

## Travel Booking System : Quick Access to Travel Services

### **Project Description:**

At Greenfield University, students and faculty often face difficulties in managing travel bookings for academic trips, conferences, and internships. Manual booking processes are slow, unorganized, and prone to miscommunication.

To address this, the university's Cloud Solutions Department developed the Travel Booking System—a virtual platform that enables students and staff to book, manage, and track travel arrangements seamlessly. Built using Flask for backend logic, AWS EC2 for hosting, DynamoDB for storing booking data, and AWS SNS for sending real-time travel notifications, this system modernizes and streamlines the entire travel management process.

---

### **Scenario 1: Streamlined Travel Booking for Students and Staff**

**With the Travel Booking System, users can register and log in securely. After logging in, students or faculty can easily access the travel booking interface to schedule transportation or accommodations for academic purposes. AWS EC2 ensures reliable performance, handling concurrent requests efficiently, even during peak usage. Flask manages user sessions and booking logic in real-time, while DynamoDB keeps track of all travel requests and bookings.**

---

### **Scenario 2: Real-Time Travel Notifications**

**Whenever a new travel request is made or updated, AWS SNS instantly notifies the requester and the travel management team. For example, a faculty member books a flight for a conference—once submitted, Flask processes the booking and SNS sends confirmation emails to both the requester and the travel administrator. This ensures prompt communication and prevents delays or miscommunication.**

---

### **Scenario 3: Easy Access to Travel Details**

**Users can log into the platform and view upcoming and past travel bookings. They can filter by date, destination, or status. The interface is intuitive, and backed by DynamoDB, which offers real-time data retrieval. AWS EC2 ensures the site remains available even under high demand, while Flask handles dynamic content rendering for each user.**

---

### **Cloud Architecture Overview :-**

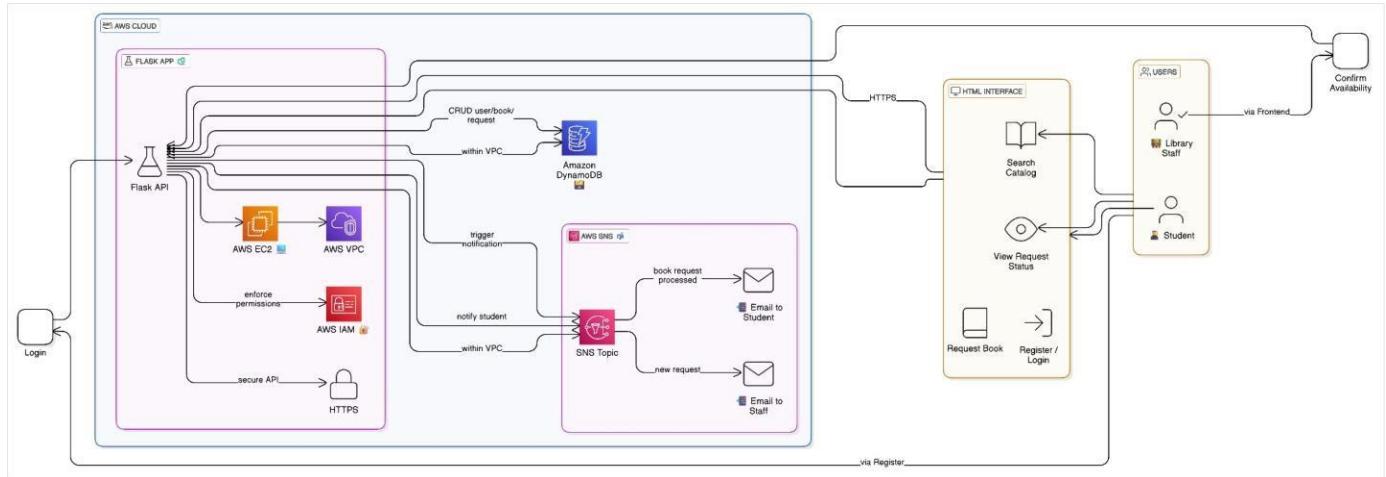
- **Frontend: HTML templates rendered by Flask (with routes for registration, login, booking, etc.)**
- **Backend: Flask (Python) application hosted on EC2**
- **Database: AWS DynamoDB (storing users and travel bookings)**

- **Notifications:** AWS SNS (email alerts for booking confirmations)
- **Deployment:** Hosted on AWS EC2 Linux instance with Flask and Boto3

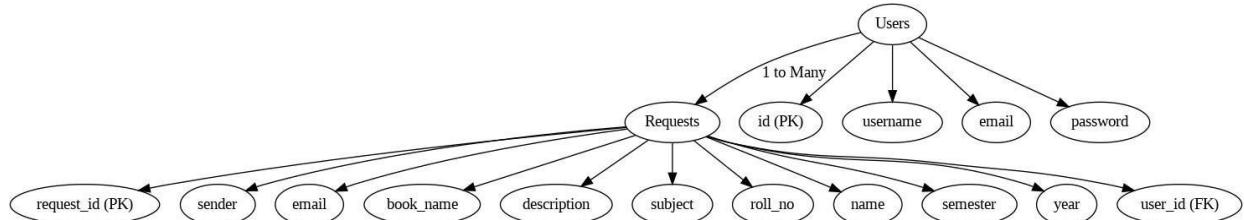
### Cloud Architecture Overview :

- **Frontend:** HTML templates rendered by Flask (with routes for registration, login, booking, etc.)
- **Backend:** Flask (Python) application hosted on EC2
- **Database:** AWS DynamoDB (storing users and travel bookings)
- **Notifications:** AWS SNS (email alerts for booking confirmations)
- **Deployment:** Hosted on AWS EC2 Linux instance with Flask and Boto3

## AWS ARCHITECTURE



### Entity Relationship (ER)Diagram:



### Pre-Requisites :-

1. [AWS Account](#)
2. [IAM Configuration](#)
3. [EC2 Instance Setup](#)
4. [DynamoDB Tables for Users and Bookings](#)
5. [SNS Topics for Travel Notifications](#)
6. [Flask App Code \(uploaded via GitHub\)](#)
7. [Git for version control](#)

## **Project WorkFlow:**

### **Milestone 1: AWS Setup**

- Create AWS account
- Configure IAM roles for EC2 instance

### **Milestone 2: DynamoDB Setup**

- Create tables: Users (with Email as partition key). Bookings (with BookingID or Email as key)

### **Milestone 3: SNS Notifications**

- Create topic: travel-booking-alerts
- Subscribe admin/staff and users via email

### **Milestone 4: Backend Development (Flask + Boto3)**

- Flask routes: /register, /login, /book-travel, /view-bookings
- Store and retrieve data using DynamoDB
- Send SNS notifications on travel booking

### **Milestone 5: EC2 Hosting**

- Launch EC2 instance
- Install Flask, Git, Boto3
- Clone GitHub repo and run the Flask server

### **Milestone 6: Testing and Deployment**



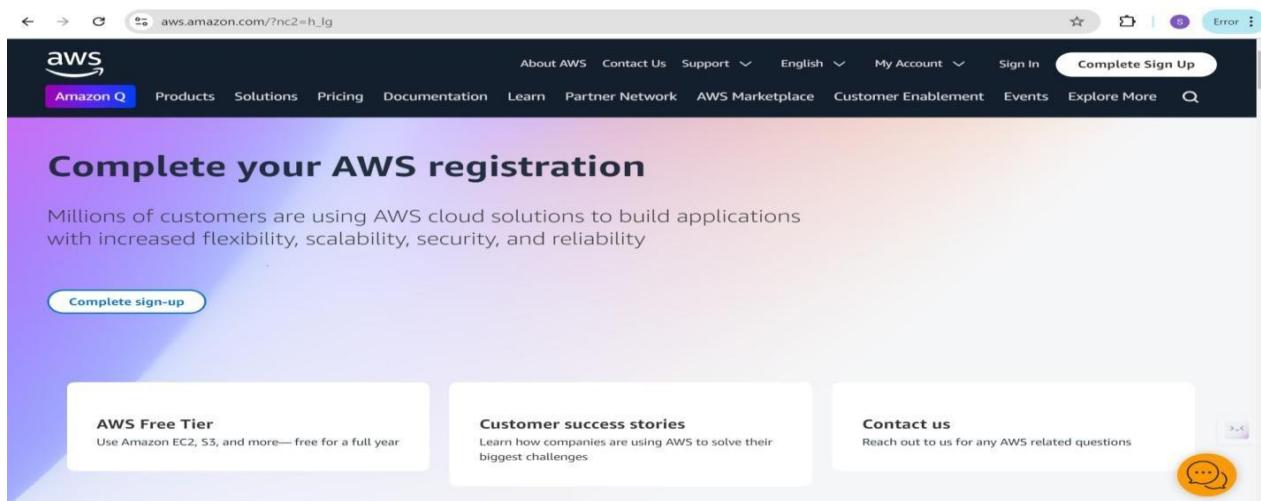
- **Test user registration, login, booking functionality, and email alerts**

## **1. Testing and Deployment**

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

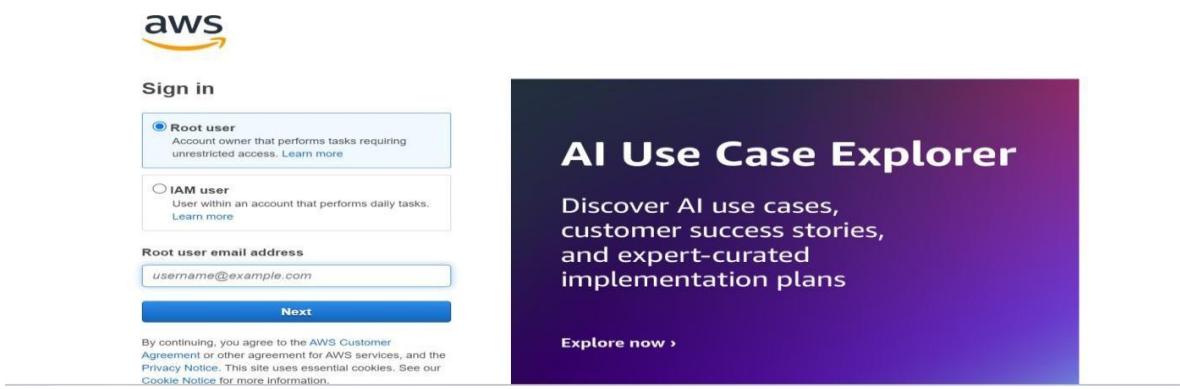
### **Milestone 1: AWS Account Setup and Login**

