

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

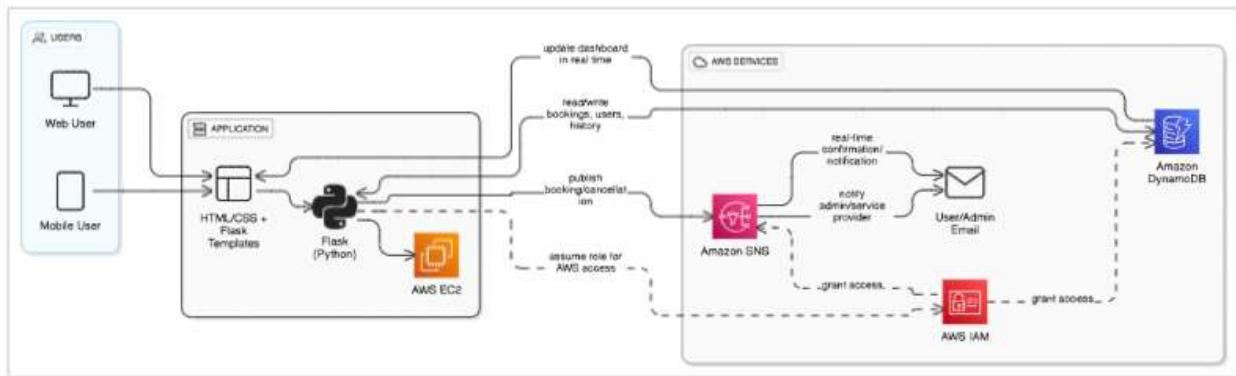
Once a booking is made—whether it's a train ticket or a hotel stay—TravelGo uses AWS SNS to instantly notify the user. For example, after a student books a hotel in Chennai, SNS sends a real-time email notification confirming the booking with all the relevant details. This notification is triggered from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

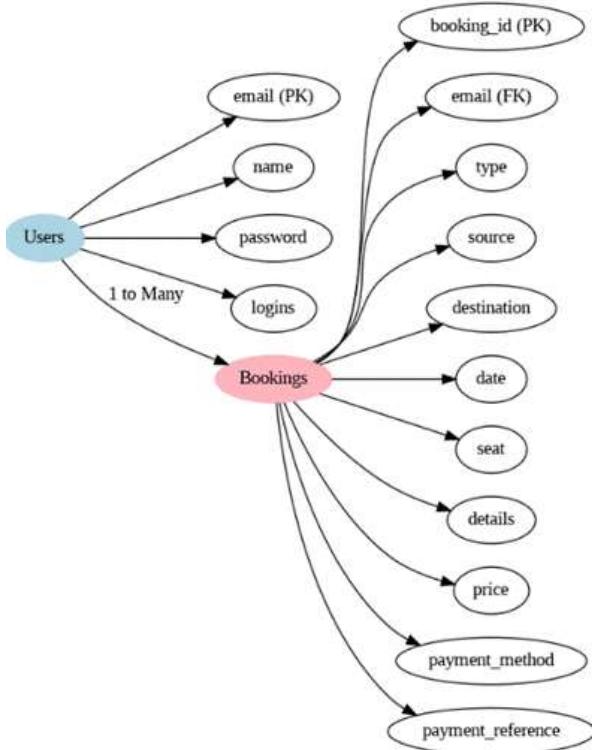
TravelGo features a dynamic user dashboard that displays all past and upcoming bookings for the logged-in user. For example, a user who has booked a flight and a hotel can view these

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

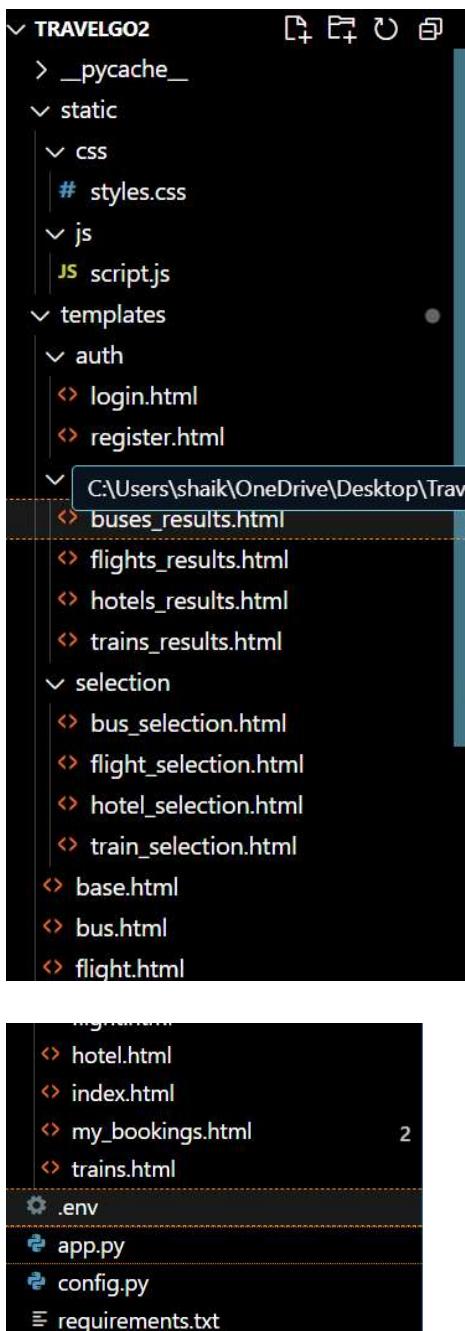
- Upload Flask Files
- Run the Flask App

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_user

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:

```
@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:

```
@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:

```
# 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.

```

# 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:

```
@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:

```
# 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:

```
1 @app.route('/add_to_checkout', methods=['POST'])
2 def add_to_checkout():
3     if 'email' not in session:
4         return jsonify({'success': False, 'message': 'Not logged in'})
5
6     item_id = request.json.get('item_id')
7     if not item_id:
8         return jsonify({'success': False, 'message': 'Item ID missing'})
9
10    # Fetch the item from wishlist
11    try:
12        response = wishlist_table.get_item(
13            Key={'email': session['email'], 'item_id': item_id}
14        )
15        item = response.get('Item')
16        if not item:
17            return jsonify({'success': False, 'message': 'Item not found in wishlist'})
18
19        # Prepare item for checkout session
20        checkout_item = {
21            'item_id': item['item_id'],
22            'item_name': item['item_name'],
23            'price': item['item_details'].split('Price: ')[-1], # e.g., "3,50,000 INR"
24            'image': item.get('item_image', ''),
25            'details': item.get('item_details', ''),
26            'quantity': 1
27        }
28
29        # Initialize or update checkout session
30        checkout_items = session.get('checkout_items', [])
31        existing = next((i for i in checkout_items if i['item_id'] == item_id), None)
32
33        if not existing:
34            checkout_items.append(checkout_item)
35            session['checkout_items'] = checkout_items
36            session.modified = True
37
38        return jsonify({'success': True, 'message': 'Item added to checkout'})
39    except Exception as e:
40        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:

