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**

<u>Cloud Architecture Overview :</u>

- **Frontend: HTML templates rendered by Flask (with routes for registration, login, booking, etc.)**

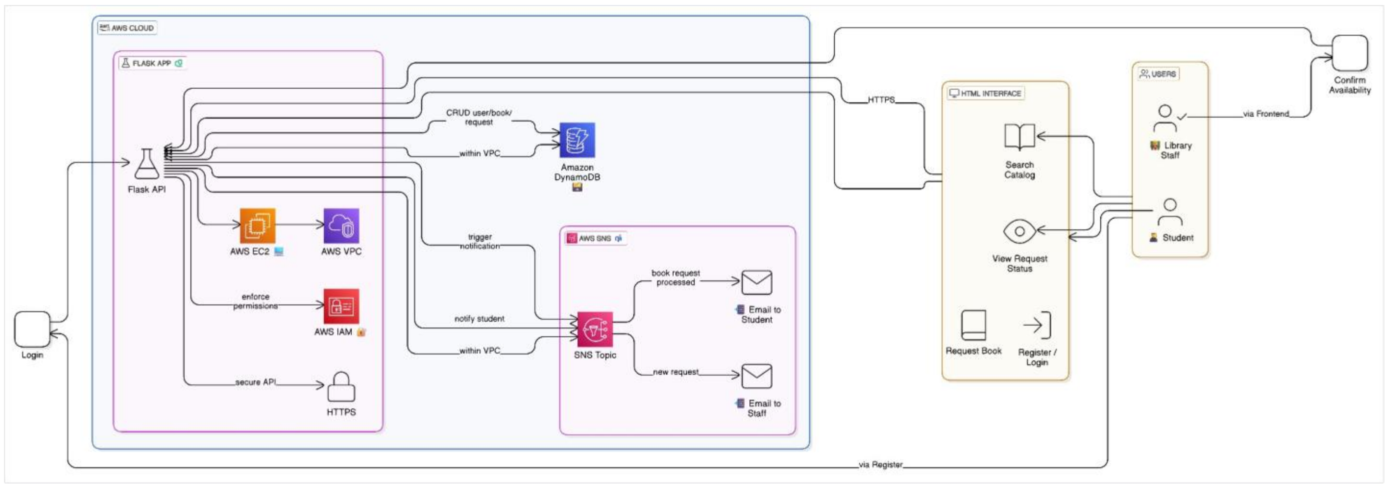
- **Backend: Flask (Python) application hosted on EC2**

- **Database: AWS DynamoDB (storing users and travel bookings)**

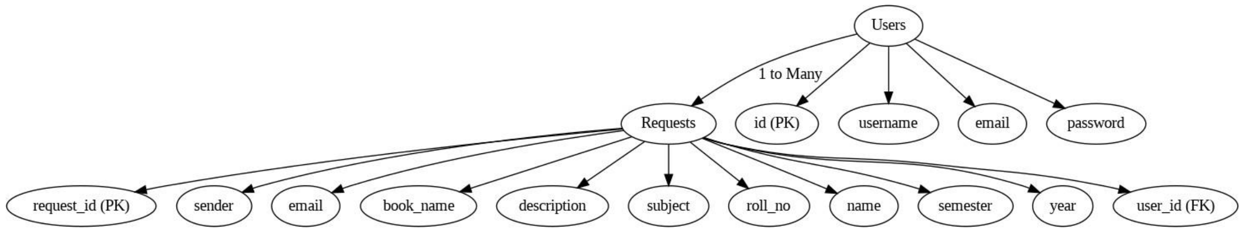
- **Notifications: AWS SNS (email alerts for booking confirmations)**

- **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

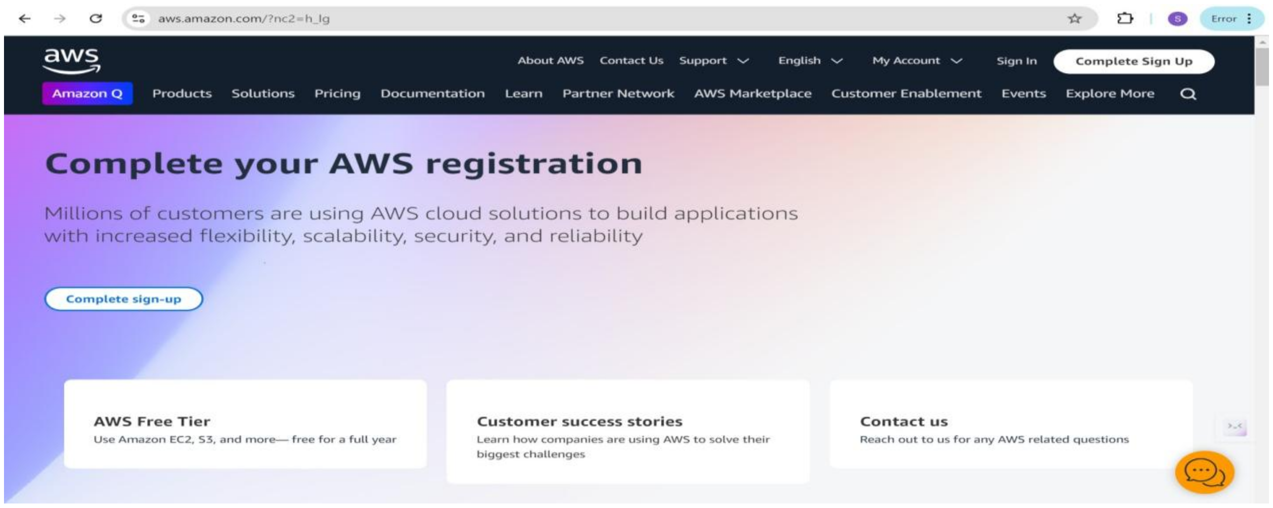
- **Test user registration, login, booking functionality, and email alerts**