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



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

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

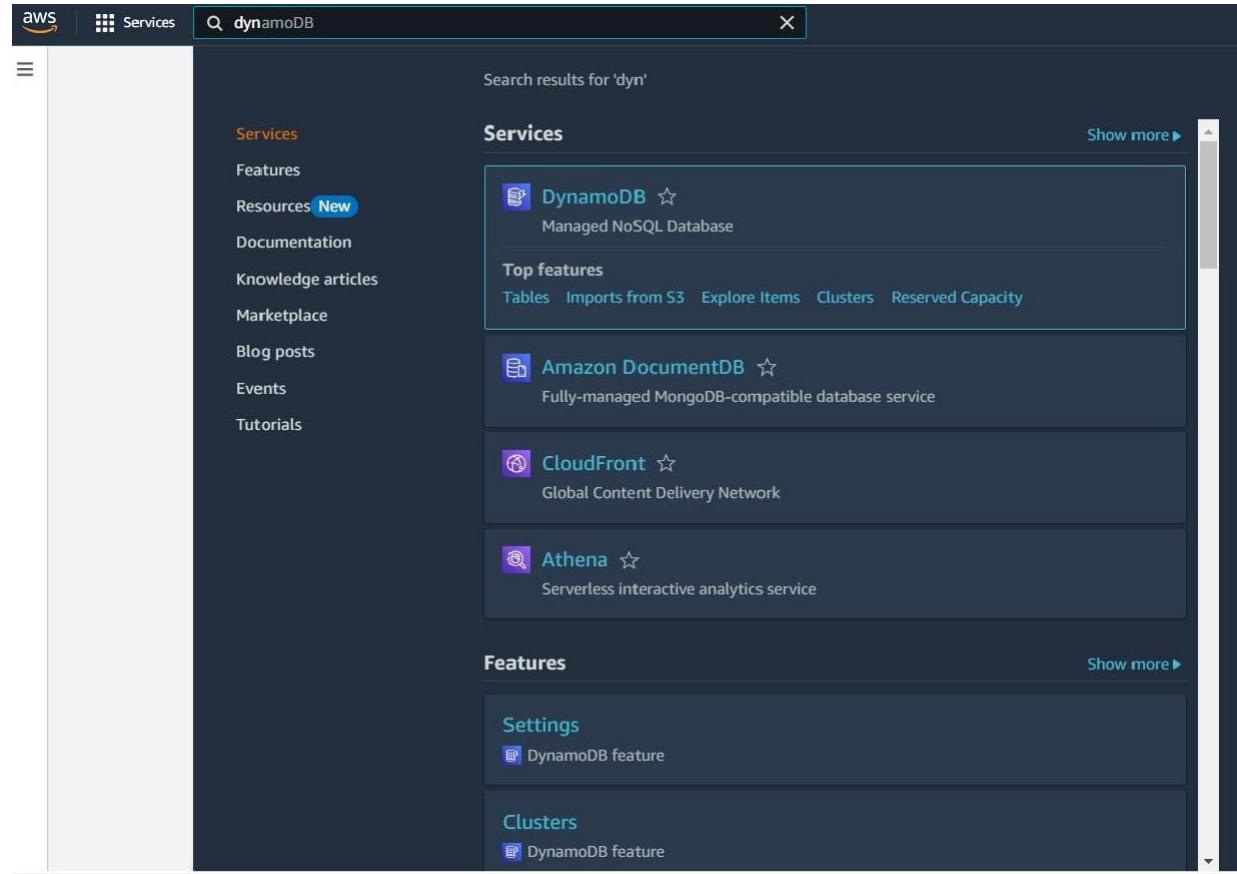


The image contains two parts. On the left, a screenshot of the AWS sign-in page. It features the AWS logo at the top, followed by a 'Sign in' section. It offers two options: 'Root user' (selected) and 'IAM user'. Below these are fields for 'Root user email address' (containing 'username@example.com') and a 'Next' button. At the bottom, there's a small note about agreeing to the AWS Customer Agreement and Privacy Notice. On the right, there's a dark purple sidebar with the heading 'AI Use Case Explorer' in white. It contains text: 'Discover AI use cases, customer success stories, and expert-curated implementation plans'. At the bottom of the sidebar is a blue 'Explore now >' button.

## Milestone 2: DynamoDB Database Creation and Setup

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

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

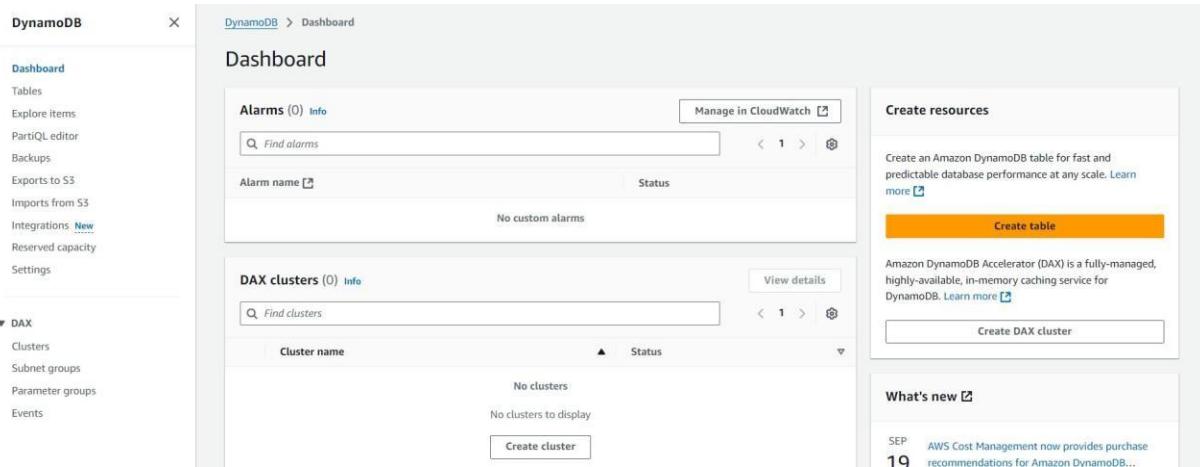


The screenshot shows the AWS Services search results page. The search bar at the top contains 'dynamoDB'. The left sidebar has a 'Services' section with links to Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main content area is titled 'Search results for 'dyn'' and shows a list of services under 'Services' and 'Features'.

- DynamoDB** (Managed NoSQL Database) - Top feature: Tables, Imports from S3, Explore Items, Clusters, Reserved Capacity.
- Amazon DocumentDB** (Fully-managed MongoDB-compatible database service)
- CloudFront** (Global Content Delivery Network)
- Athena** (Serverless interactive analytics service)

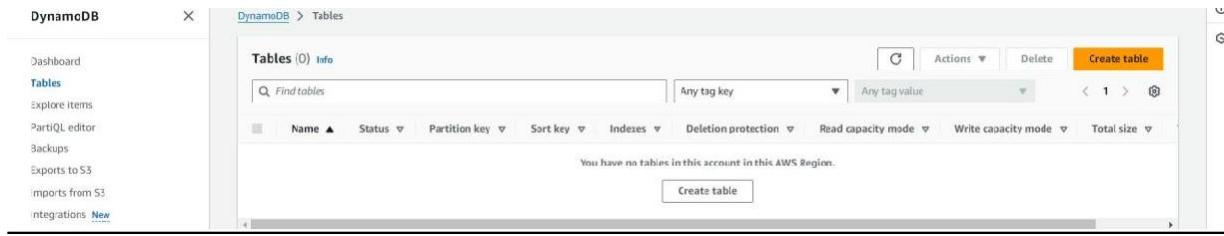
Under 'Features':

- Settings** (DynamoDB feature)
- Clusters** (DynamoDB feature)



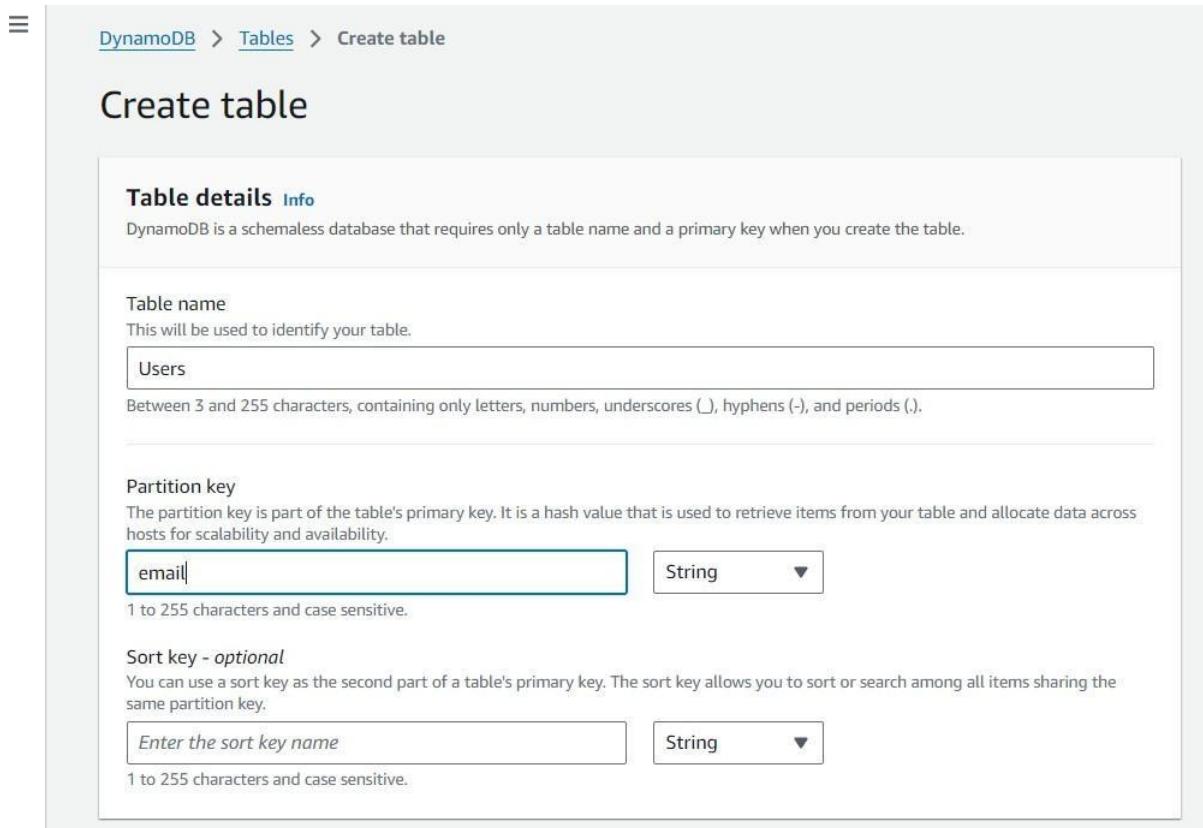
The screenshot shows the DynamoDB Dashboard. The left sidebar includes links for Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations (New), Reserved capacity, and Settings. It also has a DAX section with Clusters, Subnet groups, Parameter groups, and Events.

The main dashboard displays two sections: 'Alarms (0)' and 'DAX clusters (0)'. The 'Create resources' section on the right allows users to 'Create table' or 'Create DAX cluster'. A 'What's new' section at the bottom right shows a recent update on September 19, 2023, about AWS Cost Management providing purchase recommendations for Amazon DynamoDB.



The screenshot shows the AWS DynamoDB 'Tables' page. On the left, there is a sidebar with options like 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', and 'Integrations'. The main area has a heading 'Tables (0) Info' with a search bar and filters for 'Name', 'Status', 'Partition key', 'Sort key', 'Indexes', 'Deletion protection', 'Read capacity mode', 'Write capacity mode', and 'Total size'. A message at the bottom says 'You have no tables in this account in this AWS Region.' There is a prominent orange 'Create table' button.

- **Activity 2.2: Create a DynamoDB table for storing registration details and book requests.**
  - **Create Users table with partition key “Email” with type String and click on create tables.**



The screenshot shows the 'Create table' wizard. At the top, it says 'DynamoDB > Tables > Create table'. The main title is 'Create table'. Below it, under 'Table details', there is a section for 'Table name' with a note that it will be used to identify the table. A text input field contains 'Users'. A note below says 'Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).'. The next section is 'Partition key', which is described as part of the primary key. It includes a note about scalability and availability, a text input field with 'email', and a dropdown menu set to 'String'. A note below says '1 to 255 characters and case sensitive.'. The final section is 'Sort key - optional', which is described as the second part of the primary key. It includes a text input field with 'Enter the sort key name', a dropdown menu set to 'String', and a note below saying '1 to 255 characters and case sensitive.'

Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

**Tags**

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

**Add new tag**

You can add 50 more tags.

**Cancel** **Create table**

The Users table was created successfully.

**DynamoDB** X

**Tables** Tables (1) Info

**Actions** Actions Delete Create table

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size
Users	Active	email (S)	-	0	Off	Provisioned (5)	Provisioned (5)	0 bytes

- **Follow the same steps to create a requests table with Email as the primary key for book requests data.**

[DynamoDB](#) > [Tables](#) > [Create table](#)

## Create table

### Table details Info

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

#### Table name

This will be used to identify your table.

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

#### Partition key

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

String



1 to 255 characters and case sensitive.

#### Sort key - optional

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

String



1 to 255 characters and case sensitive.

### Table settings

#### Default settings

This feature lets you create a new table. You can modify...

#### Customize settings

Use these advanced functions to make DynamoDB work...

Search [Alt+S] United States (N. Virginia) rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew14

Tables

The bookings table was created successfully.