```
@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:

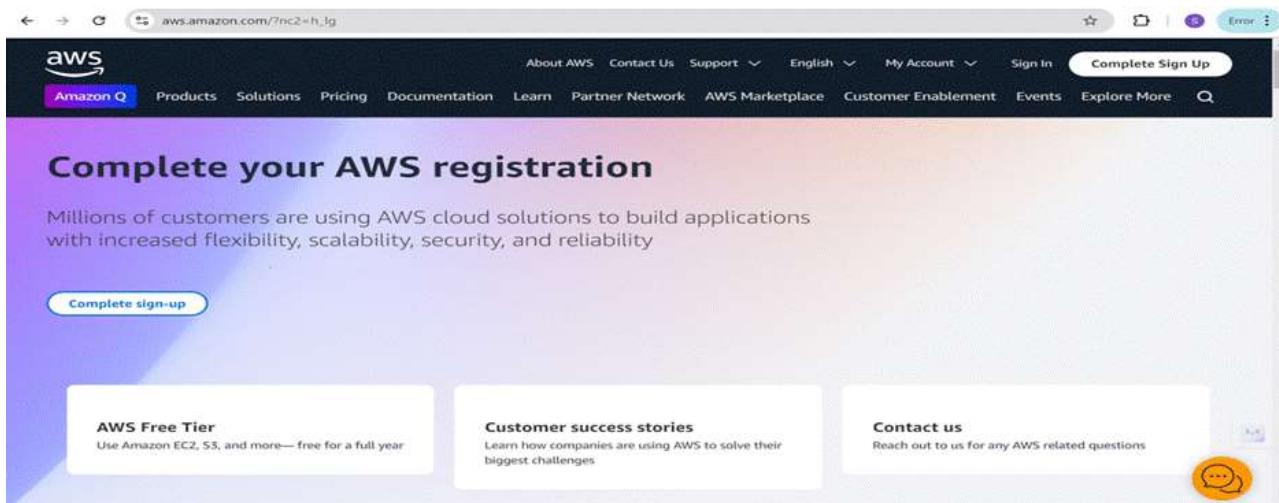
```
# Run the Flask app
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80, debug=True)
```

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

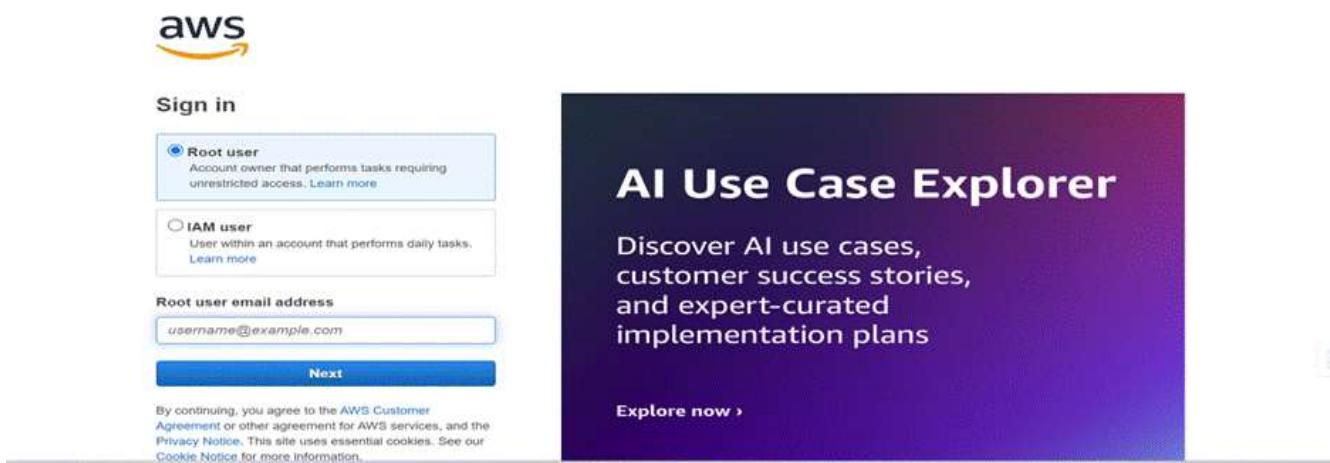
2.1 Set up an AWS account if not already done.

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.

The screenshot shows the AWS IAM search results for 'dynamoDB'. The 'Services' section is expanded, displaying three services: 'DynamoDB' (Managed NoSQL Database), 'Amazon DocumentDB' (Fully-managed MongoDB-compatible database service), and 'CloudFront' (Global Content Delivery Network). The 'DynamoDB' card is highlighted with a blue border. The 'Features' section is also visible. A sidebar on the left shows navigation links for Identity and Management, Access management, and Access reporting. A modal window titled 'View role' is open on the right, showing a single role entry with a 'Manage' button.

The screenshot shows the Amazon DynamoDB home page. The main heading is 'Amazon DynamoDB: A fast and flexible NoSQL database service for any scale'. Below it, a sub-headline states: 'DynamoDB is a fully managed, key-value, and document database that delivers single-digit-millisecond performance at any scale.' To the right, there are two callout boxes: 'Get started' (Create a new table to start exploring DynamoDB) and 'Pricing' (DynamoDB charges for reading, writing, and indexing). On the left, a sidebar lists navigation options: Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations (New), Reserved capacity, and Settings. A 'How it works' section is also present. The bottom of the page includes standard AWS footer links: CloudShell, Feedback, and the URL https://us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#create-table.

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.

Screenshot of the AWS DynamoDB 'Create table' wizard.

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
 String
1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 String
1 to 255 characters and case sensitive.

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

Detention 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 Bookings for storing booking records with email as the partition key and booking_date as the sort key.

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.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
 String
1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 String
1 to 255 characters and case sensitive.

Table settings

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

DynamoDB > Tables > Create table

Retention protection 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.

The screenshot shows the AWS DynamoDB 'Create table' wizard. The table name is 'trains'. The primary key consists of a partition key 'train_number' (String type) and a sort key 'date' (String type). The table uses a Resource-based policy for resource protection. A note indicates that auto scaling is deactivated due to lack of permissions. The 'Create table' button is visible at the bottom right.

Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
 String ▾
1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 String ▾
1 to 255 characters and case sensitive.

