

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.

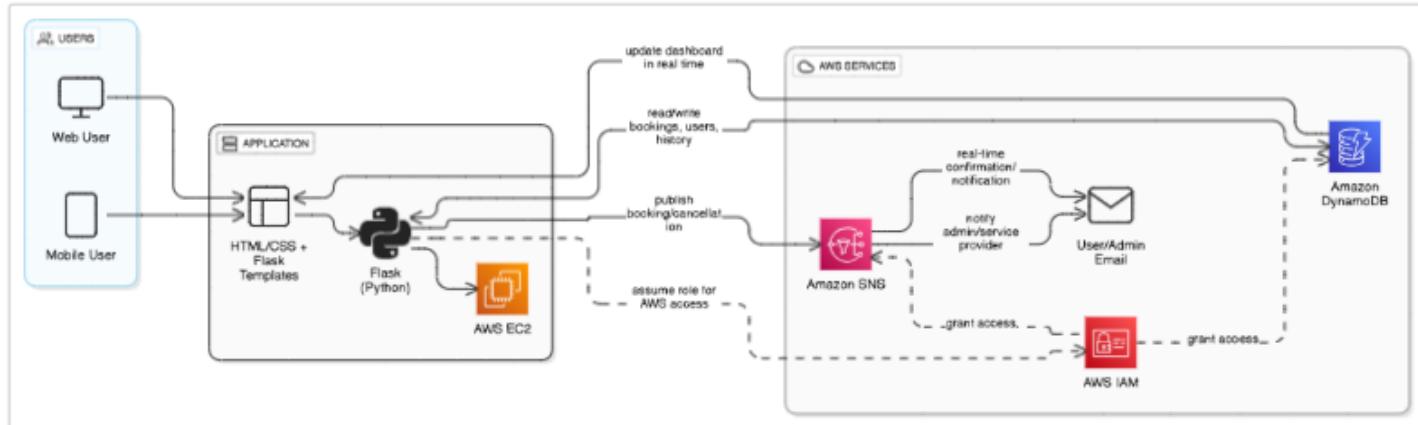
Scenario 2: Real-Time Booking Confirmation with AWS SNS

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.

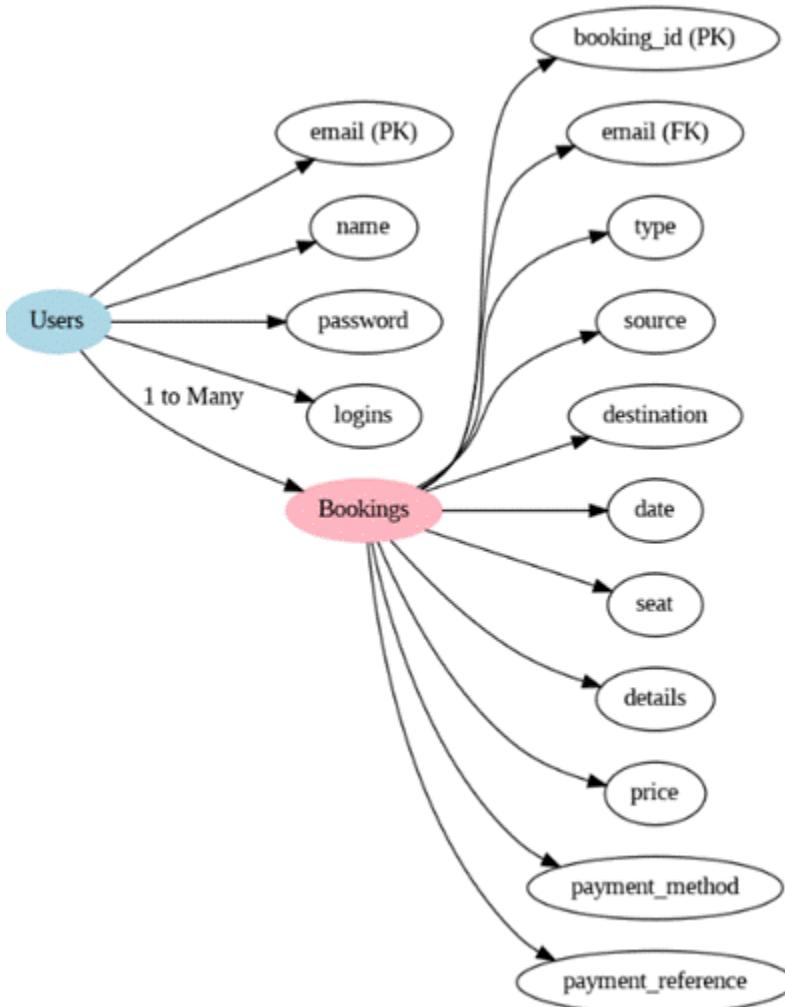
Scenario 3: Dynamic Dashboard with Personal Travel History

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

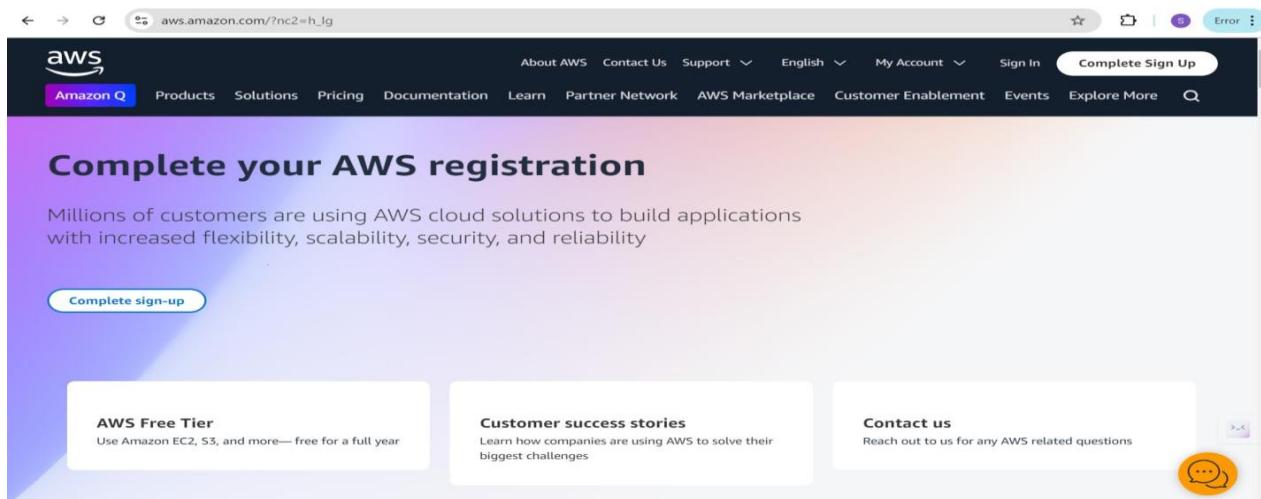
Activity 7.2: Run the Flask App

8. Testing and Deployment

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

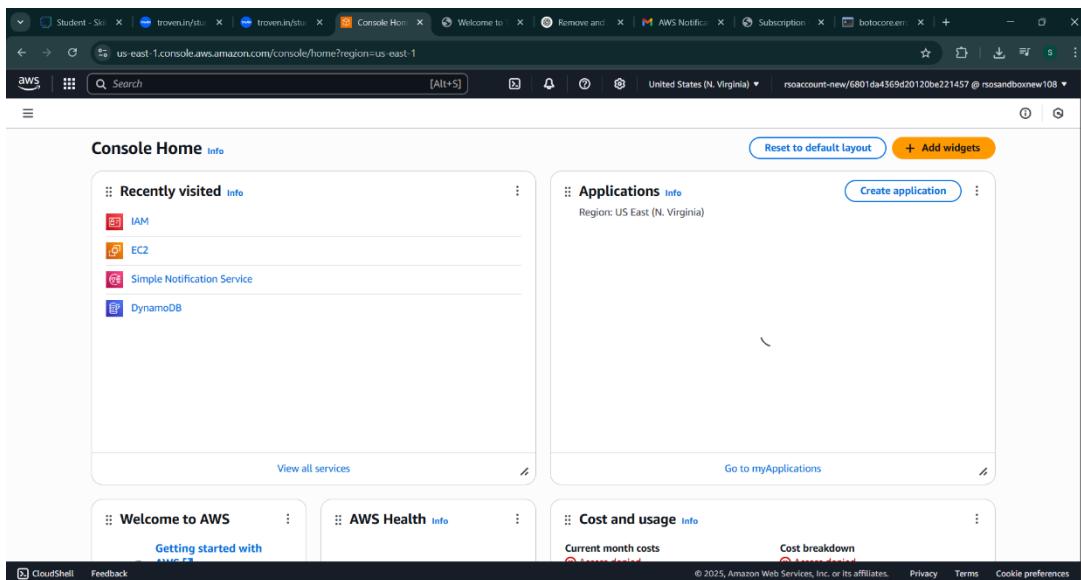
Milestone 1: AWS Account Setup and Login

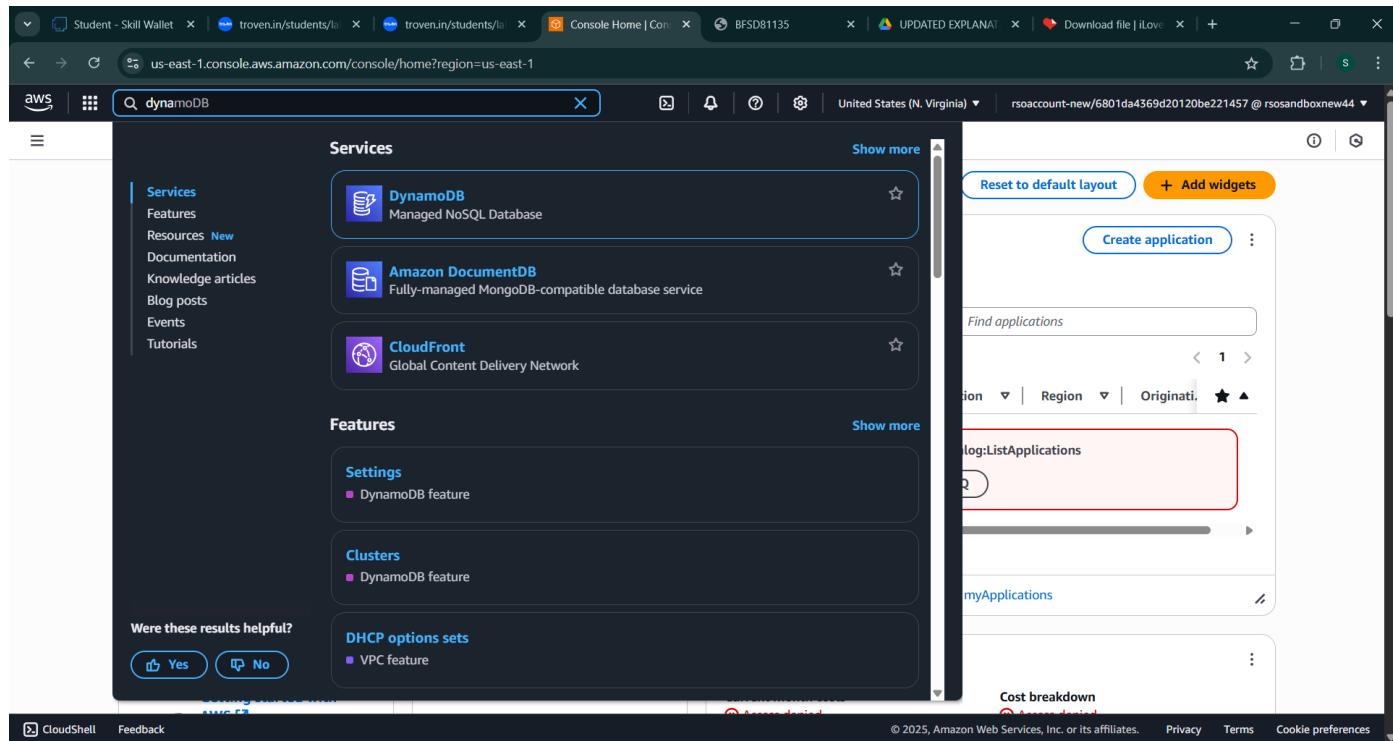
- **Activity 1.1: Set up an AWS account if not already done.**
 - Sign up for an AWS account and configure billing settings.