Tables (4/4) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity
Bookings	Active	email (S)	Booking_id (S)	0	0	Off	★	On-demand
bookings	Active	user_email (S)	booking_date (S)	0	0	Off	★	On-demand
trains	Active	train_number (S)	-	0	0	Off	★	On-demand
travelgo_users	Active	email (S)	-	0	0	Off	★	On-demand

Actions Delete Create table

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 18:13:07 30-06-2025



Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

### Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

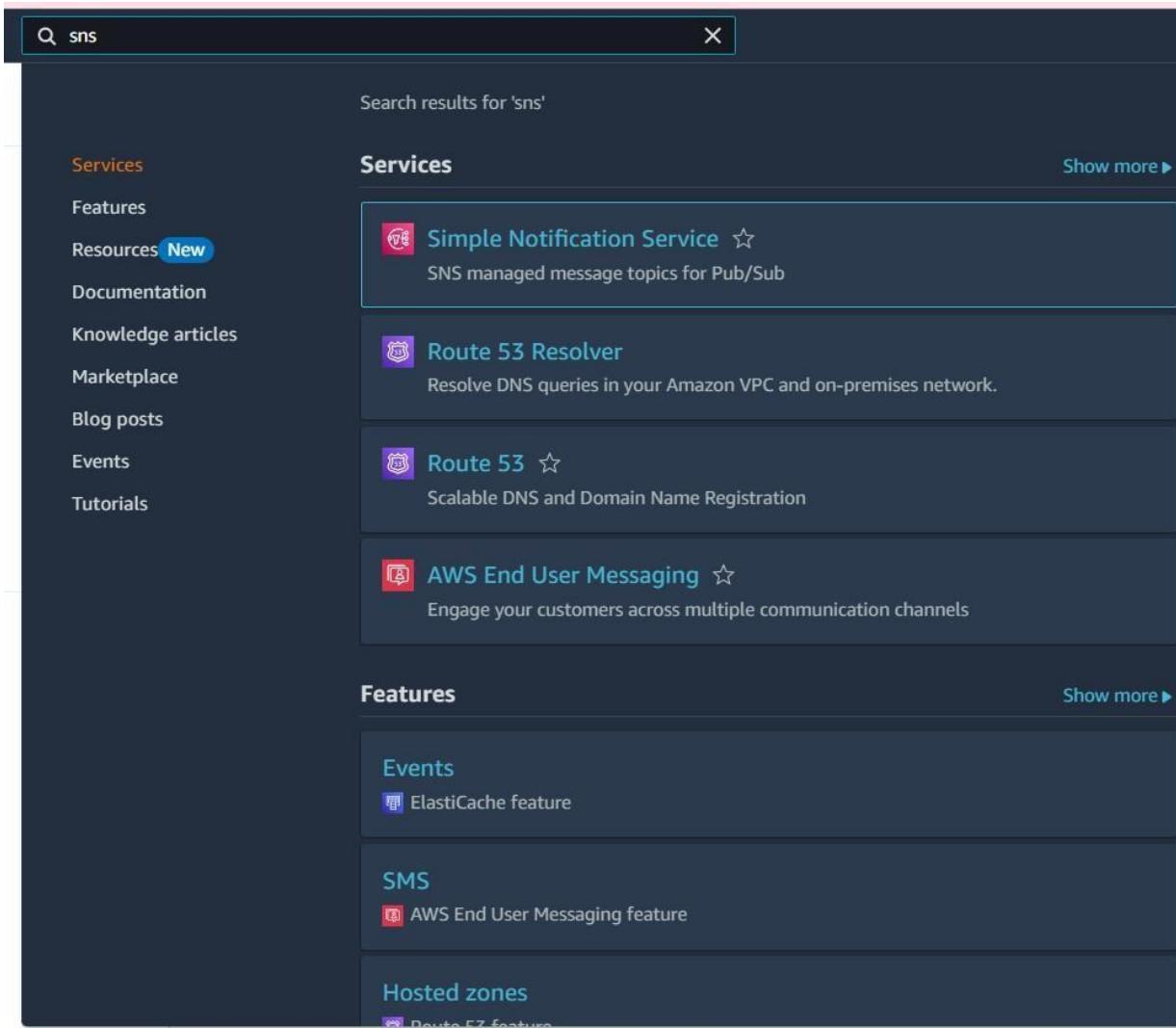
[Cancel](#)

[Create table](#)

## **Milestone 3: SNS Notification Setup**

- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**

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



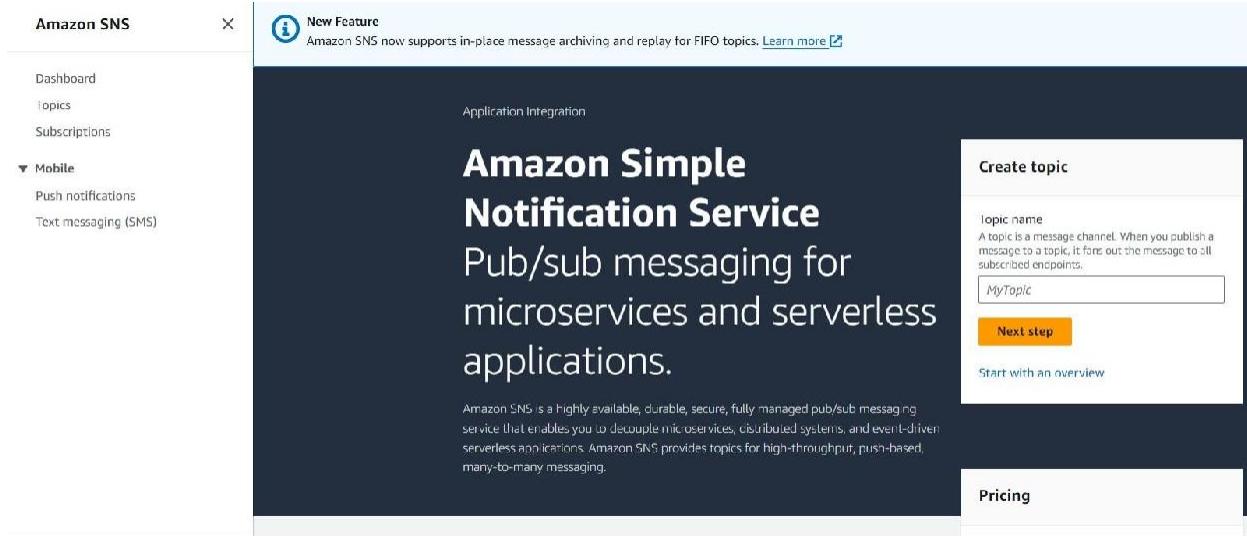
The screenshot shows the AWS search interface with the query 'sns' entered in the search bar. The results are categorized under 'Services' and 'Features'.

**Services**

- Simple Notification Service (SNS) - SNS managed message topics for Pub/Sub
- Route 53 Resolver - Resolve DNS queries in your Amazon VPC and on-premises network.
- Route 53 - Scalable DNS and Domain Name Registration
- AWS End User Messaging - Engage your customers across multiple communication channels

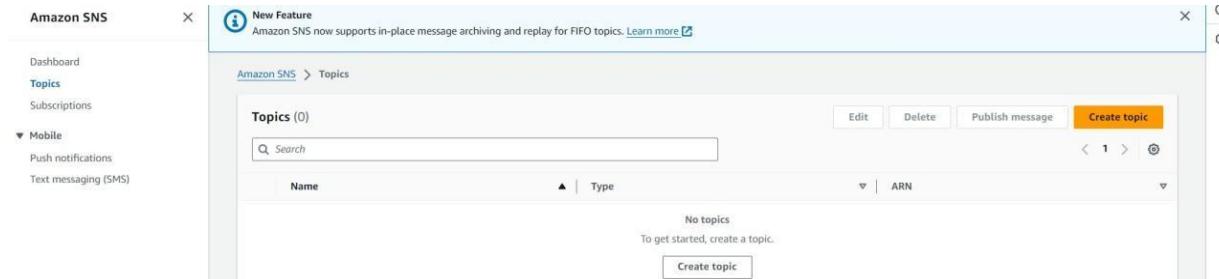
**Features**

- Events - ElastiCache feature
- SMS - AWS End User Messaging feature
- Hosted zones - Route 53 feature



The screenshot shows the Amazon Simple Notification Service (SNS) interface. On the left, there is a sidebar with options like Dashboard, Topics, Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and a New Feature notice about message archiving and replay for FIFO topics. The main content area has a dark header "Application Integration" and a large title "Amazon Simple Notification Service" followed by a subtitle "Pub/sub messaging for microservices and serverless applications." Below the title is a brief description of the service. A "Create topic" dialog box is open on the right, prompting for a "topic name" (with "MyTopic" entered) and a "Next step" button. At the bottom right of the main content area, there is a "Pricing" link.

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



The screenshot shows the "Topics" list screen in the Amazon SNS interface. The sidebar on the left includes "Topics" (which is selected and highlighted in blue). The main content area shows a table titled "Topics (0)" with columns for "Name", "Type", and "ARN". A search bar at the top of the table says "Search". Below the table, it says "No topics" and "To get started, create a topic." with a "Create topic" button. There are also "Edit", "Delete", "Publish message", and a yellow "Create topic" button at the top right of the table area.

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

[Amazon SNS](#) > [Topics](#) > [Create topic](#)

## Create topic

### Details

Type [Info](#)  
Topic type cannot be modified after topic is created

FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

Standard

- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

Name

BookRequestNotifications

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (\_).

Display name - *optional* [Info](#)

To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

My Topic

Maximum 100 characters.

► **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.

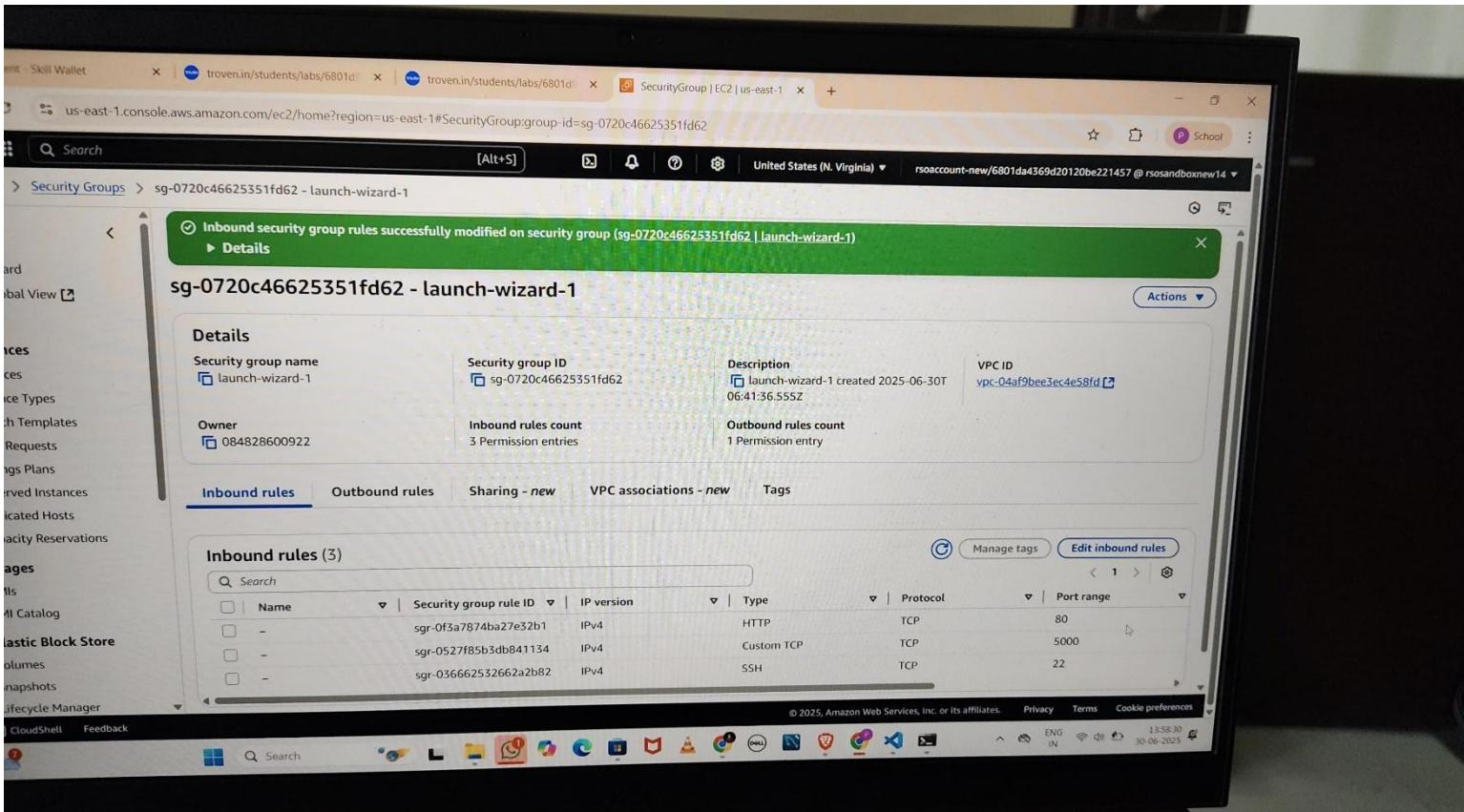
► **Delivery status logging - optional** [Info](#)  
 These settings configure the logging of message delivery status to CloudWatch Logs.

