access to Amazon DynamoDB.
- AmazonDynamoDBFullAccesswithDataPipeline**: AWS managed. This policy is on a deprecation path. See [AmazonDynamoDBFullAccess](#).
- AmazonDynamoDBReadOnlyAccess**: AWS managed. Provides read only access to Amazon DynamoDB.
- AWSLambdaDynamoDBExecutionRole**: AWS managed. Provides list and read access to DynamoDB Streams.
- AWSLambdaInvocation-DynamoDB**: AWS managed. Provides read access to DynamoDB Streams.

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 includes:

- AmazonSNSFullAccess**: AWS managed. Provides full access to Amazon SNS via the AWS Management Console and AWS CLI.
- AmazonSNSReadOnlyAccess**: AWS managed. Provides read only access to Amazon SNS.
- AmazonSNSRole**: AWS managed. Default policy for Amazon SNS service role.
- AWSElasticBeanstalkRoleSNS**: AWS managed. (Elastic Beanstalk operations role) Allows the Elastic Beanstalk service to publish messages to SNS topics.
- AWSIoTDeviceDefenderPublishFindingsToSN...**: AWS managed. Provides messages publish access to SNS topics.

Buttons at the bottom: Cancel, Previous, Next.

## Milestone 6: EC2 Instance Setup

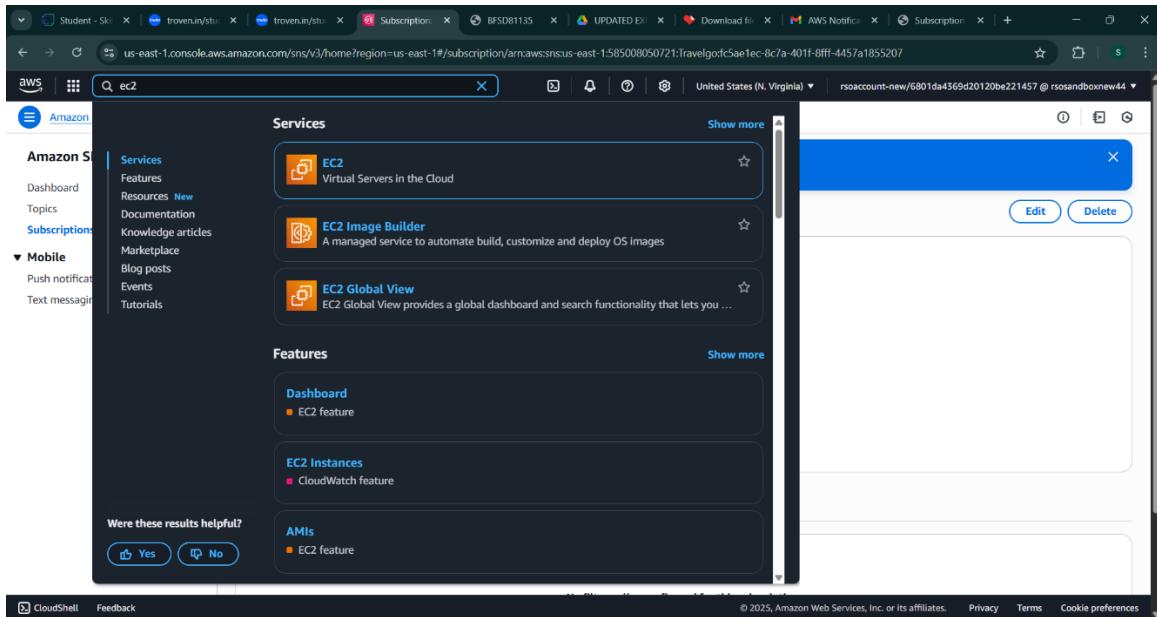
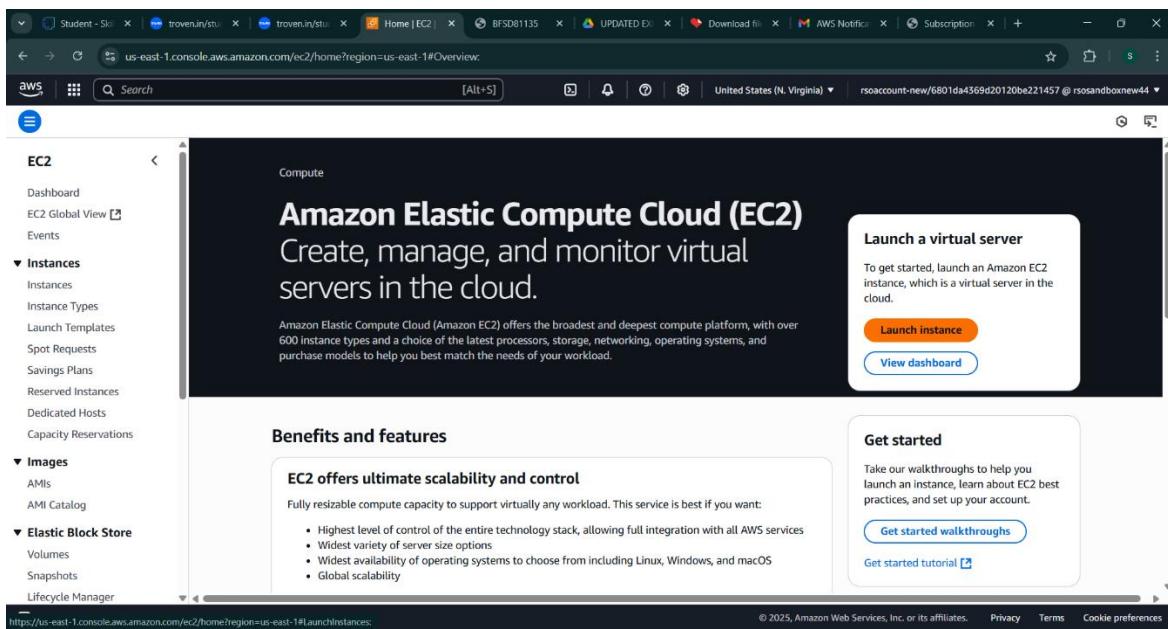
- Note: Load your Flask app and Html files into GitHub repository.