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.**



Complete your AWS registration

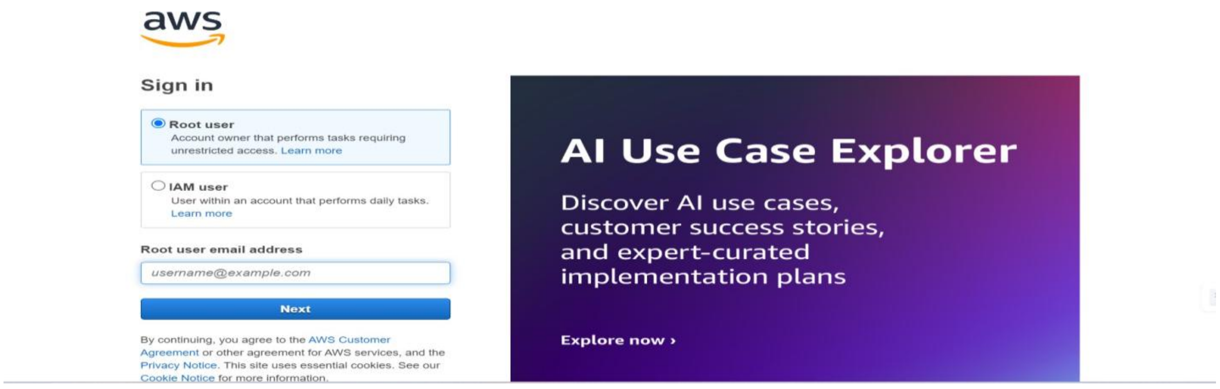
Millions of customers are using AWS cloud solutions to build applications with increased flexibility, scalability, security, and reliability

AWS Free Tier
Use Amazon EC2, S3, and more— free for a full year

Customer success stories
Learn how companies are using AWS to solve their biggest challenges

Contact us
Reach out to us for any AWS related questions

- **Activity 1.2: Log in to the AWS Management Console**
  - **After setting up your account, log in to the AWS Management Console.**



Sign in

Root user
Account owner that performs tasks requiring unrestricted access. Learn more

IAM user
User within an account that performs daily tasks. Learn more

Root user email address
username@example.com

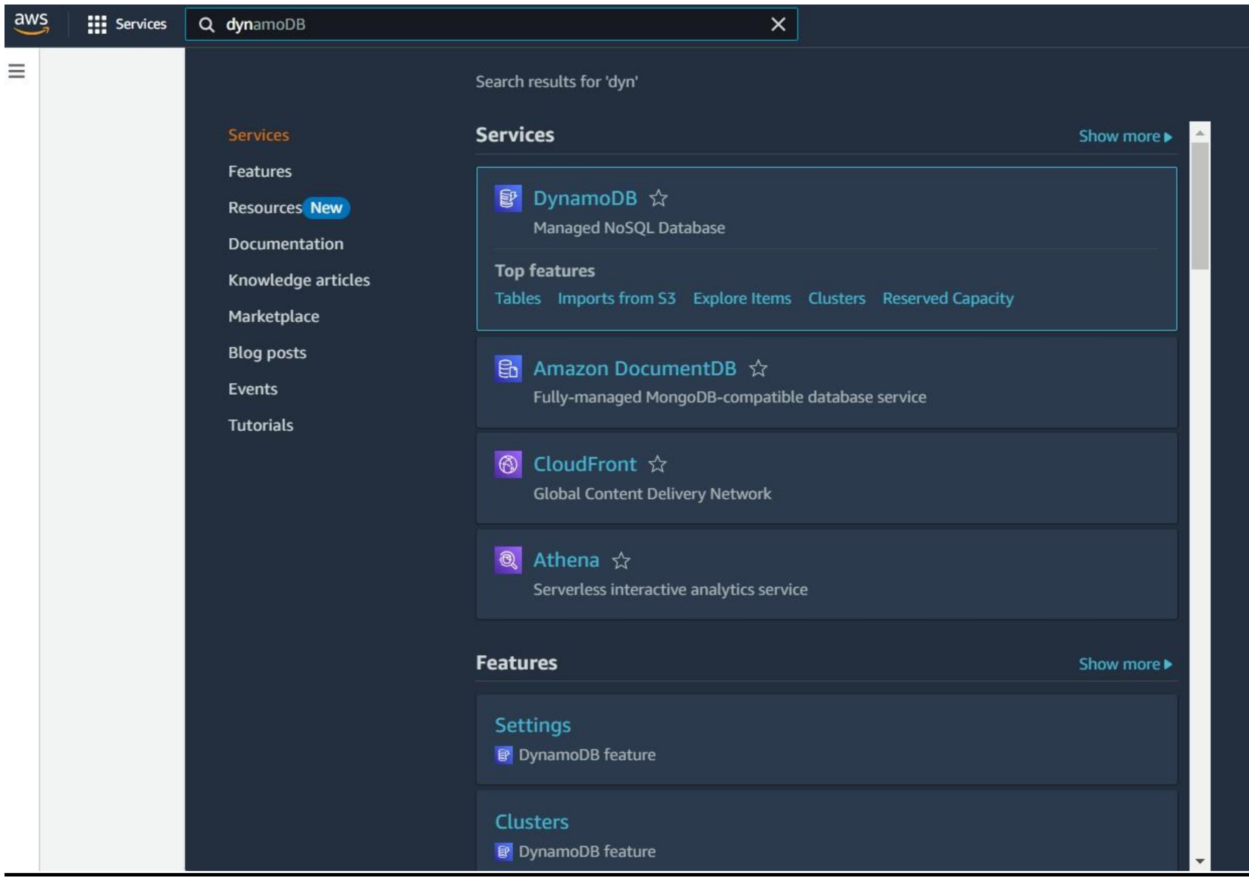
By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our Cookie Notice for more information.