Resource 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

The screenshot shows the AWS DynamoDB console. On the left, there's a navigation sidebar with links like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a section for DAX with Clusters and Subnet groups. The main area is titled 'Tables (3) Info' and shows three tables: bookings, trains, and travelgo_users. Each table has columns for Name, Status, Partition key, Sort key, Indexes, Replication Regions, and Deletion protection. The 'trains' table is highlighted as Active with user_email as the partition key and train_number as the sort key. A green banner at the top says 'The trains table was created successfully.' At the bottom, there are links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

Milestone 4: SNS Notification Setup

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

The screenshot shows the AWS SNS service page. The search bar at the top has 'sns' typed into it. The left sidebar has a 'Services' section with links for Simple Notification Service, Route 53 Resolver, and Route 53. It also has a 'Features' section with links for Events (ElastiCache feature) and SMS. A 'Were these results helpful?' poll at the bottom has 'Yes' and 'No' buttons. The main content area on the right shows a table of regions with deletion protection status (all off). At the bottom, there are links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

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

The screenshot shows the Amazon SNS Topics page. On the left, there's a sidebar with options like Dashboard, Topics (which is selected and highlighted in blue), Subscriptions, and Mobile (Push notifications and Text messaging (SMS)). The main area has a blue banner at the top stating "New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more". Below the banner, it says "Topics (0)" and has buttons for Edit, Delete, Publish message, and Create topic. A search bar is present. The table below shows "No topics" with the message "To get started, create a topic." and a prominent "Create topic" button. At the bottom, there are links for CloudShell, Feedback, and legal information.

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

The screenshot shows the "Create topic" wizard. It starts with a note that "Topic type cannot be modified after topic is created". There are two options: "FIFO (first-in, first-out)" and "Standard". "Standard" is selected and highlighted with a blue border. The "Name" field contains "Travelgo". Below it, the "Display name - optional" field contains "My Topic". A section titled "Encryption - optional" notes that "Amazon SNS provides in-transit encryption by default. Enabling server-side encryption adds at-rest encryption to your topic." At the bottom, there are links for CloudShell, Feedback, and legal information.

Access policy - optional [Info](#)
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

Data protection policy - optional [Info](#)
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

Delivery policy (HTTP/S) - optional [Info](#)
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

Delivery status logging - optional [Info](#)
These settings configure the logging of message delivery status to CloudWatch Logs.

Tags - optional
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)

Active tracing - optional [Info](#)
Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

Cancel

Create topic

Configure the SNS topic and note down the Topic ARN.

Amazon SNS <

- Dashboard
- Topics**
- Subscriptions

▼ Mobile

- Push notifications
- Text messaging (SMS)

Travelgo

Details

Name	Travelgo	Display name	-
ARN	arn:aws:sns:us-east-1:863518417312:Travelgo	Topic owner	863518417312
Type	Standard		

Publish message

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

Subscribe users

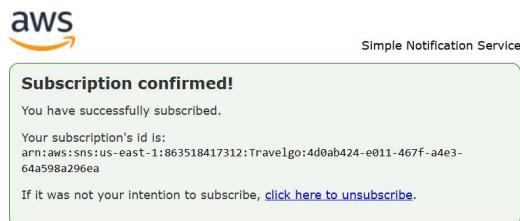
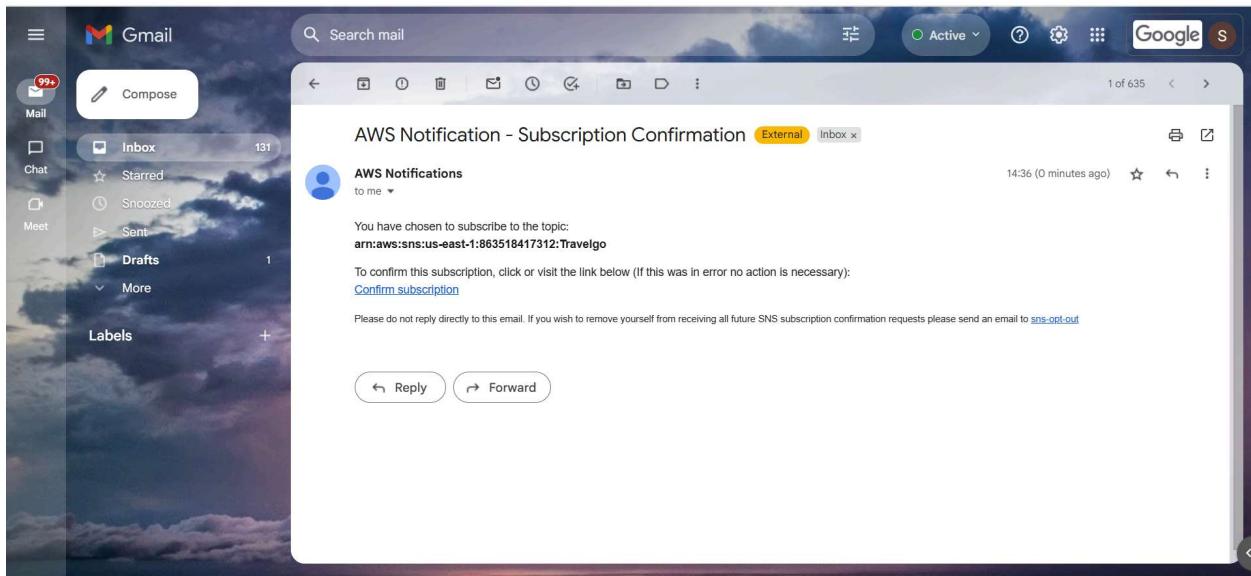
Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.

The screenshot shows the AWS SNS Topics page. On the left sidebar, there are links for Dashboard, Topics (which is selected), Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and CloudShell. The main content area shows a topic named "Travelgo" with a "Type" of "Standard". Below the topic name, there are tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), and Delivery status. The Subscriptions tab is selected, showing a table header with columns for ID, Endpoint, Status, and Protocol. A message indicates "No subscriptions found" and "You don't have any subscriptions to this topic." At the bottom of the table area is a "Create subscription" button.

After subscription request for the mail confirmation

The screenshot shows the "Create subscription" page. The top navigation bar includes links for CloudShell and Feedback, and footer links for © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences. The main form has an "Endpoint" field containing "228x1a4428@khitguntur.ac.in". A note below the endpoint says "After your subscription is created, you must confirm it." There are two expandable sections: "Subscription filter policy - optional" (with a note about filtering messages) and "Redrive policy (dead-letter queue) - optional" (with a note about sending undeliverable messages). At the bottom right are "Cancel" and "Create subscription" buttons.

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



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

The screenshot shows the AWS SNS console with a confirmation message: "Amazon SNS now supports High Throughput FIFO topics. Learn more". Below it, a subscription for topic "Travelgo" is listed with ARN: arn:aws:sns:us-east-1:863518417312:Travelgo:4d0ab424-e011-467f-a4e3-64a598a296ea. The status is "Confirmed" and the protocol is "EMAIL".

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.

The screenshot shows the AWS IAM console with a search bar containing "iam". The results list the "IAM" service, which is selected, and other services like "IAM Identity Center" and "Resource Access Manager". A sidebar on the left shows recent services: EC2, Simple Queue Service, DynamoDB, and IAM.

