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

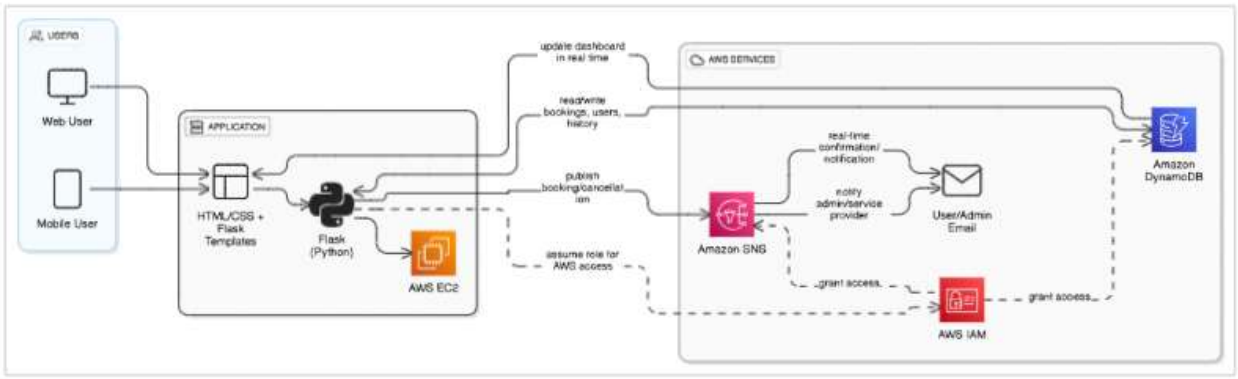
Once a booking is made—whether it's a train ticket or a hotel stay—TravelGo uses AWS SNS to instantly notify the user. For example, after a student books a hotel in Chennai, SNS sends a real-time email notification confirming the booking with all the relevant details. This notification is triggered from 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

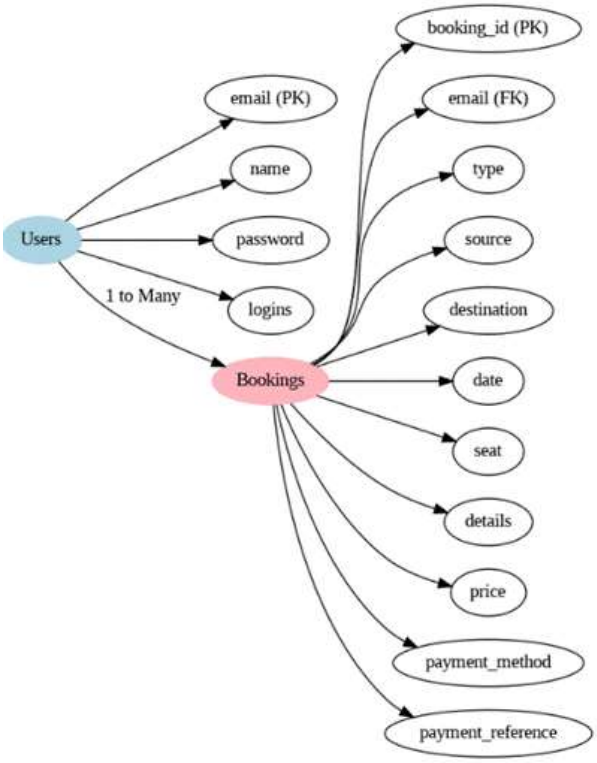
TravelGo features a dynamic user dashboard that displays all past and upcoming bookings for the logged-in user. For example, a user who has booked a flight and a hotel can view these

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.

Architecture:



Entity Relationship (ER)Diagram:



Pre-requisites:

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

Project WorkFlow:

Milestone 1. Backend Development and Application Setup

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

Milestone 2. AWS Account Setup and Login

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

Milestone 3. DynamoDB Database Creation and Setup

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

Milestone 4. SNS Notification Setup

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

Milestone 5. IAM Role Setup

- Create IAM Role
- Attach Policies

Milestone 6. EC2 Instance Setup

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

Milestone 7. Deployment on EC2

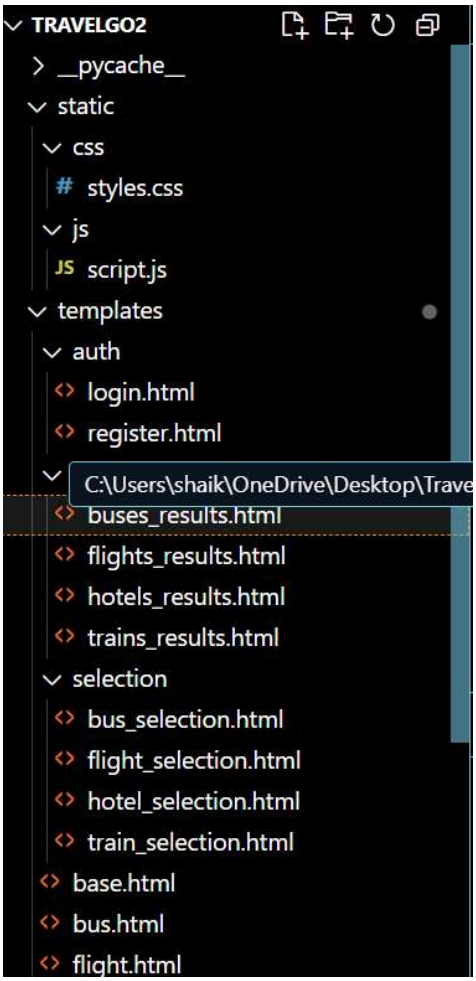
- Upload Flask Files
- Run the Flask App

Milestone 8. Testing and Deployment

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

Milestone 1. Backend Development and Application Setup

LOCAL DEPLOYMENT



Organize the project with HTML templates for each feature (e.g., login, wishlist, quiz, checkout) under the templates folder and manage backend logic in application.py.

Develop the Backend Using Flask
Import libraries:

```
import os
import json
import uuid
import random
from datetime import datetime, timedelta
from decimal import Decimal # Use Decimal for numbers in DynamoDB

from flask import Flask, render_template, request, redirect, url_for, flash, session, jsonify
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current

import boto3
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError

# --- IMPORTANT: This loads environment variables from your .env file ---
# For AWS deployment, you will set these environment variables directly on your EC2 instance
# (e.g., via user data, systemd service file, or AWS Systems Manager Parameter Store)
# The `load_dotenv()` call is primarily for local development convenience.
from dotenv import load_dotenv
load_dotenv()
# --- END IMPORTANT ---

# Assume config.py exists and contains a Config class
from config import Config
```

Import essential Flask modules for web handling, Boto3 for AWS integration, Werkzeug for password hashing, and datetime for timestamp management

Flask App Initialization:

```
app = Flask(__name__)
app.config.from_object(Config)
```

.

Initialize the Flask application and set a secret key to securely manage user sessions and form data.

Database Configuration:

```
dynamodb = boto3.resource('dynamodb', region_name=aws_region_to_use)
users_table = dynamodb.Table(app.config['DYNAMODB_USERS_TABLE'])
bookings_table = dynamodb.Table(app.config['DYNAMODB_BOOKINGS_TABLE'])

sns_client = boto3.client('sns', region_name=aws_region_to_use) # Use the determined region
```

Connect to DynamoDB using Boto3 and define references to the UserTable and WishlistTable for user and wishlist data operations.

***Routes for Core Functionalities:**

Home and registration routes:

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        username = request.form.get('username') # Added username field
        email = request.form.get('email')
        password = request.form.get('password')

        # Basic validation
        if not username or not email or not password:
            flash('All fields are required.', 'danger')
            return redirect(url_for('register')) # Assuming register.html in auth/

        try:
            # Check if email already exists (primary key lookup)
            response = users_table.get_item(Key={'email': email})
            if 'Item' in response:
                flash('Email already registered. Please use another or log in.', 'danger')
                return redirect(url_for('register'))

            # Note: Without GSI, we cannot efficiently check for unique usernames.
            # If username uniqueness is critical, a GSI would be required.

            hashed_password = generate_password_hash(password)
```

Description: Create the home and registration routes, where the registration route securely hashes user passwords and stores user data in DynamoDB upon form submission.

Login Route:

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

@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        # --- START UPDATES HERE ---

        # 1. Get raw email input
        email_raw = request.form.get('email')

        # 2. Sanitize and validate email
        if not email_raw:
            flash('Email address is required.', 'danger')
            return render_template('auth/login.html')

        email = email_raw.strip() # Remove leading/trailing whitespace
        password = request.form.get('password')
```

Description: Implement the user login route to validate credentials using DynamoDB and securely manage session data while updating the user's login count.

Booking Confirmation Routes:

```python
@app.route('/confirm_booking/<string:booking_type>/<string:item_id>', methods=['POST'])
@login_required
def confirm_booking(booking_type, item_id):
    # Retrieve details from the form submission (hidden inputs on search results page)
    booking_details = {
        "user_id": current_user.id,
        "booking_type": booking_type,
        "item_id": item_id,
        "booking_date": datetime.now(),
        "status": "confirmed", # Initial status is confirmed
        "details": request.form.to_dict() # Store all form data for details
    }

    # Remove Flask-specific internal fields that might be passed from forms
    booking_details["details"].pop('csrf_token', None)

    # Add selected seat/room to details if present
    if booking_type == 'flight':
        selected_seat = request.form.get('selected_seat')
        if selected_seat:
            booking_details['details']['selected_seat'] = selected_seat
        else:
            flash("Please select a seat for your flight.", "danger")
            # Redirect back to selection page with existing data
            return redirect(url_for('select_flight_seats', flight_id=item_id, **request.form))
    elif booking_type == 'hotel':
        selected_room_type = request.form.get('selected_room_type')
```

User dashboard and wishlist routes:

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

    return render_template('user_dashboard.html')

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

    item_id = request.json['item_id']
    item_name = request.json['item_name']
    item_details = request.json['item_details']

    # Add item to WishlistTable in DynamoDB
    wishlist_table.put_item(
        Item={
            'email': session['email'],
            'item_id': item_id,
            'item_name': item_name,
            'item_details': item_details,
            'added_date': datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S')  # Store the current UTC date
        }
    )
    return jsonify({'success': True, 'message': 'Item added to wishlist'})
```

Description: Secure the user dashboard and implement a route to add items to the wishlist,storing them in DynamoDB with item details and a timestamp.

```python
# View wishlist route
@app.route('/wishlist')
def wishlist():
    if 'email' not in session:
        return redirect(url_for('login'))

    # Retrieve wishlist items from DynamoDB for the logged-in user
    response = wishlist_table.query(
        KeyConditionExpression=boto3.dynamodb.conditions.Key('email').eq(session['email'])
    )
    wishlist_items = response.get('Items', [])  # Ensure all items are passed to the frontend

    return render_template('wishlist.html')

@app.route('/wishlist_data')
def wishlist_data():
    if 'email' not in session:
        return jsonify({'wishlist': []})

    response = wishlist_table.query(
        KeyConditionExpression=boto3.dynamodb.conditions.Key('email').eq(session['email'])
    )
    wishlist_items = response.get('Items', [])
    return jsonify({'wishlist': wishlist_items})


# Route to remove an item from the wishlist
@app.route('/remove_from_wishlist', methods=['POST'])
def remove_from_wishlist():
    if 'email' not in session:
        return jsonify({'success': False, 'message': 'Not logged in'})

    item_id = request.json.get('item_id')

    if not item_id:
        return jsonify({'success': False, 'message': 'Item ID not provided'})

    # Remove the item from WishlistTable in DynamoDB
    try:
        wishlist_table.delete_item(
            Key={
                'email': session['email'],
                'item_id': item_id
            }
        )
        return jsonify({'success': True, 'message': 'Item removed from wishlist'})
    except Exception as e:
        return jsonify({'success': False, 'message': f'Error: {str(e)}'})
```

Description: Build routes to display, fetch, and delete wishlist items for logged-in users by querying and modifying entries in DynamoDB using the user's email.

DEDICATED SEARCH PAGES ROUTES:

```python
@app.route('/flights')
def flights_page():
    return render_template('flight.html')

@app.route('/hotels')
def hotels_page():
    return render_template('hotel.html')

@app.route('/trains')
def trains_page():
    return render_template('trains.html')

@app.route('/buses')
def buses_page():
    return render_template('bus.html')
```

Description: Create a route for the virtual exhibition page where users can add detailed jewelry items to their wishlist, with proper validation and storage in DynamoDB.

Quiz Route:

```python
# Quiz page and submission
@app.route('/quiz', methods=['GET', 'POST'])
def quiz():
    if request.method == 'POST':
        score = int(request.form.get('score', 0))
        if score >= 12:
            session['won_quiz'] = True  # Set the status for quiz win
        else:
            session['won_quiz'] = False
        return redirect(url_for('user_dashboard'))
    return render_template('quiz.html')
```

Description: Create a route to handle quiz submissions, storing the result in the session if the user scores 12 or more, and redirecting to the user dashboard.

Check out route:

```
    elif data['action'] == 'update_quantity':
        # Handle quantity updates
        item_name = data['item_name']
        quantity = int(data['quantity'])
        checkout_items = session.get('checkout_items', [])

        for item in checkout_items:
            if item['name'] == item_name:
                item['quantity'] = quantity
                item['total_price'] = int(item['price'].replace(',', '').split(' ')[0]) * quantity
                break

        subtotal = sum(item['total_price'] for item in checkout_items)
        discount = session.get('discount', 0)
        total_price = subtotal - discount

        session['checkout_items'] = checkout_items
        session['subtotal'] = subtotal
        session['final_price'] = total_price
        session.modified = True

        return jsonify({'success': True, 'total_price': total_price})

    elif data['action'] == 'remove':
        # Handle item removal
        item_name = data['item_name']
        checkout_items = session.get('checkout_items', [])
        updated_checkout_items = [item for item in checkout_items if item['name'] != item_name]

        subtotal = sum(
            int(item['price'].replace(',', '').split(' ')[0]) * item.get('quantity', 1)
            for item in updated_checkout_items
        )
        discount = session.get('discount', 0)
        total_price = subtotal - discount

        session['checkout_items'] = updated_checkout_items
        session['subtotal'] = subtotal
        session['final_price'] = total_price
        session.modified = True

        return jsonify({'success': True, 'message': f"Item {item_name} removed from checkout!", 'total_price': total_price})

    elif data['action'] == 'finalize':
        # Finalize checkout and save order items
        session['order_items'] = session.get('checkout_items', [])
        session.modified = True

        return jsonify({'success': True, 'redirect': url_for('order')})

    return jsonify({'success': False, 'message': 'Invalid action!'})

@app.route('/checkout_items', methods=['GET'])
def checkout_items():
    # Retrieve checkout items for rendering on the frontend
    if 'email' not in session:
        return jsonify({'checkout_items': []})
    return jsonify({'checkout_items': session.get('checkout_items', [])})
```

Description: Implement logic for updating item quantity, removing items, and finalizing orders in the shopping cart, while maintaining session-based checkout data and price calculations.

Add to checkout route:

```python
@app.route('/add_to_checkout', methods=['POST'])
def add_to_checkout():
    if 'email' not in session:
        return jsonify({'success': False, 'message': 'Not logged in'})

    item_id = request.json.get('item_id')
    if not item_id:
        return jsonify({'success': False, 'message': 'Item ID missing'})

    # Fetch the item from wishlist
    try:
        response = wishlist_table.get_item(
            Key={'email': session['email'], 'item_id': item_id}
        )
        item = response.get('Item')
        if not item:
            return jsonify({'success': False, 'message': 'Item not found in wishlist'})

        # Prepare item for checkout session
        checkout_item = {
            'item_id': item['item_id'],
            'item_name': item['item_name'],
            'price': item['item_details'].split('Price: ')[-1],  # e.g., "3,50,000 INR"
            'image': item.get('item_image', ''),
            'details': item.get('item_details', ''),
            'quantity': 1
        }

        # Initialize or update checkout session
        checkout_items = session.get('checkout_items', [])
        existing = next((i for i in checkout_items if i['item_id'] == item_id), None)

        if not existing:
            checkout_items.append(checkout_item)
            session['checkout_items'] = checkout_items
            session.modified = True

        return jsonify({'success': True, 'message': 'Item added to checkout'})
    except Exception as e:
        return jsonify({'success': False, 'message': str(e)})
```

Description: Enable users to add wishlist items to the checkout session by retrieving item details from DynamoDB and updating the session state if the item is not already present.

Cancel Booking Route:

```python
@app.route('/cancel_booking/<string:booking_id>', methods=['POST'])
@login_required
def cancel_booking(booking_id):
    try:
        # Convert the string booking_id back to MongoDB's ObjectId
        obj_booking_id = ObjectId(booking_id)
    except Exception:
        flash('Invalid booking ID.', 'danger')
        return redirect(url_for('my_bookings'))

    # Find the booking and ensure it belongs to the current user
    booking = db.bookings.find_one({"_id": obj_booking_id, "user_id": current_user.id})

    if not booking:
        flash('Booking not found or you do not have permission to cancel it.', 'danger')
        return redirect(url_for('my_bookings'))

    if booking['status'] == 'cancelled':
        flash('This booking is already cancelled.', 'info')
        return redirect(url_for('my_bookings'))