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

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



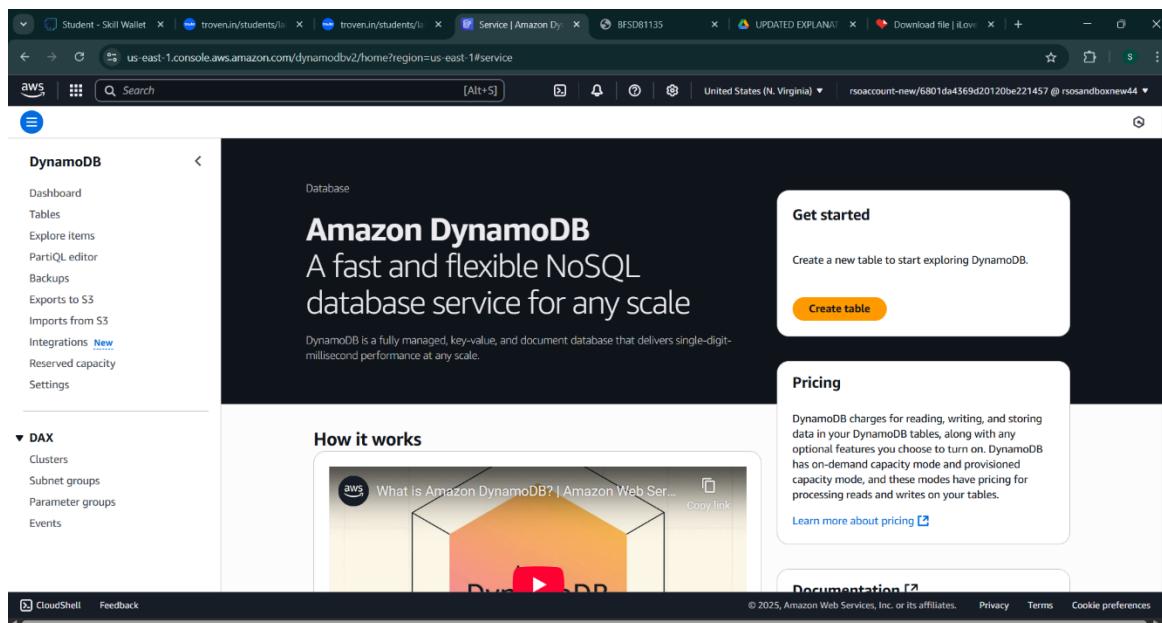


The screenshot shows the AWS Management Console search results for 'dynamoDB'. The top navigation bar includes tabs for 'Student - Skill Wallet', 'troven.in/students/...', 'troven.in/students/...', 'Console Home | Con...', 'BFSD81135', 'UPDATED EXPLAIN...', 'Download file | iLove...', and a '+' button. 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.



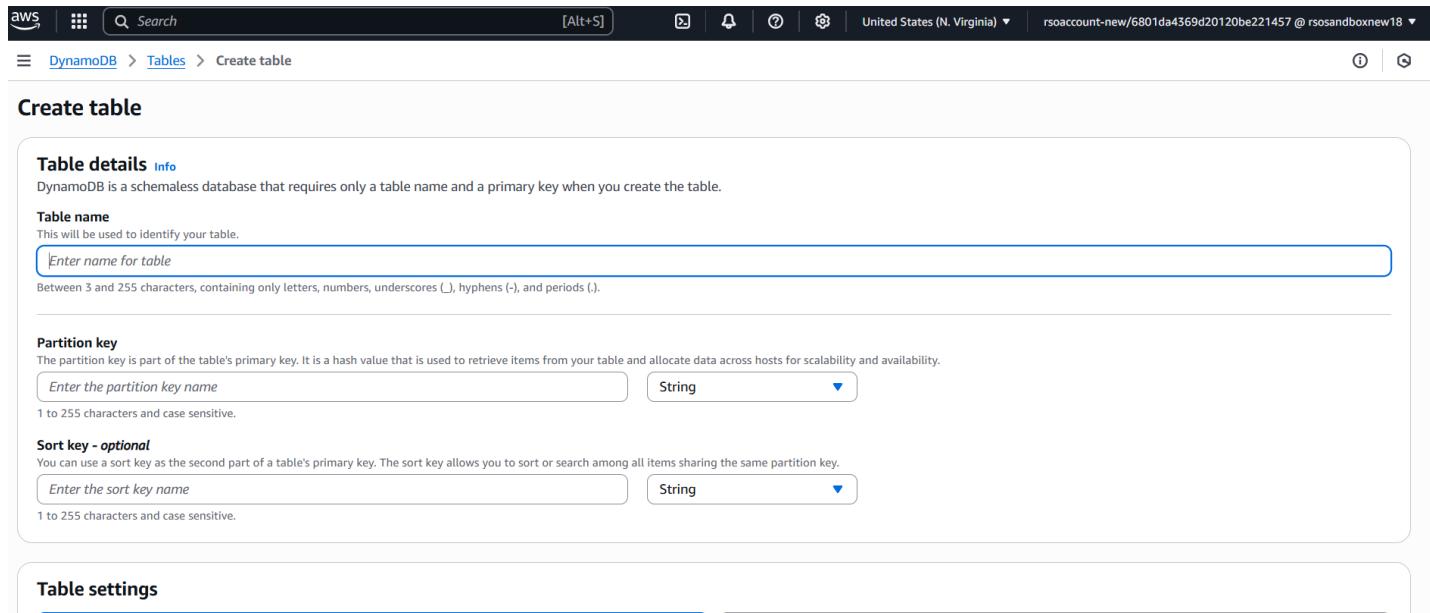
The screenshot shows the 'Amazon DynamoDB' service page. The top navigation bar includes tabs for 'Student - Skill Wallet', 'troven.in/students/...', 'troven.in/students/...', 'Service | Amazon Dy...', 'BFSD81135', 'UPDATED EXPLAIN...', 'Download file | iLove...', and a '+' button. The search bar contains '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 Services'. 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 'Create table' wizard in the AWS DynamoDB console. The 'Table details' step is active. The 'Table name' field is populated with 'Enter name for table'. The 'Partition key' section shows 'Enter the partition key name' and 'String' selected. The 'Sort key - optional' section shows 'Enter the sort key name' and 'String' selected. Below these sections is the 'Table settings' section, which is currently collapsed.



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.

Screenshot of the AWS DynamoDB 'Create table' page.

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.
 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.
 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' page.

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.
 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.
 1 to 255 characters and case sensitive.

Table settings

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.

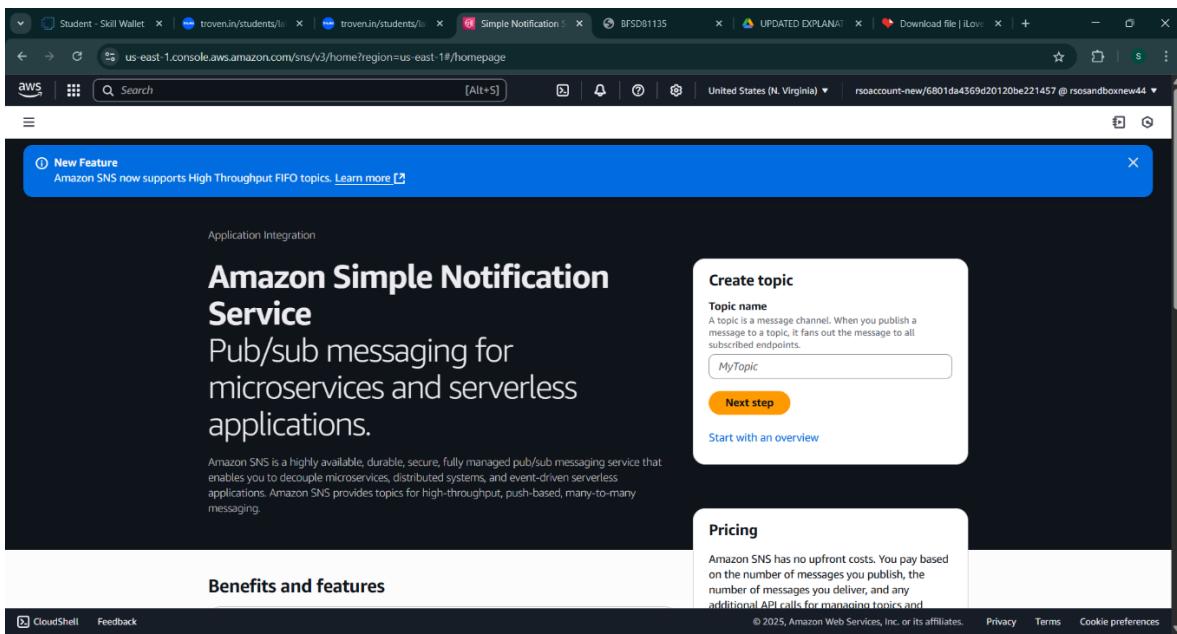
The left sidebar shows the navigation menu for SNS, including Services, Features, Resources, Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials.

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.