 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.**

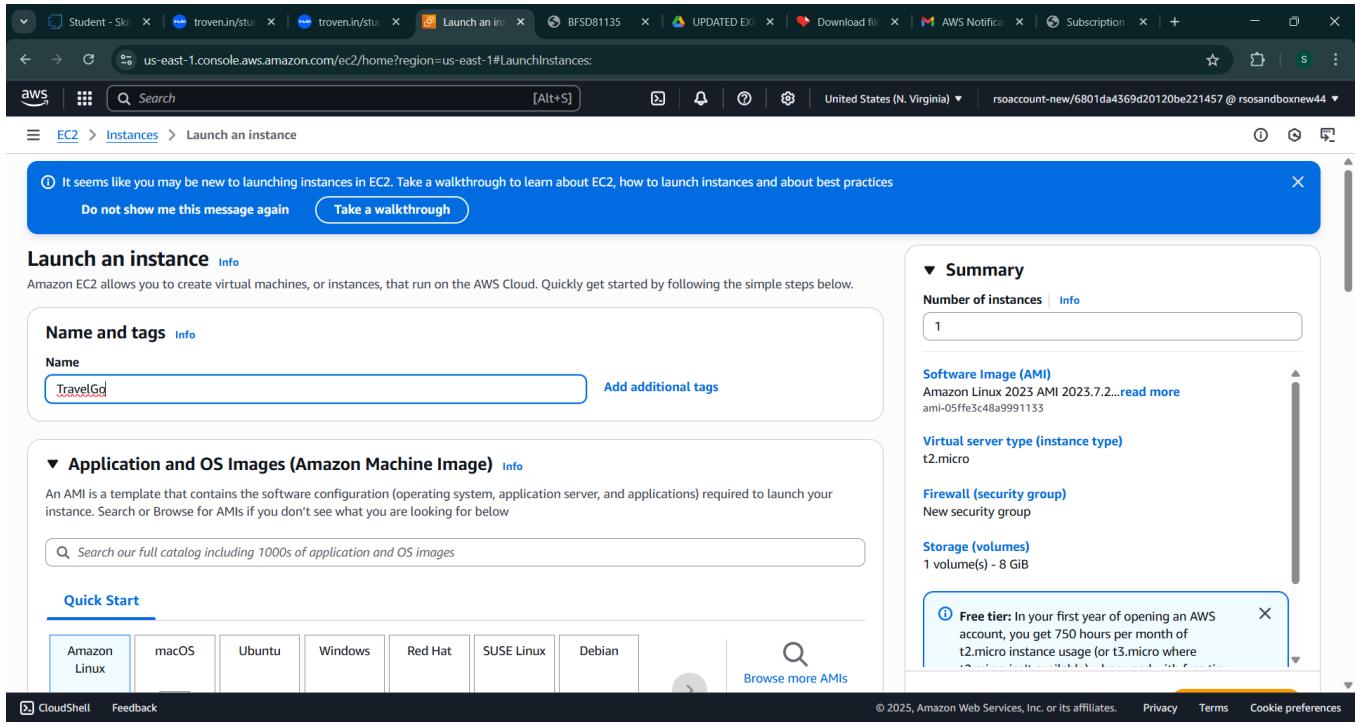
- **Launch EC2 Instance**

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

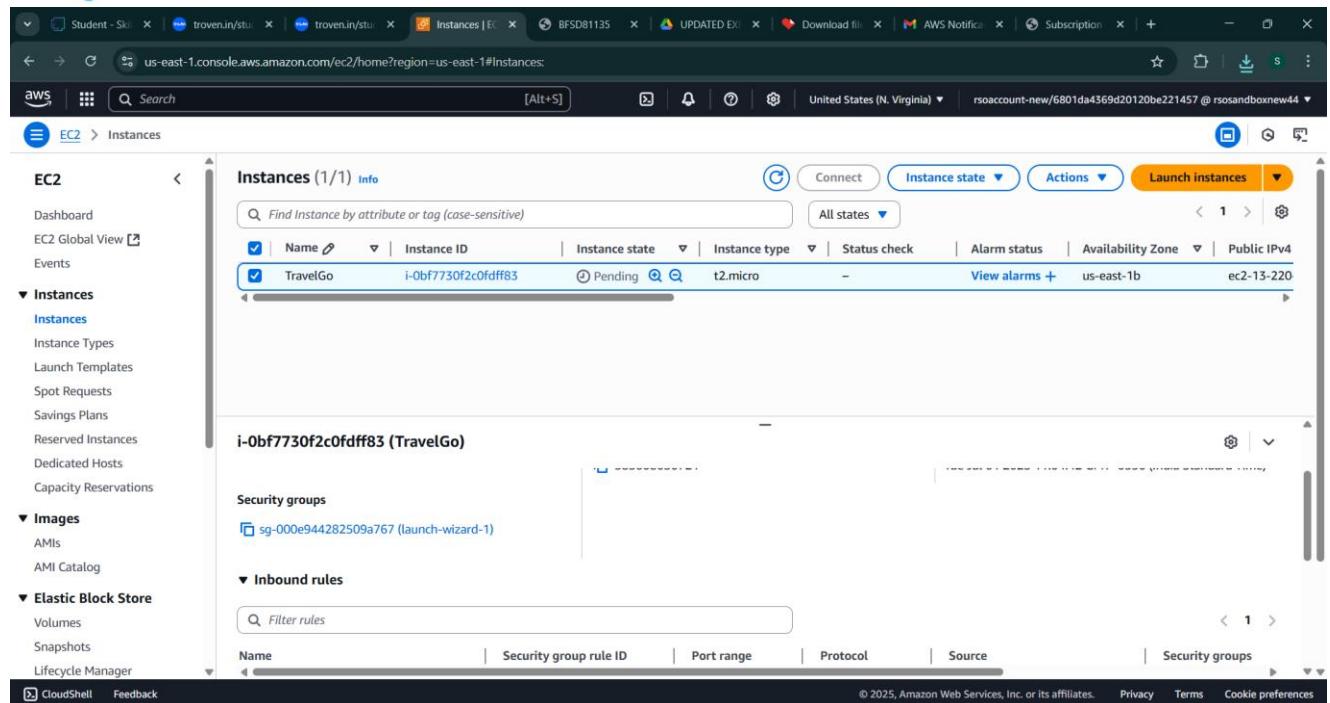
- Click on Launch instance to launch EC2 instance