    # Update the booking status to 'cancelled'
    result = db.bookings.update_one(
        {"_id": obj_booking_id},
        {"$set": {"status": "cancelled", "cancellation_date": datetime.now()}}
    )
```

Description: Handle **booking cancellation** by converting the string booking_id to an ObjectId, verifying the booking exists and belongs to the current user, checking if it's already cancelled, and updating the booking status to 'cancelled' with a cancellation_date upon successful processing.

Initiating Flask app:

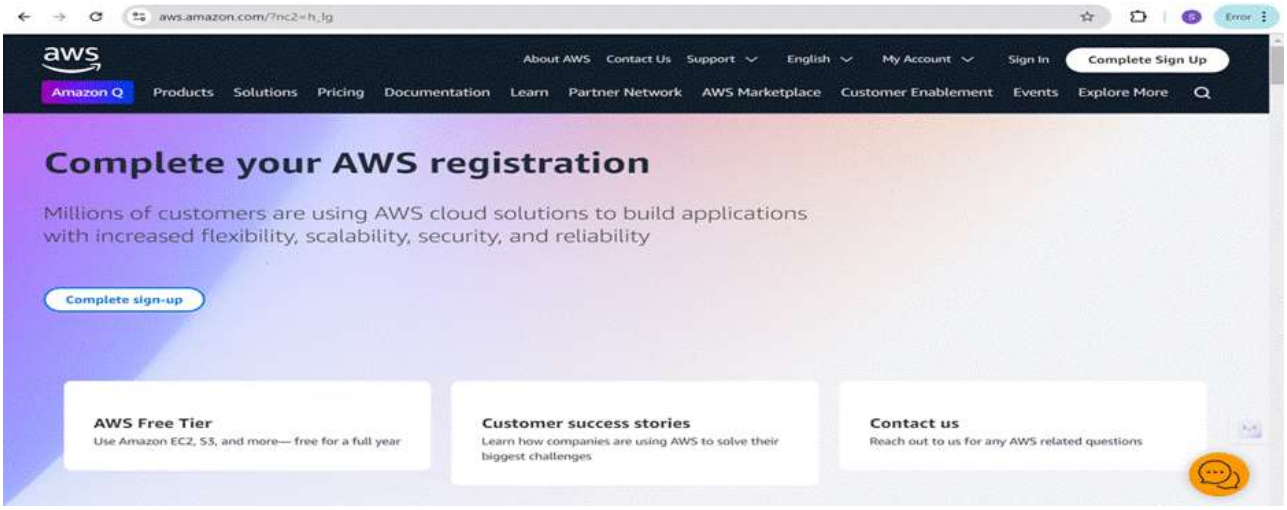
```python
# Run the Flask app
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80,debug=True)
```

Description: Start the Flask application on host 0.0.0.0 and port 80 in debug mode to enable live reloads and detailed error logs during development.

Milestone 2 : AWS Account Setup

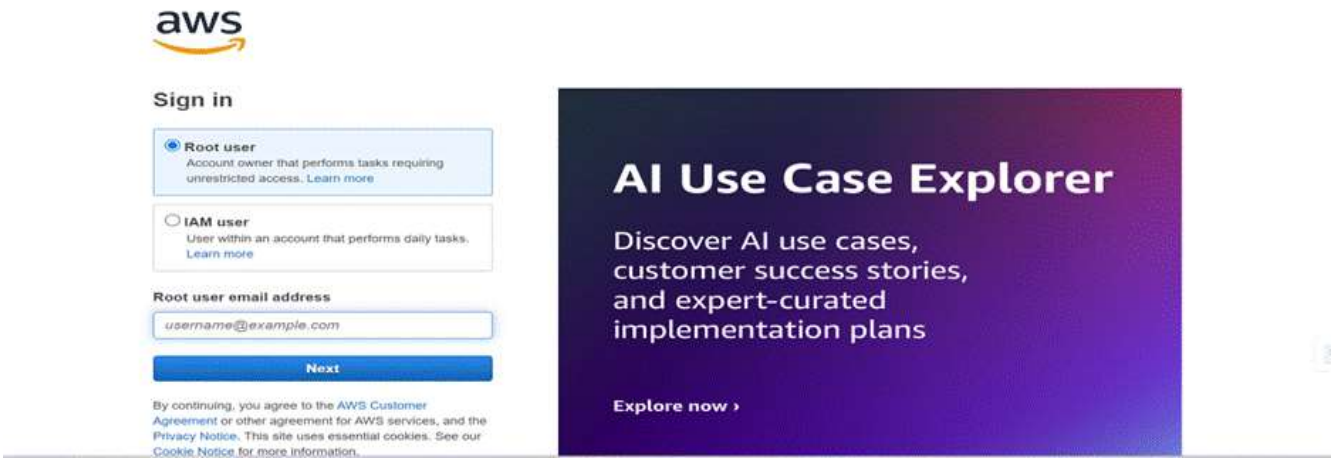
2.1 Set up an AWS account if not already done.

Sign up for an AWS account and configure billing settings.



2.2 Log in to the AWS Management Console

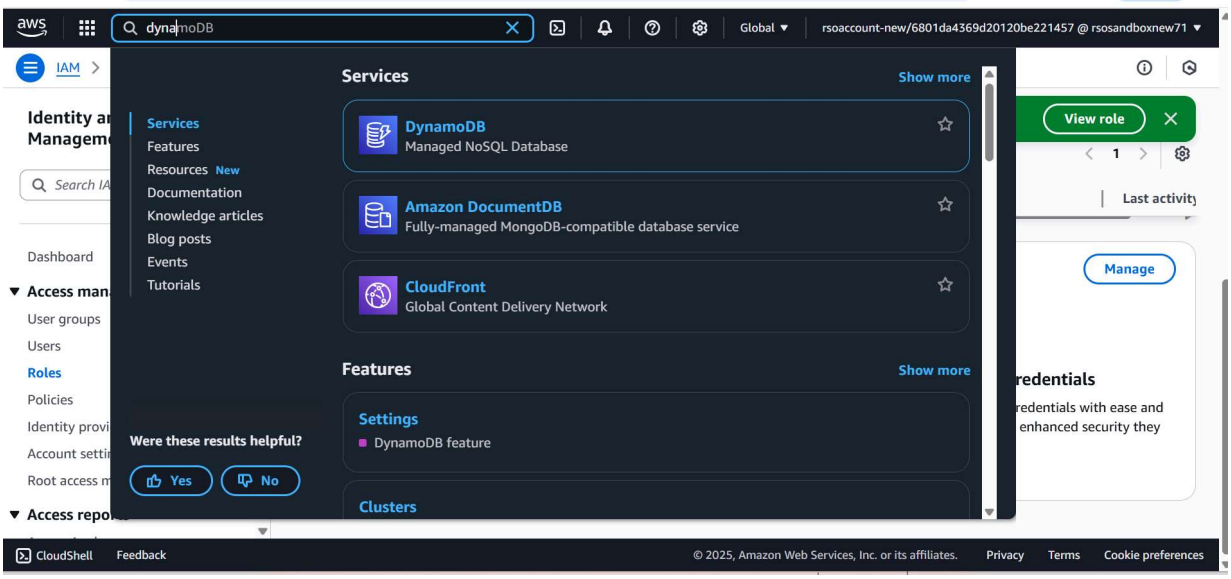
After setting up your account, log in to the AWS Management Console.

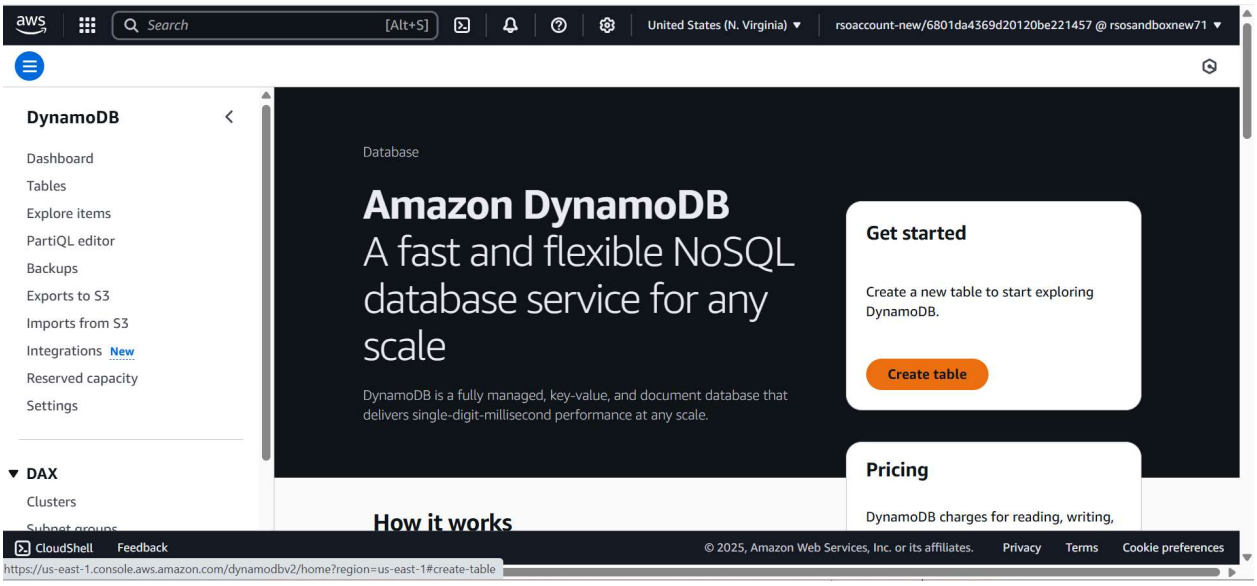


Milestone 3: DynamoDB Database Creation and Setup

Navigate to the DynamoDB

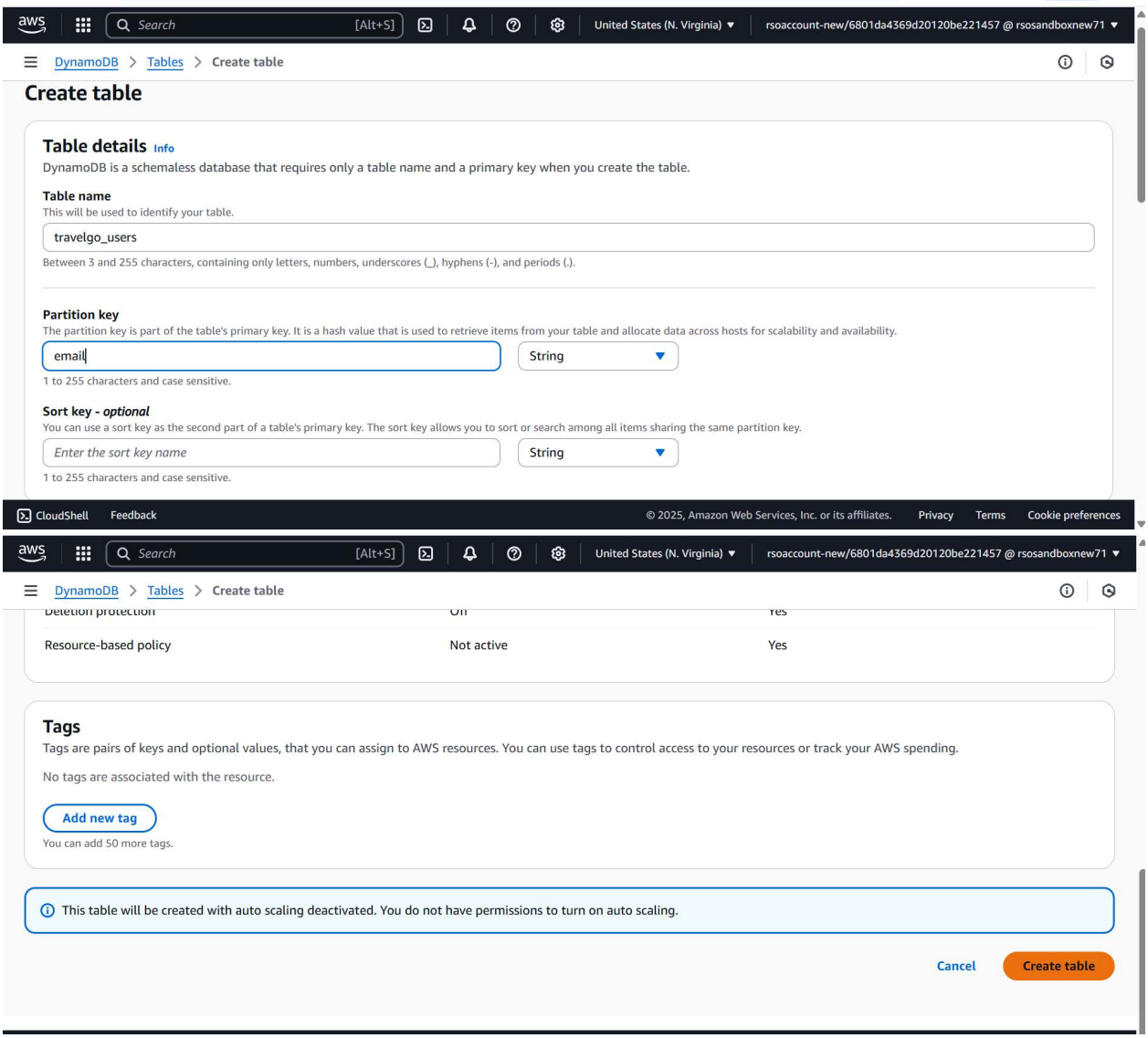
In the AWS Console, navigate to DynamoDB and click on create tables.



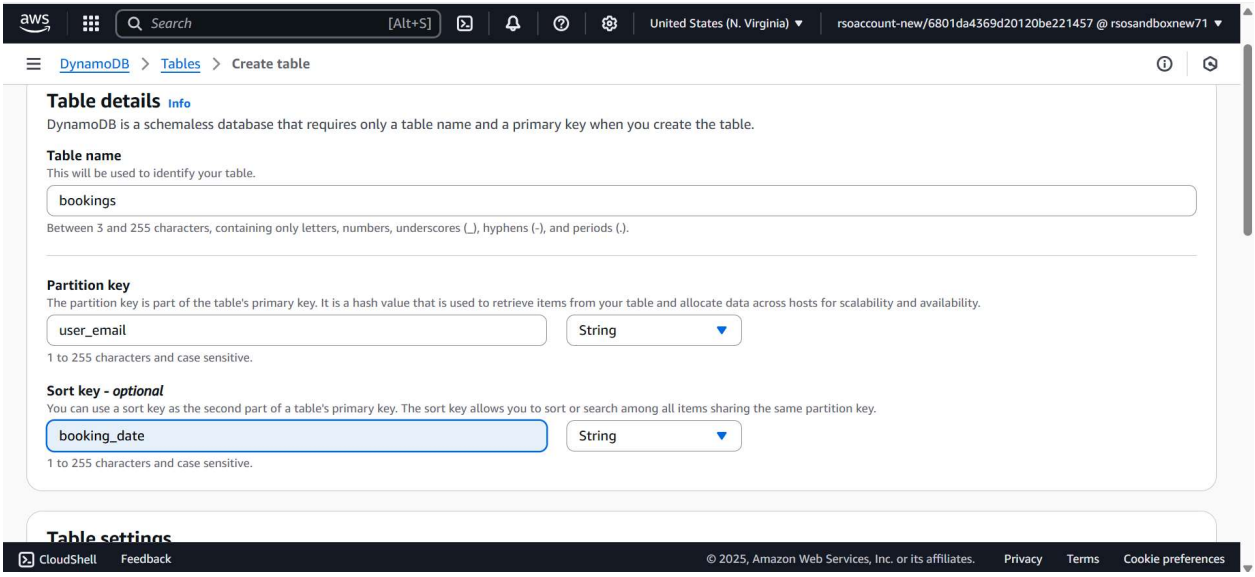


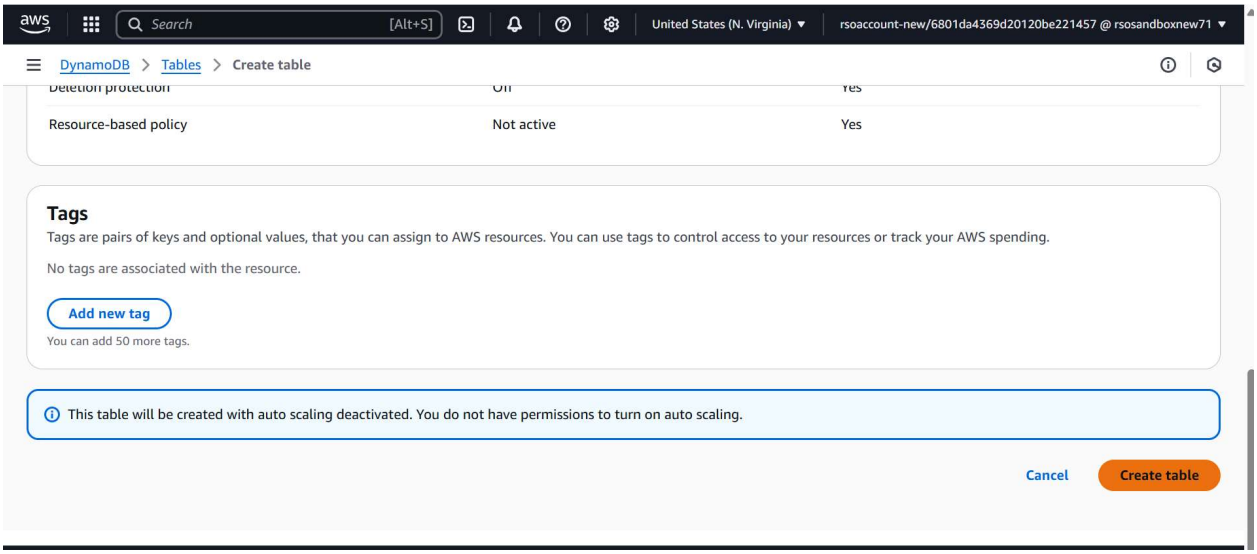
Create a DynamoDB table for storing Data.

Create a travel-Users table for storing registered user details with partition key “Email” with type String and click on create tables.

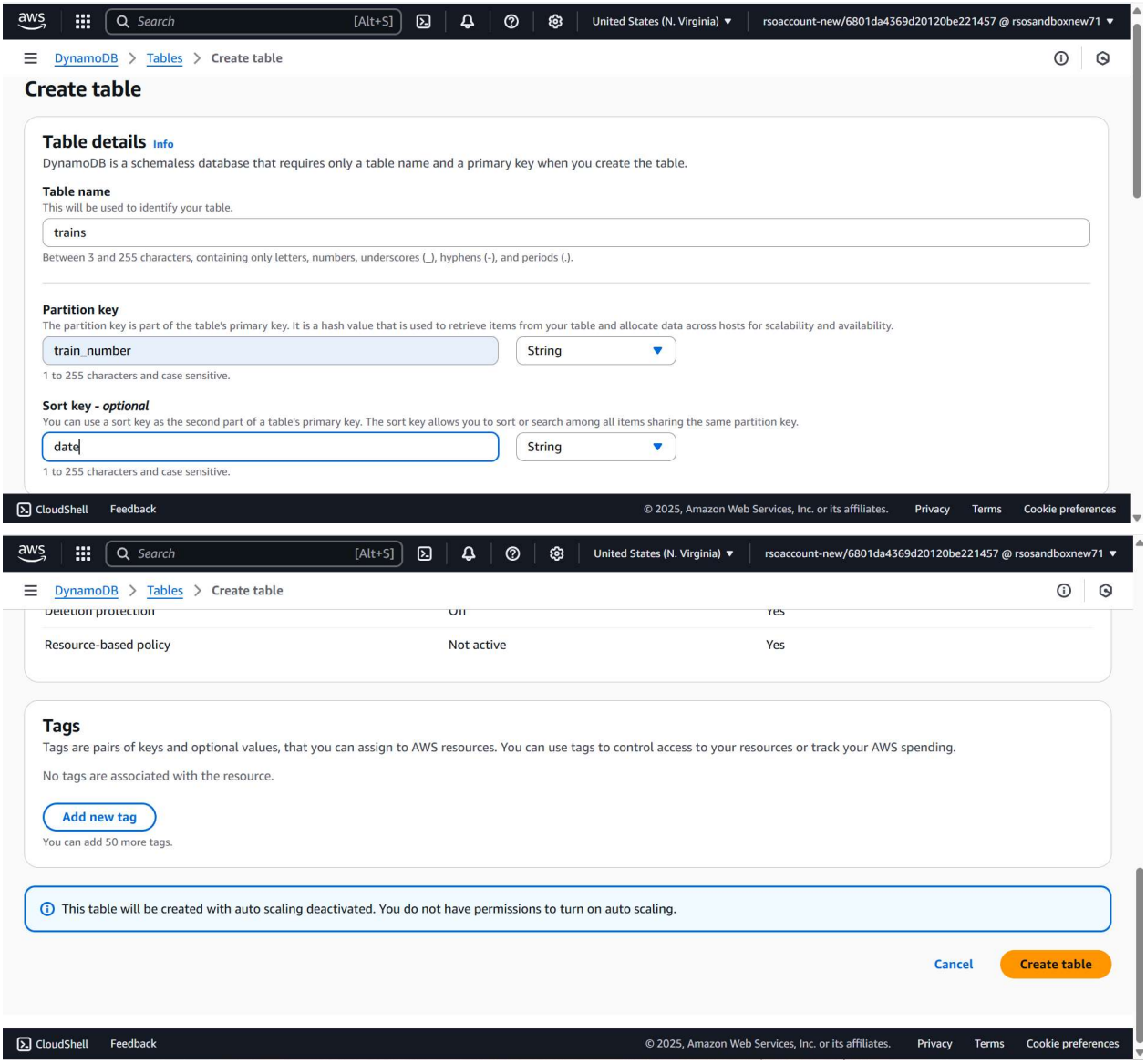


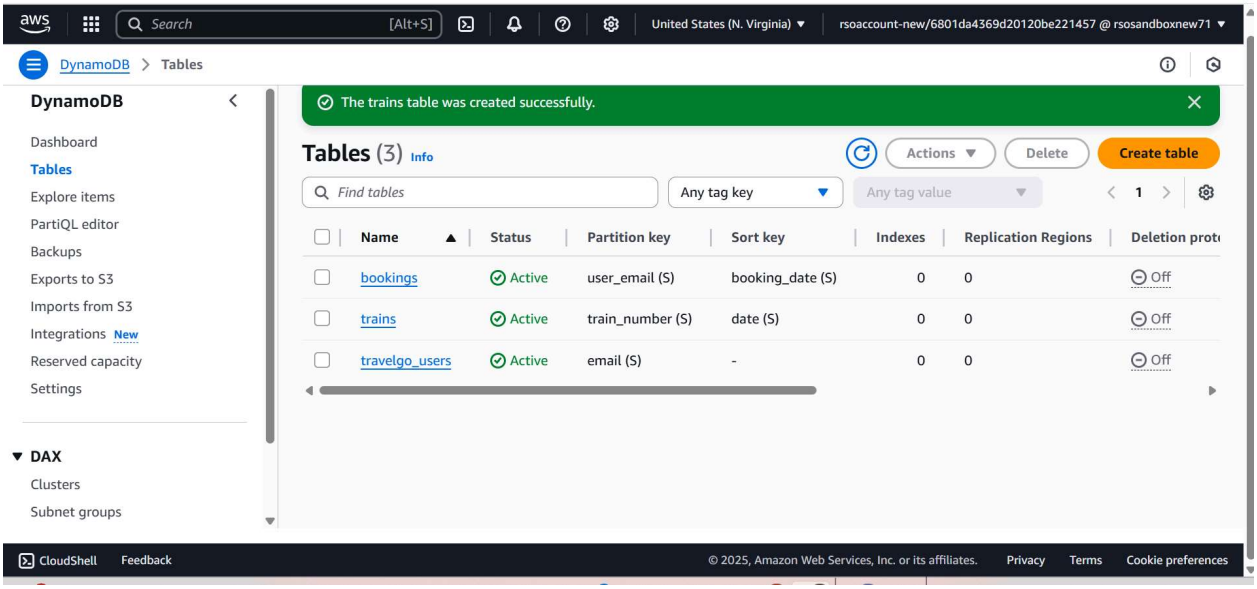
Follow the same steps to create a Bookings for storing booking records with email as the partition key and booking_date as the sort key.


aws
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
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.
bookings
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.
user_email
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.
booking_date
String
1 to 255 characters and case sensitive.
Table settings
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