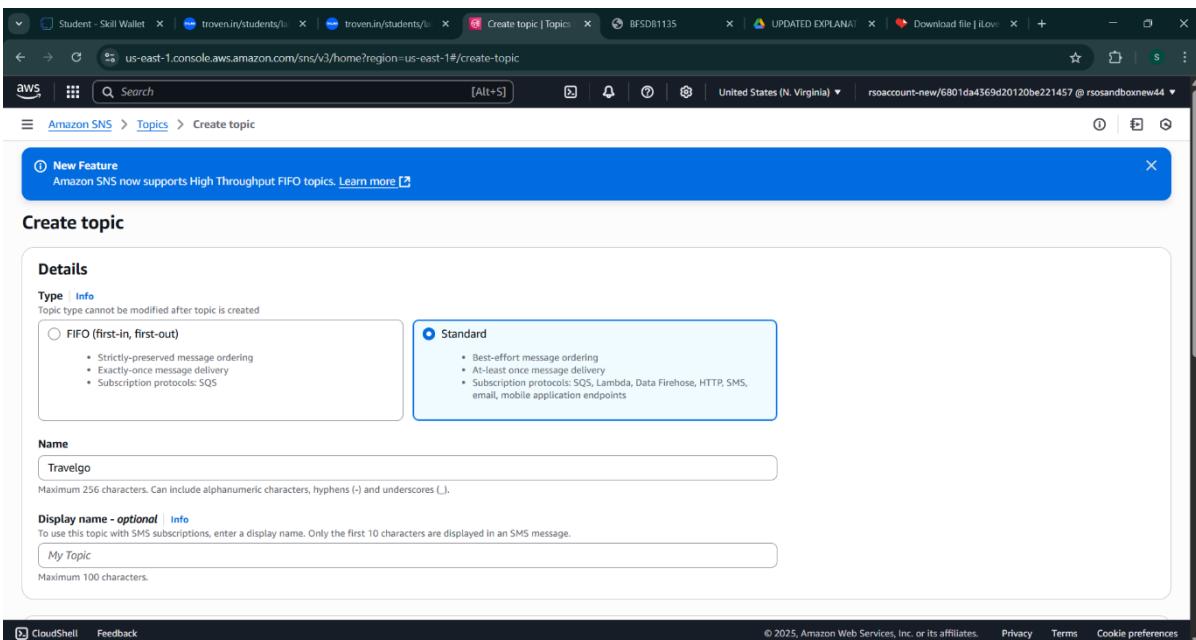
A modal window titled 'Create table' is open on the right side of the screen, showing fields for Region, Deletion protection, and Read capacity units.

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



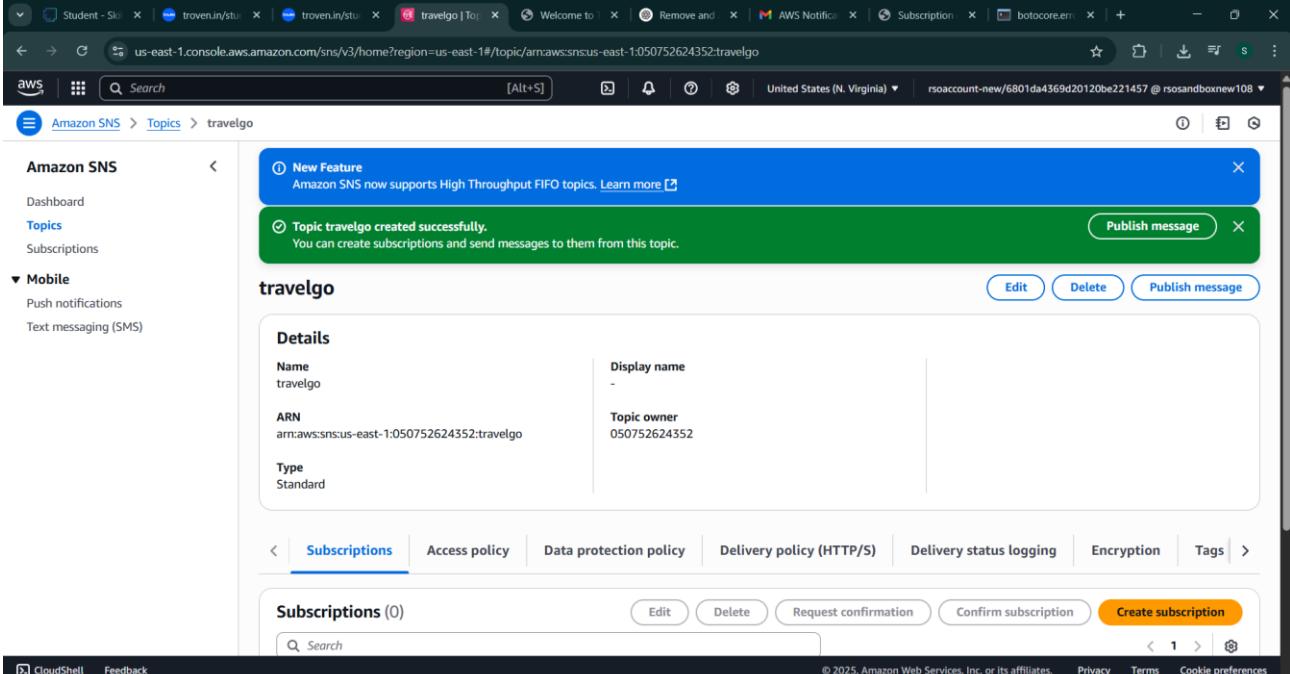
The screenshot shows the AWS SNS dashboard. At the top, there is a blue banner with the text "Amazon SNS now supports High Throughput FIFO topics. Learn more". Below the banner, the main heading is "Amazon Simple Notification Service" with the subtitle "Pub/sub messaging for microservices and serverless applications". To the right, there is a "Create topic" dialog box. Inside the dialog, there is a "Topic name" input field containing "MyTopic", a "Next step" button, and a link "Start with an overview". Below the dialog, there is a "Pricing" section with a note about no upfront costs based on message volume. At the bottom of the dashboard, there is a "Benefits and features" section and a navigation bar with links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

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



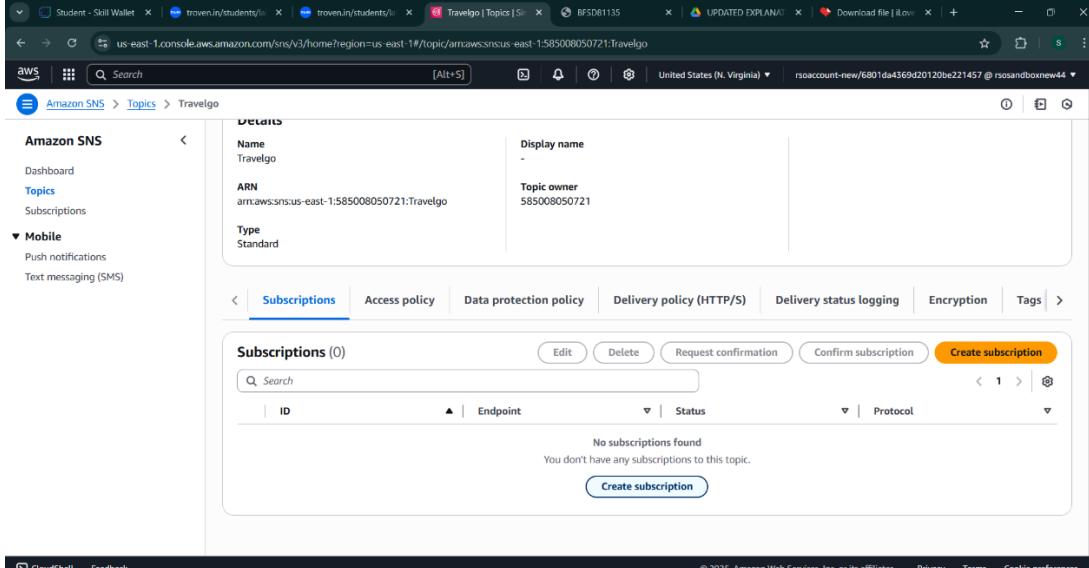
The screenshot shows the "Create topic" configuration page. At the top, there is a blue banner with the text "Amazon SNS now supports High Throughput FIFO topics. Learn more". Below the banner, the heading is "Create topic". There are two tabs: "Details" (selected) and "Info". Under "Details", there is a "Type" section with two options: "FIFO (first-in, first-out)" and "Standard". The "Standard" option is selected and highlighted with a blue border. Below the type section, there is a "Name" input field containing "Travelgo", a "Display name - optional" input field containing "My Topic", and a note stating "Maximum 100 characters.". At the bottom of the page, there is a "Create topic" button and a note about topic type being immutable after creation. The footer includes links for CloudShell, Feedback, 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.**



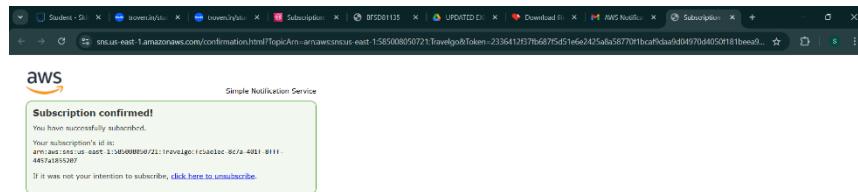
The screenshot shows the AWS SNS Topics page with the "Subscriptions" tab selected. It displays a table with columns for ID, Endpoint, Status, and Protocol. A message at the bottom says "No subscriptions found. You don't have any subscriptions to this topic." There is a "Create subscription" button at 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 Amazon SNS 'Topics' section. On the left, a sidebar lists 'Amazon SNS' with a back arrow, 'Topics' (which is selected and highlighted in blue), and 'Subscriptions'. Below these are sections for 'Mobile' (Push notifications, Text messaging (SMS)) and 'AWS Lambda' (Lambda functions). The main content area is titled 'Travelgo2' and contains a 'Details' section with fields for Name (Travelgo2), ARN (arn:aws:sns:us-east-1:699475957834:Travelgo2), Display name (-), Topic owner (699475957834), and Type (Standard). Below this are tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags. The 'Subscriptions' tab is active, showing one entry: an endpoint (228x1a05f3@khitguntur.ac.in) with ID 23791d21-5659-43f5-8ba4-cf55f... and Status Confirmed (indicated by a green checkmark). There are buttons for Edit, Delete, Request confirmation, Confirm subscription, and Create subscription.