► **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.**



The screenshot shows the AWS Security Groups console for the 'sg-0720c46625351fd62 | launch-wizard-1' group. The 'Details' tab is selected, displaying the following information:

- Security group name:** launch-wizard-1
- Security group ID:** sg-0720c46625351fd62
- Description:** launch-wizard-1 created 2025-06-30T06:41:36.555Z
- VPC ID:** vpc-04af9bee3ec4e58fd
- Owner:** 084828600922
- Inbound rules count:** 3 Permission entries
- Outbound rules count:** 1 Permission entry

The 'Inbound rules' tab is selected, showing three rules:

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-0f3a7874ba27e32b1	IPv4	HTTP	TCP	80
-	sgr-0527f85b3db841134	IPv4	Custom TCP	TCP	5000
-	sgr-036662532662a2b82	IPv4	SSH	TCP	22

- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**
- **Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.**

Amazon SNS > Subscriptions > Create subscription

### Create subscription

**Details**

**Topic ARN**  

X

**Protocol**  
The type of endpoint to subscribe

**Endpoint**  
An email address that can receive notifications from Amazon SNS.

i After your subscription is created, you must confirm it. Info

**► Subscription filter policy - optional** Info  
This policy filters the messages that a subscriber receives.

**► Redrive policy (dead-letter queue) - optional** Info  
Send undeliverable messages to a dead-letter queue.

Cancel
Create subscription

Amazon SNS

New Feature  
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Subscription to BookRequestNotifications created successfully  
The ARN of the subscription is arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications:d78e0371-9235-404d-952c-85c2743607c4.

Amazon SNS > Topics > BookRequestNotifications > Subscription: d78e0371-9235-404d-952c-85c2743607c4

Subscription: d78e0371-9235-404d-952c-85c2743607c4

Edit Delete

Details	
ARN	arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications:d78e0371-9235-404d-952c-85c2743607c4
Endpoint	instantlibrary2@gmail.com
Topic	BookRequestNotifications
Subscription Principal	arn:aws:iam:557690616836:root
Status	Pending confirmation
Protocol	EMAIL

Subscription filter policy [Edit](#) | [Redrive policy \(dead-letter queue\)](#)

Subscription filter policy [Info](#)  
This policy filters the messages that a subscriber receives.

No filter policy configured for this subscription.  
To apply a filter policy, edit this subscription.

Edit

- **After subscription request for the mail confirmation**

Amazon SNS

New Feature  
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Confirmation request was sent successfully.  
The ARN of the subscription is arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications:7ee5d16-12ad-4731-971d-c342c2213ed5.

Amazon SNS > Topics > BookRequestNotifications

BookRequestNotifications

Edit Delete Publish message

Details	
Name	BookRequestNotifications
ARN	arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications
Type	Standard
Display name	-
Topic owner	557690616836

Subscriptions Access policy Data protection policy Delivery policy (HTTP/S) Delivery status logging Encryption Tags Integrations

Subscriptions (2)

ID	Endpoint	Status	Protocol
Pending confirmation	instantlibrary2@gmail.com	Pending confirmation	EMAIL

Q: Search Edit Delete Request confirmation Confirm subscription Create subscription < 1 >

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

AWS Notification - Subscription Confirmation Inbox ×**AWS Notifications** <no-reply@sns.amazonaws.com>

9

to me ▾

You have chosen to subscribe to the topic:

**arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications**

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

---

**AWS Notifications** <no-reply@sns.amazonaws.com>

to me ▾

[...](#)

You have chosen to subscribe to the topic:

**arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications**

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

---



Simple Notification Service

**Subscription confirmed!**

You have successfully subscribed.

Your subscription's id is:

**arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications:d78e0371-9235-404d-952c-85c2743607c4**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.**

Dashboard    Topics    Subscriptions

▼ Mobile    Push notifications    Text messaging (SMS)

Amazon SNS > Topics > BookRequestNotifications

### BookRequestNotifications

**Details**

Name	BookRequestNotifications	Display name	
ARN	arn:aws:sns:ap-south-1:55769616836:BookRequestNotifications	Topic owner	55769616836
Type	Standard		

**Subscriptions**    Access policy    Data protection policy    Delivery policy (HTTP/S)    Delivery status logging    Encryption    Tags    Integrations

**Subscriptions (2)**

ID	Endpoint	Status	Protocol
d78e0371-9235-404d-952c-85c2743607c4	irotantilibrary2@gmail.com	Confirmed	EMAIL

**Actions:** Edit    Delete    Request confirmation    Confirm subscription    Create subscription

## Milestone 4:Backend Development and Application Setup

- Activity 4.1: Develop the backend using Flask

- File Explorer Structure



Description: set up the **INSTANT LIBRARY** project with an `app.py` file, a `static/` folder for assets, and a `templates/` directory containing all required HTML pages like `home`, `login`, `register`, subject-specific pages (e.g., `computer science.html`, `data science.html`), and utility pages (e.g., `request-form.html`, `statistics.html`).

### Description of the code :

- Flask App Initialization

```
from flask import Flask, render_template, request, redirect, url_for
import boto3
from boto3.dynamodb.conditions import Key
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from bcrypt import hashpw, gensalt, checkpw
```

Description: import essential libraries including `Flask` utilities for routing, `Boto3` for `DynamoDB` operations, `SMTP` and `email` modules for sending mails, and `Bcrypt` for password hashing and verification

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

Description: initialize the `Flask` application instance using `Flask( name )` to start building the web app.

- Dynamodb Setup:

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

users_table = dynamodb.Table('travelgo_users')
trains_table = dynamodb.Table('trains')
bookings_table = dynamodb.Table('bookings')
```

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

- **SNS Connection**

**Description: Configure SNS to send notifications when a book request is submitted. Paste your stored ARN link in the sns topic arn space, along with the region name where the SNS topic is created. Also, specify the chosen email service in SMTP SERVER (e.g., Gmail, Yahoo, etc.) and enter the subscribed email in the SENDER EMAIL section. Create an 'App password' for the email ID and store it in the SENDER PASSWORD section.**

- **Routes for Web Pages**

- **Home Route:**

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

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

```
SNS_TOPIC_ARN = 'arn:aws:sns:ap-south-1:353250843450:TravelGoBookingTopic'
```

```
def send_sns_notification(subject, message):
    try:
        sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Subject=subject,
            Message=message
        )
    except Exception as e:
        print(f"SNS Error: {e}")
```

```
def send_sns_notification(subject, message):
    try:
        sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Subject=subject,
            Message=message
        )
    except Exception as e:
```

- Register Route:

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

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

- login Route (GET/POST):

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = users_table.get_item(Key={'email': email})
        if 'Item' in user and check_password_hash(user['Item']['password'], password):
            session['email'] = email
            flash('Logged in successfully!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid email or password!', 'error')
            return render_template('login.html')
    return render_template('login.html')
```

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

### Logout and Dashboard :-

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

@app.route('/dashboard')
def dashboard():
    if 'email' not in session:
        return redirect(url_for('login'))
    user_email = session['email']
    response = bookings_table.query(
        KeyConditionExpression=Key('user_email').eq(user_email),
        ScanIndexForward=False
    )
    bookings = response.get('Items', [])
    for booking in bookings:
        if 'total_price' in booking:
            try:
                booking['total_price'] = float(booking['total_price'])
            except Exception:
                booking['total_price'] = 0.0
    return render_template('dashboard.html', username=user_email, bookings=bookings)
```

### Train :-

```

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

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

    booking_details = {
        'name': request.args.get('name'),
        'train_number': request.args.get('trainNumber'),
        'source': request.args.get('source'),
        'destination': request.args.get('destination'),
        'departure_time': request.args.get('departureTime'),
        'arrival_time': request.args.get('arrivalTime'),
        'price_per_person': Decimal(request.args.get('price')),
        'travel_date': request.args.get('date'),
        'num_persons': int(request.args.get('persons')),
        'item_id': request.args.get('trainId'),
        'booking_type': 'train',
        'user_email': session['email'],
        'total_price': Decimal(request.args.get('price')) * int(request.args.get('persons'))
    }
}

```

```

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

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

all_seats = [f"S{i}" for i in range(1, 101)]
available_seats = [seat for seat in all_seats if seat not in booked_seats]

if len(available_seats) < booking_details['num_persons']:
    flash("Not enough seats available.", "error")
    return redirect(url_for("train"))

session['pending_booking'] = booking_details
return render_template('confirm_train_details.html', booking=booking_details, available_seats=available_seats[:booking_details['num_persons']])

```

## Buses:-

```

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

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

all_seats = ["S{i}" for i in range(1, 41)]
session['pending_booking'] = booking

return render_template("select_bus_seats.html", booking=booking, booked_seats=booked_seats, all_seats=all_seats)

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

    booking = session.pop('pending_booking', None)
    selected_seats = request.form['selected_seats']

    if not booking or not selected_seats:
        flash("Booking failed! Missing data.", "error")
        return redirect(url_for("bus"))

    # Prevent double booking
    response = bookings_table.query(
        IndexName='GSI_ItemDate',
        KeyConditionExpression=Key('item_id').eq(booking['item_id']) & Key('travel_date').eq(booking['travel_date'])
    )
    existing = set()
    for b in response.get('Items', []):
        if 'seats_display' in b:
            |   existing.update(b['seats_display'].split(', '))

    selected = selected_seats.split(',')
    if any(s in existing for s in selected):
        flash("One or more selected seats are already booked!", "error")
        return redirect(url_for("bus"))

    booking['seats_display'] = selected_seats
    booking['booking_id'] = str(uuid.uuid4())
    booking['booking_date'] = datetime.now().isoformat()

    bookings_table.put_item(Item=booking)

    send sns_notification(
        subject="Bus Booking Confirmed",
        message=f'Your bus from {booking["source"]} to {booking["destination"]} on {booking["travel_date"]} is confirmed.\nSeats: {booking["seats_display"]}\nTotal: ${booking["total_price"]}')
    )

    flash('Bus booking confirmed!', 'success')
    return redirect(url_for('dashboard'))

```

```

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

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

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

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

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

```

## Flight:

```

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

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

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

    booking = {
        'booking_type': 'flight',
        'flight_id': request.form['flight_id'],
        'airline': request.form['airline'],
        'flight_number': request.form['flight_number'],
        'source': request.form['source'],
        'destination': request.form['destination'],
        'departure_time': request.form['departure_time'],
        'arrival_time': request.form['arrival_time'],
        'travel_date': request.form['travel_date'],
        'num_persons': int(request.form['num_persons']),
        'price_per_person': Decimal(request.form['price_per_person']),
        'total_price': Decimal(request.form['total_price']),
        'user_email': session['email'],
        'booking_date': datetime.now().isoformat(),
        'booking_id': str(uuid.uuid4())
    }

    bookings_table.put_item(Item=booking)
    # SNS for Flight
    send sns_notification(
        subject="Flight Booking Confirmed",
        message=f"Your flight booking on {booking['travel_date']} from {booking['source']} to {booking['destination']} with {booking['airline']} is confirmed.\nTotal: ${booking['total_price']}"
    )
    flash('Flight booking confirmed successfully!', 'success')
    return redirect(url_for('dashboard'))
}

```

```

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

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

    booking = {
        'name': request.args.get('name'),
        'location': request.args.get('location'),
        'checkin_date': request.args.get('checkin'),
        'checkout_date': request.args.get('checkout'),
        'num_rooms': int(request.args.get('rooms')),
        'num_guests': int(request.args.get('guests')),
        'price_per_night': Decimal(request.args.get('price')),
        'rating': int(request.args.get('rating'))
    }

    ci = datetime.fromisoformat(booking['checkin_date'])
    co = datetime.fromisoformat(booking['checkout_date'])
    nights = (co - ci).days
    booking['nights'] = nights
    booking['total_price'] = booking['price_per_night'] * booking['num_rooms'] * nights

    return render_template('confirm_hotel_details.html', booking=booking)