aws
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
DynamoDB
Tables
Create table
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.
This table will be created with auto scaling deactivated. You do not have permissions to turn on auto scaling.
Cancel
Create table

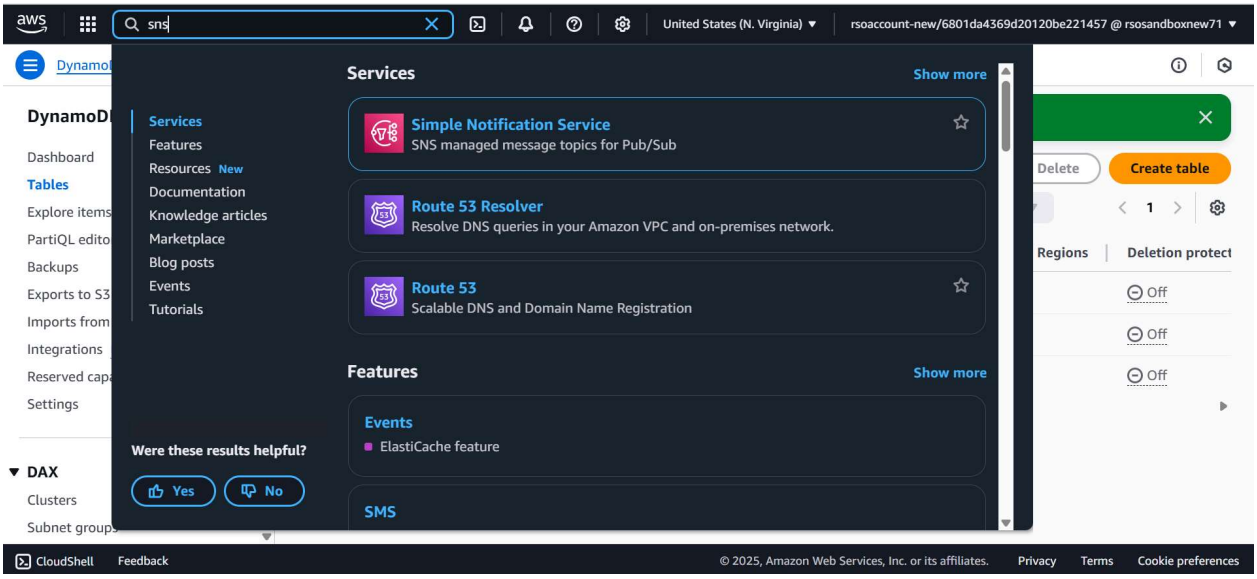
Follow the same steps to create a trains for storing train number with date as the partition key and train_number as the sort key.



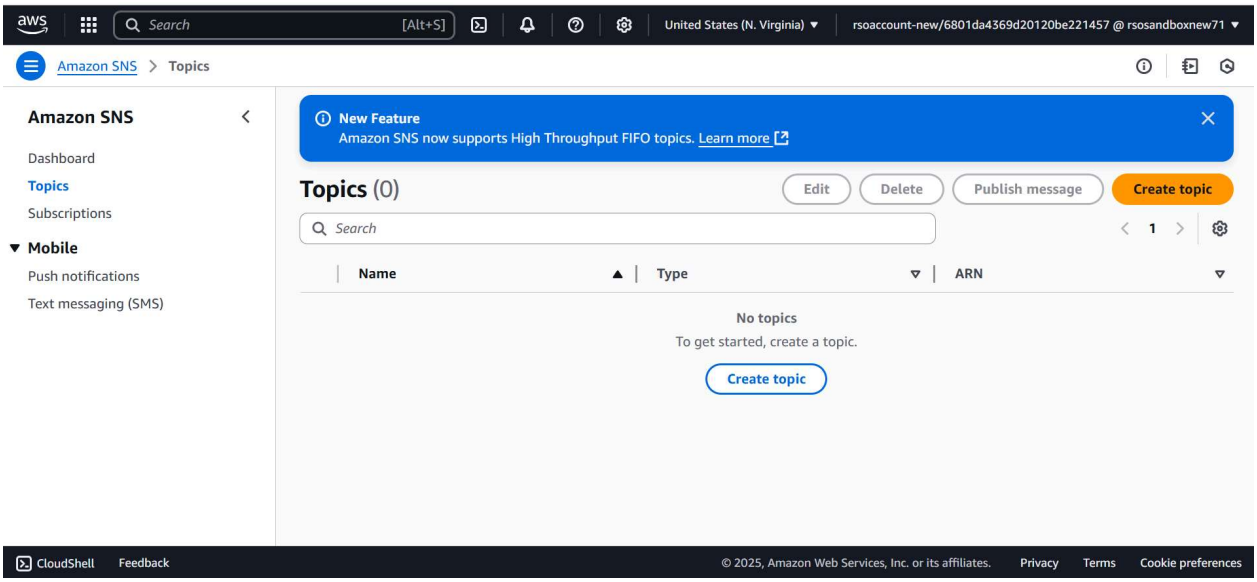


Milestone 4: SNS Notification Setup

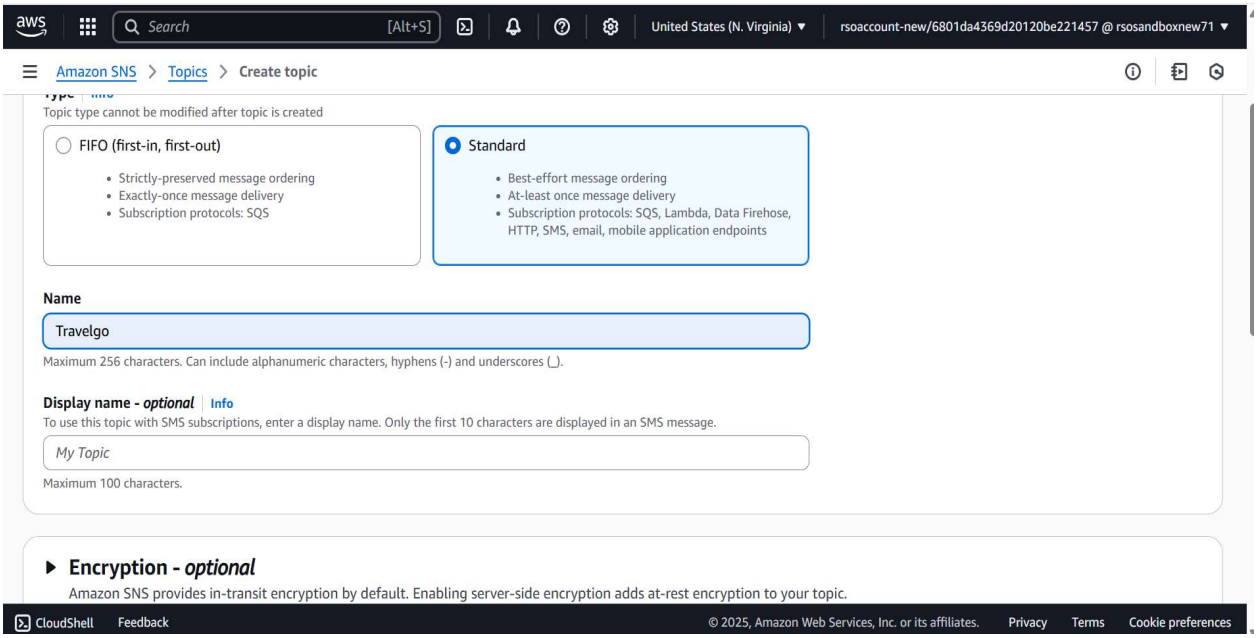
In the AWS Console, search for SNS and navigate to the SNS Dashboard.

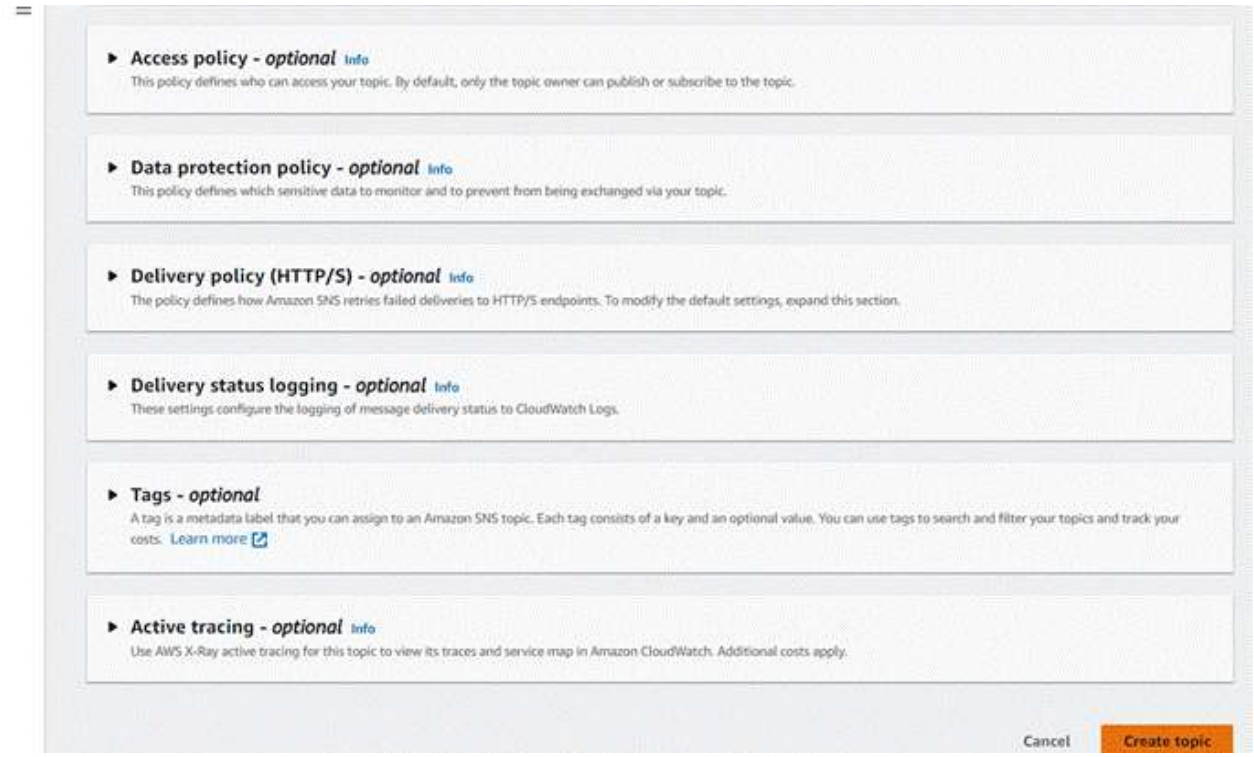


Click on Create Topic and choose a name for the topic.

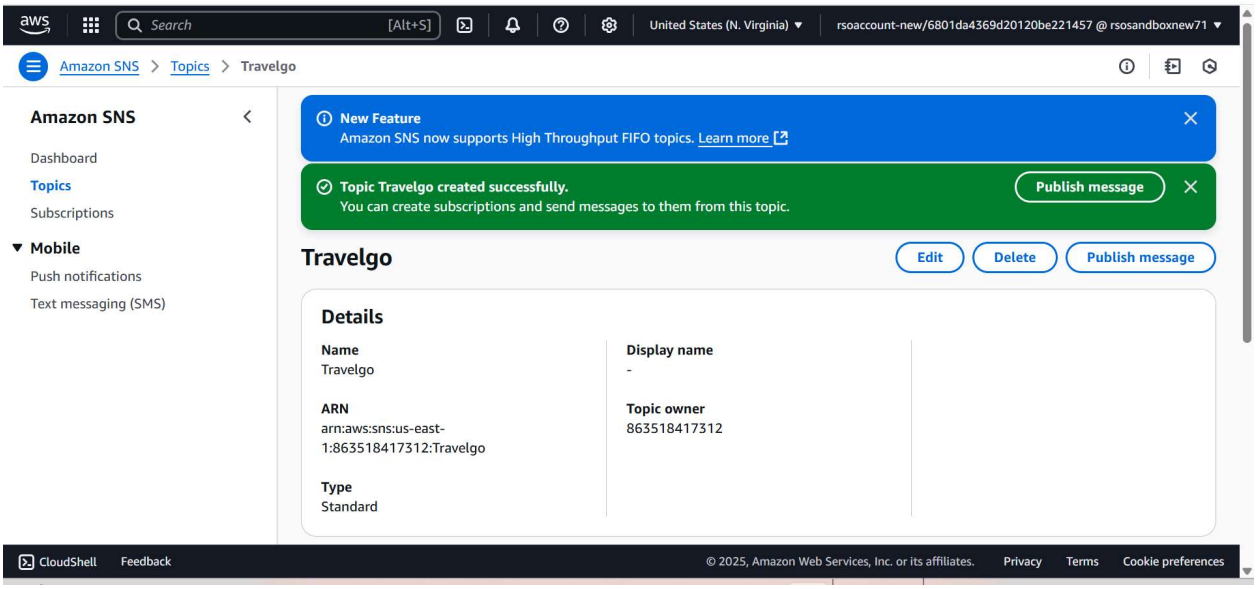


Choose Standard type for general notification use cases and name as Travelgo and Click on Create Topic.



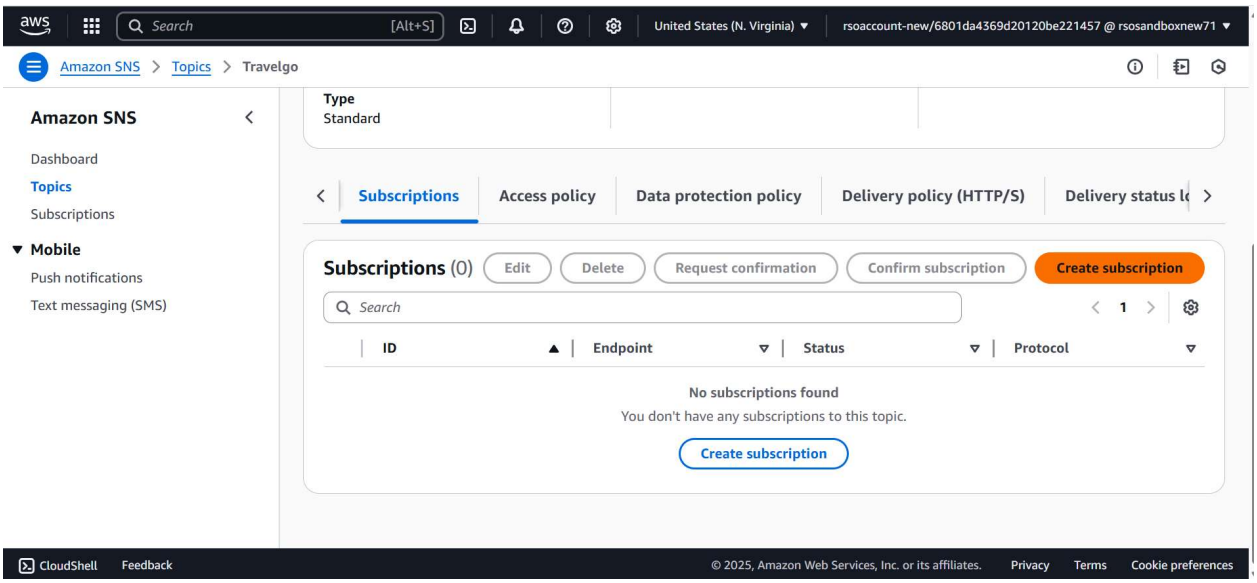


Configure the SNS topic and note down the Topic ARN.

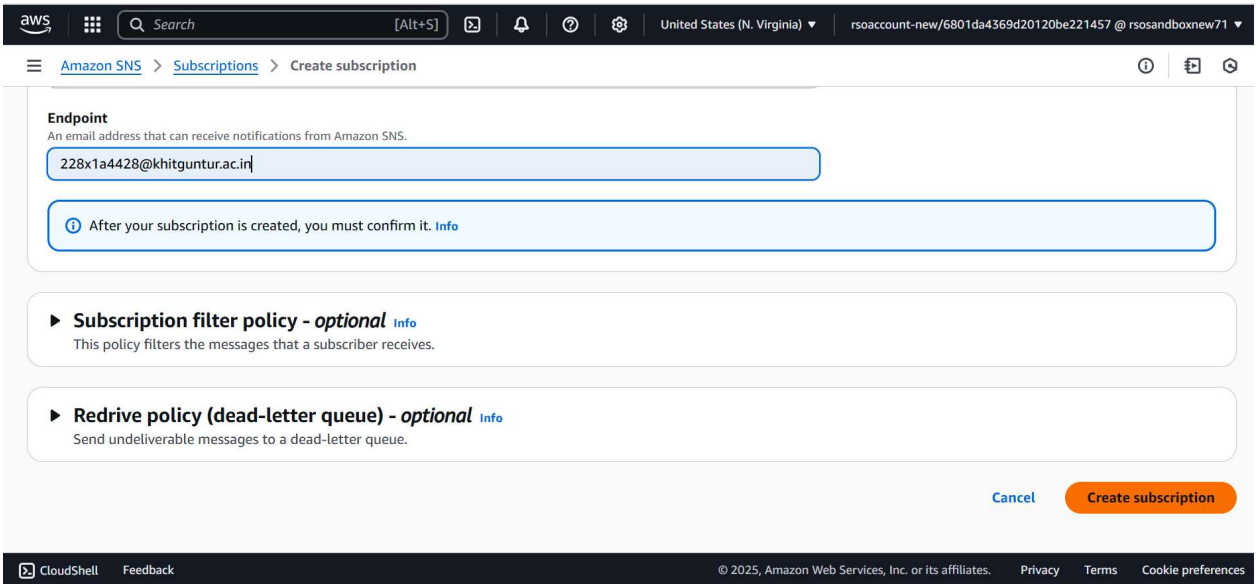


Subscribe users

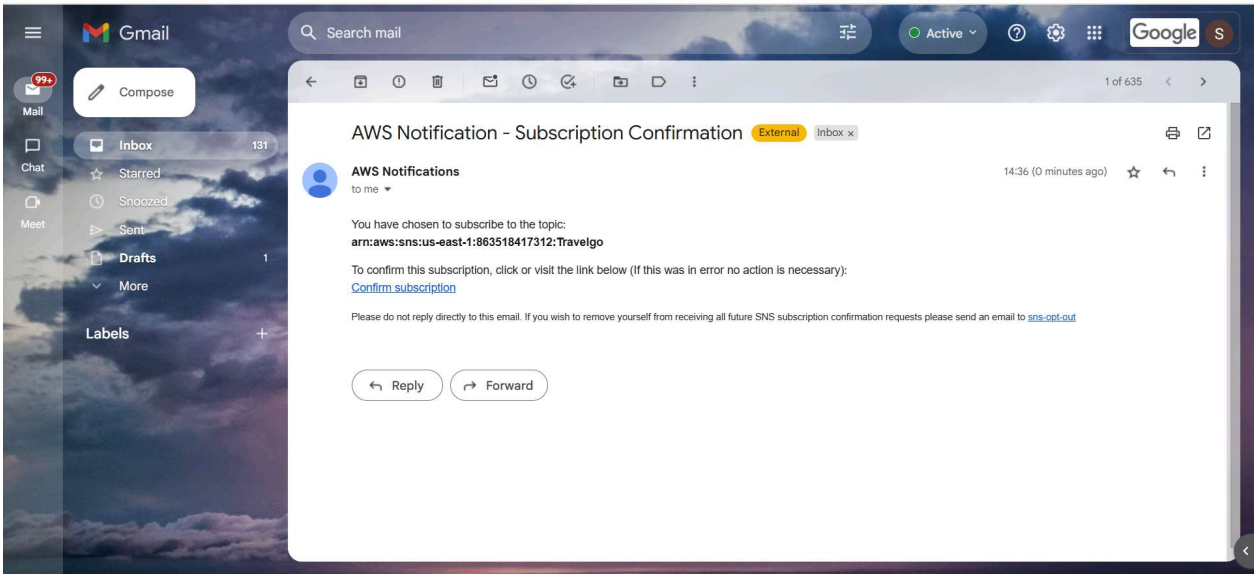
Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.



After subscription request for the mail confirmation



Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.



aws

Simple Notification Service

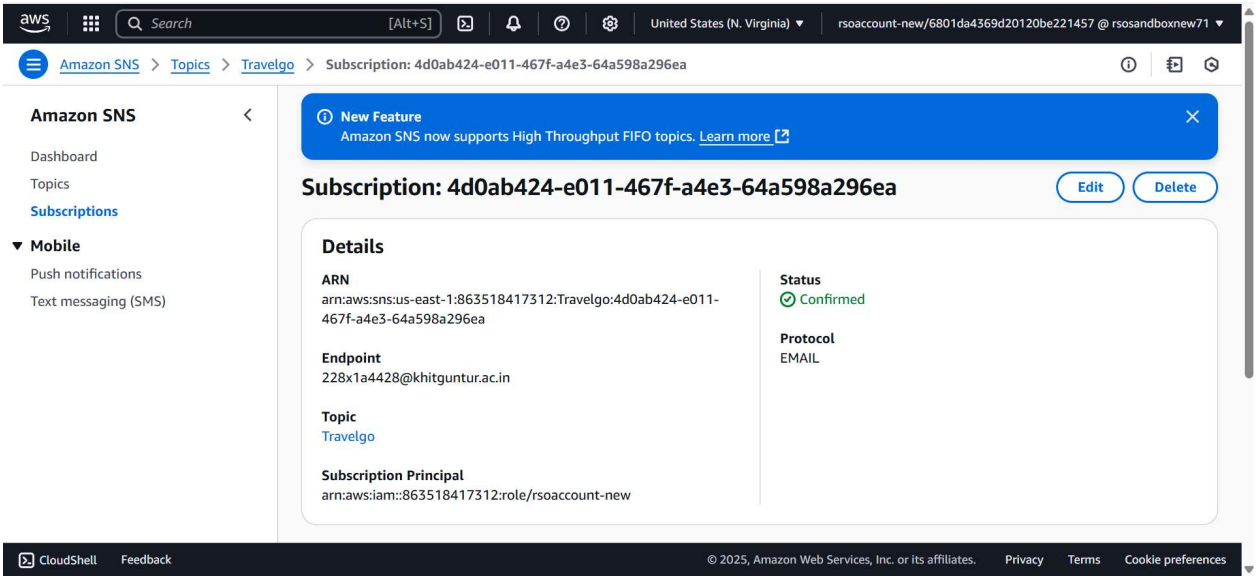
Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:
arn:aws:sns:us-east-1:863518417312:Travelgo:4d0ab424-e011-467f-a4e3-64a598a296ea

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

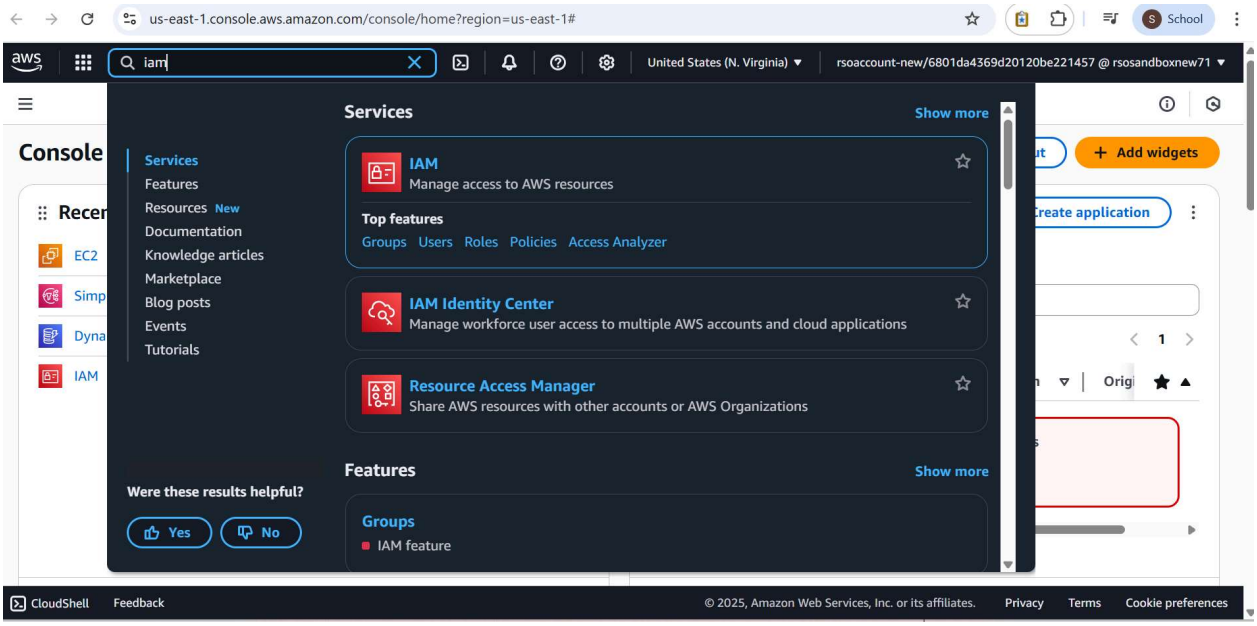
Successfully done with the SNS mail subscription and setup, now store the ARN link.

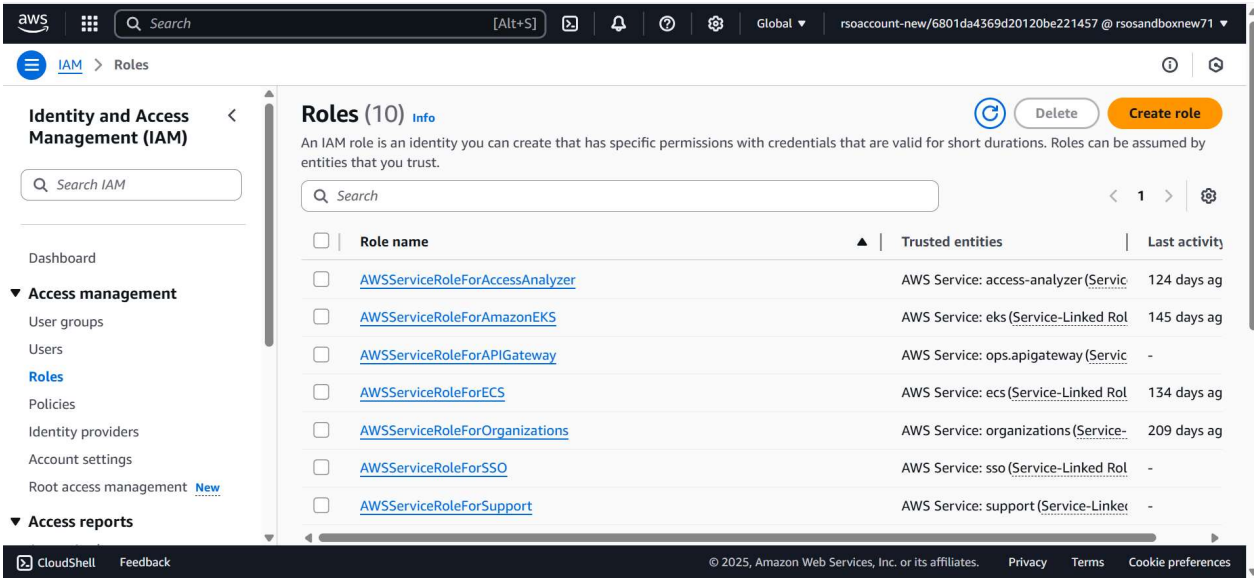


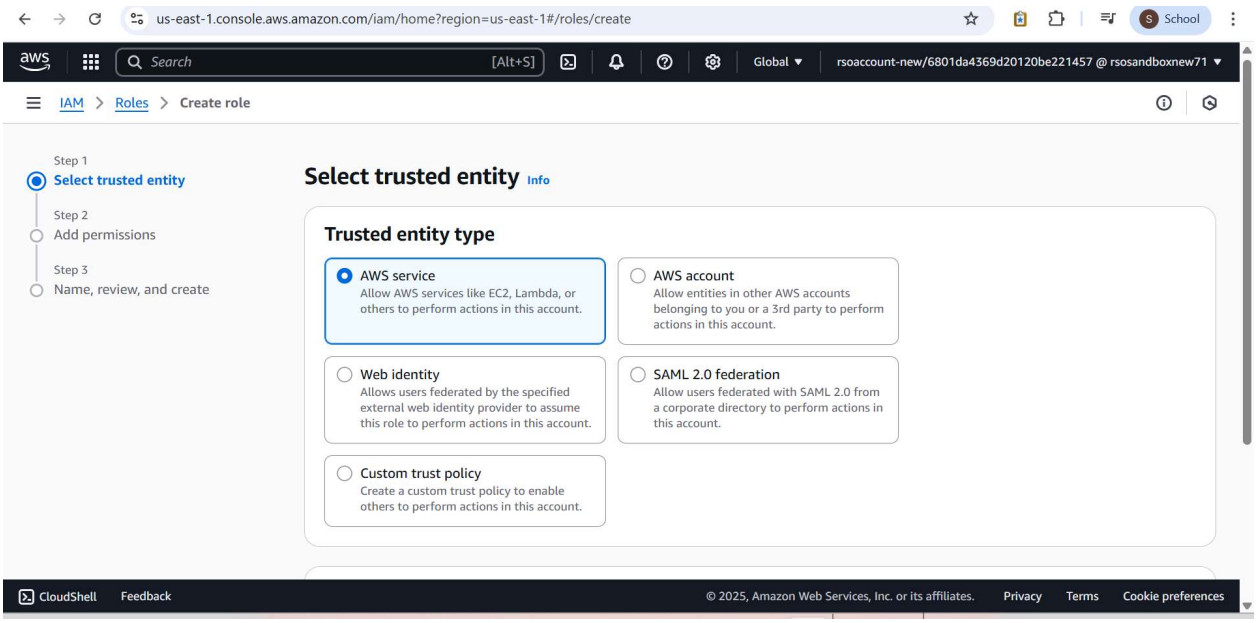
Milestone 5 : IAM Role Setup

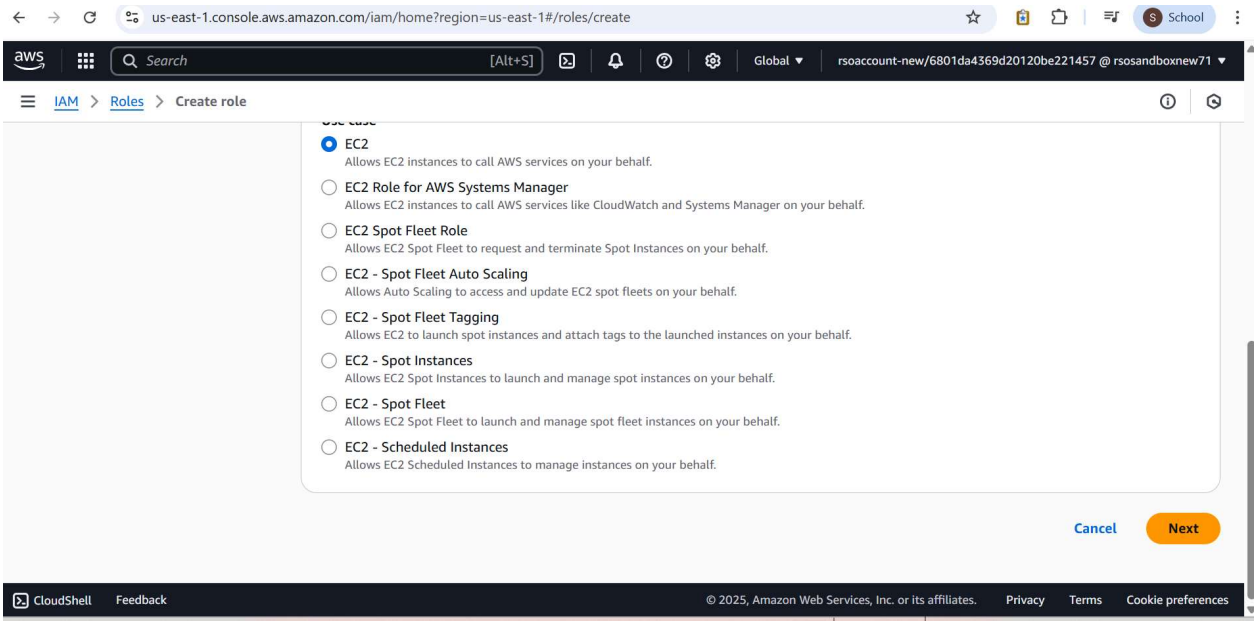
Create IAM Role.

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



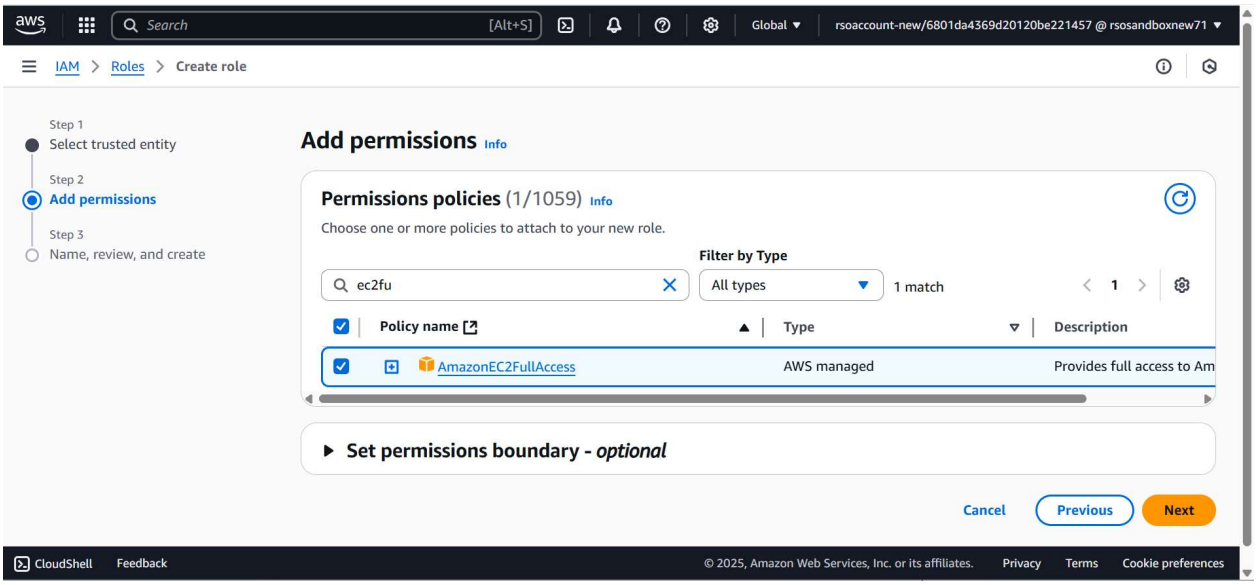

Search [Alt+S]
Global
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
IAM > Roles
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Roles (10) Info
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.
Delete
Create role
Search
Role name
Trusted entities
Last activity
AWSServiceRoleForAccessAnalyzer