Screenshot of the AWS IAM Roles page showing 10 roles listed:

Role name	Trusted entities	Last activity
AWSServiceRoleForAccessAnalyzer	AWS Service: access-analyzer (Service-Linked Role)	124 days ago
AWSServiceRoleForAmazonEKS	AWS Service: eks (Service-Linked Role)	145 days ago
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service-Linked Role)	-
AWSServiceRoleForECS	AWS Service: ecs (Service-Linked Role)	134 days ago
AWSServiceRoleForOrganizations	AWS Service: organizations (Service-Linked Role)	209 days ago
AWSServiceRoleForSSO	AWS Service: sso (Service-Linked Role)	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-

Screenshot of the 'Create role' wizard Step 1: Select trusted entity.

Step 1: Select trusted entity

Step 2: Add permissions

Step 3: Name, review, and create

Select trusted entity

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.

School

us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/create

Search [Alt+S]

Global rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71

IAM > Roles > Create role

EC2 Role

EC2
Allows EC2 instances to call AWS services on your behalf.

EC2 Role for AWS Systems Manager
Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.

EC2 Spot Fleet Role
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.

EC2 - Spot Fleet Auto Scaling
Allows Auto Scaling to access and update EC2 spot fleets on your behalf.

EC2 - Spot Fleet Tagging
Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.

EC2 - Spot Instances
Allows EC2 Spot Instances to launch and manage spot instances on your behalf.

EC2 - Spot Fleet
Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.

EC2 - Scheduled Instances
Allows EC2 Scheduled Instances to manage instances on your behalf.

Cancel Next

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.

Step 1 Select trusted entity

Step 2 Add permissions

Step 3 Name, review, and create

Add permissions

Permissions policies (1/1059) Info

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

Filter by Type

ec2fu All types 1 match

Policy name Type Description

AmazonEC2FullAccess AWS managed Provides full access to Am

Set permissions boundary - optional

Cancel Previous Next

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

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

The search bar shows 'dynamodbfull'. The results table lists three AWS managed policies:

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

Below the table, a note says: ▶ Set permissions boundary - optional

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

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

The search bar shows 'sns'. The results table lists five AWS managed policies:

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 operatio
AWSIoTDeviceDefenderPublishFindingsToSNS...	AWS managed	Provides messages publish

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

Screenshot of the AWS IAM 'Create role' wizard, Step 3: Name, review, and create.

Role details

Role name: Studentuser

Description: Allows EC2 instances to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy:

```
1 "Version": "2012-10-17",
2 "Statement": [
3     {
4         "Action": "sts:AssumeRole",
5         "Effect": "Allow",
6         "Principal": "*"
7     }
]
```

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

Screenshot of the AWS IAM 'Create role' wizard, Step 3: Add tags.

Add tags - optional

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

The screenshot shows the AWS Identity and Access Management (IAM) service. In the top navigation bar, the user is signed in as 'rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71'. The main page displays a success message: 'Role Studentuser created.' Below this, there's a search bar and a table with two rows: 'Role name' and 'Trusted entities'. The 'Role name' row contains the 'Studentuser' role. To the right of the table are buttons for 'View role' and 'Manage'. At the bottom of the page, there are links for 'CloudShell', 'Feedback', and copyright information: '© 2025, Amazon Web Services, Inc. or its affiliates.' followed by 'Privacy', 'Terms', and '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.

The screenshot shows the AWS EC2 Instances page. The search bar at the top contains 'ec2'. The left sidebar has sections for 'Amazon S' (Dashboard, Topics, Subscriptions), 'Mobile' (Push notifications, Blog posts, Events, Tutorials), and 'Services' (Features, Resources, Documentation, Knowledge articles, Marketplace). The main content area is titled 'Services' and lists three items: 'EC2' (Virtual Servers in the Cloud), 'EC2 Image Builder' (A managed service to automate build, customize and deploy OS images), and 'EC2 Global View' (Provides a global dashboard and search functionality). Below this is a 'Features' section with a 'Dashboard' item. At the bottom of the page, there's a feedback poll: 'Were these results helpful?' with 'Yes' and 'No' buttons, and a 'Were these results helpful?' link. The footer includes 'CloudShell', 'Feedback', and copyright information: '© 2025, Amazon Web Services, Inc. or its affiliates.' followed by 'Privacy', 'Terms', and 'Cookie preferences'.

- Click on Launch instance to launch EC2 instance

The screenshot shows the AWS EC2 home page. The main content area features a large heading "Amazon Elastic Compute Cloud (EC2)" and a sub-headline "Create, manage, and monitor virtual servers in the cloud." Below this, a paragraph describes the service's capabilities. To the right, a callout box titled "Launch a virtual server" contains two buttons: "Launch instance" and "View dashboard". The left sidebar has sections for "Instances" (with sub-links for Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations) and "Images" (with a link for AMIs). The top navigation bar includes the AWS logo, search bar, and various account and region settings.

This screenshot shows the "Launch an instance" wizard. The first step, "Name and tags", is completed with the name "TravelGoproject". The second step, "Application and OS Images (Amazon Machine Image)", is currently selected. It displays a search bar and a list of AMIs, with "Amazon Linux 2023 AMI 2023.7.2..." being the top result. Other visible fields include "Virtual server type (instance type)" set to "t2.micro" and "Firewall (security group)" set to "New security group". At the bottom are "Cancel", "Launch instance", and "Preview code" buttons.

- 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. On the left, under 'Key pair (login)', it says 'Key pair name - required' and has a dropdown menu with 'Select'. To its right is a 'Create new key pair' button. Below this is the 'Network settings' section, which includes 'Network' (vpc-063cbc01c43c74b9a), 'Subnet' (no preference), and 'Auto-assign public IP' (Enable). On the right, the 'Summary' section shows 'Number of instances' (1), 'Software image (AMI)' (Amazon Linux 2023 AMI 2023.7.2...), 'Virtual server type (instance type)' (t2.micro), and a 'Launch instance' button.

Create and download the key pair for Server access.

The screenshot shows the 'Create key pair' dialog box overlaid on the EC2 launch wizard. In the 'Create key pair' dialog, the 'Key pair name' field is filled with 'travel-go-appl'. The 'Key pair type' section shows 'RSA' selected (RSA encrypted private and public key pair) and 'ED25519' as an option (ED25519 encrypted private and public key pair). The 'Private key file format' section shows '.pem' selected (For use with OpenSSH) and '.ppk' as an option (For use with PuTTY). At the bottom of the dialog are 'Cancel' and 'Create key pair' buttons. The background shows the EC2 launch wizard's 'Summary' step.



InstantLibrary.pem

The screenshot shows the AWS EC2 'Launch an instance' wizard. On the left, under 'Firewall (security groups)', it says 'Create security group' is selected. Under 'Rules with source of 0.0.0.0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.', there is a note with a warning icon. On the right, the 'Summary' section shows 1 instance, AMI 'Amazon Linux 2023 AMI 2023.7.2...', instance type 't2.micro', and a 'New security group'. At the bottom right are 'Cancel', 'Launch instance', and 'Preview code' buttons.