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

    booking = {
        'booking_type': 'hotel',
        'name': request.form['hotel_name'],
        'location': request.form['location'],
        'checkin_date': request.form['checkin'],
        'checkout_date': request.form['checkout'],
        'num_rooms': int(request.form['rooms']),
        'num_guests': int(request.form['guests']),
        'price_per_night': Decimal(request.form['price']),
        'rating': int(request.form['rating']),
        'user_email': session['email'],
        'booking_date': datetime.now().isoformat(),
        'booking_id': str(uuid.uuid4())
    }

    ci = datetime.fromisoformat(booking['checkin_date'])
    co = datetime.fromisoformat(booking['checkout_date'])
    nights = (co - ci).days
    booking['total_price'] = booking['price_per_night'] * booking['num_rooms'] * nights

    bookings_table.put_item(Item=booking)
}

```

### Response :-

```
# SNS for Flight
send_sns_notification(
    subject="Flight Booking Confirmed",
    message=f"Your flight booking on {booking['travel_date']} from {booking['source']} to {booking['destination']} with {booking['airline']} is confirmed.\nTotal: ${booking['total_price']}"
)

flash('Flight booking confirmed successfully!', 'success')
return redirect(url_for('dashboard'))
```

```
# SNS for Hotel
send_sns_notification(
    subject="Hotel Booking Confirmed",
    message=f"Hotel booking at {booking['name']} in {booking['location']} from {booking['checkin_date']} to {booking['checkout_date']} is confirmed.\nTotal: ${booking['total_price']}"
)

flash('Hotel booking confirmed successfully!', 'success')
return redirect(url_for('dashboard'))

app.route('/cancel_booking', methods=['POST'])
# cancel booking():
if 'email' not in session:
    return redirect(url_for('login'))

booking_id = request.form.get('booking_id')
user_email = session['email']

if not booking_id:
    flash("Error: Booking ID is missing for cancellation.", 'error')
    return redirect(url_for('dashboard'))

try:
    bookings_table.delete_item(
        Key={'user_email': user_email, 'booking_date': request.form.get('booking_date')}}
    flash(f"Booking cancelled successfully!", 'success')
except Exception as e:
    flash(f"Failed to cancel booking: {str(e)}", 'error')

return redirect(url_for('dashboard'))
```

### Deployment Code:

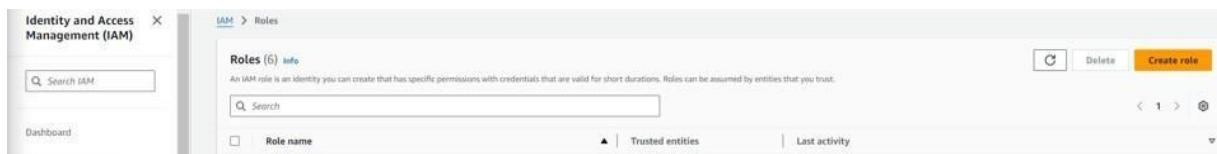
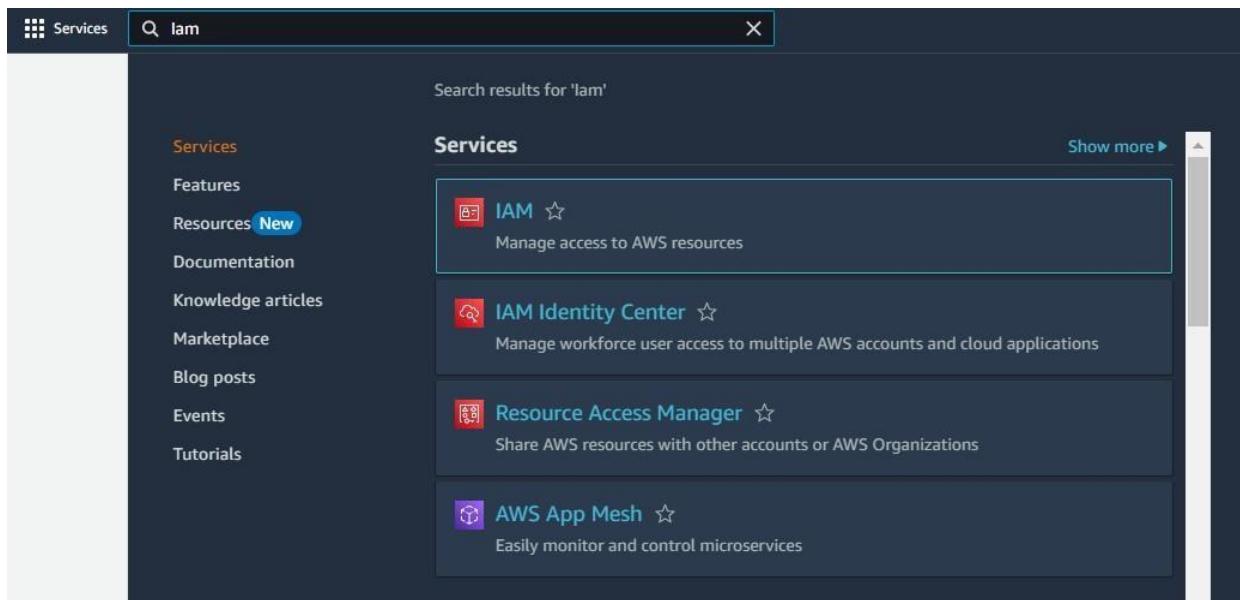
```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

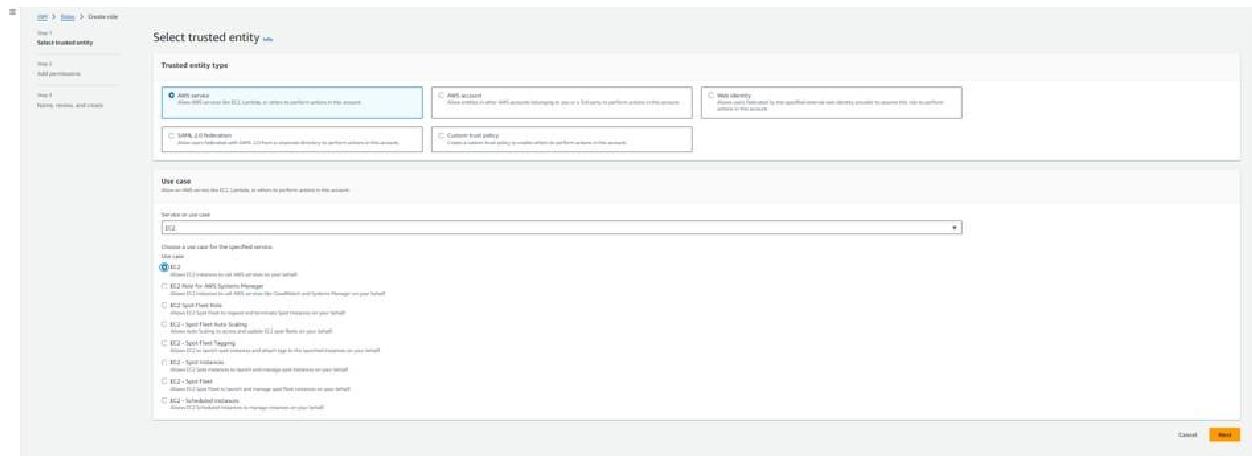
Description: start the Flask server to listen on all network interfaces (0.0.0.0) at port 80 with debug mode enabled for development and testing.

## Milestone 5: IAM Role Setup

- Activity 5.1: Create IAM Role.

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





Step 1  
Select trusted entity

Step 2  
Add permissions

Step 3  
Name, review, and create

Select trusted entity

Trusted entity type

AWS service Allows AWS services like S3, Lambda, or CloudWatch to perform actions in this account.

AWS account Allows users in other AWS accounts to assume this identity to perform actions in this account.

AWS Lambda function Allows Lambda functions to assume this identity to perform actions in this account.

Custom trust policy Creates a custom trust policy giving the specified entities permission to assume this role on your behalf.

Use a role for this task

EC2

Choose a use case for the specified service

EC2

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

EC2 Role for AWS Systems Manager Allows EC2 instances to call AWS Systems Manager on your behalf.

EC2 Spot Fleet Role Allows EC2 Spot Fleet to request automatically terminate spot instances on your behalf.

EC2 - Spot Fleet Role Scaling Allows EC2 Spot Fleet Role Scaling to manage EC2 Spot Fleet on your behalf.

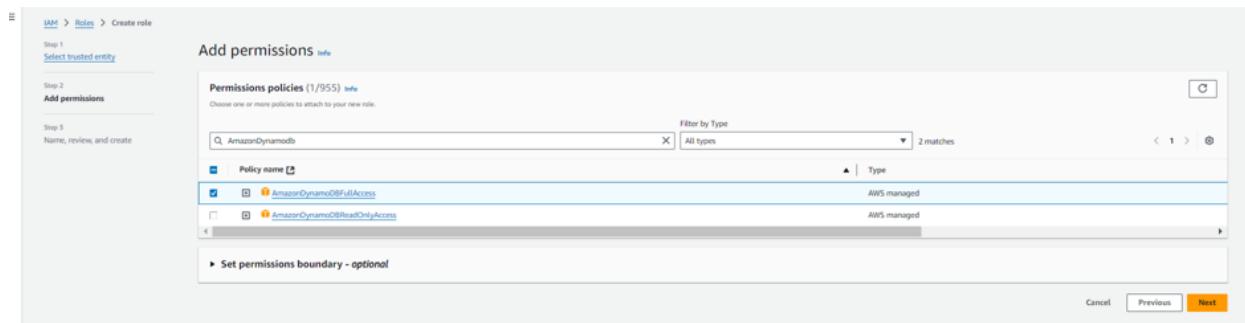
EC2 - Spot Fleet Tagging Allows EC2 Spot Fleet Tagging to tag EC2 Spot Fleet instances and project tags to the specified instances on your behalf.

EC2 - Agent Instances Allows EC2 Agent instances to report and manage agent instances on your behalf.

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

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

Cancel Next



IAM > Roles > Create role

Step 1  
Select trusted entity

Step 2  
Add permissions

Step 3  
Name, review, and create

Add permissions

Permissions policies (1/955)

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

Filter by Type

Q AmazonDynamoDB All types 2 matches

Policy name

AmazonDynamoDBFullAccess AWS managed

AmazonDynamoDBReadOnlyAccess AWS managed

Set permissions boundary - optional

Cancel Previous Next

- **Activity 5.2: Attach Policies.**

**Attach the following policies to the role:**

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

AMI > Roles > Create role

Step 1: Select trusted entities

### Add permissions

Permissions policies (0/955) [Info](#)

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

Policy name	Type
<input checked="" type="checkbox"/>  AmazonS3FullAccess	AWS managed
<input type="checkbox"/>  AmazonS3ReadWriteAccess	AWS managed
<input type="checkbox"/>  AmazonVPCFullAccess	AWS managed
<input type="checkbox"/>  AWSLambdaBasicExecutionRole	AWS managed
<input type="checkbox"/>  AWSLambdaDefenderEvaluationRole	AWS managed

Set permissions boundary - optional

[Cancel](#) [Previous](#) [Next](#)

Step 1: Select trusted entities

### Name, review, and create

Role details

Role name:

Description: Add a short description for this role.

Allows EC2 instances to call AWS services on your behalf.

Step 2: Add permissions

Trust policy:

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

Permissions policy summary

Policy name	Type	Attached as
<input checked="" type="checkbox"/> <a href="#">AmazonS3FullAccess</a>	AWS managed	Permissions policy
<input type="checkbox"/> <a href="#">AmazonVPCFullAccess</a>	AWS managed	

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 this resource.

[Add new tag](#)

You can add up to 50 more tags.

[Cancel](#) [Previous](#) [Create role](#)

IAM > Roles > sns\_Dynamodb\_role

**Sns\_Dynamodb\_role** Info

Allows EC2 instances to call AWS services on your behalf.

**Summary** Edit

Creation date October 13, 2024, 23:00 (UTC+00:00)	ARN arn:aws:iam::557090616836:role/sns_Dynamodb_role	Instance profile ARN arn:aws:iam::557090616836:instance-profile/sns_Dynamodb_role
Last activity <span style="color: green;">6 days ago</span>	Maximum session duration 1 hour	

**Permissions** Trust relationships Tags Last Accessed Revoke sessions

**Permissions policies (2) Info** You can attach up to 10 managed policies.

**Attached policies**

Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	4
AmazonSNSFullAccess	AWS managed	2

**Add permissions** Add policy Simulate Remove Add permissions

## Milestone 6: EC2 Instance Setup

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

---

 static	Initial commit
 templates	Update statistics.html
 app.py	Update app.py

---

Local      Codespaces

---

 [Clone](#)      

[HTTPS](#)    [SSH](#)    [GitHub CLI](#)

---

<https://github.com/AlekhyaPenubakula/InstantLit> 

Clone using the web URL.