AWS Service: access-analyzer (Servic
124 days ag
AWSServiceRoleForAmazonEKS
AWS Service: eks (Service-Linked Rol
145 days ag
AWSServiceRoleForAPIGateway
AWS Service: ops.apigateway (Servic
-
AWSServiceRoleForECS
AWS Service: ecs (Service-Linked Rol
134 days ag
AWSServiceRoleForOrganizations
AWS Service: organizations (Service-
209 days ag
AWSServiceRoleForSSO
AWS Service: sso (Service-Linked Rol
-
AWSServiceRoleForSupport
AWS Service: support (Service-Linke
-
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

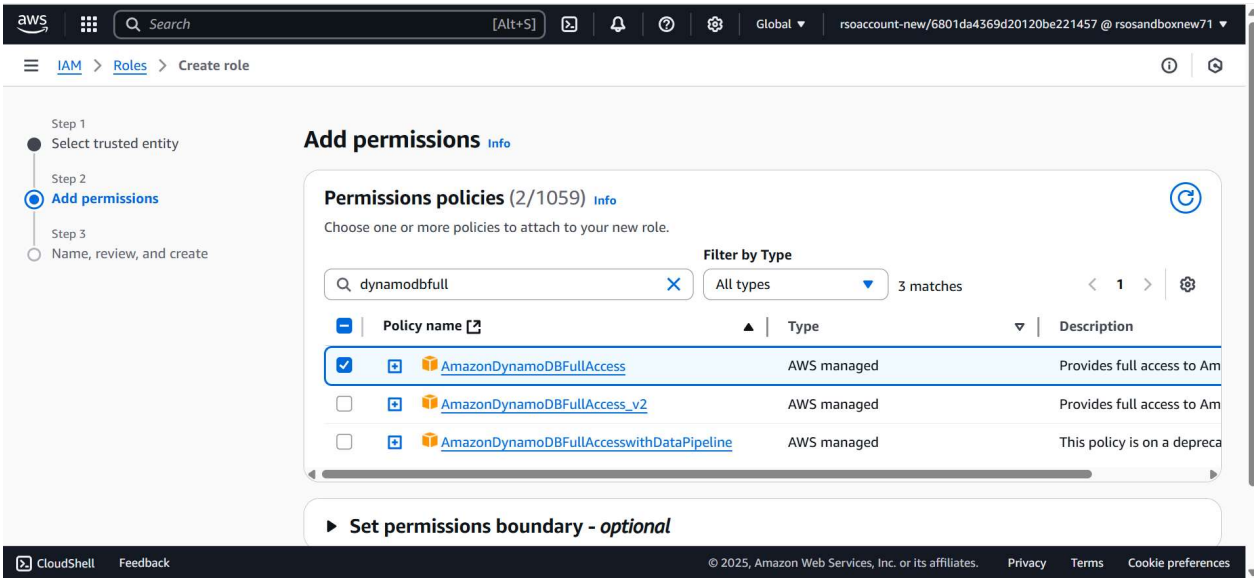

us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/create
School
Search [Alt+S]
Global
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Select trusted entity Info
Trusted 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
Allows users federated 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.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

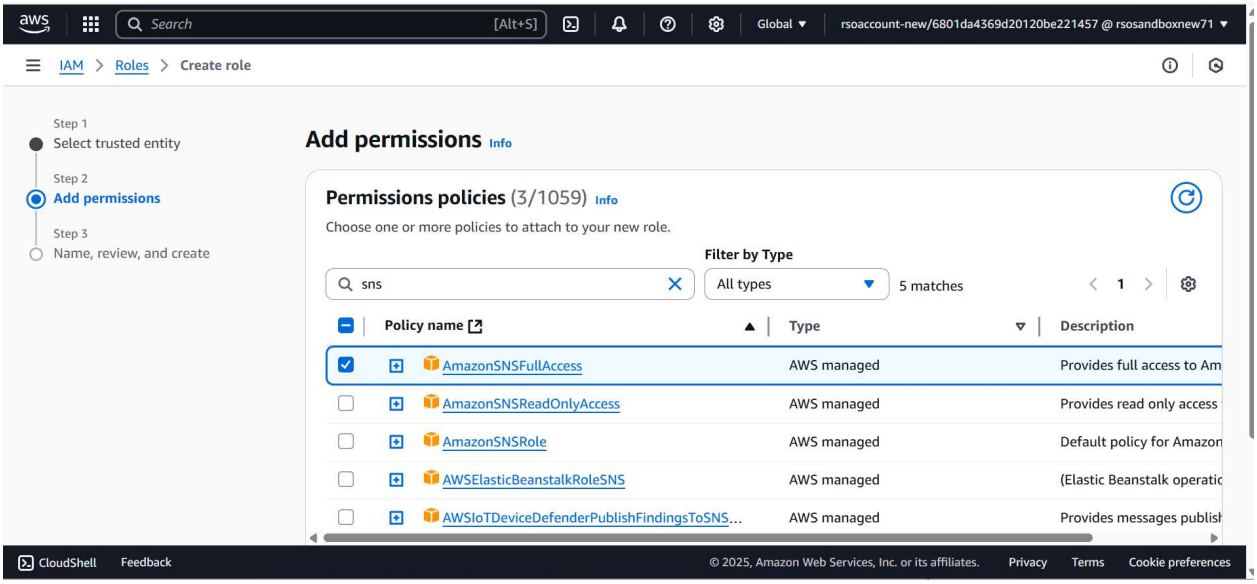


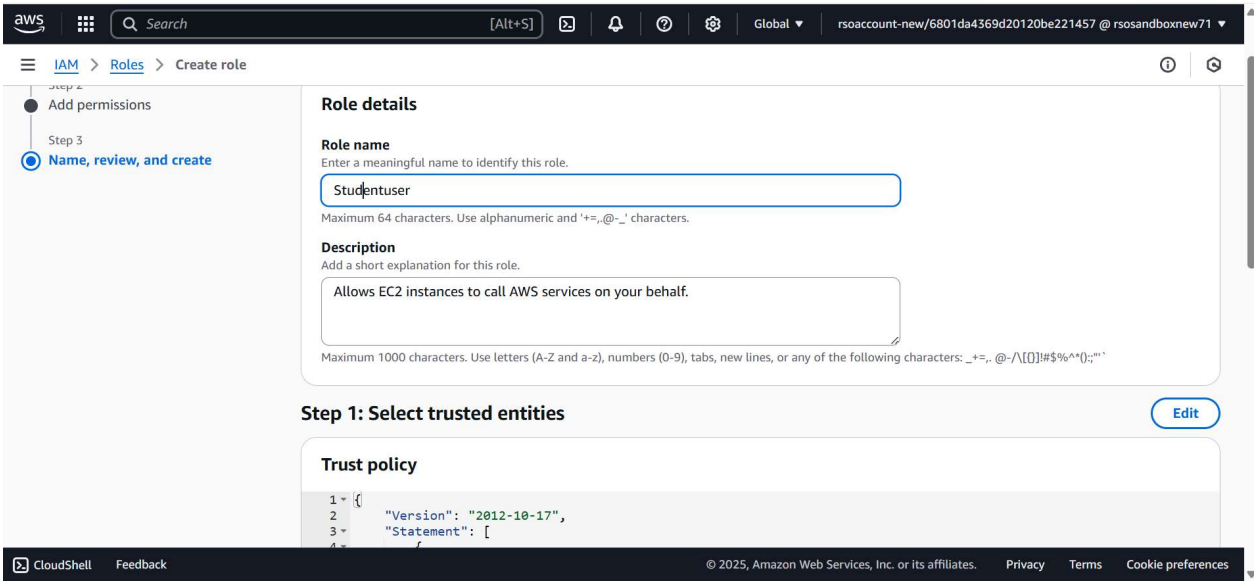
Attach the following policies to the role:

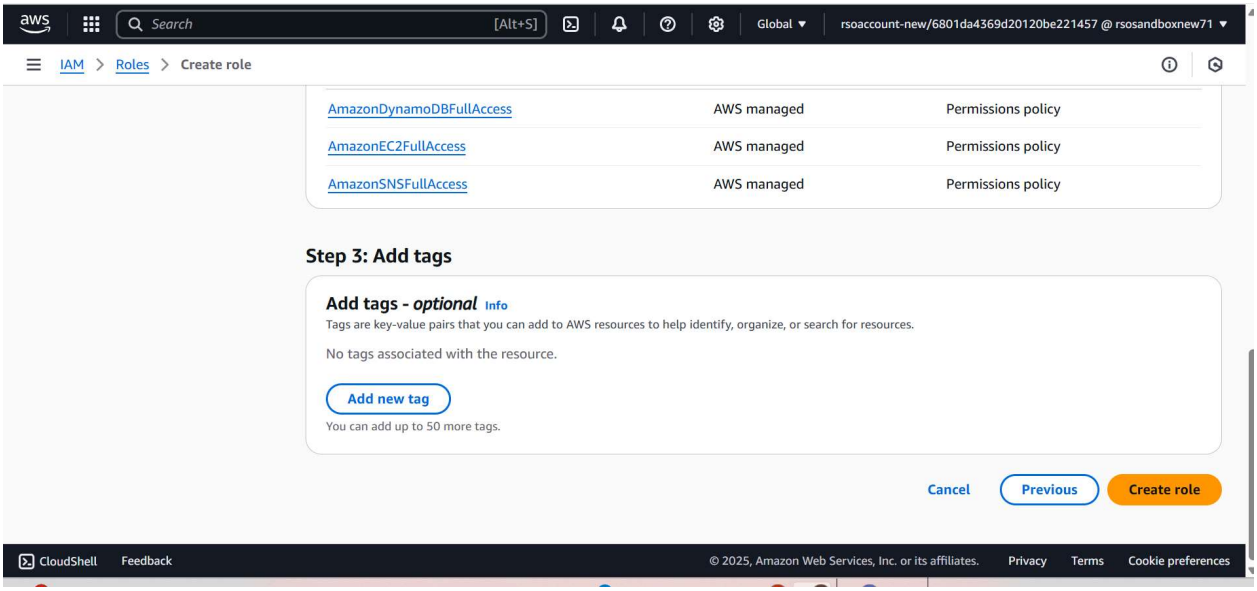
- **AmazonDynamoDBFullAccess**: Allows EC2 to perform read/write operations on DynamoDB.
- **AmazonSNSFullAccess**: Grants EC2 the ability to send notifications via SNS.

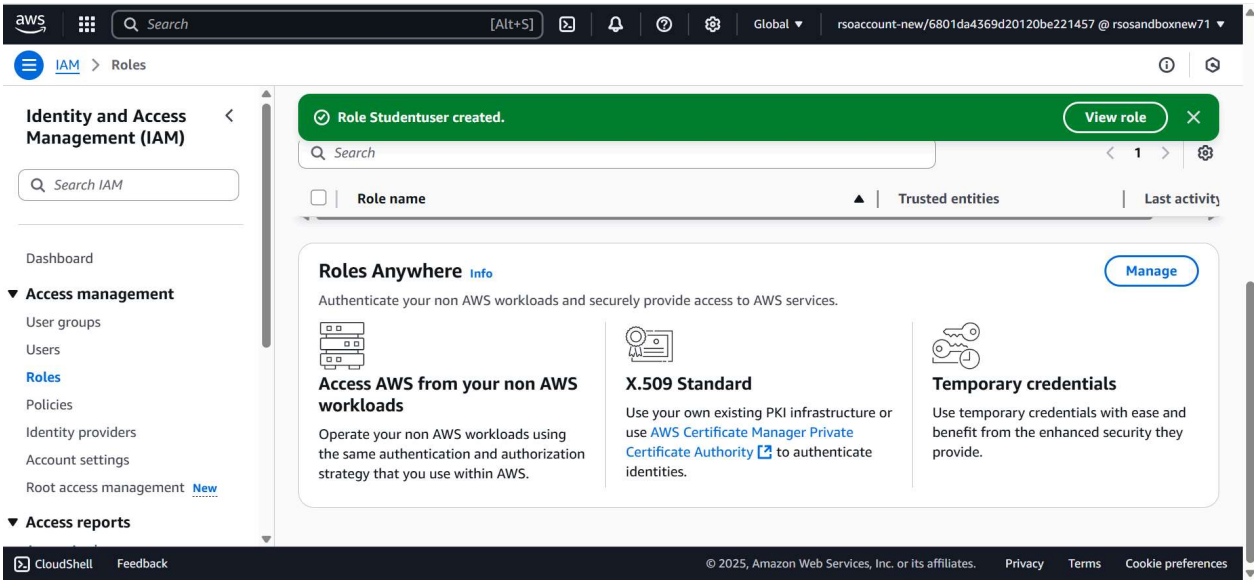



Search [Alt+S]
Global
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (2/1059) Info
Choose one or more policies to attach to your new role.
Filter by Type
dynamodbfull
All types
3 matches
1
Policy name
Type
Description
AmazonDynamoDBFullAccess
AWS managed
Provides full access to Am
AmazonDynamoDBFullAccess_v2
AWS managed
Provides full access to Am
AmazonDynamoDBFullAccesswithDataPipeline
AWS managed
This policy is on a depreca
Set permissions boundary - optional
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
Global
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (3/1059) Info
Choose one or more policies to attach to your new role.
Filter by Type
sns
All types
5 matches
1
Policy name
Type
Description
AmazonSNSFullAccess
AWS managed
Provides full access to Am
AmazonSNSReadOnlyAccess
AWS managed
Provides read only access
AmazonSNSRole
AWS managed
Default policy for Amazon
AWSElasticBeanstalkRoleSNS