The screenshot shows the AWS EC2 'Launch an instance' wizard after launch. A green success message at the top says 'Successfully initiated launch of instance (i-021c696cac90133b7)'. Below it, a 'Next Steps' section lists four options: 'Create billing and free tier usage alerts', 'Connect to your instance', 'Connect an RDS database', and 'Create EBS snapshot policy'. Each option has a brief description and a 'Create' or 'Connect' button. At the bottom right are 'Cancel', 'Launch instance', and 'Preview code' buttons.

Configure security groups for HTTP, and SSH access:

aws | Search [Alt+S] | United States (N. Virginia) | rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71

EC2 Instances

Instances (1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
TravelGoproject	i-021c696cac90133b7	Running	t2.micro	Initializing	View alarms +

Select an instance

© 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

Inbound rules

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0485a2bbdb90566bf	SSH	TCP	22	Cus... ▾	<input type="text"/> 0.0.0.0/0 X
sgr-0a3b97d577de88311	HTTPS	TCP	443	Cus... ▾	<input type="text"/> 0.0.0.0/0 X
-	Custom TCP	TCP	5000	An... ▾	<input type="text"/> 0.0.0.0/0 X

Add rule

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

Cancel Preview changes Save rules

© 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

EC2

- Dashboard
- EC2 Global View
- Events
- Instances**
 - Instances
 - Instance Types
 - Launch Templates
 - Spot Requests
 - Savings Plans
 - Reserved Instances
 - Dedicated Hosts
 - Capacity Reservations
- Images**
 - AMIs

sg-001cb936ef308ba42 - launch-wizard-1

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)

Name	Security group rule ID	Type	Protocol
allow-ssh	allow-ssh	TCP	tcp
allow-https	allow-https	TCP	tcp
allow-aws-ec2	allow-aws-ec2	TCP	tcp

© 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 > 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

Name	Instance ID	Instance state	Instance type
TravelGoproject	i-021c696cac90133b7	Running	t2.micro

i-021c696cac90133b7 (TravelGoproject)

Actions

- Connect
- Instance state
- Launch instances
- Instance diagnostics
- Instance settings
- Networking
- Security
- Image and templates
- Modify IAM role
- Monitor and troubleshoot

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

Instance summary

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

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

Screenshot of the AWS EC2 Instances Modify IAM role page.

The URL is [rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71](#).

Instance ID: [i-021c696cac90133b7](#) (TravelGoproject)

IAM role: Studentuser (selected) | Create new IAM role

Buttons: Cancel | Update IAM role

Screenshot of the AWS EC2 Instances Connect to instance page.

The URL is [rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71](#).

Instance ID: [i-021c696cac90133b7](#) (TravelGoproject)

Connect using a Public IP (selected) | Connect using a Private IP

Public IPv4 address

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

The screenshot shows the AWS EC2 Connect interface. At the top, there's a navigation bar with the AWS logo, search bar, and account information: United States (N. Virginia) and rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71. Below the navigation bar, the path is EC2 > Instances > i-021c696cac90133b7 > Connect to instance. The main area is titled "Connect" with a "Info" link. It says "Connect to an instance using the browser-based client." There are four tabs: EC2 Instance Connect, Session Manager, SSH client (which is selected and highlighted in blue), and EC2 serial console. Under the "Instance ID" section, it shows i-021c696cac90133b7 (TravelGoproject). Below this, there's a numbered list of steps: 1. Open an SSH client, 2. Locate your private key file. The key used to launch this instance is travel-go-appl.pem, 3. Run this command, if necessary, to ensure your key is not publicly viewable: chmod 400 "travel-go-appl.pem", and 4. Connect to your instance using its Public DNS: ec2-50-19-159-121.compute-1.amazonaws.com. An "Example:" section shows the command ssh -i "travel-go-appl.pem" ec2-user@ec2-50-19-159-121.compute-1.amazonaws.com. A note in a callout box says: "Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username." At the bottom, there are links for CloudShell, Feedback, and legal notices: © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

- Now connect the EC2 with the files

The screenshot shows a Windows PowerShell window titled "ec2-user@ip-172-31-18-105:~". The window displays the following text:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\shaik> ssh -i "C:\Users\shaik\Downloads\travel-go-appl.pem" ec2-user@ec2-50-19-159-121.compute-1.amazonaws.com
The authenticity of host 'ec2-50-19-159-121.compute-1.amazonaws.com (50.19.159.121)' can't be established.
ED25519 key fingerprint is SHA256:CgTHKigFwEhwLw9OMyNU7jhou9NqtgXYELahsVj8aN00.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-50-19-159-121.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

[ec2-user@ip-172-31-18-105 ~]$ |
```

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](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:8b84921fb2a4-41a4-965b-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
```

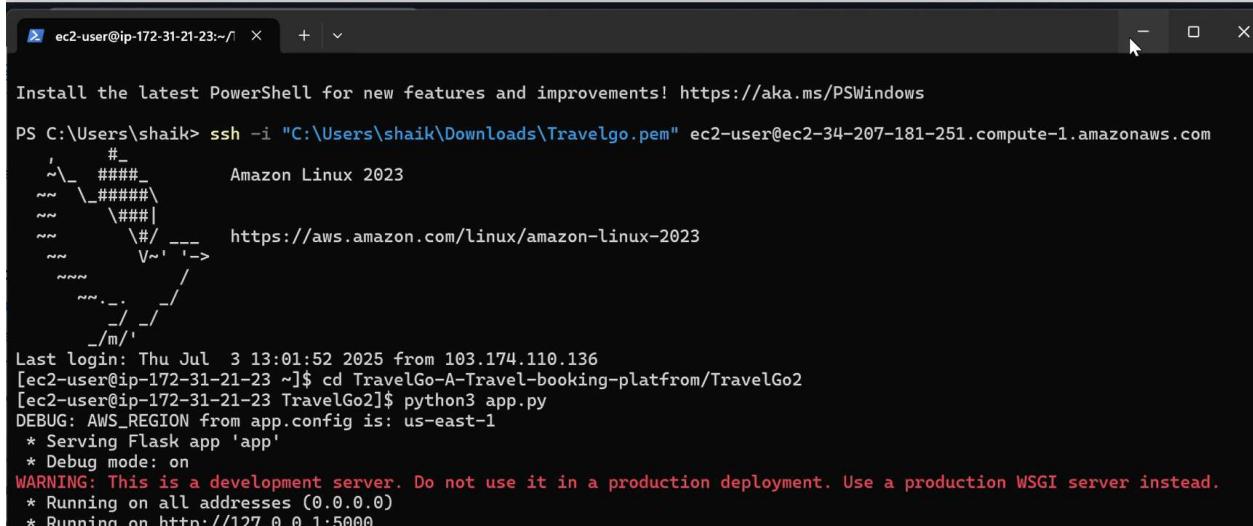
- Launch the Flask app:

```
.... 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