---

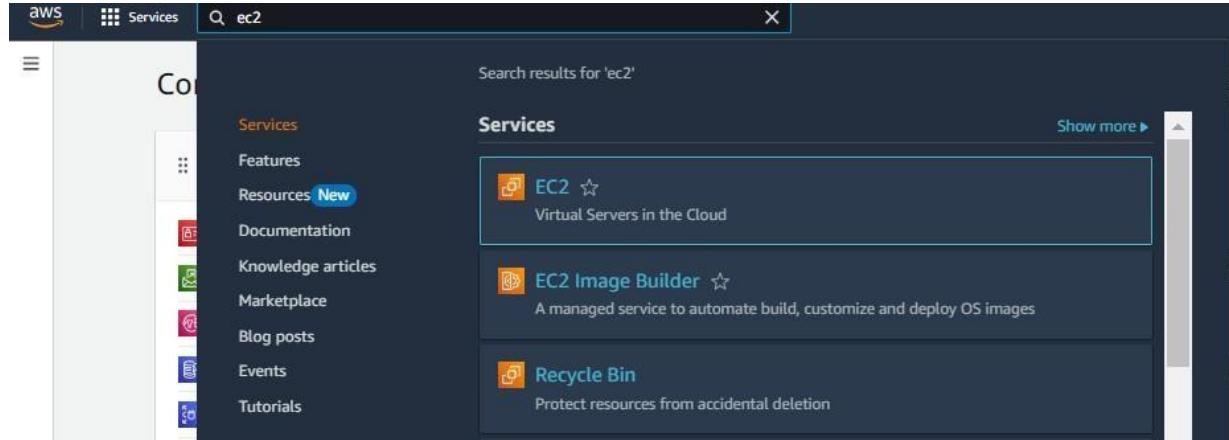
 [Open with GitHub Desktop](#)

---

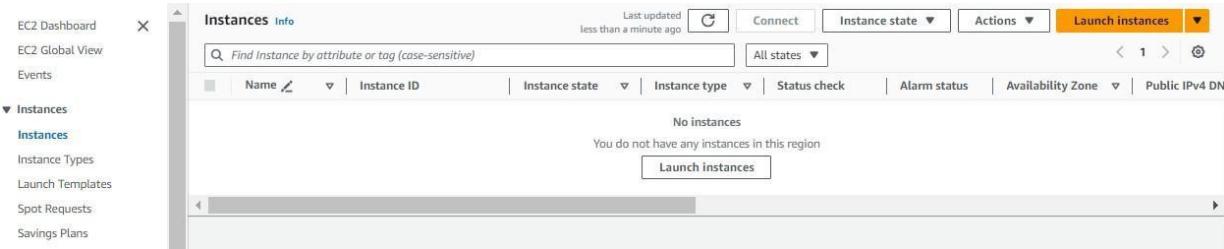
 [Download ZIP](#)

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

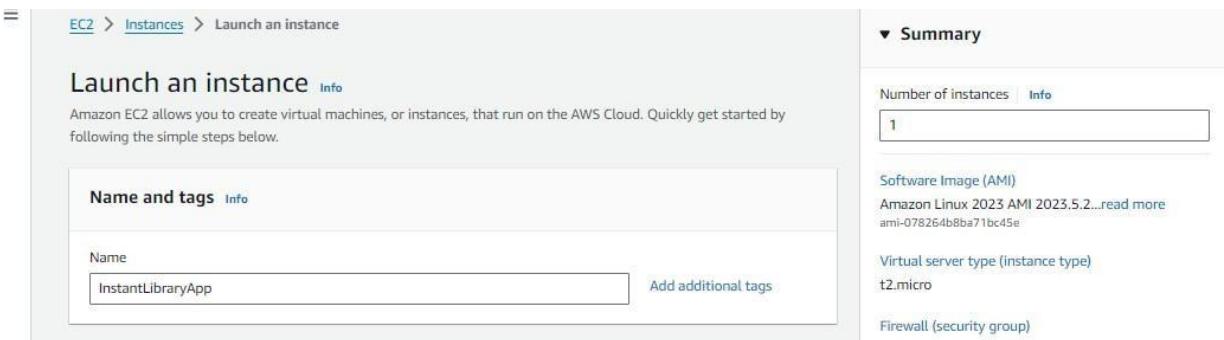
- **Launch EC2 Instance**
  - **In the AWS Console, navigate to EC2 and launch a new instance.**



- **Click on Launch instance to launch EC2 instance**

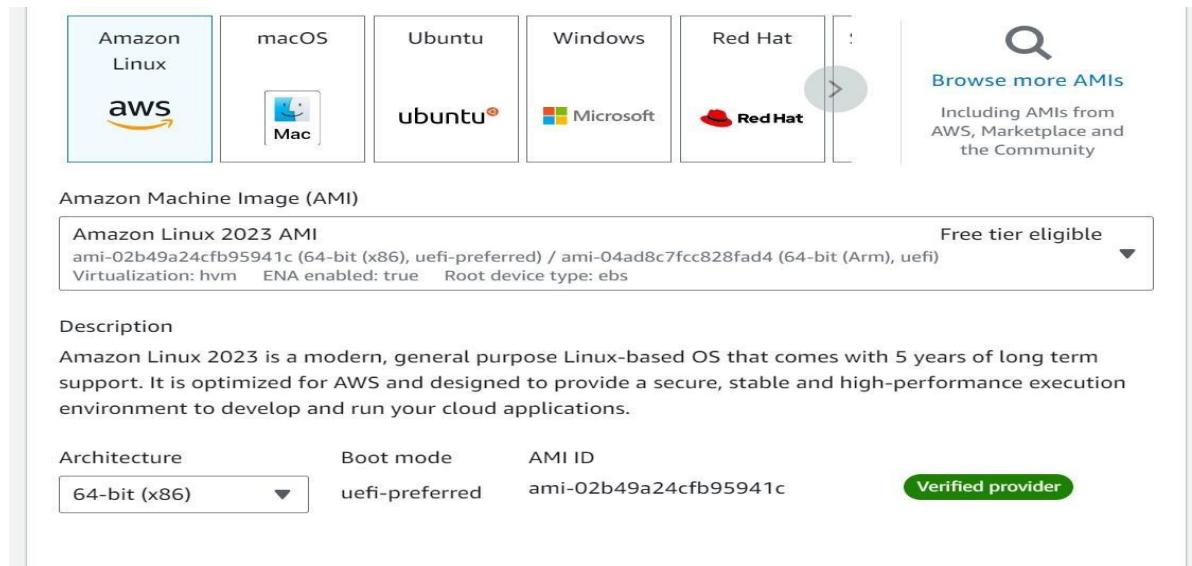


The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with options like EC2 Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, and Savings Plans. The main area has a header with 'Instances Info' and filters for 'Find Instance by attribute or tag (case-sensitive)' and 'All states'. Below that is a table with columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DNS. A message says 'No instances' and 'You do not have any instances in this region'. At the bottom right is a large blue 'Launch Instances' button.



This is the first step of the 'Launch an instance' wizard. It shows the URL: EC2 > Instances > Launch an instance. The title is 'Launch an instance' with an 'Info' link. The sub-instruction reads: 'Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.' Below this is a 'Name and tags' section with a 'Name' input field containing 'InstantLibraryApp' and a 'Add additional tags' link. To the right, under 'Summary', it shows 'Number of instances' set to 1. Further down are sections for 'Software Image (AMI)', 'Virtual server type (instance type)', and 'Firewall (security group)'. A search icon and a 'Browse more AMIs' link are also present.

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



This screenshot shows the 'Amazon Machine Image (AMI)' selection screen. It features a grid of OS icons: Amazon Linux (aws logo), macOS (Mac logo), Ubuntu (ubuntu logo), Windows (Microsoft logo), and Red Hat (Red Hat logo). To the right is a search icon and a link to 'Browse more AMIs'. Below the grid, a section for 'Amazon Linux 2023 AMI' is shown with details: 'ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)'. It also lists 'Virtualization: hvm', 'ENA enabled: true', and 'Root device type: ebs'. A 'Free tier eligible' badge with a dropdown arrow is visible. Under the heading 'Description', it says: 'Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.' At the bottom, there are dropdown menus for 'Architecture' (set to '64-bit (x86)'), 'Boot mode' (set to 'uefi-preferred'), and 'AMI ID' (set to 'ami-02b49a24cfb95941c'). A green 'Verified provider' badge is also present.

- **Create and download the key pair for Server access.**

▼ **Instance type** [Info](#) | [Get advice](#)

Instance type

t2.micro	Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true	
On-Demand Linux base pricing: 0.0124 USD per Hour	
On-Demand Windows base pricing: 0.017 USD per Hour	
On-Demand RHEL base pricing: 0.0268 USD per Hour	
On-Demand SUSE base pricing: 0.0124 USD per Hour	

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

▼ **Key pair (login)** [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select Create new key pair

### Create key pair

Key pair name  
 Key pairs allow you to connect to your instance securely.  
 The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

<input checked="" type="radio"/> RSA RSA encrypted private and public key pair	<input type="radio"/> ED25519 ED25519 encrypted private and public key pair
---	--

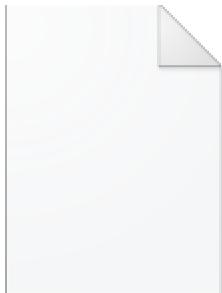
Private key file format

.pem  
For use with OpenSSH

.ppk  
For use with PuTTY

**⚠ When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#)**

[Cancel](#) [Create key pair](#)



## InstantLibrary.pem

**Description**

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture	Boot mode	AMI ID	Username
64-bit (x86)	uefi-preferred	ami-078264b8ba71bc45e	ec2-user <span style="border: 1px solid green; border-radius: 50%; padding: 2px;">Verified provider</span>

**Instance type** Info | Get advice

Instance type

<b>t2.micro</b> Family: t2 1 vCPU 1 GiB Memory Current generation: true On-Demand Linux base pricing: 0.0124 USD per Hour On-Demand Windows base pricing: 0.017 USD per Hour On-Demand RHEL base pricing: 0.0268 USD per Hour On-Demand SUSE base pricing: 0.0124 USD per Hour	Free tier eligible <input checked="" type="radio"/> All generations <a href="#">Compare instance types</a>
---	--

Additional costs apply for AMIs with pre-installed software

**Key pair (login)** Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - **required**

Create new key pair

Cancel
Preview code
Launch instance

**Summary**

Number of instances Info

Software Image (AMI)  
 Amazon Linux 2023 AMI 2023.5.2... [read more](#)  
 ami-078264b8ba71bc45e

Virtual server type (instance type)  
 t2.micro

Firewall (security group)  
 New security group

Storage (volumes)  
 1 volume(s) - 8 GiB

**Free tier:** In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

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

**▼ Network settings [Info](#)**

VPC - required [Info](#)  
 vpc-03cdc7b6f19dd7211 (default)  
 172.31.0.0/16

Subnet [Info](#)  
 No preference [▼](#) [C](#) Create new subnet [↗](#)

Auto-assign public IP [Info](#)  
 Enable [▼](#)

Additional charges apply when outside of free tier allowance

Firewall (security groups) [Info](#)  
 A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group [↗](#)  Select existing security group

Security group name - required  
 launch-wizard

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and \_-:/()#,@[]+=&;!\$^\*

Description - required [Info](#)  
 launch-wizard created 2024-10-13T17:49:56.622Z

**Inbound Security Group Rules**

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0) [Remove](#)

Type <a href="#">Info</a> ssh <a href="#">▼</a>	Protocol <a href="#">Info</a> TCP	Port range <a href="#">Info</a> 22
Source type <a href="#">Info</a> Anywhere <a href="#">▼</a>	Source <a href="#">Info</a> <a href="#">Add CIDR, prefix list or security</a> 0.0.0.0/0 <a href="#">X</a>	Description - optional <a href="#">Info</a> e.g. SSH for admin desktop

▼ Security group rule 2 (TCP, 80, 0.0.0.0/0) [Remove](#)

Type <a href="#">Info</a> HTTP <a href="#">▼</a>	Protocol <a href="#">Info</a> TCP	Port range <a href="#">Info</a> 80
Source type <a href="#">Info</a> Custom <a href="#">▼</a>	Source <a href="#">Info</a> <a href="#">Add CIDR, prefix list or security</a> 0.0.0.0/0 <a href="#">X</a>	Description - optional <a href="#">Info</a> e.g. SSH for admin desktop

▼ Security group rule 3 (TCP, 5000, 0.0.0.0/0) [Remove](#)

Type <a href="#">Info</a> Custom TCP <a href="#">▼</a>	Protocol <a href="#">Info</a> TCP	Port range <a href="#">Info</a> 5000
Source type <a href="#">Info</a> Custom <a href="#">▼</a>	Source <a href="#">Info</a> <a href="#">Add CIDR, prefix list or security</a> 0.0.0.0/0 <a href="#">X</a>	Description - optional <a href="#">Info</a> e.g. SSH for admin desktop

[Add security group rule](#)

EC2 > ... > Launch an instance

**Success**  
Successfully initiated launch of instance i-001861022fbcac290

▶ Launch log

**Next Steps**

Q: What would you like to do next with this instance, for example "Create volume" or "Create backup"?