AWS managed
(Elastic Beanstalk operatic
AWSIoTDeviceDefenderPublishFindingsToSNS...
AWS managed
Provides messages publish
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
Global
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
IAM > Roles > Create role
Add permissions
Step 3
Name, review, and create
Role details
Role name
Enter a meaningful name to identify this role.
Studentuser
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.
Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*();:"'`
Step 1: Select trusted entities
Edit
Trust policy
{
"Version": "2012-10-17",
"Statement": [
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

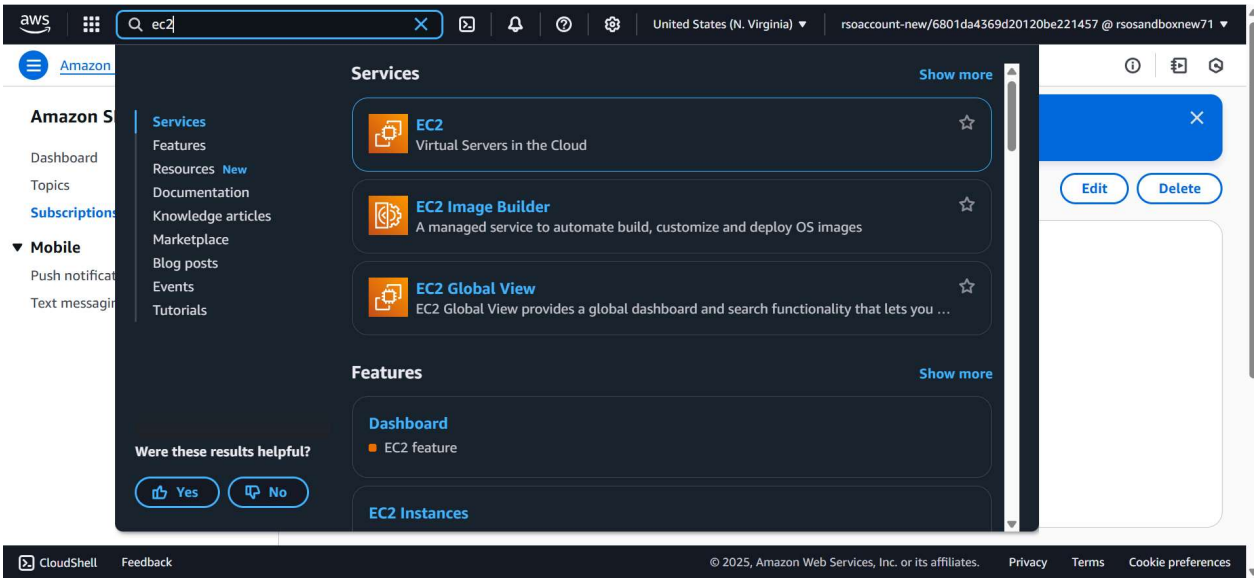

Search [Alt+S]
Global
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
IAM > Roles > Create role
AmazonDynamoDBFullAccess AWS managed Permissions policy
AmazonEC2FullAccess AWS managed Permissions policy
AmazonSNSFullAccess AWS managed Permissions policy
Step 3: Add tags
Add tags - optional Info
Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.
No tags associated with the resource.
Add new tag
You can add up to 50 more tags.
Cancel
Previous
Create role
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



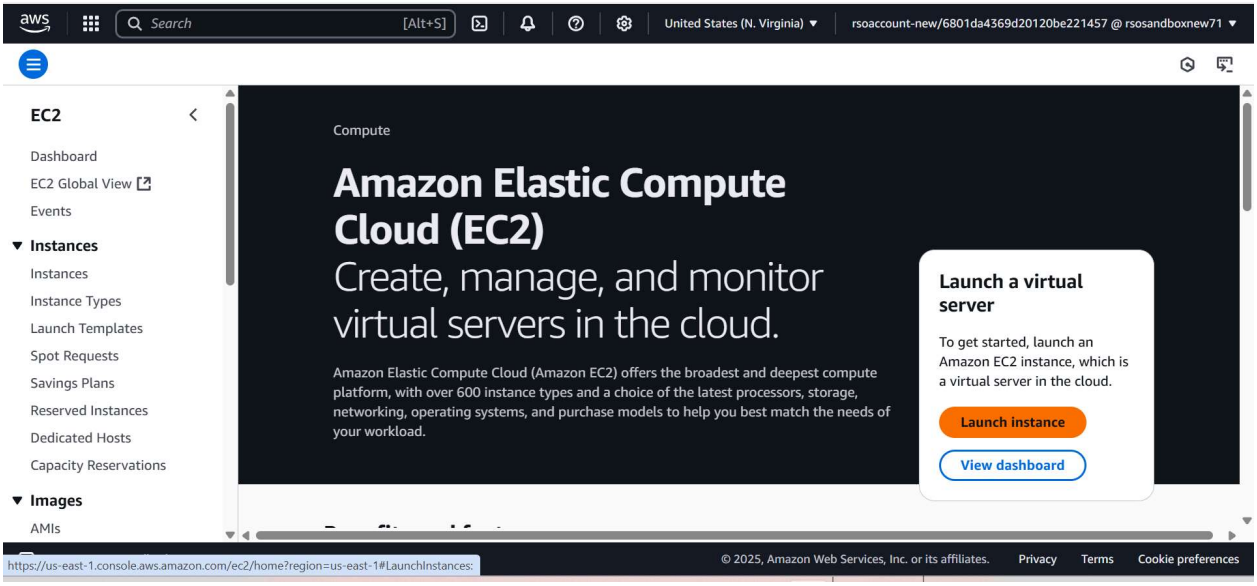
Milestone 6 : EC2 Instance Setup

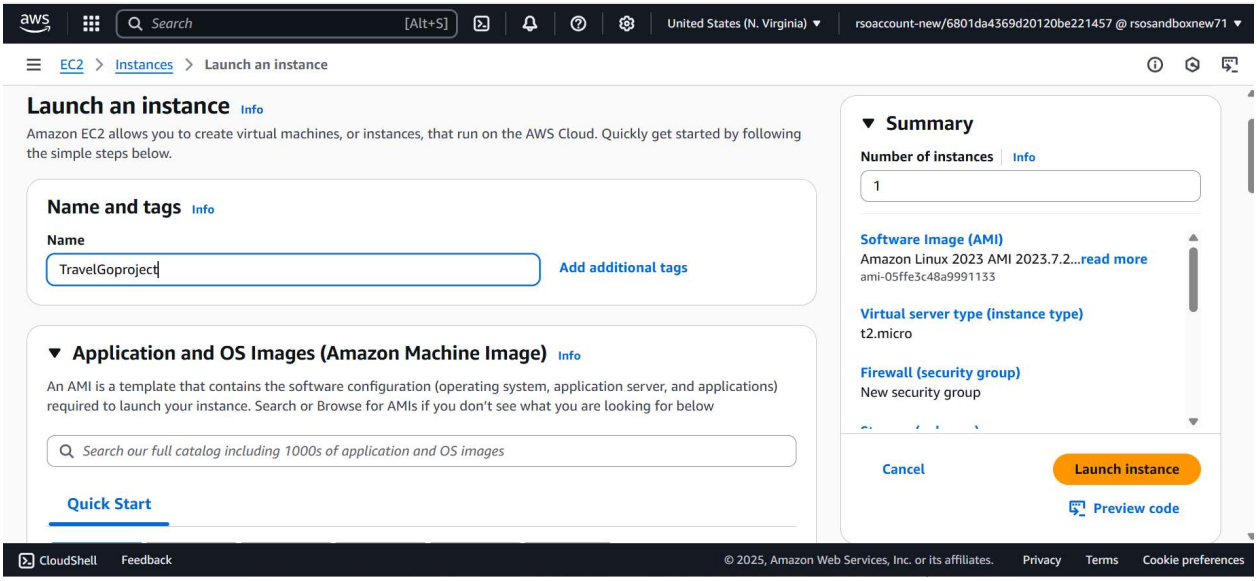
Load your Project Files to Github

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

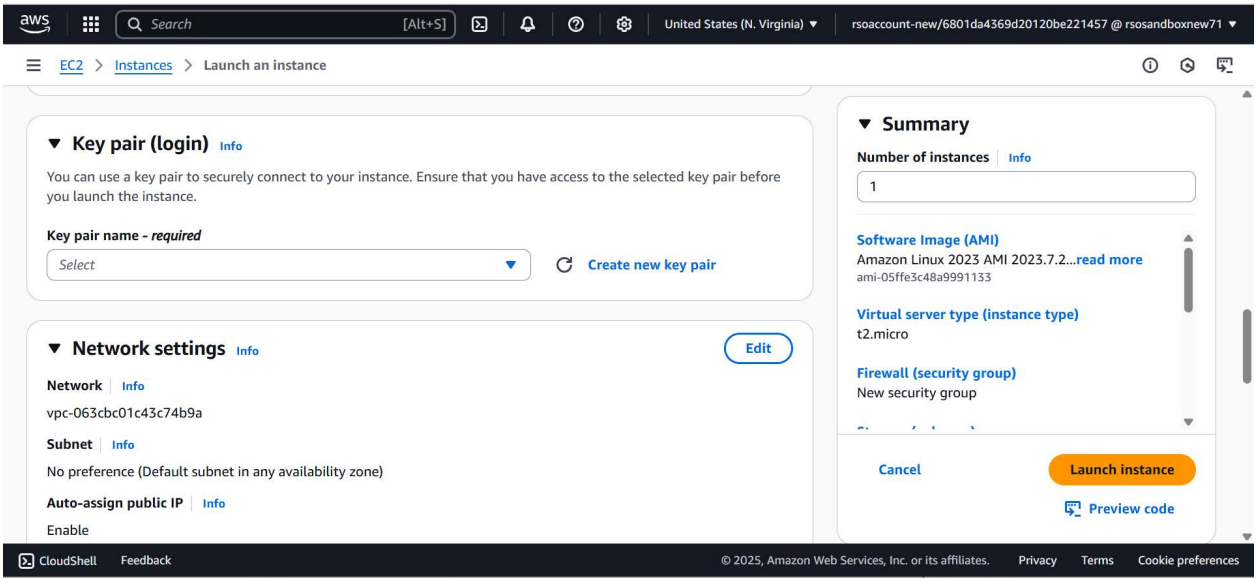


- Click on Launch instance to launch EC2 instance

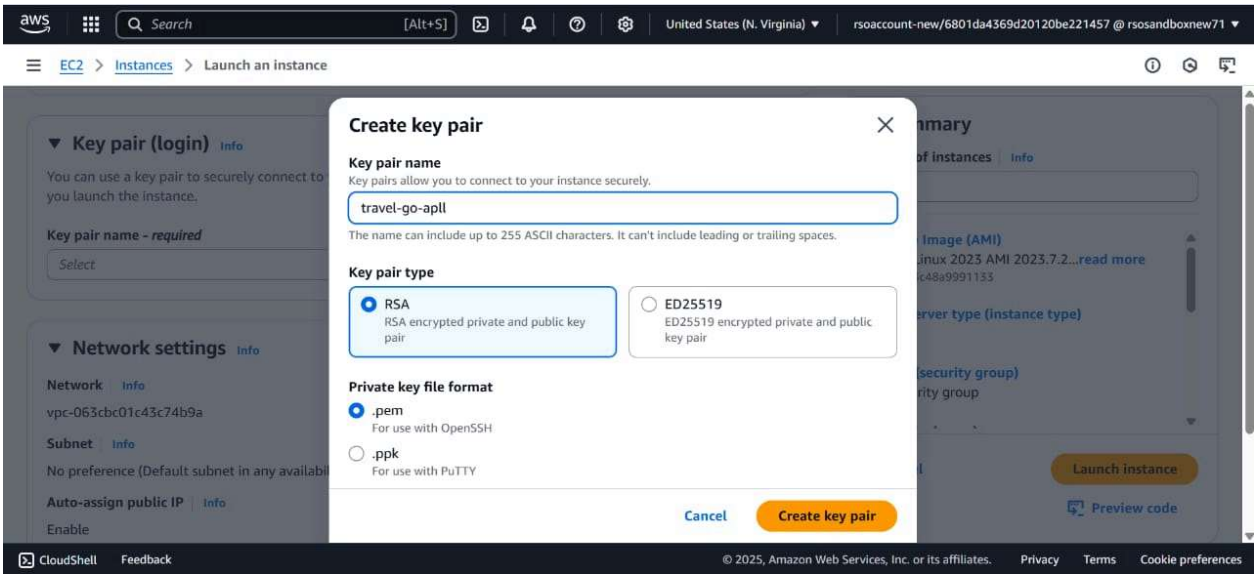


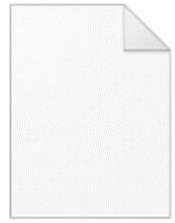


- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

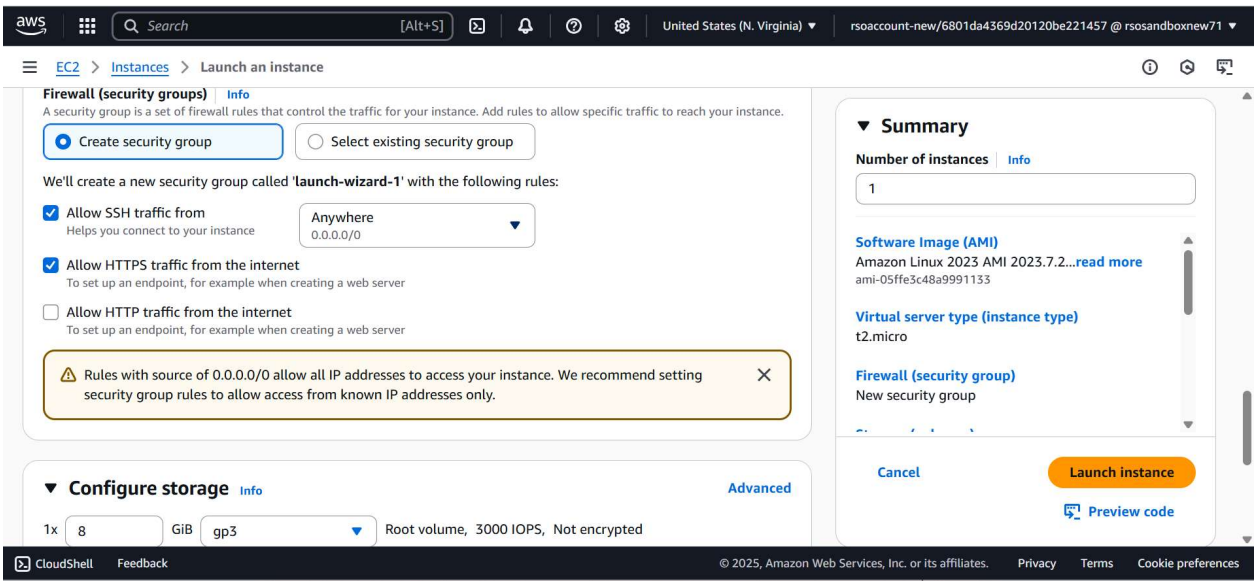


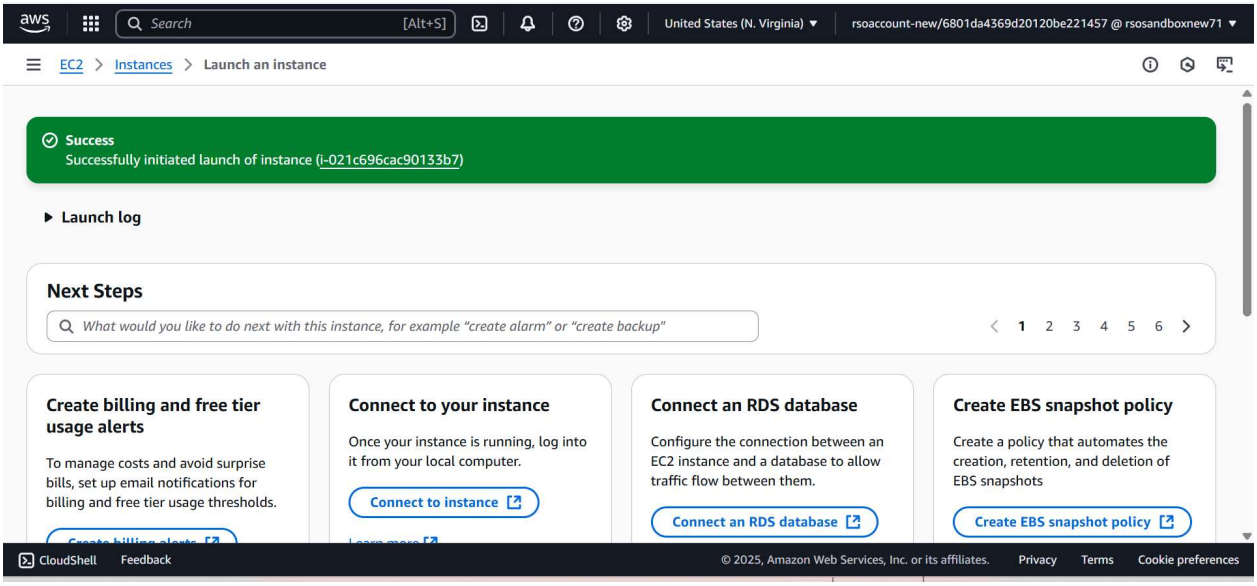
Create and download the key pair for Server access.



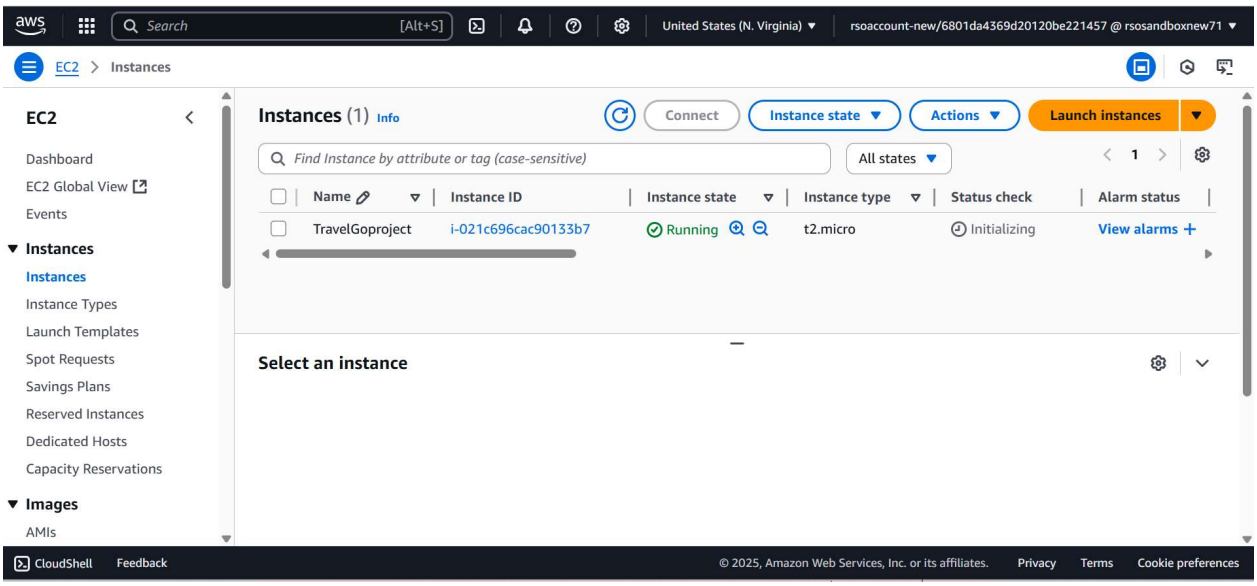


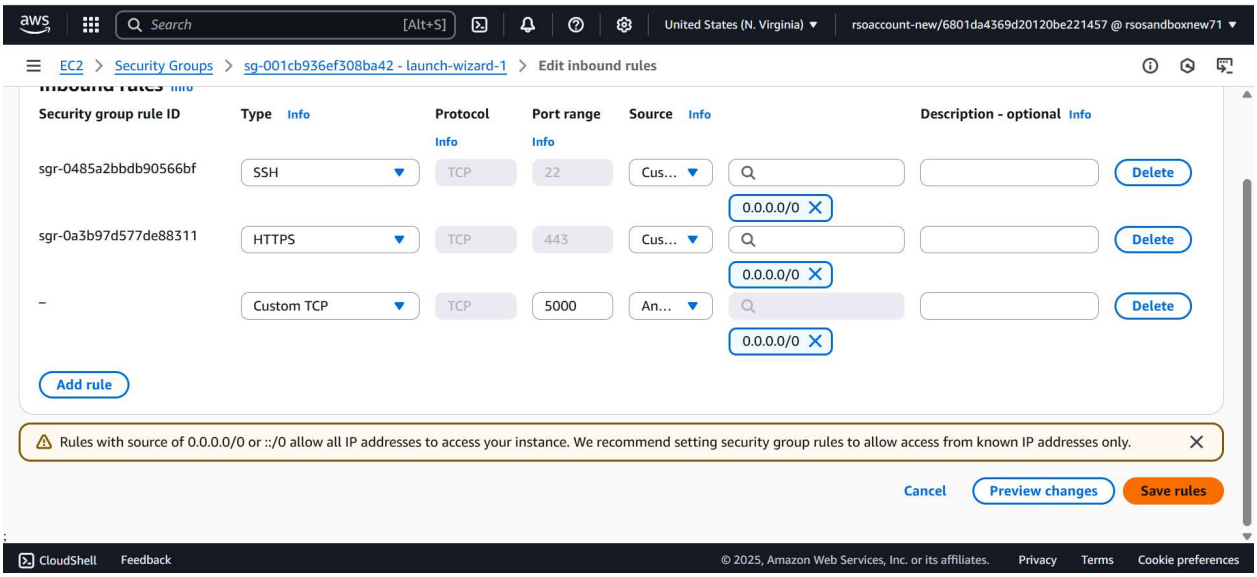
InstantLibrary.pem

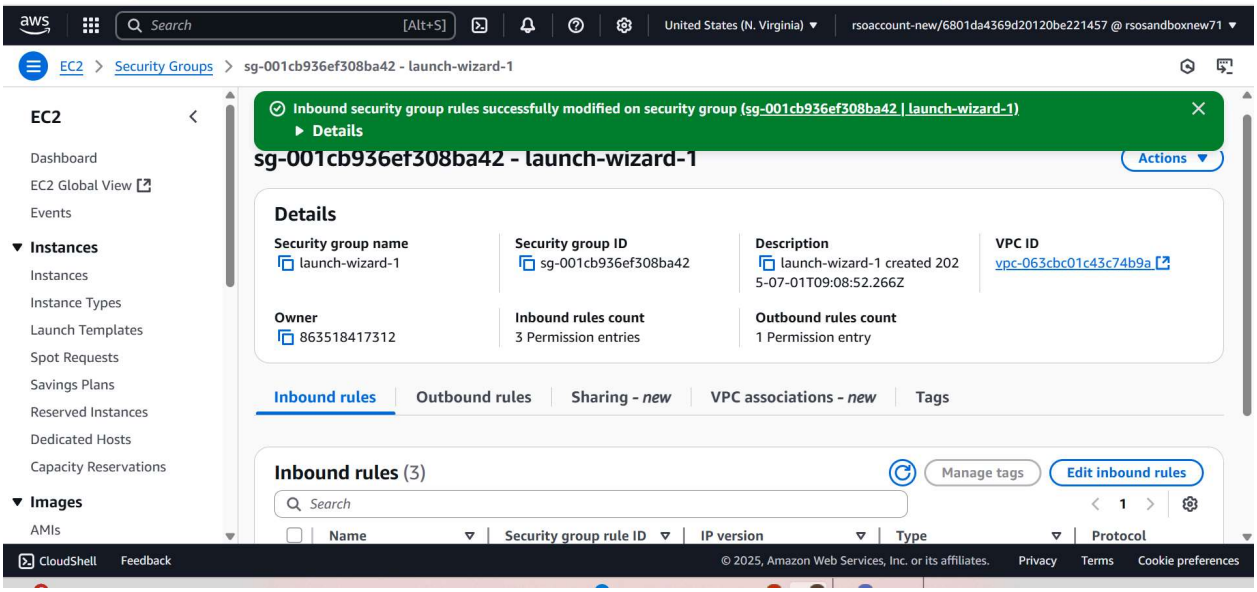


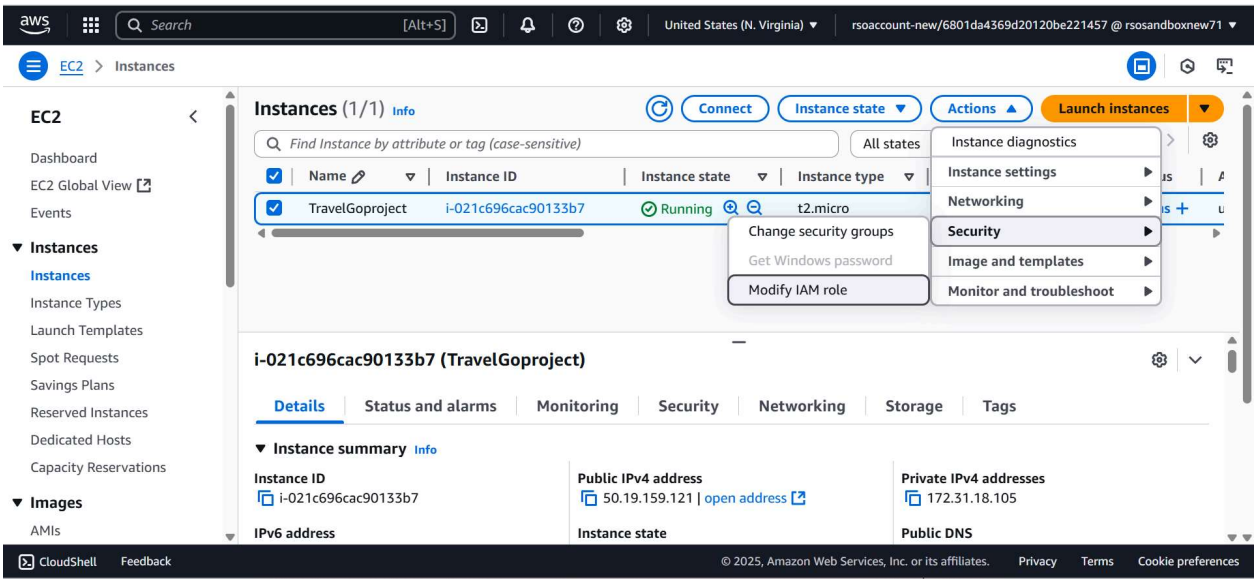


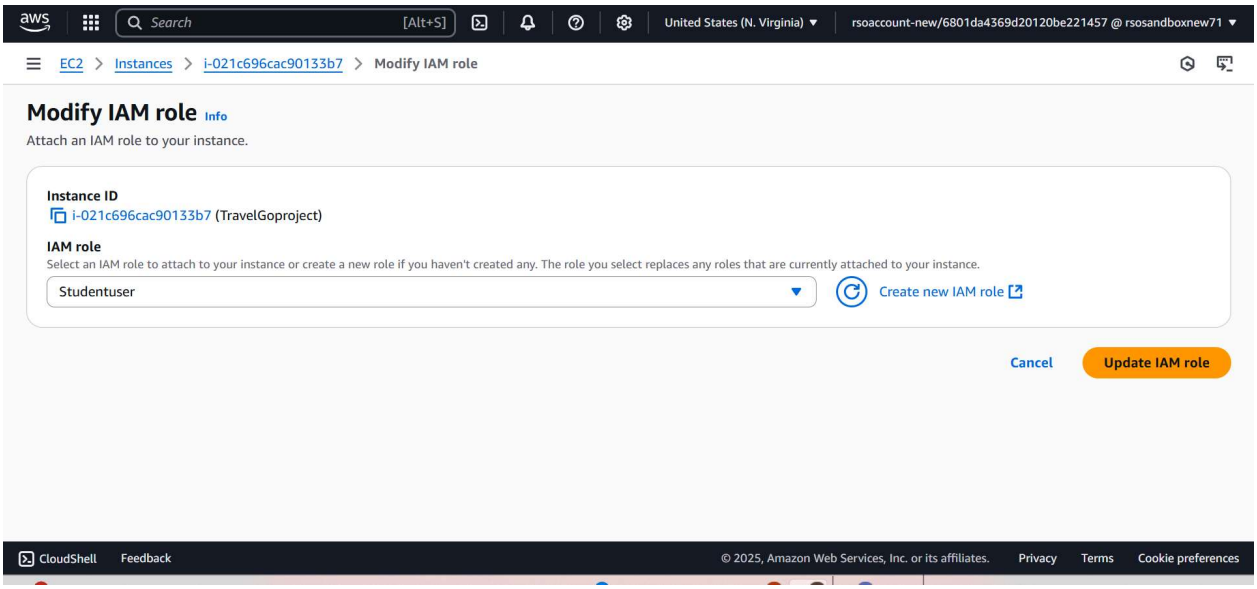
Configure security groups for HTTP, and SSH access:

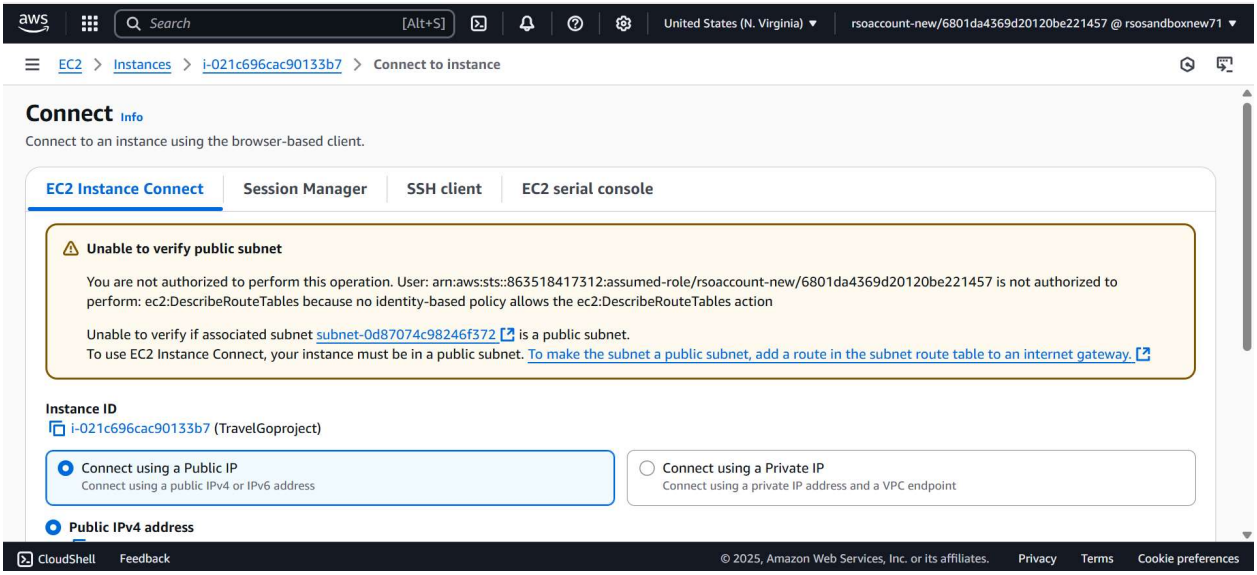

aws
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
Instances (1) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