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



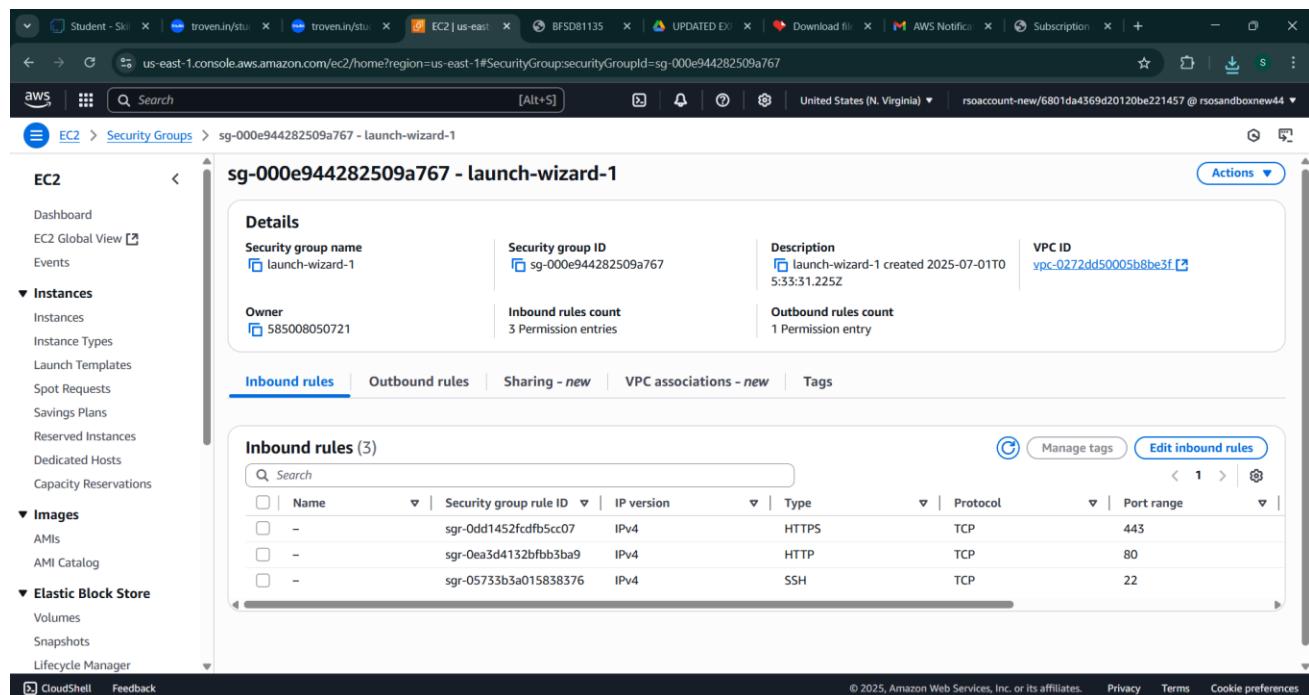
The screenshot shows the AWS EC2 'Launch an instance' wizard. The first step, 'Name and tags', has 'TravelGd' entered in the 'Name' field. The second step, 'Application and OS Images (Amazon Machine Image)', shows 'Amazon Linux' selected from a list. The third step, 'Summary', shows the configuration: 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 sidebar, under the EC2 section, the 'Instances' option is selected. The main content area displays a table titled 'Instances (1/1) Info' with one row. The row details an instance named 'TravelGo' with the ID 'i-0bf7730f2c0fdff83'. The instance is currently 'Pending' and is of type 't2.micro'. It is associated with the 'sg-000e944282509a767 (launch-wizard-1)' security group. The table includes columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4.

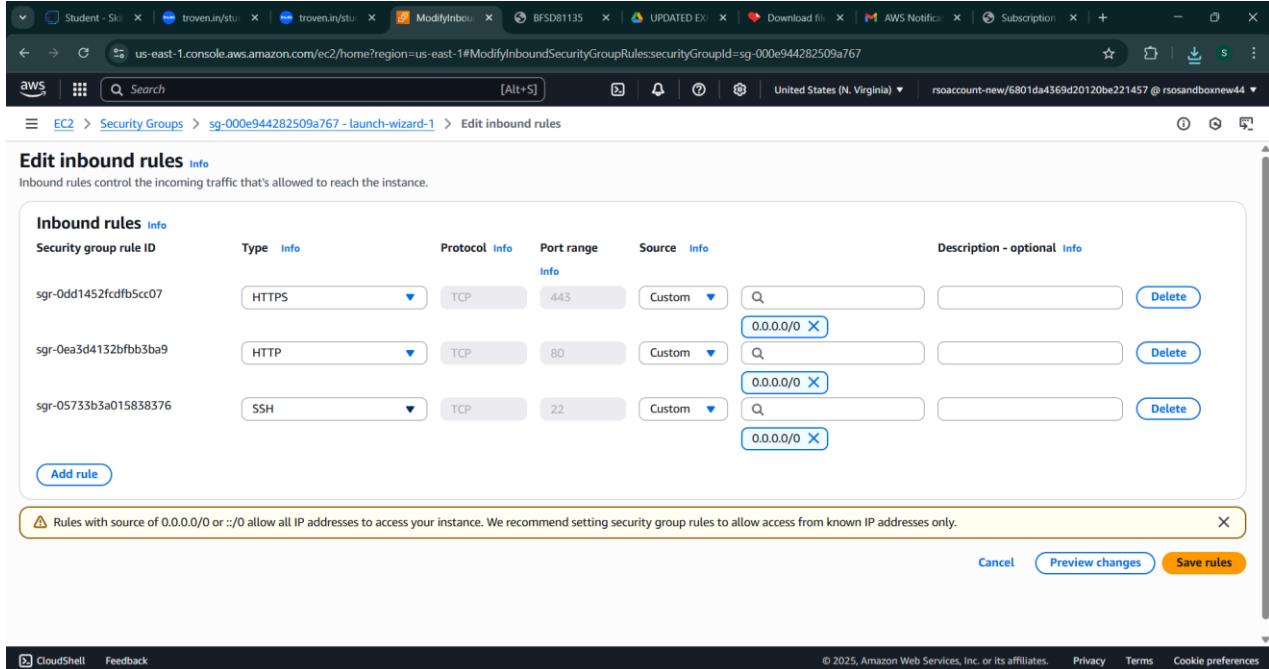
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**