```
ec2-user@ip-172-31-21-23:~/ + | 
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\Users\shaik> ssh -i "C:\Users\shaik\Downloads\Travelgo.pem" ec2-user@ec2-34-207-181-251.compute-1.amazonaws.com
'`#_
~\_\####_          Amazon Linux 2023
~~ \_\#####\_
~~  \###|_
~~   \#/ ___  https://aws.amazon.com/linux/amazon-linux-2023
~~   V~' `-->
~~   /`_
~~ .-: /`_
~~ /_ /`_
~/m/'_
Last login: Thu Jul  3 13:01:52 2025 from 103.174.110.136
[ec2-user@ip-172-31-21-23 ~]$ cd TravelGo-A-Travel-booking-platfrom/TravelGo2
[ec2-user@ip-172-31-21-23 TravelGo2]$ python3 app.py
DEBUG: AWS_REGION from app.config is: us-east-1
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on https://127.0.0.1:5000
```

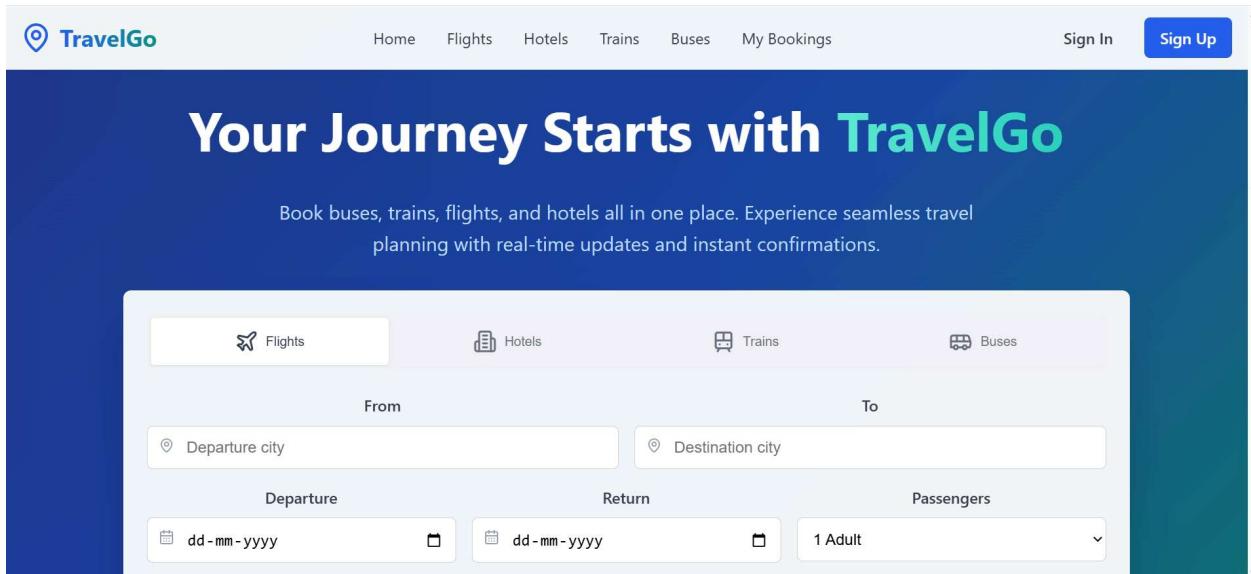
- Access the website through:
PublicIPs: http://34.207.181.251:5000

Milestone 8 : Testing and Deployment

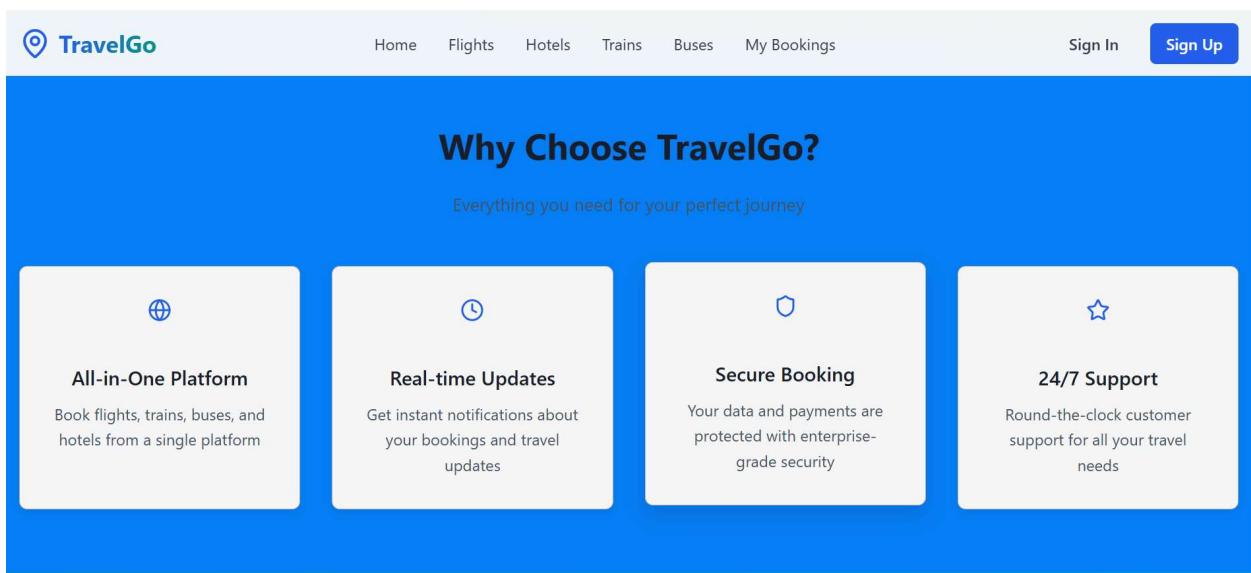
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and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

Functional Testing to verify the Project

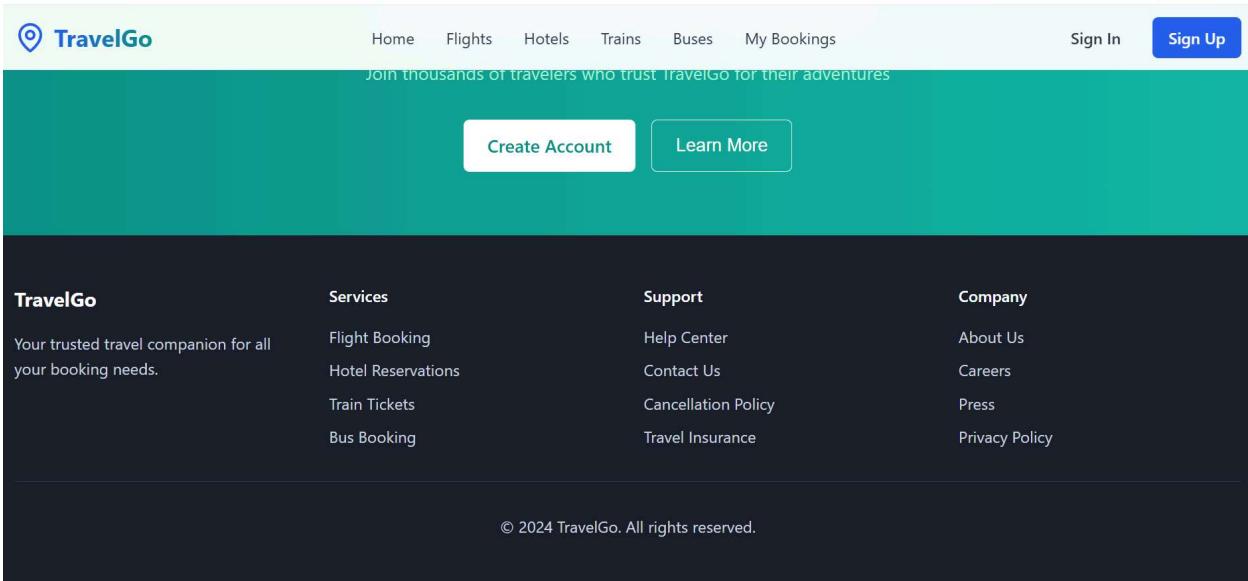
Home Page:



The screenshot shows the TravelGo home page. At the top, there's a navigation bar with links for Home, Flights, Hotels, Trains, Buses, and My Bookings, along with Sign In and Sign Up buttons. The main header features the text "Your Journey Starts with TravelGo". Below it, a sub-header reads: "Book buses, trains, flights, and hotels all in one place. Experience seamless travel planning with real-time updates and instant confirmations." A large search bar is centered, containing tabs for Flights, Hotels, Trains, and Buses, and fields for "From" and "To" locations, "Departure" and "Return" dates, and "Passengers".

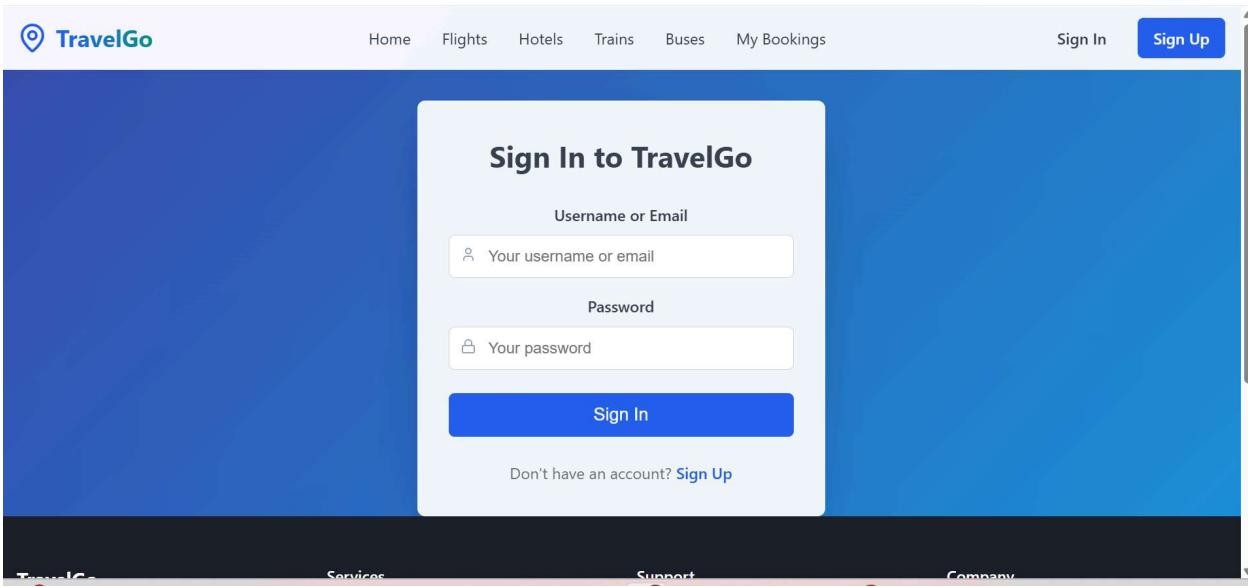


The screenshot shows the "Why Choose TravelGo?" section of the website. It features four cards with icons and descriptions: "All-in-One Platform" (book flights, trains, buses, and hotels from a single platform), "Real-time Updates" (get instant notifications about bookings and travel updates), "Secure Booking" (data and payments protected with enterprise-grade security), and "24/7 Support" (round-the-clock customer support). Above this section, the TravelGo logo and navigation bar are visible.