< 1 2 3 4 >

Create billing and free tier usage alerts  To manage costs and avoid surprise bills, set up usage notifications for billing and free tier usage thresholds.  Create billing alerts [?]	Connect to your instance  Once your instance is running, log into it from your local computer.  Connect to instance [?]  Learn more [?]	Connect an RDS database  Configure the connection between an EC2 instance and a database to allow traffic flow between them.  Connect an RDS database [?]  Create a new RDS database [?]  Learn more [?]	Create EBS snapshot policy  Create a policy that automates the creation, retention, and deletion of EBS snapshots.  Create EBS snapshot policy [?]	Manage detailed monitoring  Enable or disable detailed monitoring for the instance. If you enable detailed monitoring, the Amazon CloudWatch Metrics displays monitoring graphs with 1-minute periods.  Manage detailed monitoring [?]	Create Load Balancer  Create a application, network gateway, or classic Elastic Load Balancer.  Create Load Balancer [?]
Create AWS budget  AWS Budgets allows you to create budgets, forecast spend, and take action on your costs and usage from a single location.  Create AWS budget [?]	Manage CloudWatch alarms  Create or update Amazon CloudWatch alarms for the instance.  Manage CloudWatch alarms [?]	Disaster recovery for your instances  Recover the instance you just launched into a different Availability Zone or a different Region using AWS Family Disaster Recovery (FDR).  Disaster recovery for your instances [?]	Monitor for suspicious runtime activities  Amazon GuardDuty enables you to continuously monitor for malicious runtime activity and unauthorized behavior, with near real-time visibility into on-going activities occurring across your Amazon EC2 workloads.  Monitor for suspicious runtime activities [?]	Get instance screenshot  Capture a screenshot from the instance and view it as an image. This is useful for troubleshooting an unreachable instance.  Get instance screenshot [?]	Get system log  View the instance's system log to troubleshoot issues.  Get system log [?]

**View all instances**

- To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected, navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the EC2 Dashboard, click the Connect button. From the connection methods presented, choose EC2 Instance Connect. Finally, click Connect again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.**

Instances (1/2) **Info**

Last updated less than a minute ago

Q: Find Instance by attribute or tag (case-sensitive)

All states ▾

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6 IPs	Monitoring	Security g
InstantLibrary...	i-001861022fbcac290	Stopped	t2.micro	-	View alarms +	ap-south-1b	-	-	-	-	disabled	launch-wi

EC2 > Instances > i-001861022fbcac290

Instance summary for i-001861022fbcac290 (InstantLibraryApp) **Info**

Updated less than a minute ago

Instance ID i-001861022fbcac290	Public IPv4 address -	Private IPv4 addresses 172.31.5.5
IPv6 address -	Instance state Stopped	Public IPv4 DNS -
Hostname type IP name: ip-172-31-5-ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-5-ap-south-1.compute.internal	Elastic IP addresses -
Answer private resource DNS name IPV4 (A)	Instance type t2.micro	AWS Compute Optimizer findings Opt-in to AWS Compute Optimizer for recommendations.   Learn more [?]
Auto-assigned IP address -	VPC ID vpc-03cdcf7b6f19dd7211	Auto Scaling Group name -
IAM Role sns_Dynamodb_role	Subnet ID subnet-0d9fa3144480cc9a9	
IMDSv2 Required	Instance ARN arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290	

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

EC2 > Instances > i-001861022fbcac290

Instance summary for i-001861022fbcac290 (instantLibraryApp) [Info](#)

Updated less than a minute ago

Instance ID	i-001861022fbcac290	Public IPv4 address	172.31.3.4
IPv6 address	-	Private IPv4 address	172.31.3.4
Instance type	t2.micro	Public IPv6 DNS	-
IP name	ip-172-31-3-5.ap-south-1.compute.internal	Private IPv6 DNS	-
Answer private resource DNS name	ip-172-31-3-5.ap-south-1.compute.internal	Change security groups	<a href="#">Get Windows password</a>
IPv4 (A)	-	Networking	<a href="#">Modify IAM role</a>
Auto assigned IPv4 address	-	Security	<a href="#">Image and templates</a>
AMI Role	sns_Dynamodb_role	Elastic IP addresses	<a href="#">Monitor and troubleshoot</a>
IMDSv2	Requires	AWS Compute Optimizer findings	-

EC2 > Instances > i-001861022fbcac290 > Modify IAM role

## Modify IAM role [Info](#)

Attach an IAM role to your instance.

Instance ID

i-001861022fbcac290 (InstantLibraryApp)

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.

[▼](#) [Create new IAM role](#)

[Cancel](#) [Update IAM role](#)

- **Now connect the EC2 with the files**

## Connect to instance Info

Connect to your instance i-001861022fbcac290 (InstantLibraryApp) using any of these options

[EC2 Instance Connect](#) | [Session Manager](#) | [SSH client](#) | [EC2 serial console](#)



### Port 22 (SSH) is open to all IPv4 addresses

Port 22 (SSH) is currently open to all IPv4 addresses, indicated by **0.0.0.0/0** in the inbound rule in [your security group](#). For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: **13.233.177.0/29**. [Learn more](#).

#### Instance ID

i-001861022fbcac290 (InstantLibraryApp)

#### Connection Type

[Connect using EC2 Instance Connect](#)

Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.

[Connect using EC2 Instance Connect Endpoint](#)

Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

[Public IPv4 address](#)

13.200.229.59

[IPv6 address](#)

-

#### Username

Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

ec2-user



**Note:** In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

[Cancel](#)

[Connect](#)

```
A newer release of "Amazon Linux" is available.
Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
[ec2-user@ip-172-31-3-5 ~]$
```

i-001861022fbcac290 (InstantLibraryApp)

PublicIPs: 13.201.74.42 PrivateIPs: 172.31.3.5

## **Milestone 7: Deployment on EC2**

### **Activity 7.1: Install Software on the EC2 Instance**

Install Python3, Flask, and

Git: On Amazon Linux

2:

`sudo yum update -y`

`sudo yum install python3 git`

`sudo pip3 install flask boto3`

Verify Installations:

`flask --version`

`git --version`

### **Activity 7.2: Clone Your Flask Project from GitHub**

Clone your project repository from GitHub into the EC2 instance using Git.

Run: 'git clone https://github.com/vamsi200527/travelgoaws

Note: change your github-username and your-repository-name with your credentials here:

---

'git clone https://github.com/SoftwareKarthik/travelgoproject

- This will download your project to the EC2 instance.

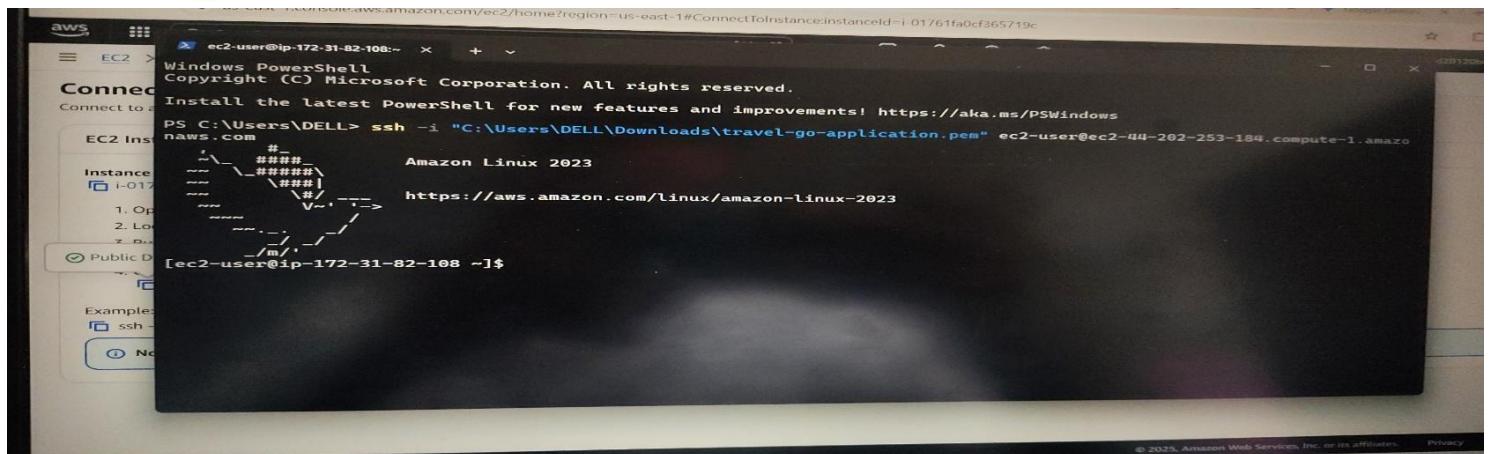
To navigate to the project directory, run the following command:

`cd InstantLibrary`

Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application

`sudo flask run --host=0.0.0.0 --port=80`

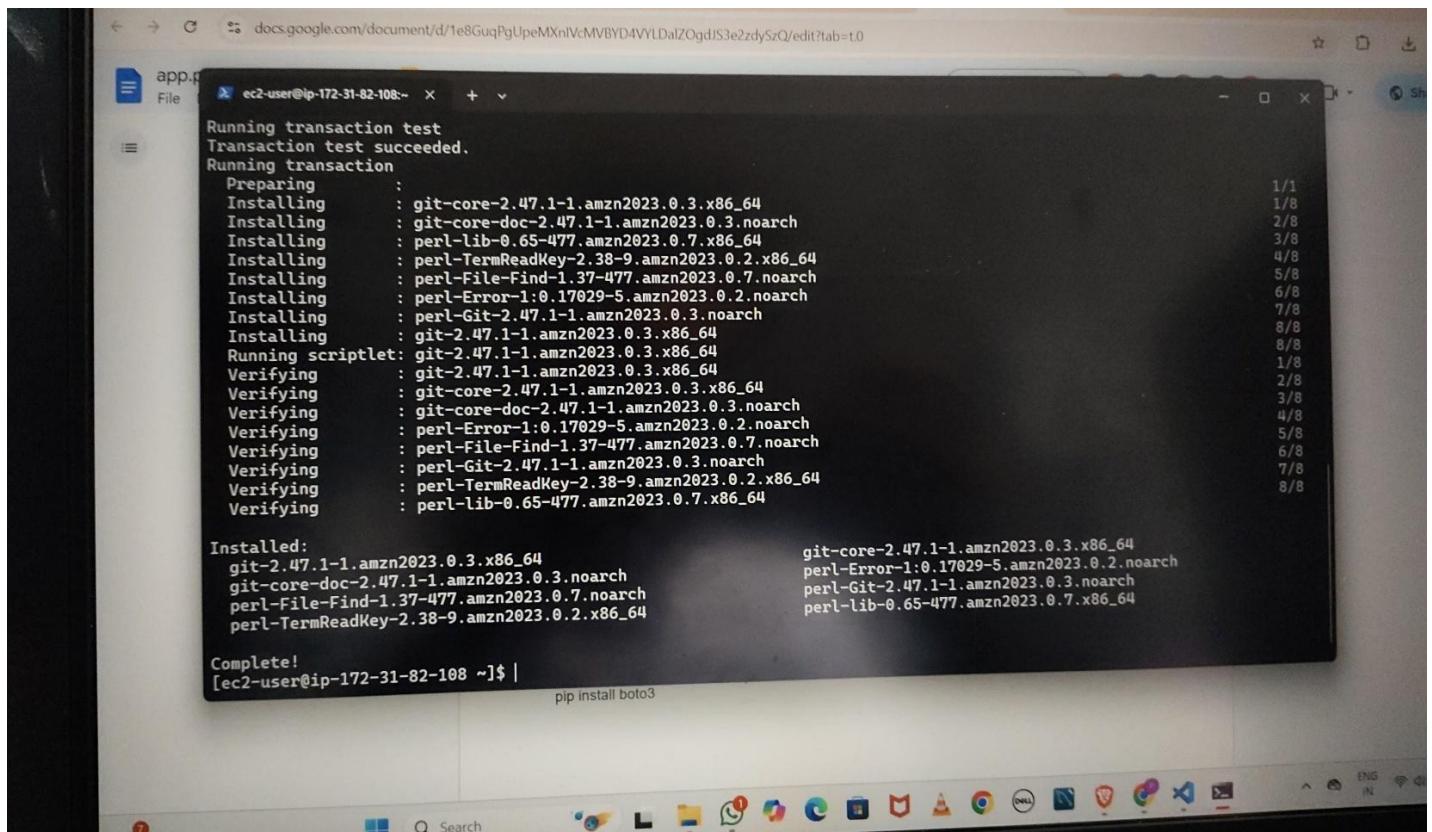


## Verify the Flask app is running:

[http://your-ec2-public-](http://your-ec2-public-ip)