The screenshot shows the AWS Security Groups page. Under the 'Instances' section of the sidebar, the 'Security Groups' option is selected. The main content area shows the details for the security group 'sg-000e944282509a767 - launch-wizard-1'. The 'Details' section provides information such as the security group name ('launch-wizard-1'), security group ID ('sg-000e944282509a767'), description ('launch-wizard-1 created 2025-07-01T05:33:31Z'), and VPC ID ('vpc-0272dd50005b8be3f'). The 'Inbound rules' tab is selected, showing three entries:

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

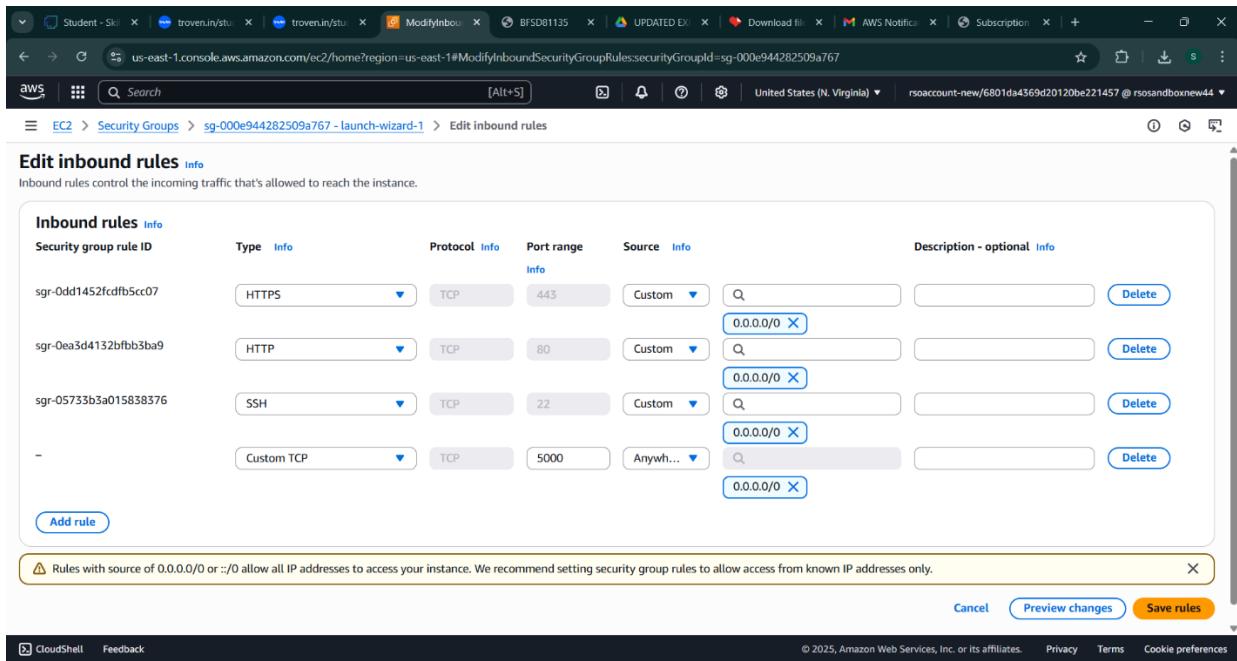
Inbound rules control the incoming traffic that's allowed to reach the instance.

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: red;">X</span> <span style="float: right;"><a href="#">Delete</a></span>
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span> <span style="float: right;"><a href="#">Delete</a></span>
sgr-05733b3a015838376	SSH	TCP	22	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span> <span style="float: right;"><a href="#">Delete</a></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. X