1
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
TravelGoproject
i-021c696cac90133b7
Running
t2.micro
Initializing
View alarms +
Select an instance
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

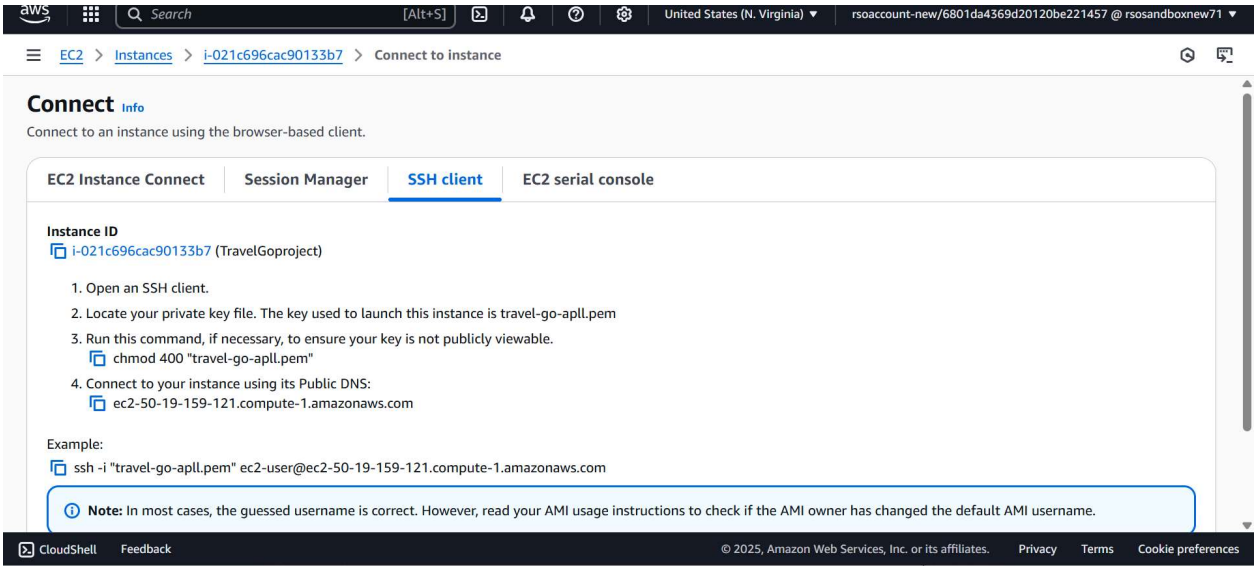

aws
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
EC2 > Security Groups > sg-001cb936ef308ba42 - launch-wizard-1 > Edit inbound rules
Security group rule ID
Type Info
Protocol Info
Port range Info
Source Info
Description - optional Info
sgr-0485a2bbdb90566bf
SSH
TCP
22
Cus...
0.0.0.0/0
Delete
sgr-0a3b97d577de88311
HTTPS
TCP
443
Cus...
0.0.0.0/0
Delete
–
Custom TCP
TCP
5000
An...
0.0.0.0/0
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
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
EC2 > Security Groups > sg-001cb936ef308ba42 - launch-wizard-1
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
Inbound security group rules successfully modified on security group (sg-001cb936ef308ba42 | launch-wizard-1)
Details
sg-001cb936ef308ba42 - launch-wizard-1
Actions
Details
Security group name
launch-wizard-1
Security group ID
sg-001cb936ef308ba42
Description
launch-wizard-1 created 202
5-07-01T09:08:52.266Z
VPC ID
vpc-063cbc01c43c74b9a
Owner
863518417312
Inbound rules count
3 Permission entries
Outbound rules count
1 Permission entry
Inbound rules
Outbound rules
Sharing - new
VPC associations - new
Tags
Inbound rules (3)
Manage tags
Edit inbound rules
Search
Name
Security group rule ID
IP version
Type
Protocol
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

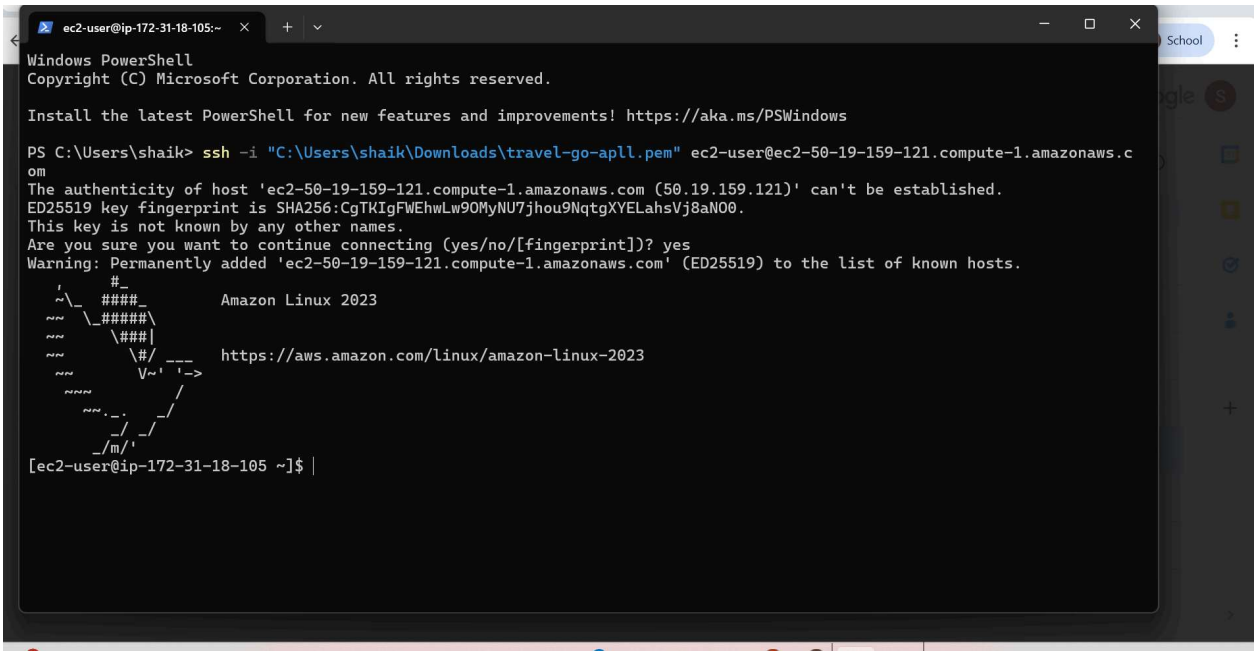

Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
Instances (1/1) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
TravelGoproject
i-021c696cac90133b7
Running
t2.micro
Instance diagnostics
Instance settings
Networking
Security
Image and templates
Monitor and troubleshoot
Change security groups
Get Windows password
Modify IAM role
i-021c696cac90133b7 (TravelGoproject)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-021c696cac90133b7
Public IPv4 address
50.19.159.121 | open address
Private IPv4 addresses
172.31.18.105
IPv6 address
Instance state
Public DNS
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