ip

- Run the Flask app on the EC2 instance



```

docs.google.com/document/d/1e8GuqPgUpemXnIVcMVBYD4VYLDalZQgdJS3e2zdySzQ/edit?tab=t.0

app.py
ec2-user@ip-172-31-82-108:~$ pip install boto3
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : git-core-2.47.1-1.amzn2023.0.3.x86_64 1/8
Installing : git-core-doc-2.47.1-1.amzn2023.0.3.noarch 2/8
Installing : perl-lib-0.65-477.amzn2023.0.7.x86_64 3/8
Installing : perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64 4/8
Installing : perl-File-Find-1.37-477.amzn2023.0.7.noarch 5/8
Installing : perl-Error-1.0.17029-5.amzn2023.0.2.noarch 6/8
Installing : perl-Git-2.47.1-1.amzn2023.0.3.noarch 7/8
Installing : git-2.47.1-1.amzn2023.0.3.x86_64 8/8
Running scriptlet: git-2.47.1-1.amzn2023.0.3.x86_64 8/8
Verifying : git-2.47.1-1.amzn2023.0.3.x86_64 1/8
Verifying : git-core-2.47.1-1.amzn2023.0.3.x86_64 2/8
Verifying : git-core-doc-2.47.1-1.amzn2023.0.3.noarch 3/8
Verifying : perl-Error-1.0.17029-5.amzn2023.0.2.noarch 4/8
Verifying : perl-File-Find-1.37-477.amzn2023.0.7.noarch 5/8
Verifying : perl-Git-2.47.1-1.amzn2023.0.3.noarch 6/8
Verifying : perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64 7/8
Verifying : perl-lib-0.65-477.amzn2023.0.7.x86_64 8/8

Installed:
git-2.47.1-1.amzn2023.0.3.x86_64
git-core-doc-2.47.1-1.amzn2023.0.3.noarch
perl-File-Find-1.37-477.amzn2023.0.7.noarch
perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64

git-core-2.47.1-1.amzn2023.0.3.x86_64
perl-Error-1.0.17029-5.amzn2023.0.2.noarch
perl-Git-2.47.1-1.amzn2023.0.3.noarch
perl-lib-0.65-477.amzn2023.0.7.x86_64

Complete!
[ec2-user@ip-172-31-82-108 ~]$ |

```

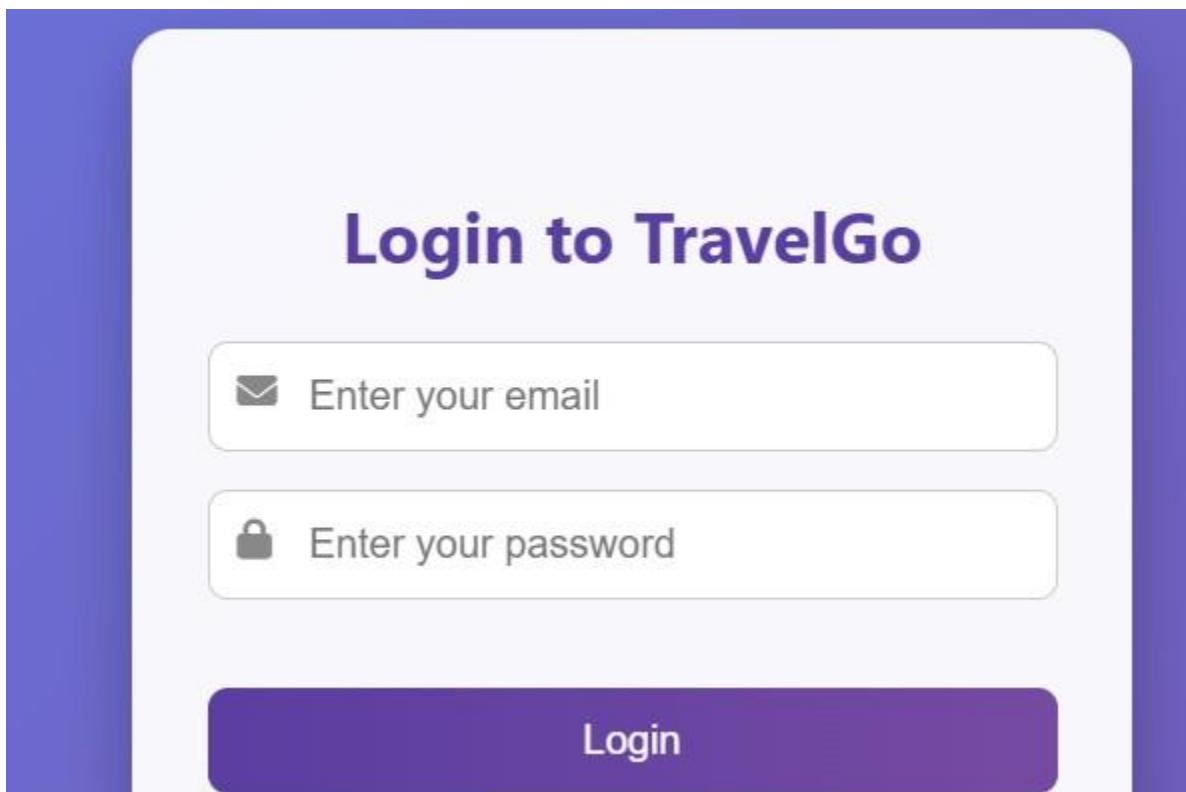
**Access the website through:**

<http://172.31.16.68:5000/>

**Milestone 8: Testing and Deployment**

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

**Login Page:**



## Registration Page:

### Register for TravelGo

 Enter your name

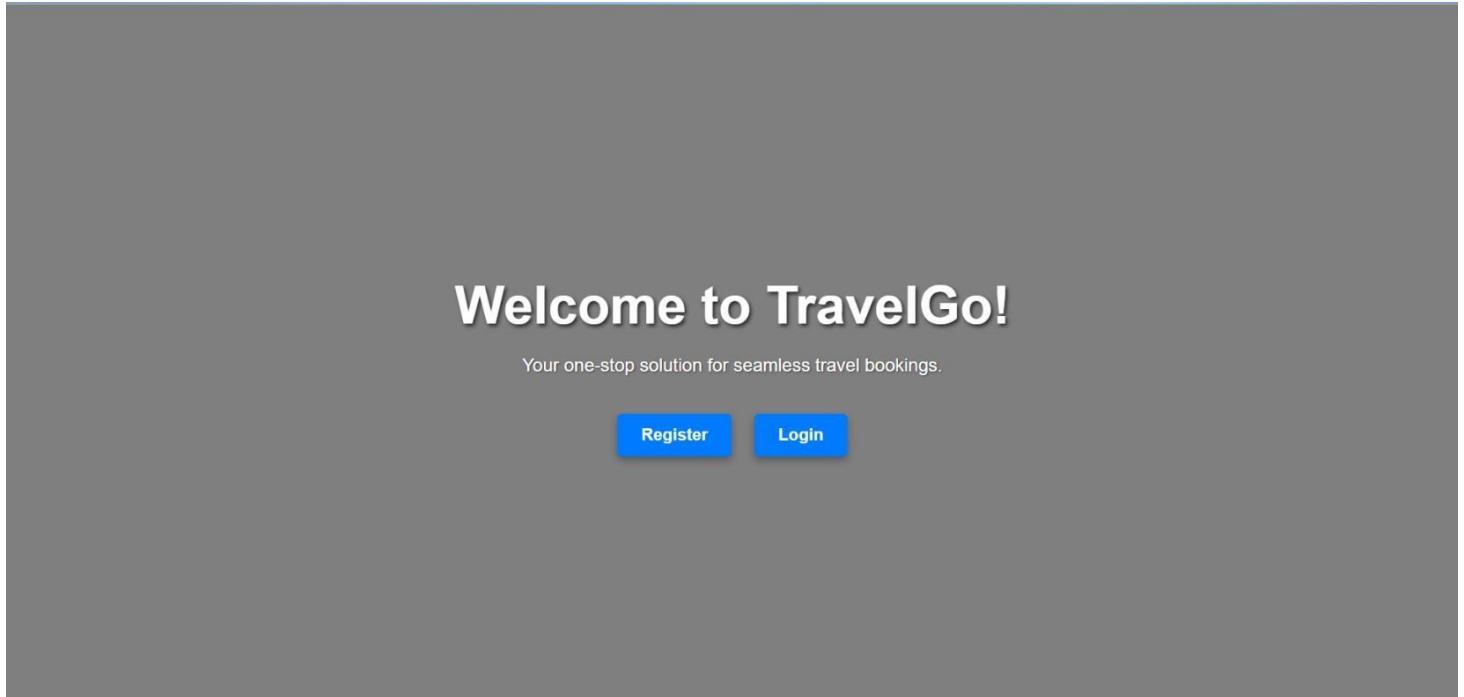
 Enter your email

 Enter your password

 Confirm your password

**Register**

Already have an account? [Login here](#)

**Front Page:-****Confirm Booking Page:**

## **Conclusion:**

The Greenfield University Travel Booking System has been successfully developed and deployed using a robust and scalable cloud architecture. By integrating Flask, AWS EC2, DynamoDB, and SNS, the platform offers a seamless experience for students and faculty to manage academic travel requests and bookings.

This cloud-based solution overcomes the limitations of traditional, manual travel management by automating booking processes, ensuring real-time communication, and enabling efficient tracking of travel data. The use of DynamoDB guarantees fast and secure storage, while AWS SNS keeps all stakeholders informed with instant email notifications.

The system's responsive interface, backed by scalable AWS services, ensures smooth operation even under high usage. From registration and login to travel request submissions and status tracking, all functionalities have been thoroughly tested and optimized.

In summary, this platform significantly improves the efficiency and transparency of academic travel planning at Greenfield University, demonstrating the power of modern cloud technologies in solving real-world administrative challenges.

**Experience Gained :-**

Working on the Greenfield University Travel Booking System project provided valuable hands-on experience across multiple areas of cloud computing, backend development, and full-stack deployment. The following skills and insights were gained:

 **Cloud Services Integration**

Gained practical experience in setting up and integrating AWS services like **EC2**, **DynamoDB**, and **SNS**, learning how these services interact to support real-world web applications.

 **Flask Web Development**

Built a scalable backend using **Flask**, implementing routing, user authentication, and session management while connecting it to a cloud-based NoSQL database.

 **DynamoDB Database Management**

Designed and managed DynamoDB tables, learned how to perform CRUD operations using **boto3**, and understood best practices for data modeling in NoSQL environments.

 **Real-time Notification System**

Used **AWS SNS** to send email alerts, improving user interaction and system responsiveness. This reinforced the importance of real-time communication in modern applications.

 **Deployment on AWS EC2**

Learned how to launch, configure, and secure an **EC2 instance**, and successfully deployed a Flask app to a live production environment.

 **IAM Roles and Security Best Practices**

Understood the importance of **IAM policies and permissions** by configuring roles for EC2 to securely interact with other AWS services.

 **Version Control and GitHub Integration**



Managed project code using **Git**, enabling better team collaboration and version tracking through GitHub.

#### **Problem Solving and Debugging**

Encountered and resolved various technical challenges during development and deployment, enhancing debugging and troubleshooting skills.

#### **Project Management and Documentation**

Documented the entire workflow, from architecture to deployment, ensuring the project can be maintained, scaled, or enhanced in the future.