[Cancel](#) [Preview changes](#) **Save rules**



**Edit inbound rules** Info

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

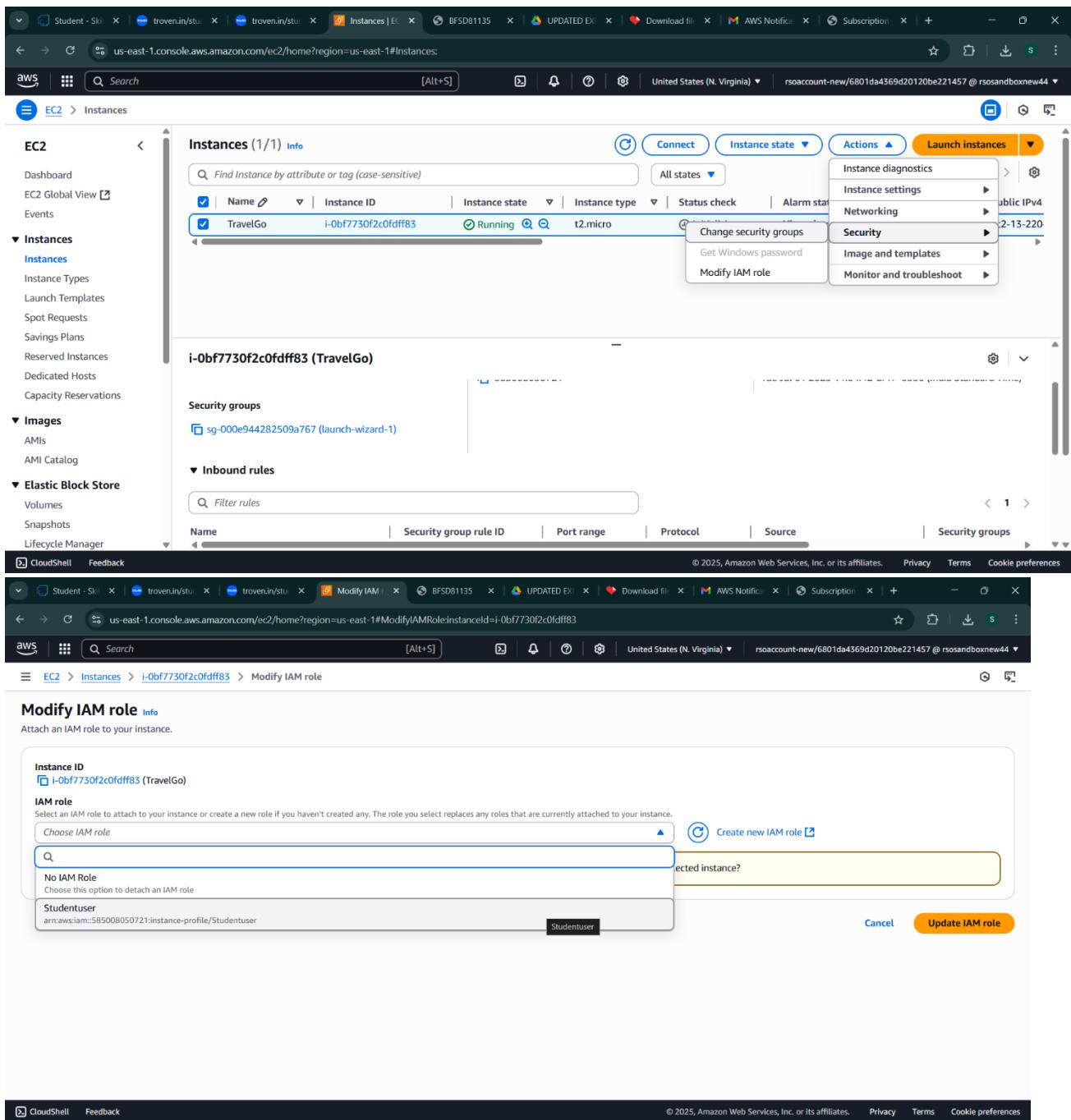
Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdfb5cc07	HTTPS	TCP	443	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span> <span style="float: right;"><a href="#">Delete</a></span>