After subscription request for the mail confirmation

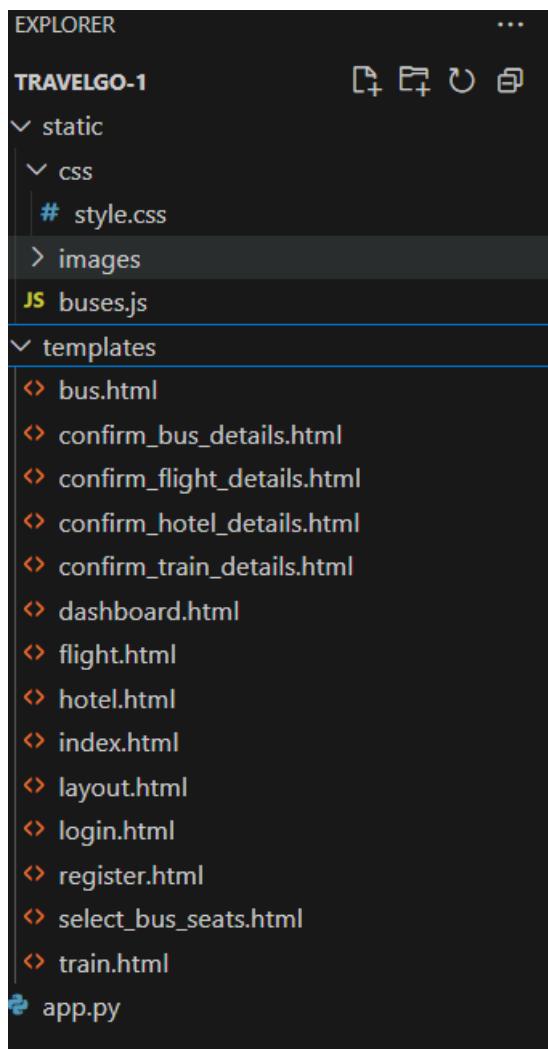


- Successfully done with the SNS mail subscription and setup, now store the ARN link.

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
app = Flask(__name__) #ensure correctly after pushing into github
```

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:**

```
"AWS Setup using IAM Role"
REGION = 'ap-south-1' # Replace with your actual AWS region
dynamodb = boto3.resource('dynamodb', region_name=REGION)
sns_client = boto3.client('sns', region_name=REGION)

users_table = dynamodb.Table('travelgo_users')
trains_table = dynamodb.Table('trains')
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:**

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

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:**

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

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

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = users_table.get_item(Key={'email': email})
        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')
```

```

@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'))

    booking_details = {
        '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': Decimal(request.args.get('price')),
        'travel_date': request.args.get('date'),
        'num_persons': int(request.args.get('persons')),
        'item_id': request.args.get('trainId'),
        'booking_type': 'train',
        'user_email': session['email'],
        'total_price': Decimal(request.args.get('price')) * int(request.args.get('persons'))
    }

    response = bookings_table.query(
        IndexName='GSI_ItemDate',
        KeyConditionExpression=Key('item_id').eq(booking_details['item_id']) & Key('travel_date').eq(booking_details['travel_date'])
    )

    booked_seats = set()
    for b in response.get('Items', []):
        if 'seats_display' in b:
            booked_seats.add(b['seats_display'].split(','))

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

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

    booking_details = {
        'name': request.args.get('name'),
        'source': request.args.get('source'),
        'destination': request.args.get('destination'),
        'time': request.args.get('time'),
        'type': request.args.get('type'),
        'price_per_person': Decimal(request.args.get('price')),
        'travel_date': request.args.get('date'),
        'num_persons': int(request.args.get('persons')),
        'item_id': request.args.get('busId'),
        'booking_type': 'bus',
        'user_email': session['email'],
        'total_price': Decimal(request.args.get('price')) * int(request.args.get('persons'))
    }
    session['pending_booking'] = booking_details
    return render_template('confirm_bus_details.html', booking=booking_details)

```

```

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

@app.route('/confirm_flight_details')
def confirm_flight_details():
    booking = {
        'flight_id': request.args['flight_id'],
        'airline': request.args['airline'],
        'flight_number': request.args['flight_number'],
        'source': request.args['source'],
        'destination': request.args['destination'],
        'departure_time': request.args['departure'],
        'arrival_time': request.args['arrival'],
        'travel_date': request.args['date'],
        'num_persons': int(request.args['passengers']),
        'price_per_person': float(request.args['price']),
    }
    booking['total_price'] = booking['price_per_person'] * booking['num_persons']
    return render_template('confirm_flight_details.html', booking=booking)

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

    booking = {
        'booking_type': 'flight',
        'flight_id': request.form['flight_id'],
        'airline': request.form['airline'],
        'flight_number': request.form['flight_number'],
        'source': request.form['source'],
        'destination': request.form['destination'],
        'departure_time': request.form['departure_time'],
        'arrival_time': request.form['arrival_time'],
    }

```

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:

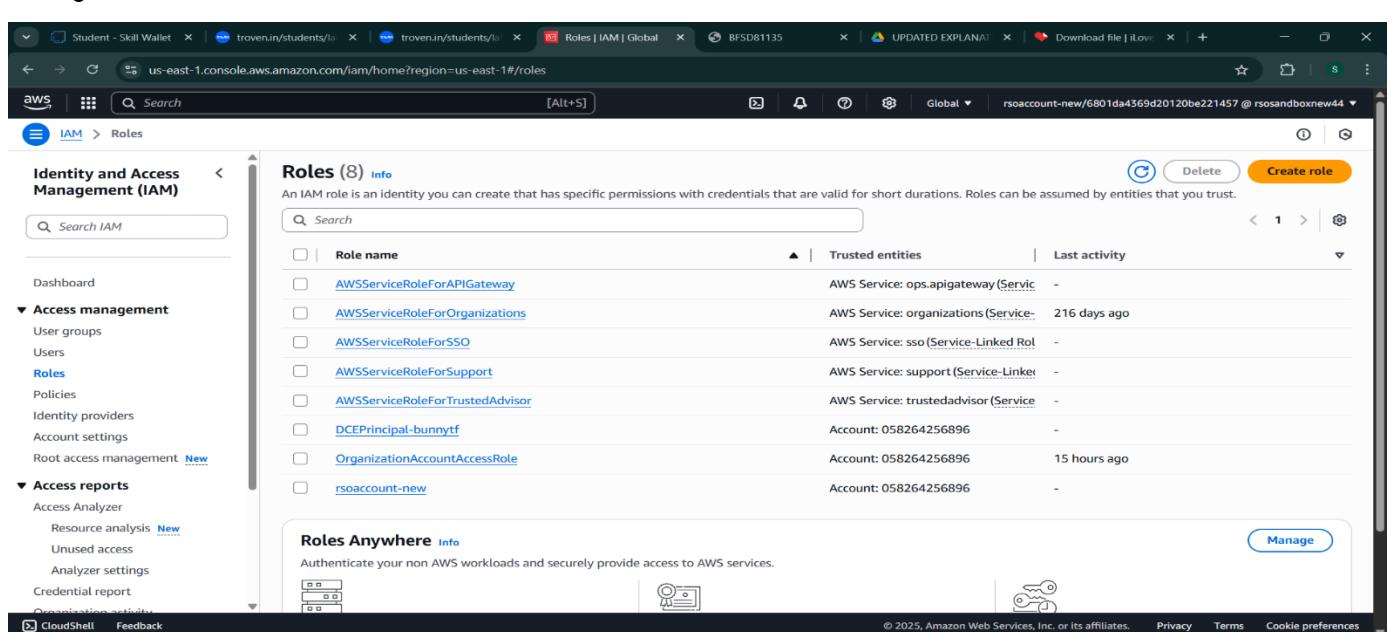
```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

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

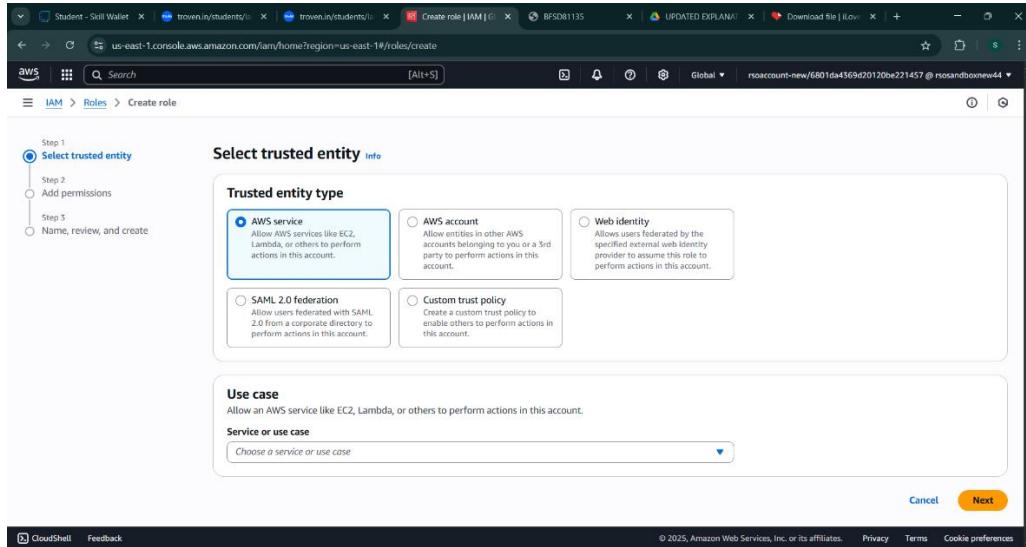
- **Activity 5.1:Create IAM Role.**

- 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

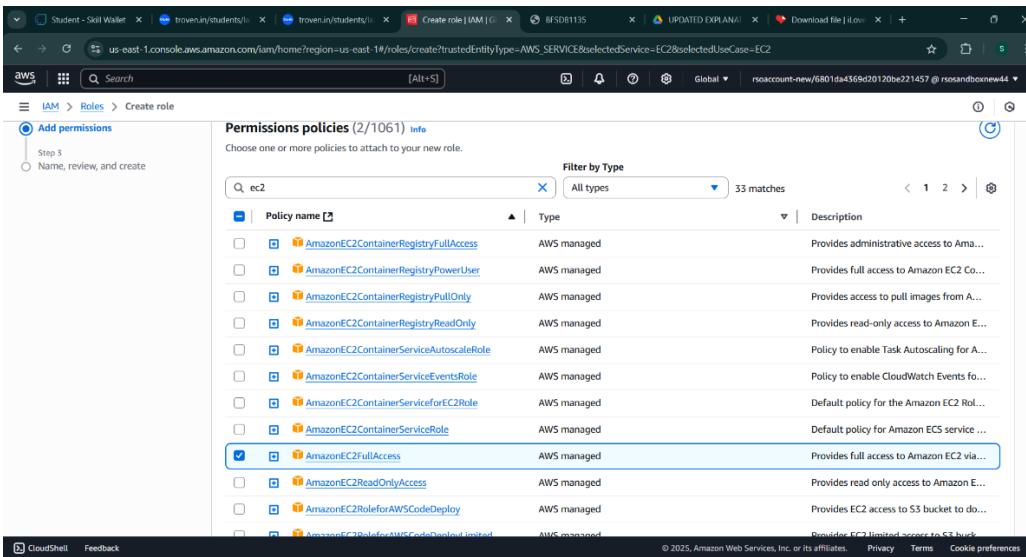
Trusted entity type

- 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 entities associated 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