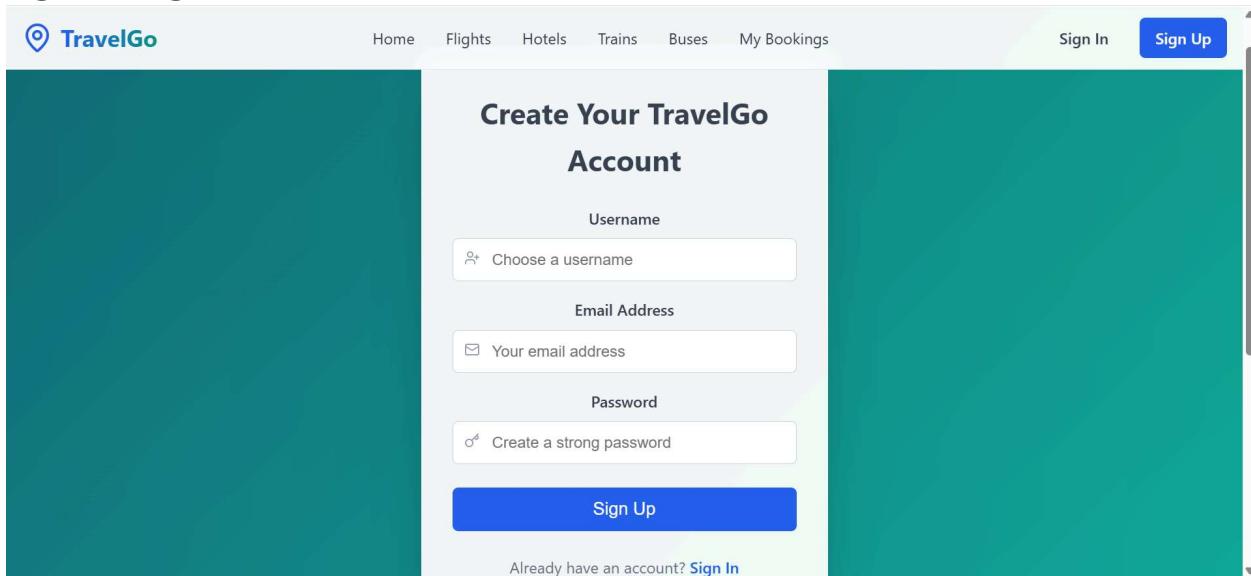
The screenshot shows the TravelGo homepage. At the top left is the TravelGo logo with a blue circle icon. To its right is a navigation bar with links: Home, Flights, Hotels, Trains, Buses, and My Bookings. On the far right are Sign In and Sign Up buttons. Below the navigation is a teal-colored banner with the text "Join thousands of travelers who trust TravelGo for their adventures". Underneath the banner are two buttons: "Create Account" and "Learn More". The main content area has a dark background. On the left, under the heading "TravelGo", is the text "Your trusted travel companion for all your booking needs.". In the center, there's a section titled "Services" with links to Flight Booking, Hotel Reservations, Train Tickets, and Bus Booking. To the right, under "Support", are links to Help Center, Contact Us, Cancellation Policy, and Travel Insurance. On the far right, under "Company", are links to About Us, Careers, Press, and Privacy Policy. At the bottom of the page, a dark footer bar contains the copyright notice "© 2024 TravelGo. All rights reserved."

Sign In Page:



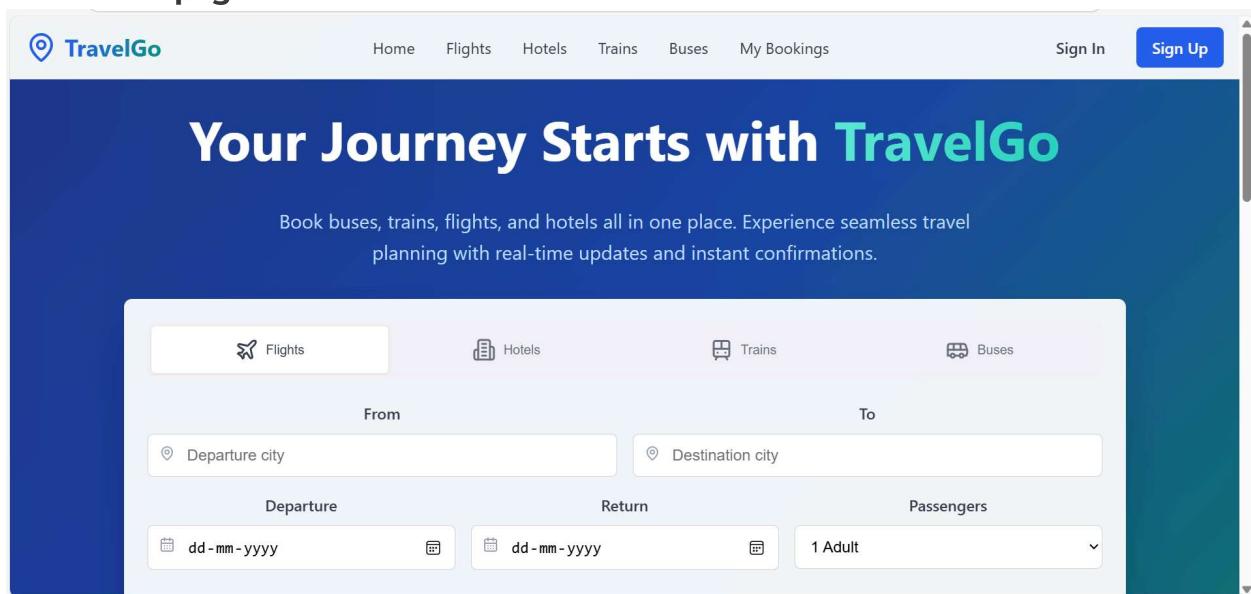
The screenshot shows the sign-in page for TravelGo. The page has a blue header with the TravelGo logo and navigation links: Home, Flights, Hotels, Trains, Buses, and My Bookings. On the far right are Sign In and Sign Up buttons. The main content is a white form box titled "Sign In to TravelGo". It contains fields for "Username or Email" and "Password", each with a placeholder text ("Your username or email" and "Your password"). Below the fields is a blue "Sign In" button. At the bottom of the form box is the text "Don't have an account? [Sign Up](#)". The footer of the page is black and contains links for TravelGo, Services, Support, and Company.

Sign Up Page:



The sign up page for TravelGo features a teal header and footer. The main content area is titled "Create Your TravelGo Account". It includes fields for "Username" (placeholder: "Choose a username"), "Email Address" (placeholder: "Your email address"), and "Password" (placeholder: "Create a strong password"). A blue "Sign Up" button is at the bottom, and a link to "Sign In" is at the very bottom.

Dashboard page:



The dashboard page for TravelGo has a dark blue header and footer. The main section is titled "Your Journey Starts with TravelGo" and encourages users to book buses, trains, flights, and hotels. Below this, there's a search bar with tabs for "Flights", "Hotels", "Trains", and "Buses". The search form includes fields for "From" and "To" locations, "Departure" and "Return" dates, and "Passengers".

Buses page:

The screenshot shows the 'Find Your Perfect Bus' search form. It includes fields for 'From' (Departure city) and 'To' (Destination city), a date selector for 'Travel Date' (dd-mm-yyyy), and a passenger selector for 'Passengers' (1 Adult). A large blue 'Search Buses' button is at the bottom.

From To

Departure city Destination city

Travel Date Passengers

dd-mm-yyyy 1 Adult

Search Buses

The screenshot shows the 'Find Your Perfect Bus' search form with populated fields: 'From' set to Hyderabad and 'To' set to Bengaluru, 'Travel Date' set to 07-07-2025, and 'Passengers' set to 2 Adults. A large blue 'Search Buses' button is at the bottom.

From To

Hyderabad Bengaluru

Travel Date Passengers

07-07-2025 2 Adults

Search Buses

Buses from Hyderabad to Bengaluru

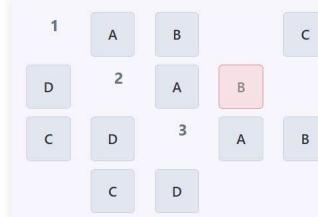
Date: 2025-07-07

BusCo 1 (8:15 AM - 12:45 AM)**\$35** From: Hyderabad Date: 2025-07-07 To: Bengaluru Passengers: 2 Adults**Select Seats**BusCo 2 (9:15 AM - 13:45 AM)**\$40** From: Hyderabad Date: 2025-07-07 To: Bengaluru Passengers: 2 Adults**Select Seats**BusCo 3 (10:15 AM - 14:45 AM)**\$45** From: Hyderabad Date: 2025-07-07 To: Bengaluru Passengers: 2 Adults**Select Seats**

BusCo 1 from Hyderabad to Bengaluru on 2025-07-07

Departure: 8:15 AM | Price: \$35.0

Choose Your Seat

Selected Seat: **None**

My Bookings

View all your confirmed and upcoming travel plans.

Bus Booking (ID: 6868dacb) \$35.0

From: Hyderabad

Date: 2025-07-07

Passengers: 1 Adult

Status: **Confirmed**

To: Bengaluru

Operator: BusCo 1

Seat: 3A

 Cancel Booking

Hotel Booking (ID: 6868d06f) \$95.0 / night

Destination: Hyderabad

Check-in: 2025-07-25

Check-out: 2025-07-30

Hotel: Hotel 1 Hyderabad

Room Type: Deluxe King

Check-in: 2025-07-25

Guests/Rooms: 2 Guests, 1 Room

Rating: 3.7

Status: **Confirmed**

 Cancel 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.