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	<input type="text"/>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

[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, where a new IAM role named 'Studentuser' is being assigned.

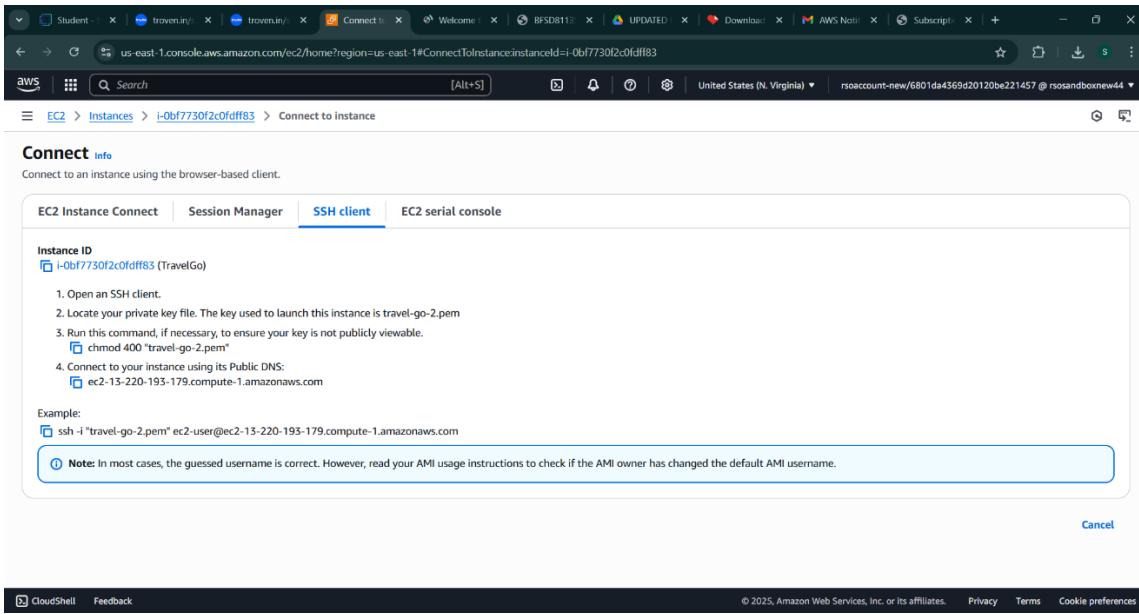
**AWS Management Console - EC2 Instances Page:**