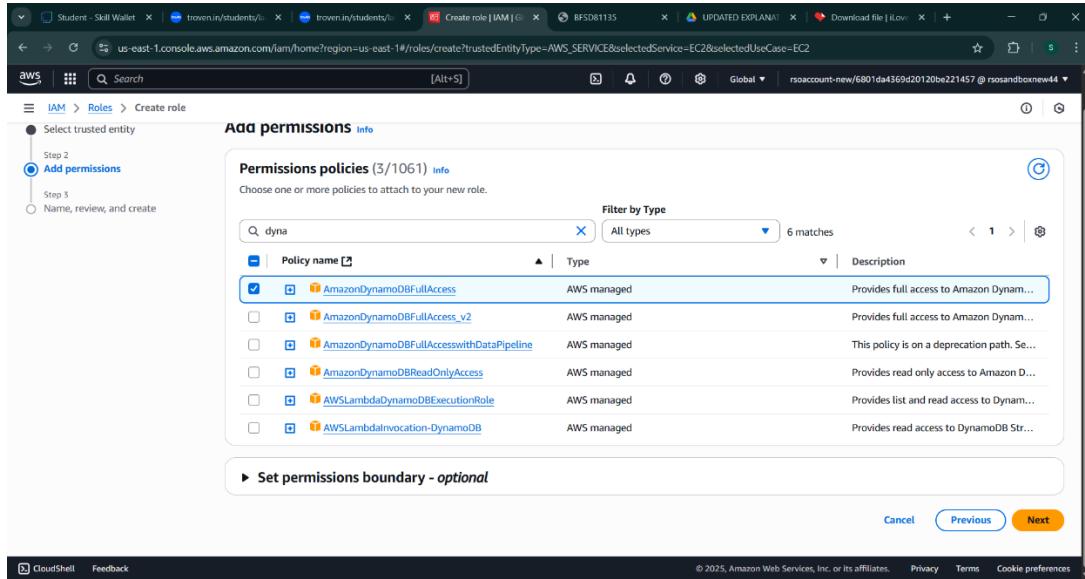
Policy name	Type	Description
AmazonEC2ContainerRegistryFullAccess	AWS managed	Provides administrative access to Amazon E...
AmazonEC2ContainerRegistryPowerUser	AWS managed	Provides full access to Amazon EC2 Co...
AmazonEC2ContainerRegistryPullOnly	AWS managed	Provides access to pull images from A...
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 ...
AmazonEC2FullAccess	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.

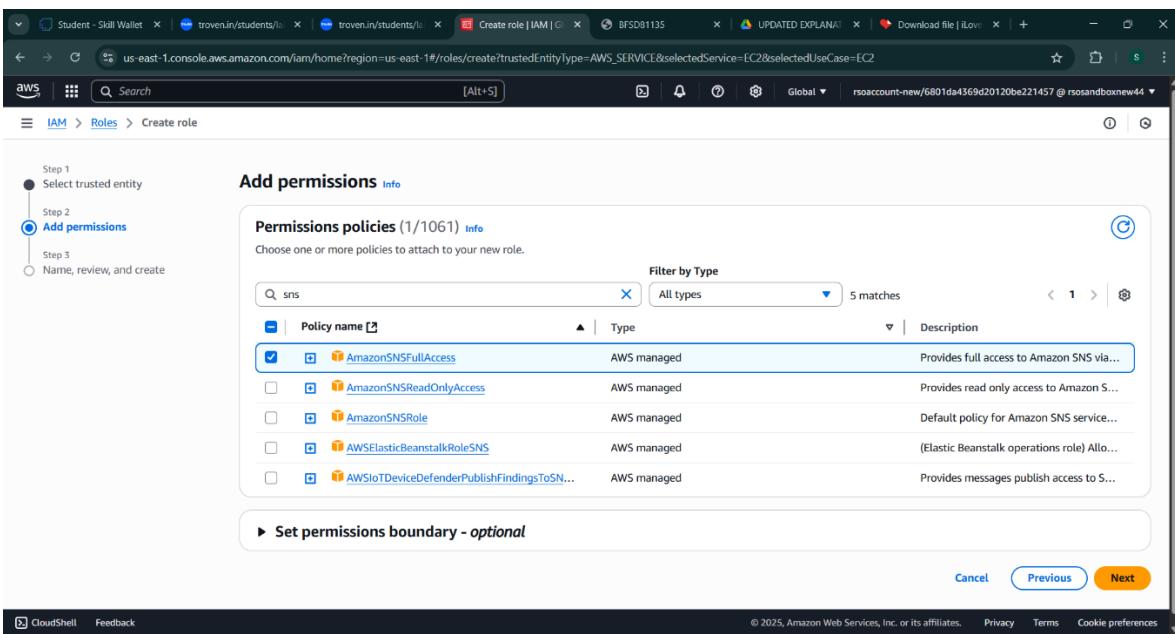
Filter by Type All types 6 matches

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 AmazonDynamoDBFullAccess .
<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

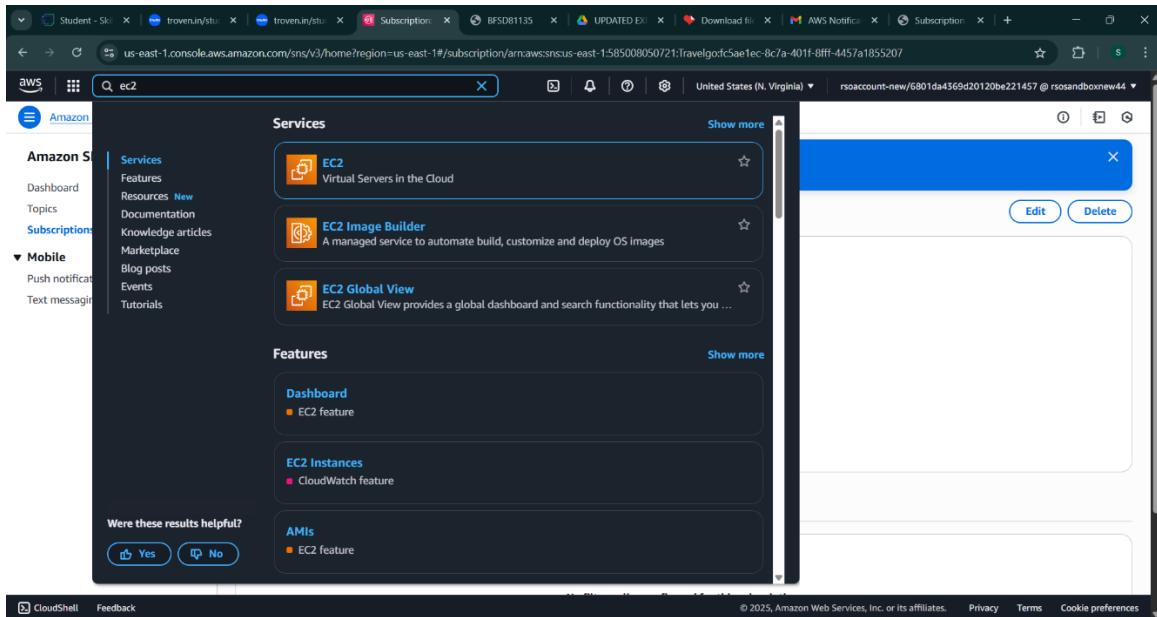
- Note: Load your Flask app and Html files into GitHub repository.

 static	commit	last week
 templates	commit	last week
 app.py	Update app.py	last week

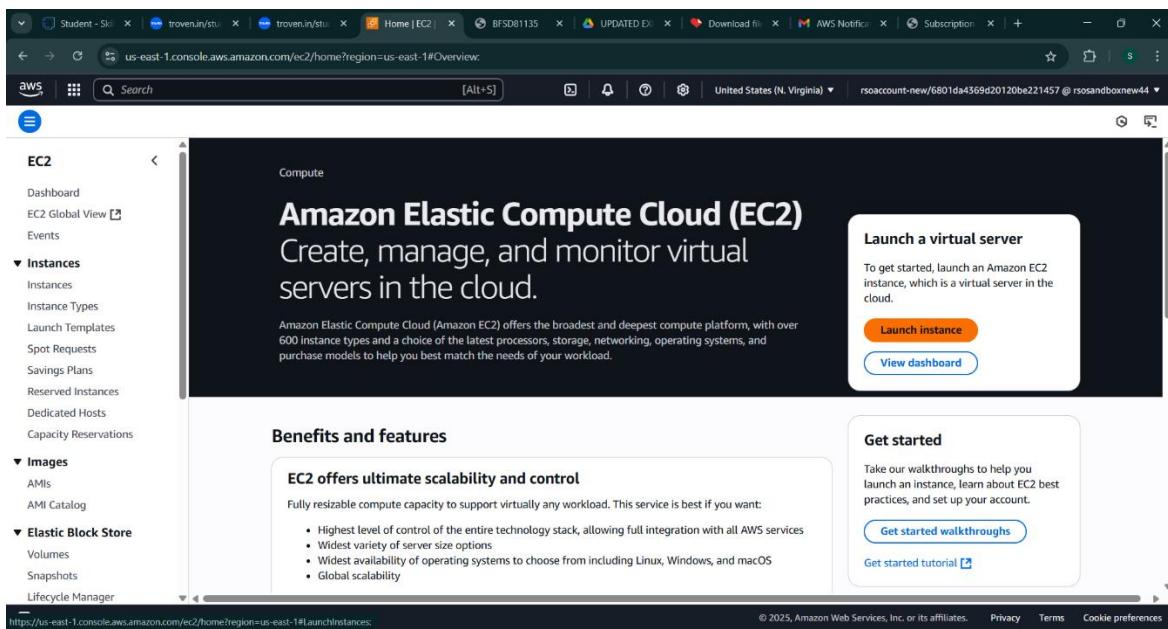
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- **Launch EC2 Instance**