EC2 > Instances > i-021c696cac90133b7 > Modify IAM role
Modify IAM role Info
Attach an IAM role to your instance.
Instance ID
i-021c696cac90133b7 (TravelGoproject)
IAM role
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.
Studentuser
Create new IAM role
Cancel
Update IAM role
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71
EC2 > Instances > i-021c696cac90133b7 > Connect to instance
Connect Info
Connect to an instance using the browser-based client.
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Unable to verify public subnet
You are not authorized to perform this operation. User: arn:aws:sts::863518417312:assumed-role/rsoaccount-new/6801da4369d20120be221457 is not authorized to perform: ec2:DescribeRouteTables because no identity-based policy allows the ec2:DescribeRouteTables action
Unable to verify if associated subnet subnet-0d87074c98246f372 is a public subnet.
To use EC2 Instance Connect, your instance must be in a public subnet. To make the subnet a public subnet, add a route in the subnet route table to an internet gateway.
Instance ID
i-021c696cac90133b7 (TravelGoproject)
Connect using a Public IP
Connect using a public IPv4 or IPv6 address
Connect using a Private IP
Connect using a private IP address and a VPC endpoint
Public IPv4 address
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



- Now connect the EC2 with the files



Milestone 7: Deployment using EC2

Install Software on the EC2 Instance

Install Python3, Flask, and Git:

- **On Amazon Linux 2:**

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

- **Verify Installations:**

  flask --version
  git --version

Clone Your Flask Project from GitHub

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: https://github.com/2004tanveer/TravelGo-A-Travel-booking-platfrom

This will download your project to the EC2 instance.

- To navigate to the project directory, run the following command: cd Travelgo2
- Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:
- Run the Flask Application
- export SNS_TOPIC_ARN=ARN:arn:aws:sns:us-east-1:664418997405:Travelgo:8b84921f-b2a4-41a4-965b-10a7622a81fd
- git pull origin main (Only if you made changes in git and want to reflect them in EC2 terminal.)