- Instances (1/1)**: Shows the 'TravelGo' instance (i-0bf7730f2c0fdff83).
- Actions** dropdown menu: Includes options like 'Instance diagnostics', 'Instance settings', 'Networking', 'Security' (selected), 'Get Windows password', 'Image and templates', and 'Monitor and troubleshoot'.

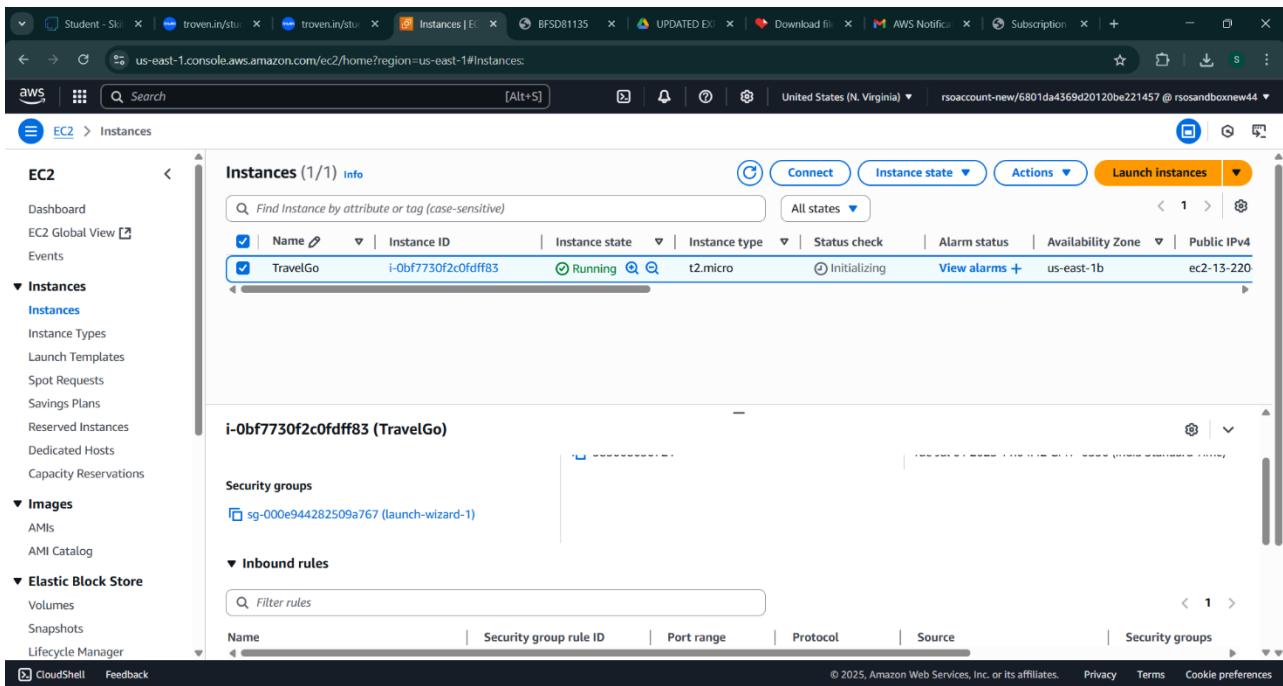
**Modify IAM role Dialog:**

- Instance ID:** i-0bf7730f2c0fdff83 (TravelGo)
- IAM role:** A dropdown menu where 'Studentuser' is selected. Other options include 'Choose IAM role' and 'Create new IAM role'.
- Attached instance?**: A question asking if the selected role should be attached to the instance.
- Role details:** Shows 'arn:aws:iam::585008050721:instance-profile/Studentuser'.
- Buttons:** 'Cancel' and 'Update IAM role'.