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



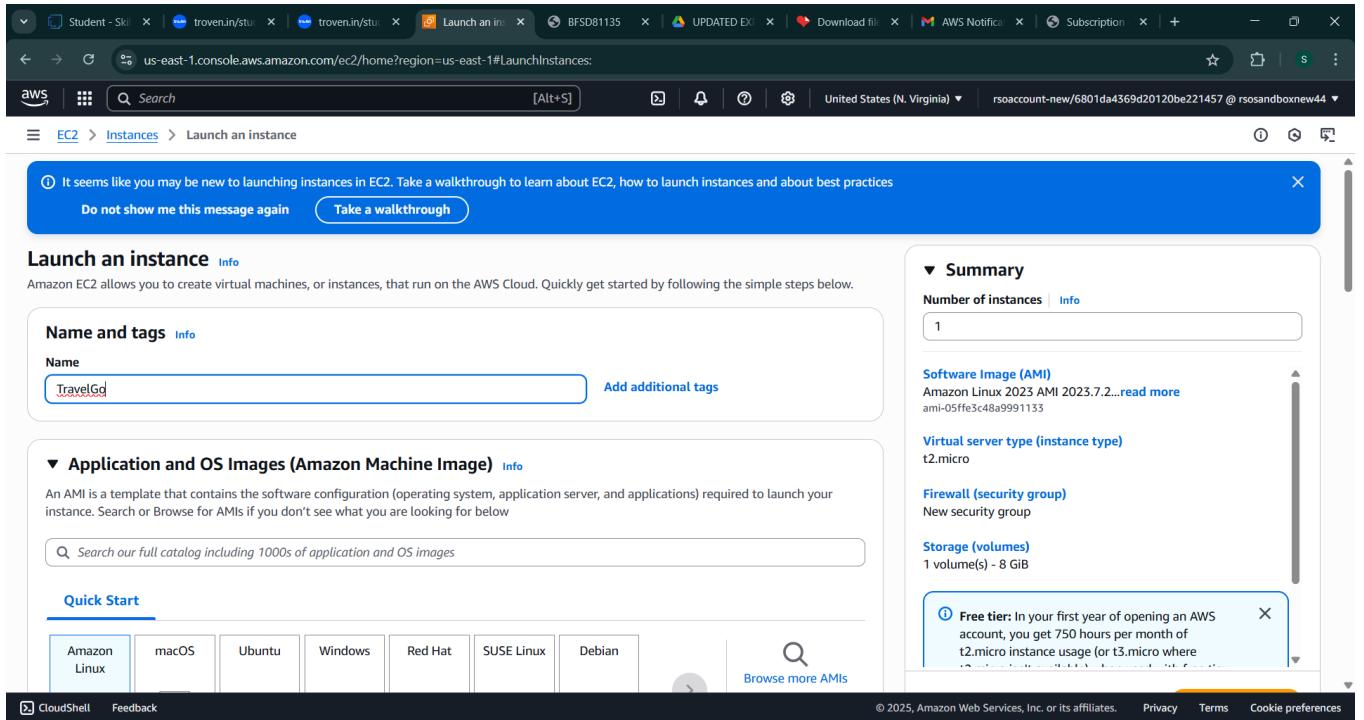
The screenshot shows the AWS search results for the query "ec2". The results are categorized under "Services" and "Features". The "Services" section includes cards for EC2, EC2 Image Builder, and EC2 Global View. The "Features" section includes cards for Dashboard, EC2 Instances, and AMIs. A modal window is open on the right side of the screen, showing a single result card for EC2 with "Edit" and "Delete" buttons.



The screenshot shows the AWS EC2 Home Overview page. The left sidebar navigation includes EC2, Dashboard, EC2 Global View, Events, Instances, Instances Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, AMI Catalog, and Elastic Block Store. The main content area features a large heading "Amazon Elastic Compute Cloud (EC2)" with the subtext "Create, manage, and monitor virtual servers in the cloud." Below this, there is a "Benefits and features" section with a box titled "EC2 offers ultimate scalability and control" and a "Get started" section with links to "Get started walkthroughs" and "Get started tutorial".

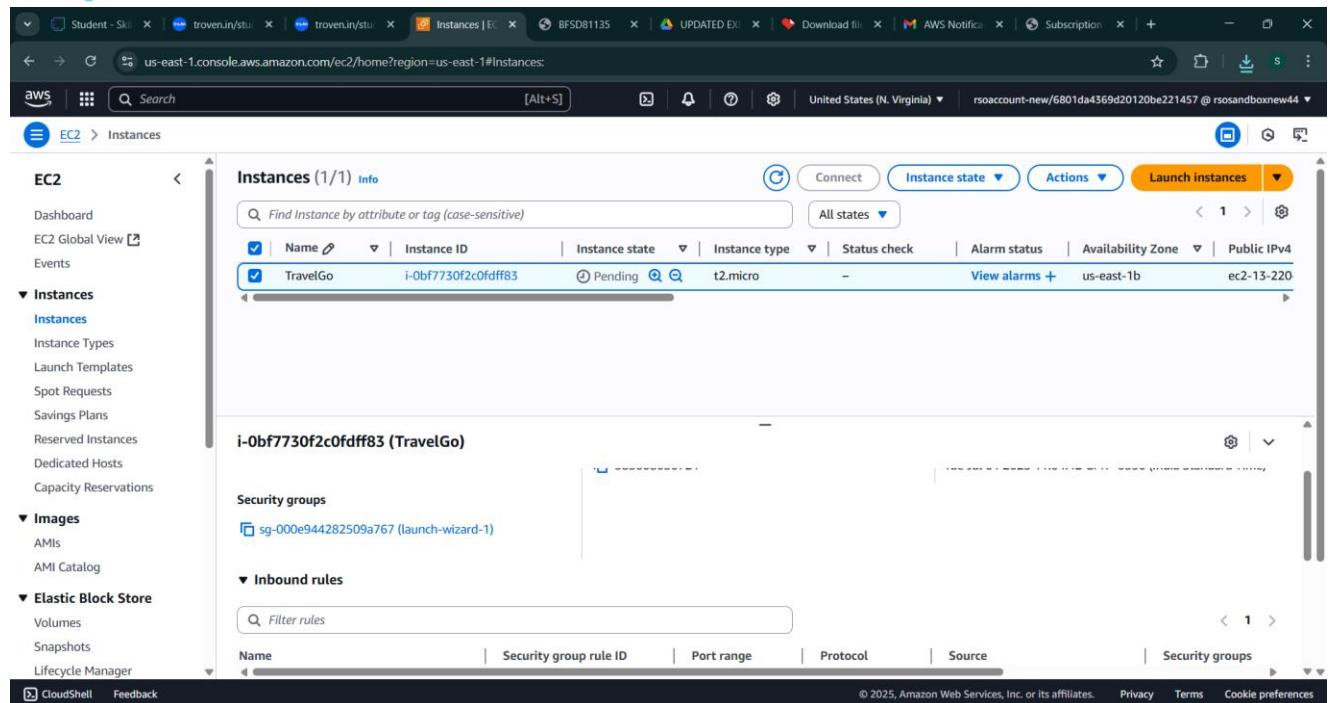
- Click on Launch instance to launch EC2 instance

- 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 of operating systems. 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 storage. A note about the free tier is visible on the right.

- Create and download the key pair for Server access.

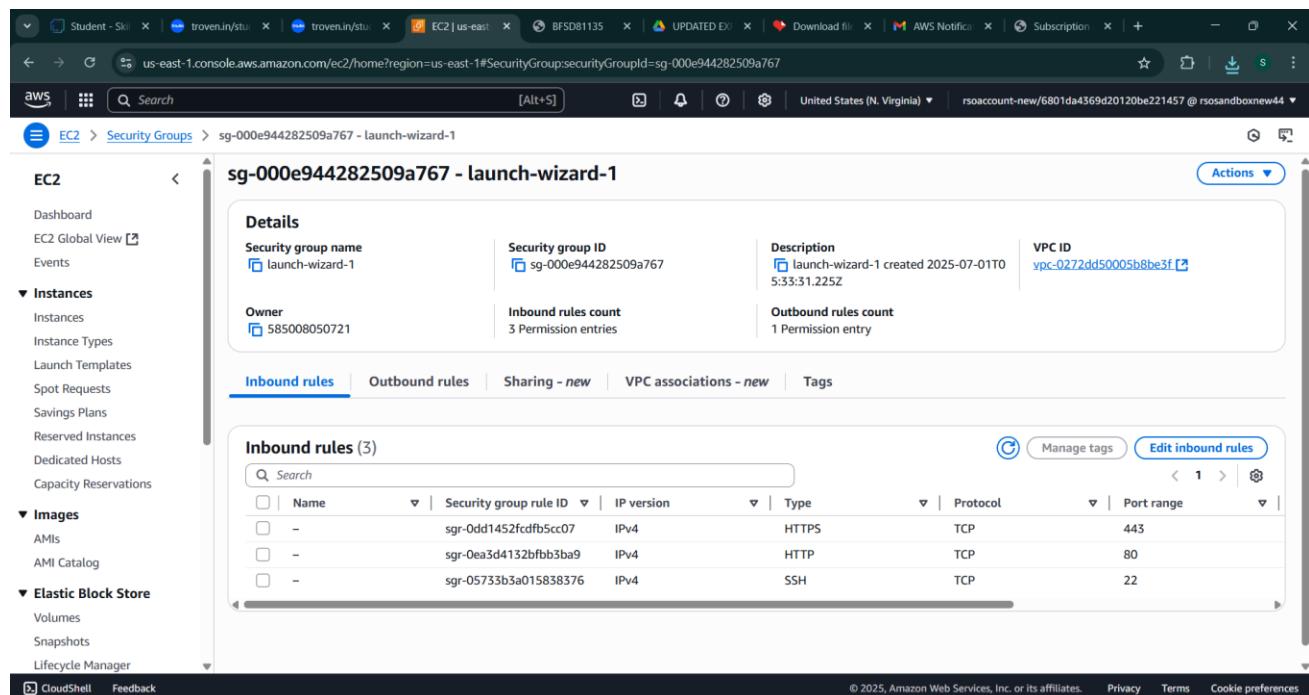


The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with sections like EC2, Instances, Images, and Elastic Block Store. The main area displays a table of instances with one entry:

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

Below the table, there's a detailed view for the selected instance (i-0bf7730f2c0fdff83). It shows the Security groups (sg-000e944282509a767) and Inbound rules.

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



The screenshot shows the AWS Security Groups page. The left sidebar includes sections for EC2, Instances, Images, and Elastic Block Store. The main content is for the security group sg-000e944282509a767, which has the name "sg-000e944282509a767 - launch-wizard-1".