```
# Install requirements

pip install flask boto3
```

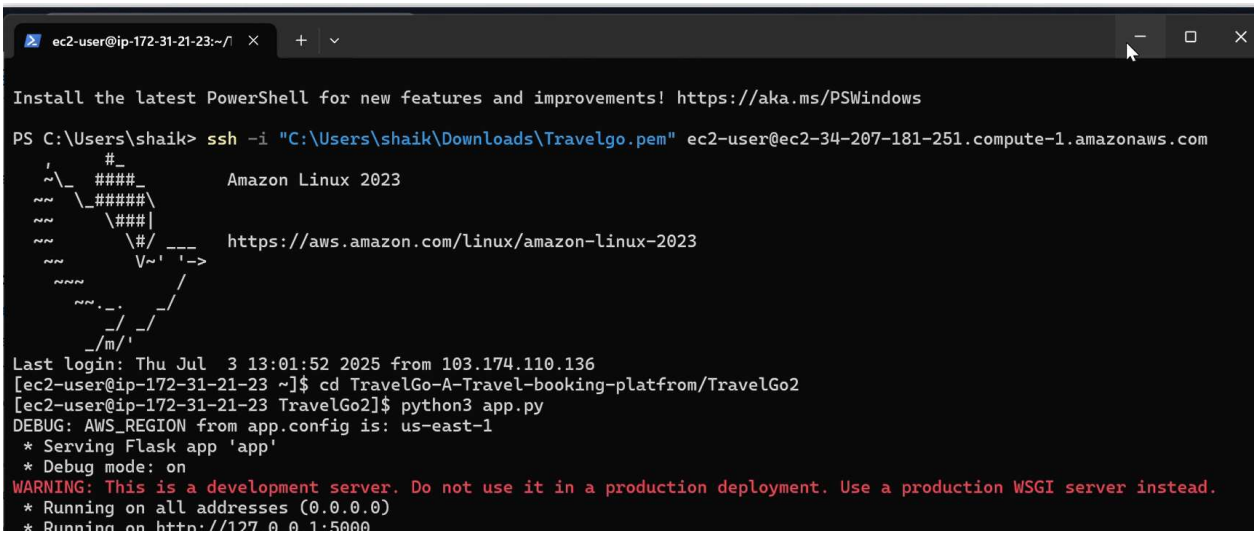
- Launch the Flask app:

sudo -E venv/bin/python3 app.py

Verify the Flask app is running:

http://your-ec2-public-ip

- Run the Flask app on the EC2 instance



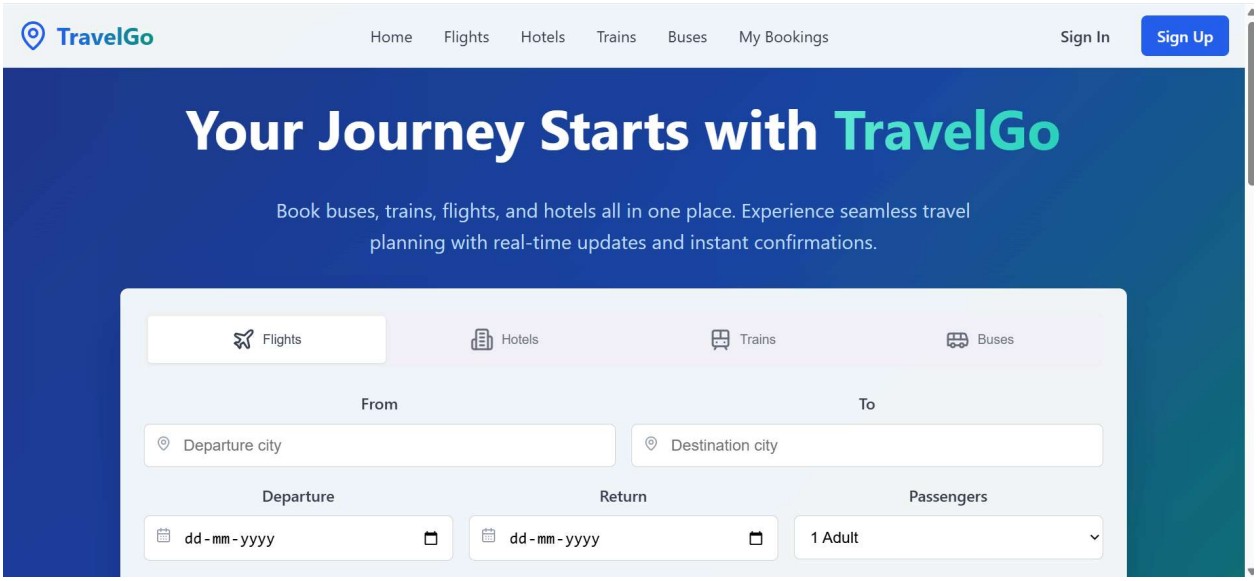
- Access the website through:
  PublicIPs: http://34.207.181.251:5000

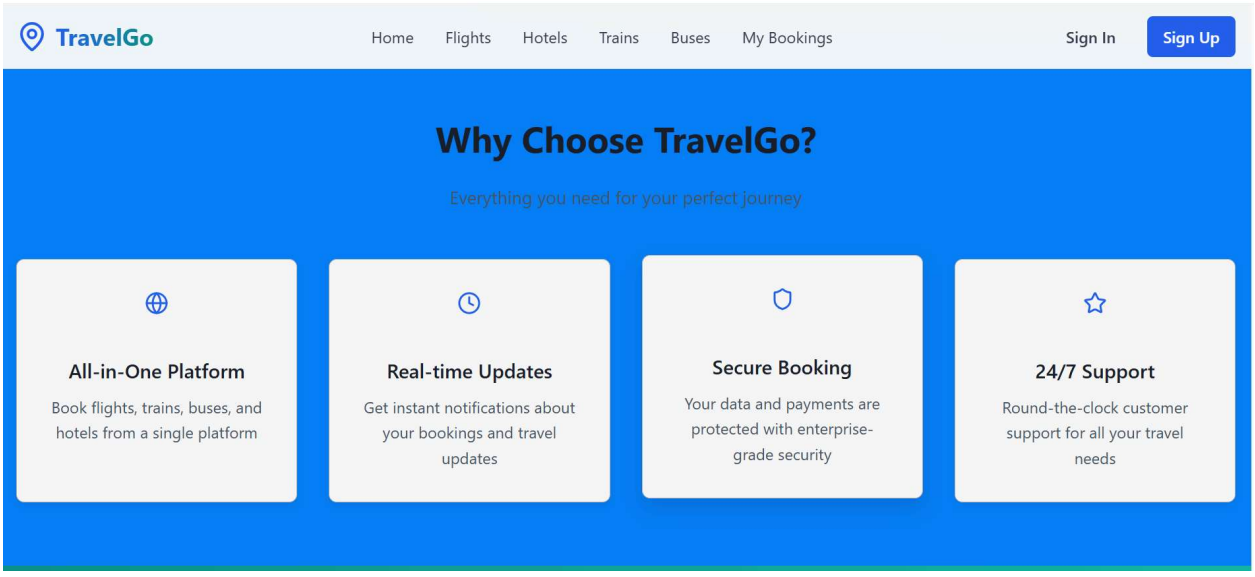
Milestone 8 : Testing and Deployment

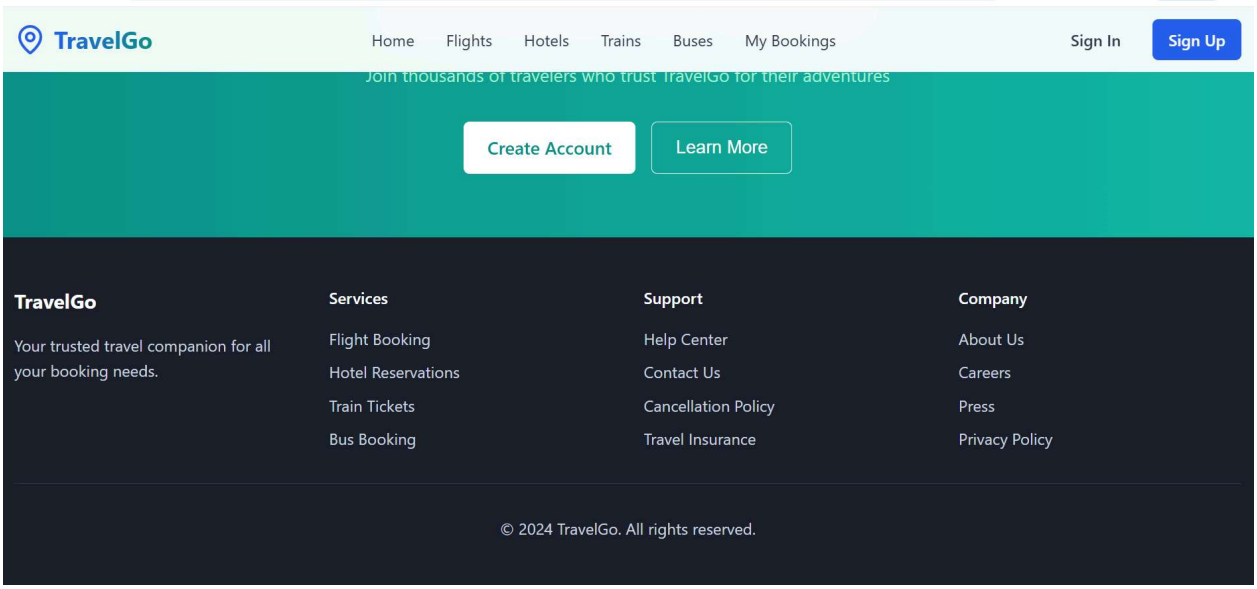
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment.

Functional Testing to verify the Project

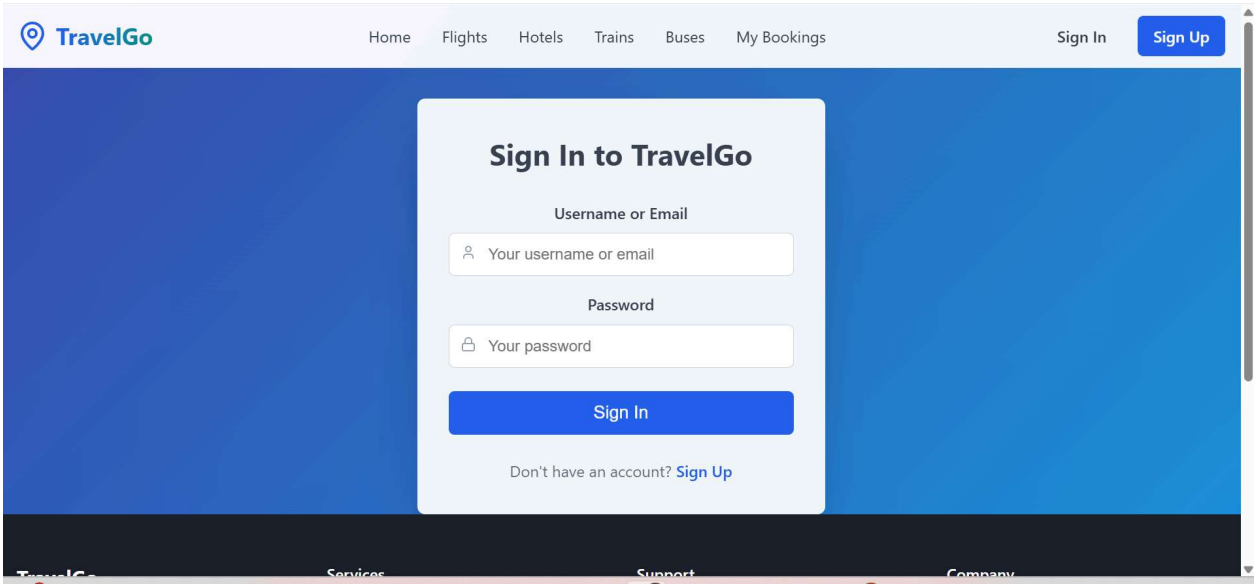
Home Page:



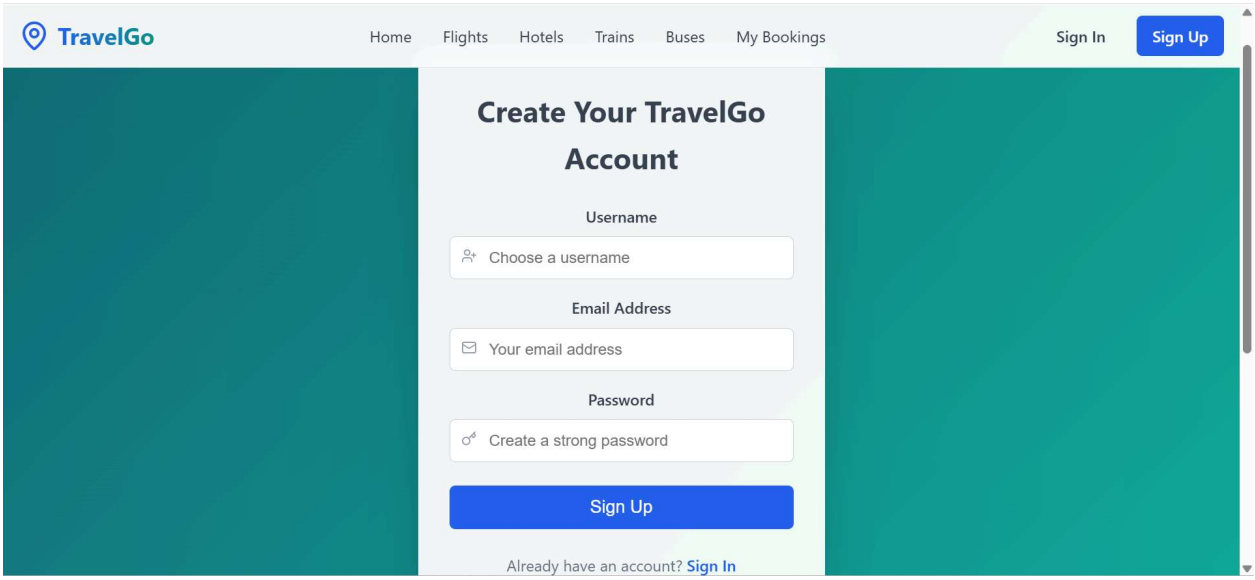



TravelGo
Home
Flights
Hotels
Trains
Buses
My Bookings
Sign In
Sign Up
Join thousands of travelers who trust TravelGo for their adventures
Create Account
Learn More
TravelGo
Your trusted travel companion for all your booking needs.
Services
Flight Booking
Hotel Reservations
Train Tickets
Bus Booking
Support
Help Center
Contact Us
Cancellation Policy
Travel Insurance
Company
About Us
Careers
Press
Privacy Policy
© 2024 TravelGo. All rights reserved.

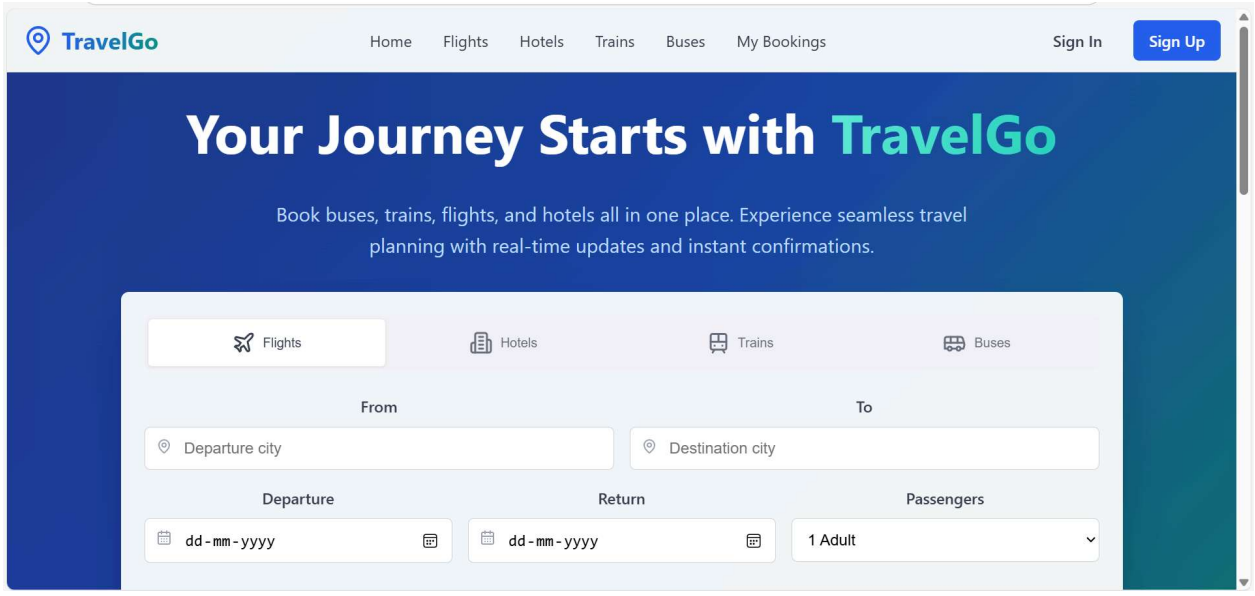
Sign In Page:


TravelGo
Home
Flights
Hotels
Trains
Buses
My Bookings
Sign In
Sign Up
Sign In to TravelGo
Username or Email
Your username or email
Password
Your password
Sign In
Don't have an account? Sign Up

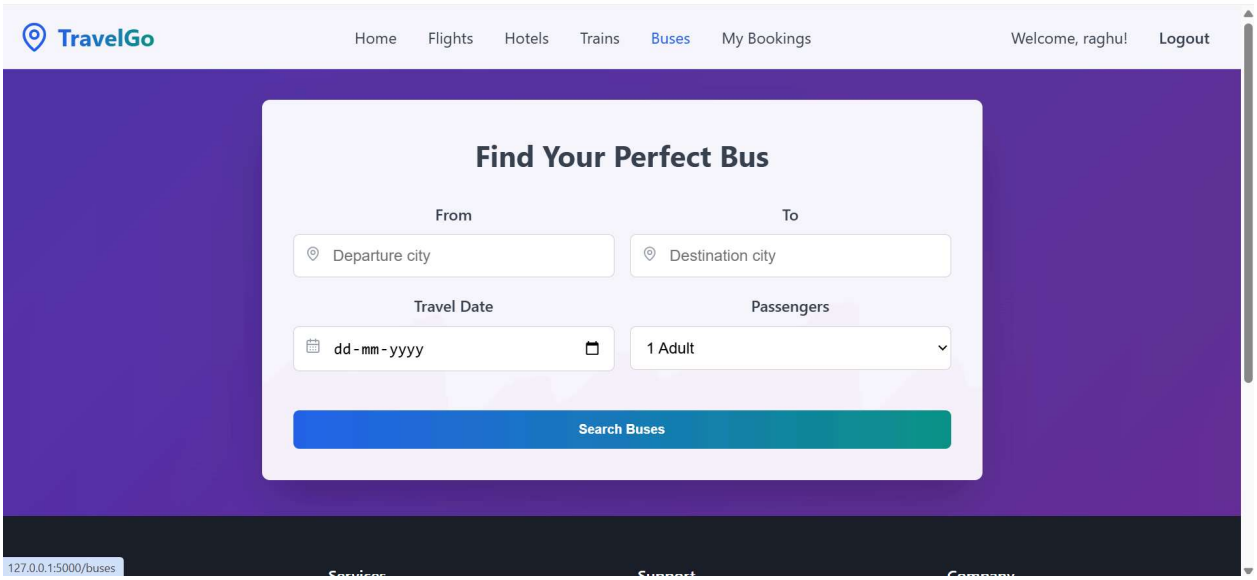
Sign Up Page:

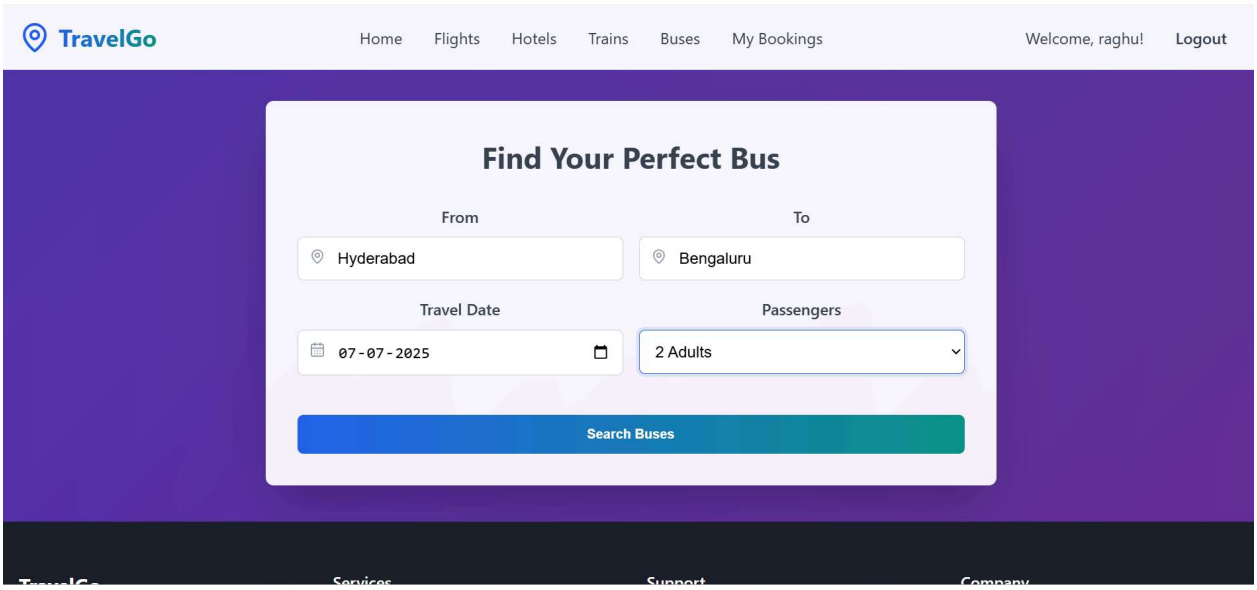


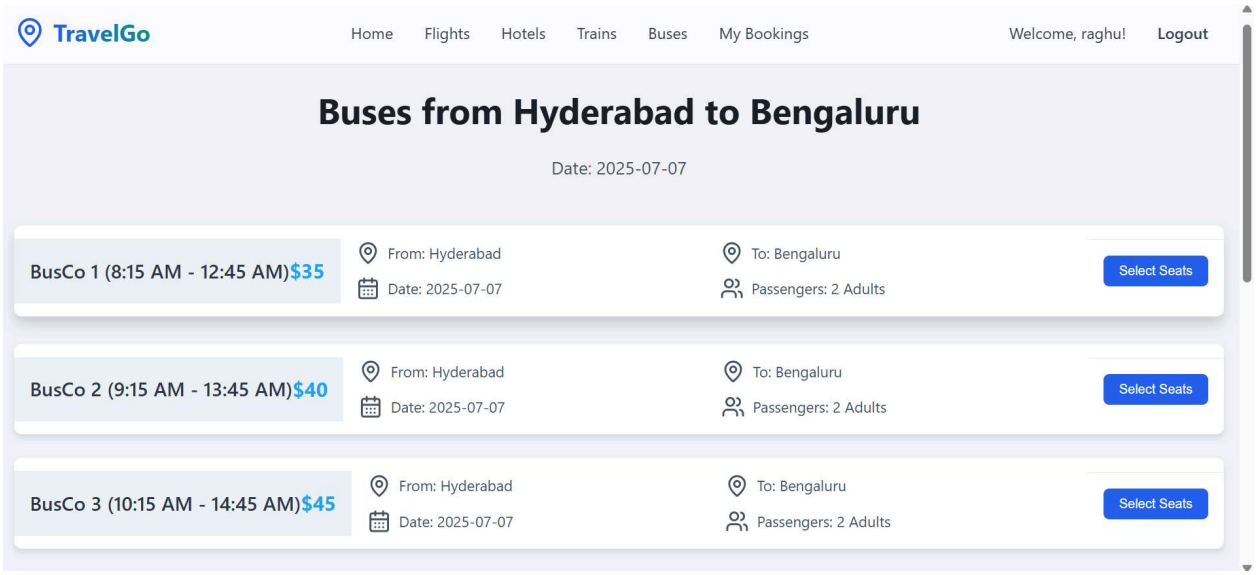
Dashboard page:

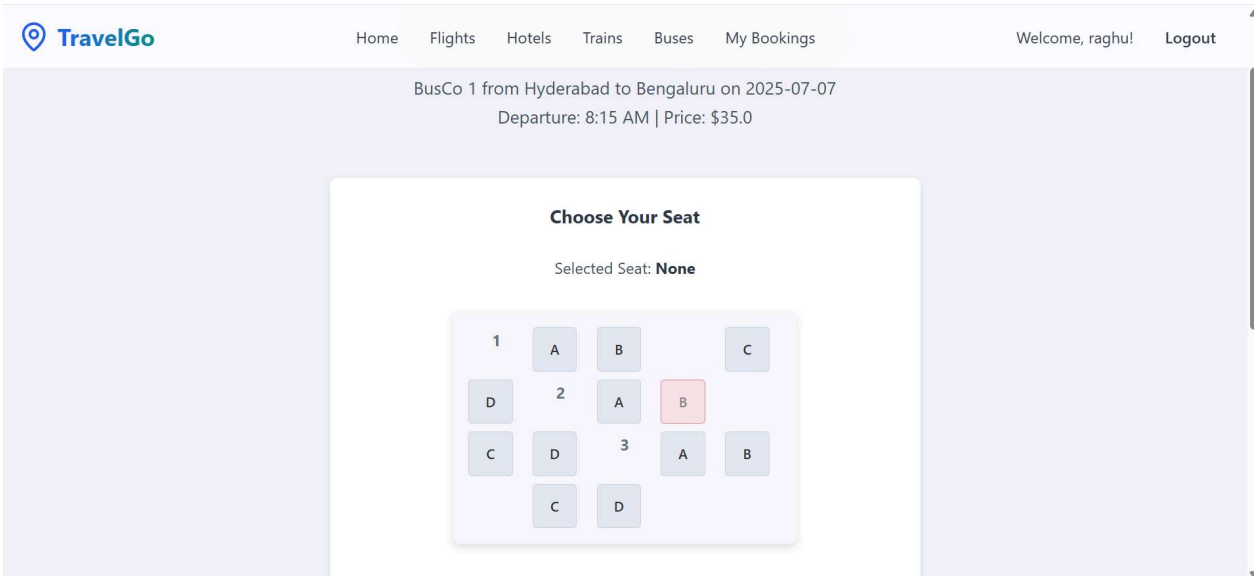


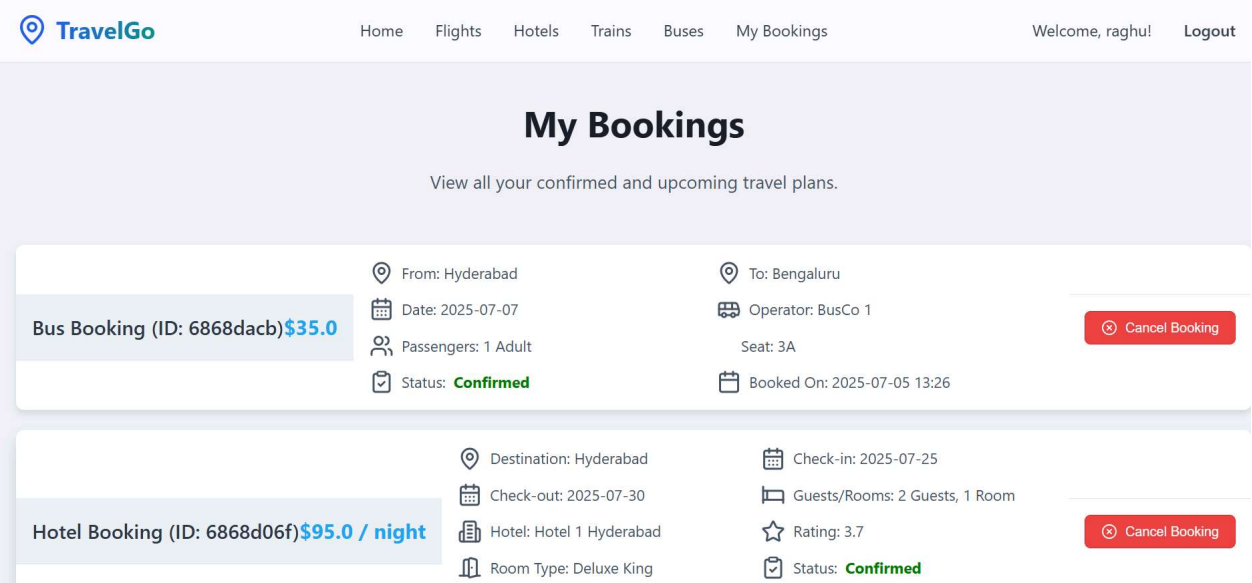
Buses page:


TravelGo
Home
Flights
Hotels
Trains
Buses
My Bookings
Welcome, raghu!
Logout
Find Your Perfect Bus
From
Departure city
To
Destination city
Travel Date
dd-mm-yyyy
Passengers
1 Adult
Search Buses
127.0.0.1:5000/buses


TravelGo
Home
Flights
Hotels
Trains
Buses
My Bookings
Welcome, raghu!
Logout
Find Your Perfect Bus
From
Hyderabad
To
Bengaluru
Travel Date
07-07-2025
Passengers
2 Adults
Search Buses


TravelGo
Home
Flights
Hotels
Trains
Buses
My Bookings
Welcome, raghu!
Logout
Buses from Hyderabad to Bengaluru
Date: 2025-07-07
BusCo 1 (8:15 AM - 12:45 AM)$35
From: Hyderabad
To: Bengaluru
Date: 2025-07-07
Passengers: 2 Adults
Select Seats
BusCo 2 (9:15 AM - 13:45 AM)$40
From: Hyderabad
To: Bengaluru
Date: 2025-07-07
Passengers: 2 Adults
Select Seats
BusCo 3 (10:15 AM - 14:45 AM)$45
From: Hyderabad
To: Bengaluru
Date: 2025-07-07
Passengers: 2 Adults
Select Seats


TravelGo
Home
Flights
Hotels
Trains
Buses
My Bookings
Welcome, raghu!
Logout
BusCo 1 from Hyderabad to Bengaluru on 2025-07-07
Departure: 8:15 AM | Price: $35.0
Choose Your Seat
Selected Seat: None
1
A
B
C
D
2
A
B
C
D
3
A
B
C
D



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.