- 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 steps like opening an SSH client, locating the private key file, running commands to ensure the key is not publicly viewable, and connecting using the 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 is titled "Instances (1/1) Info". It lists one instance: "TravelGo" (i-0bf7730f2c0fdff83), which is "Running" and has an "t2.micro" instance type. The "Actions" dropdown menu is open, showing options like "Launch instances", "Stop", "Start", "Reboot", "Terminate", and "Edit". Below the table, the instance details for "i-0bf7730f2c0fdff83 (TravelGo)" are shown, including its security group ("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/Mohan-143-coder/travelgo.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:**

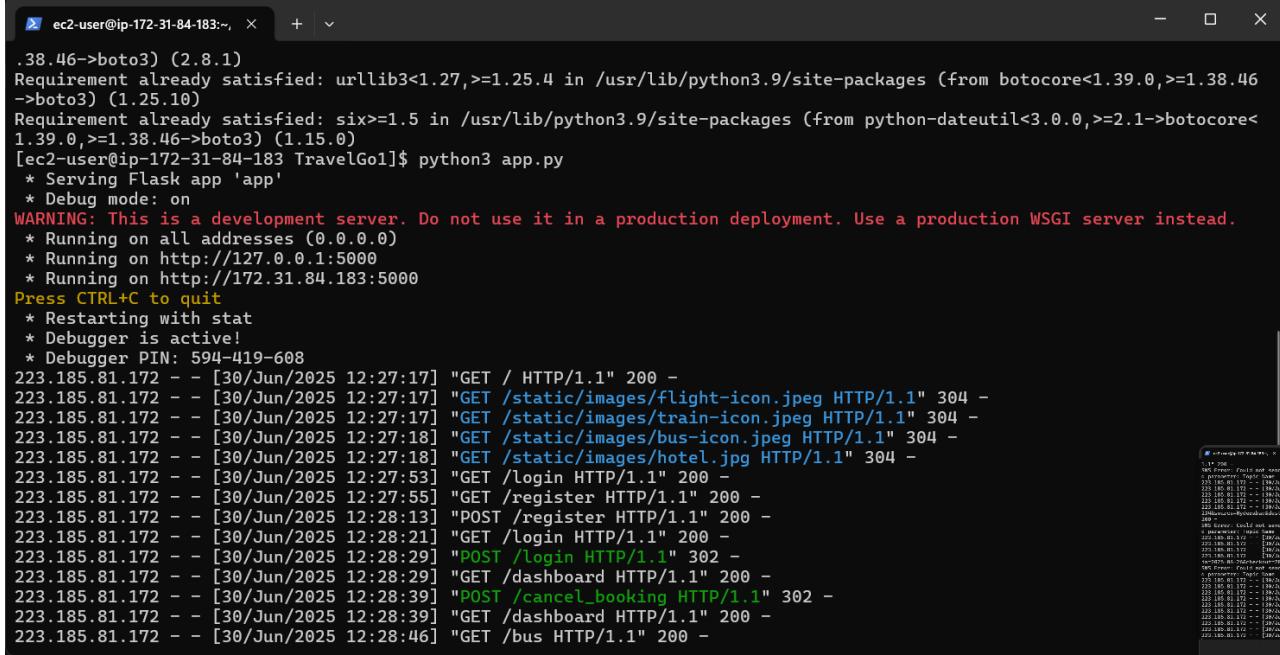
**Run the Flask Application**

```
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



```
.38.46->boto3) (2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.39.0,>=1.38.46->boto3) (1.25.10)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.39.0,>=1.38.46->boto3) (1.15.0)
[ec2-user@ip-172-31-84-183 TravelGo1]$ 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://172.31.84.183:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 594-419-608
223.185.81.172 - - [30/Jun/2025 12:27:17] "GET / HTTP/1.1" 200 -
223.185.81.172 - - [30/Jun/2025 12:27:17] "GET /static/images/flight-icon.jpeg HTTP/1.1" 304 -
223.185.81.172 - - [30/Jun/2025 12:27:17] "GET /static/images/train-icon.jpeg HTTP/1.1" 304 -
223.185.81.172 - - [30/Jun/2025 12:27:18] "GET /static/images/bus-icon.jpeg HTTP/1.1" 304 -
223.185.81.172 - - [30/Jun/2025 12:27:18] "GET /static/images/hotel.jpg HTTP/1.1" 304 -
223.185.81.172 - - [30/Jun/2025 12:27:53] "GET /login HTTP/1.1" 200 -
223.185.81.172 - - [30/Jun/2025 12:27:55] "GET /register HTTP/1.1" 200 -
223.185.81.172 - - [30/Jun/2025 12:28:13] "POST /register HTTP/1.1" 200 -
223.185.81.172 - - [30/Jun/2025 12:28:21] "GET /login HTTP/1.1" 200 -
223.185.81.172 - - [30/Jun/2025 12:28:29] "POST /Login HTTP/1.1" 302 -
223.185.81.172 - - [30/Jun/2025 12:28:29] "GET /dashboard HTTP/1.1" 200 -
223.185.81.172 - - [30/Jun/2025 12:28:39] "POST /cancel_booking HTTP/1.1" 302 -
223.185.81.172 - - [30/Jun/2025 12:28:39] "GET /dashboard HTTP/1.1" 200 -
223.185.81.172 - - [30/Jun/2025 12:28:46] "GET /bus HTTP/1.1" 200 -
```

## 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.