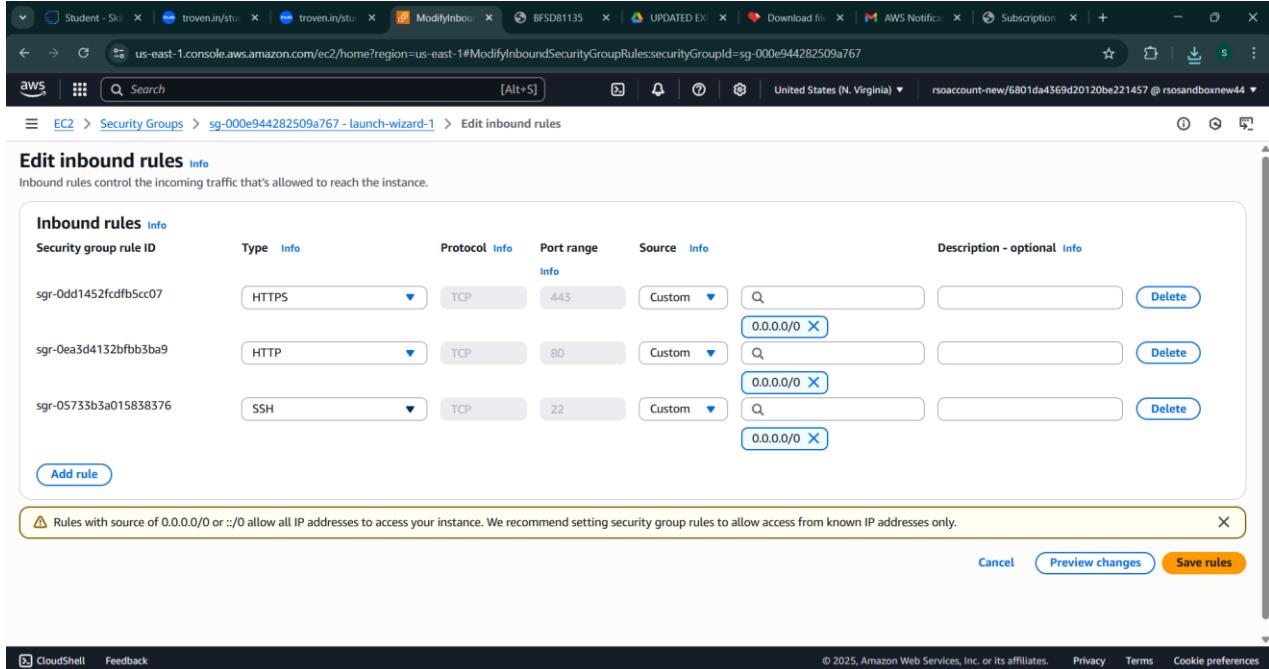
Details:

- Security group name: launch-wizard-1
- Security group ID: sg-000e944282509a767
- Description: launch-wizard-1 created 2025-07-01T05:33:31.225Z
- VPC ID: 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

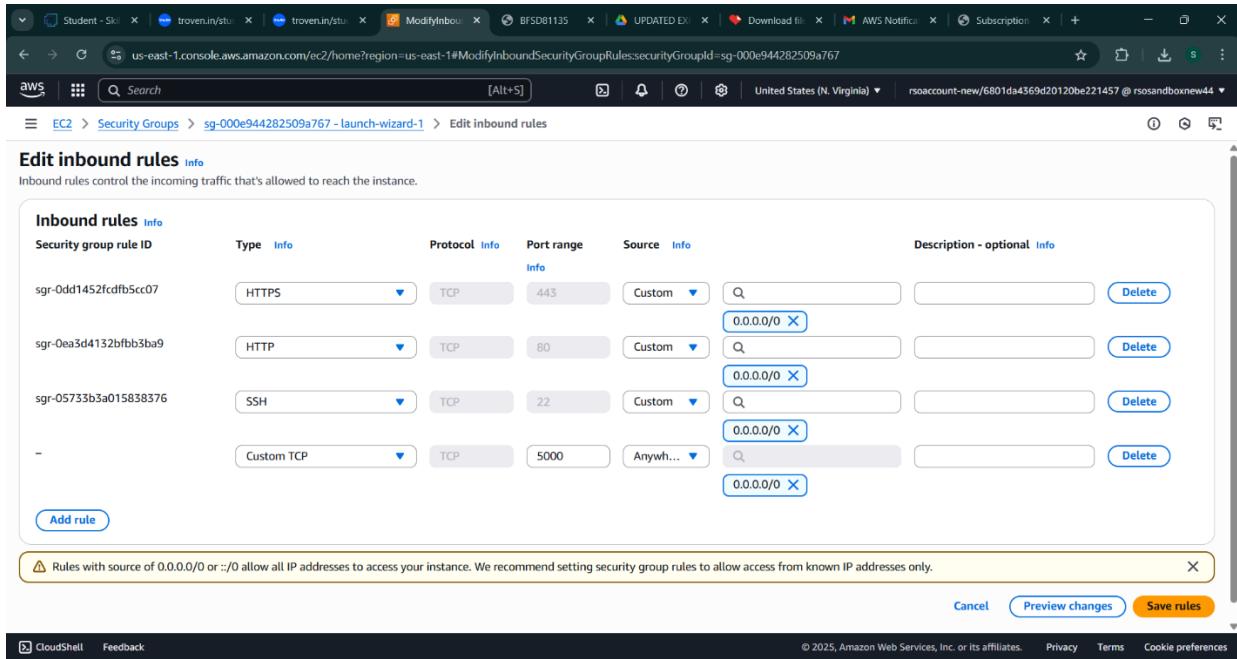
Inbound rules control the incoming traffic that's allowed to reach the instance.

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 X Delete
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	<input type="text"/> 0.0.0.0/0 X Delete
sgr-05733b3a015838376	SSH	TCP	22	Custom	<input type="text"/> 0.0.0.0/0 X Delete

[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.

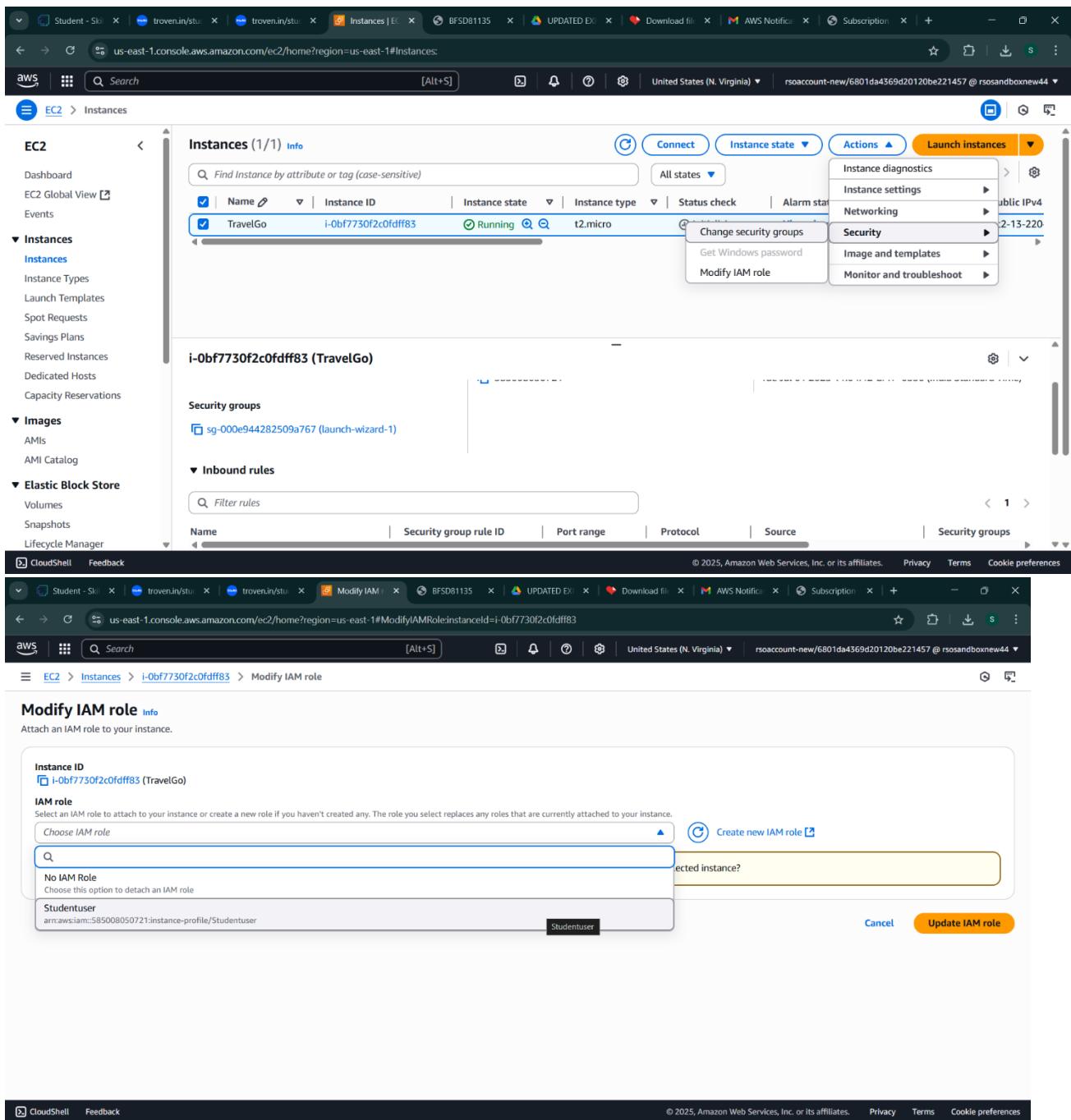
Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdfb5cc07	HTTPS	TCP	443	Custom	<input type="text"/> 0.0.0.0/0 X Delete
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	<input type="text"/> 0.0.0.0/0 X Delete
sgr-05733b3a015838376	SSH	TCP	22	Custom	<input type="text"/>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

[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](#)

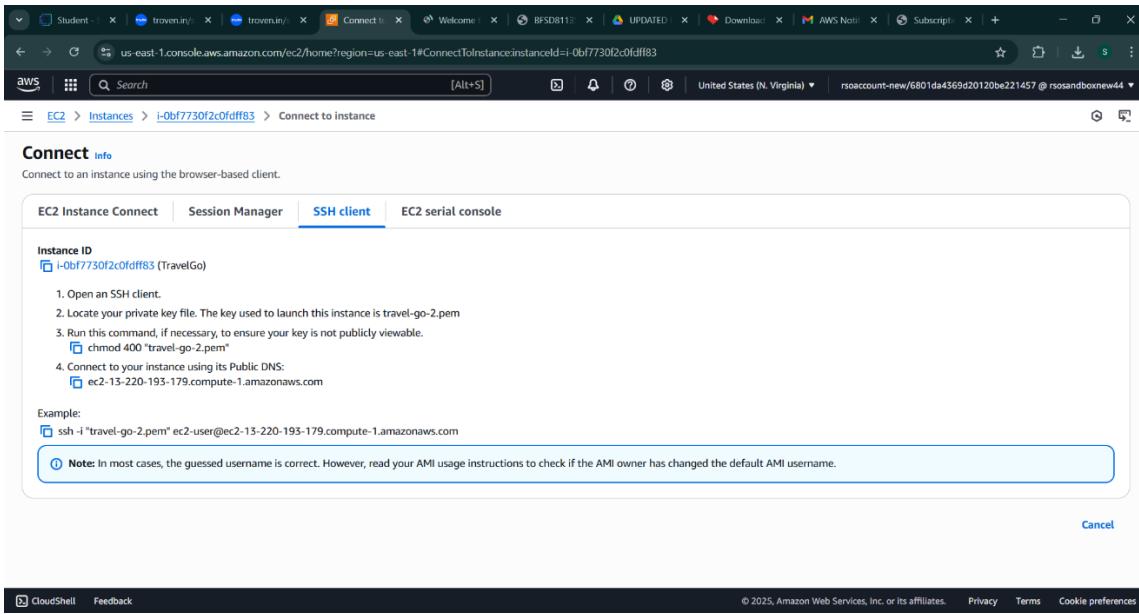
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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



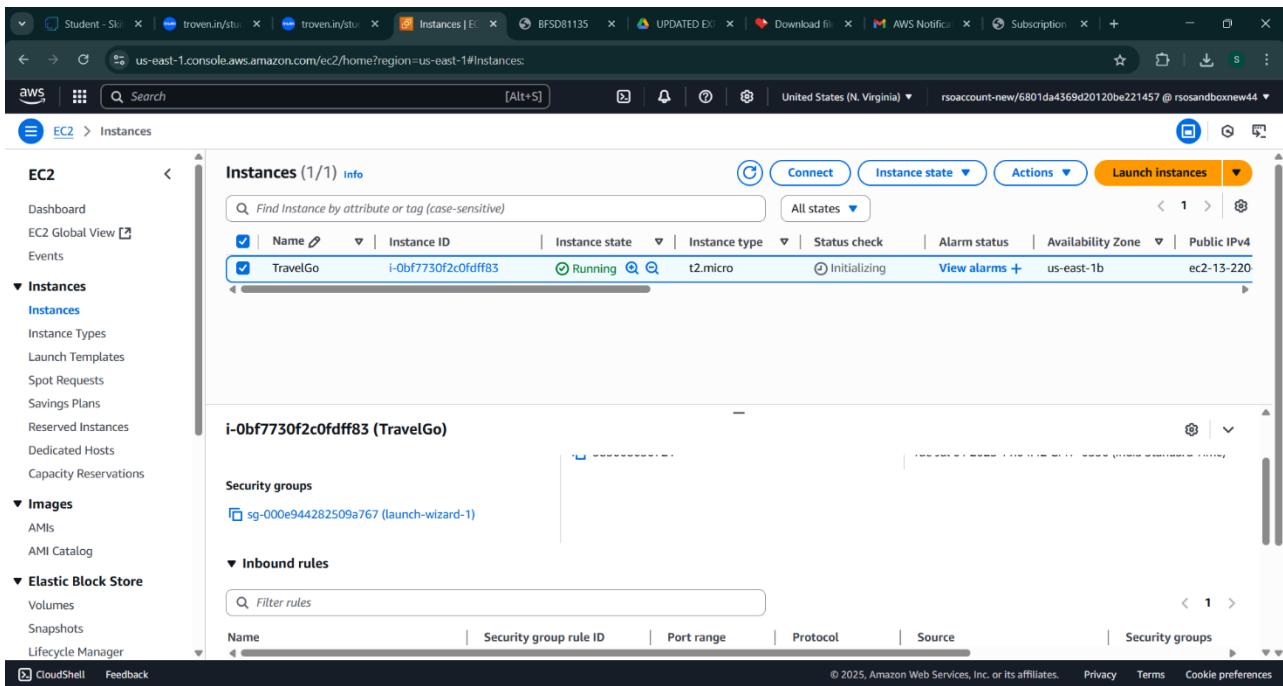
The screenshot shows two overlapping windows from the AWS Management Console:

- Top Window (EC2 Instances):** The user is viewing the EC2 Instances page. A context menu is open over an instance named "TravelGo" (i-0bf7730f2c0fdf83). The "Actions" menu is expanded, showing options like "Instance diagnostics", "Instance settings", "Networking", "Security" (which is currently selected), "Get Windows password", "Image and templates", and "Monitor and troubleshoot".
- Bottom Window (Modify IAM role):** The user is in the "Modify IAM role" dialog for the "TravelGo" instance. The "IAM role" section is active, showing a dropdown menu where "Studentuser" is selected. The "Attached instance?" dropdown also has "TravelGo" selected. 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) 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/HarishwarReddy5485/Travel.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

Search & Book Trains

Guntur ▾ Hyderabad ▾ 27 - 06 - 2025  1

Gouthami Express (12737)

From: Guntur **To:** Hyderabad

Departure: 08:00 PM **Arrival:** 06:00 AM

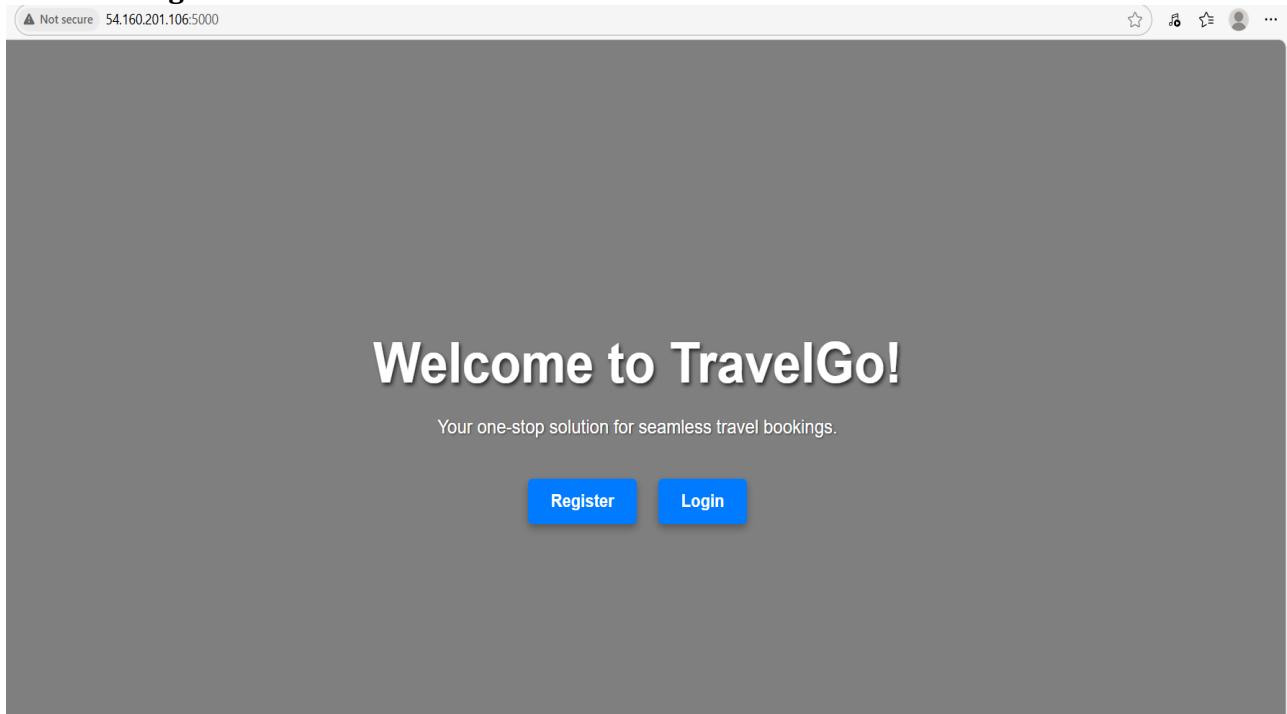
Date: 2025-06-27

Price per person: ₹600

Milestone 8: Testing and Deployment

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

Welcome Page:

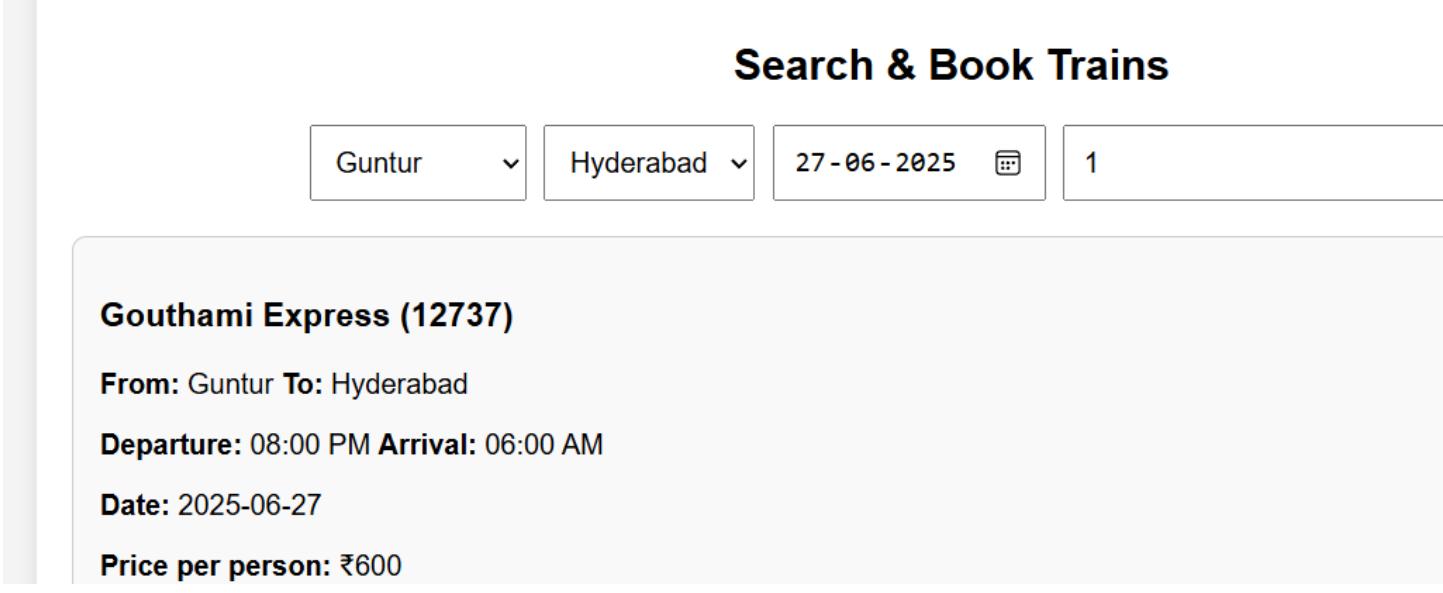


Register Page:

Login Page:

Bus Booking Page:

Train Booking page



Search & Book Trains

Guntur ▾ Hyderabad ▾ 27-06-2025 1

Gouthami Express (12737)

From: Guntur **To:** Hyderabad

Departure: 08:00 PM **Arrival:** 06:00 AM

Date: 2025-06-27

Price per person: ₹600

Flight Booking Page:

🔍 Search & Book Flights

Chennai

Hyderabad

01-07-2025

1

Search

Air India AI-543

₹2800 x 1 = ₹2800

Chennai → Hyderabad

2025-07-01 | 06:00 - 07:20

Book

Confirm Your Flight Booking

Airline: Air India (AI-543)**Route:** Chennai → Hyderabad**Date:** 2025-07-01**Departure:** 06:00**Arrival:** 07:20**Passengers:** 1**Price/person:** ₹2800**Total Price:** ₹2800**Confirm Booking**

Hotel Booking Page:

Confirm Your Hotel Booking

Hotel: The Park

Location: Hyderabad

Check-in: 2025-07-01

Check-out: 2025-07-02

Rooms: 1

Guests: 1

Price/night: ₹5500

Total nights: 1

Total Cost: ₹5500

Confirm Booking

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.