AI Use Case Explorer

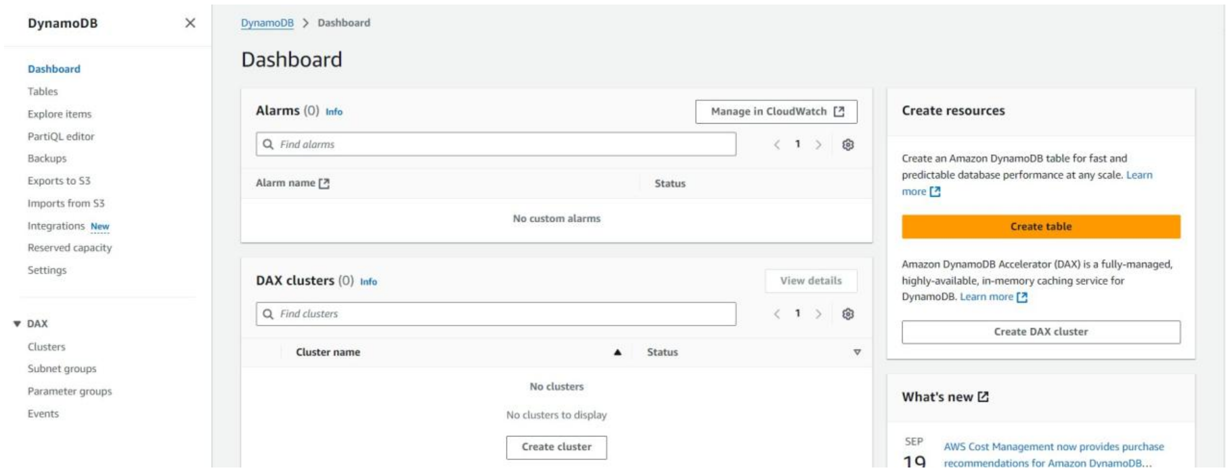
Discover AI use cases, customer success stories, and expert-curated implementation plans

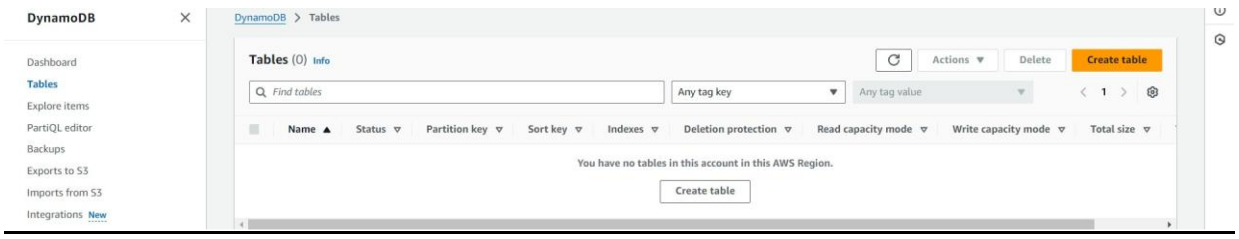
Milestone 2: DynamoDB Database Creation and Setup

- **Activity 2.1:Navigate to the DynamoDB**
    - **In the AWS Console, navigate to DynamoDB and click on create tables.**



-

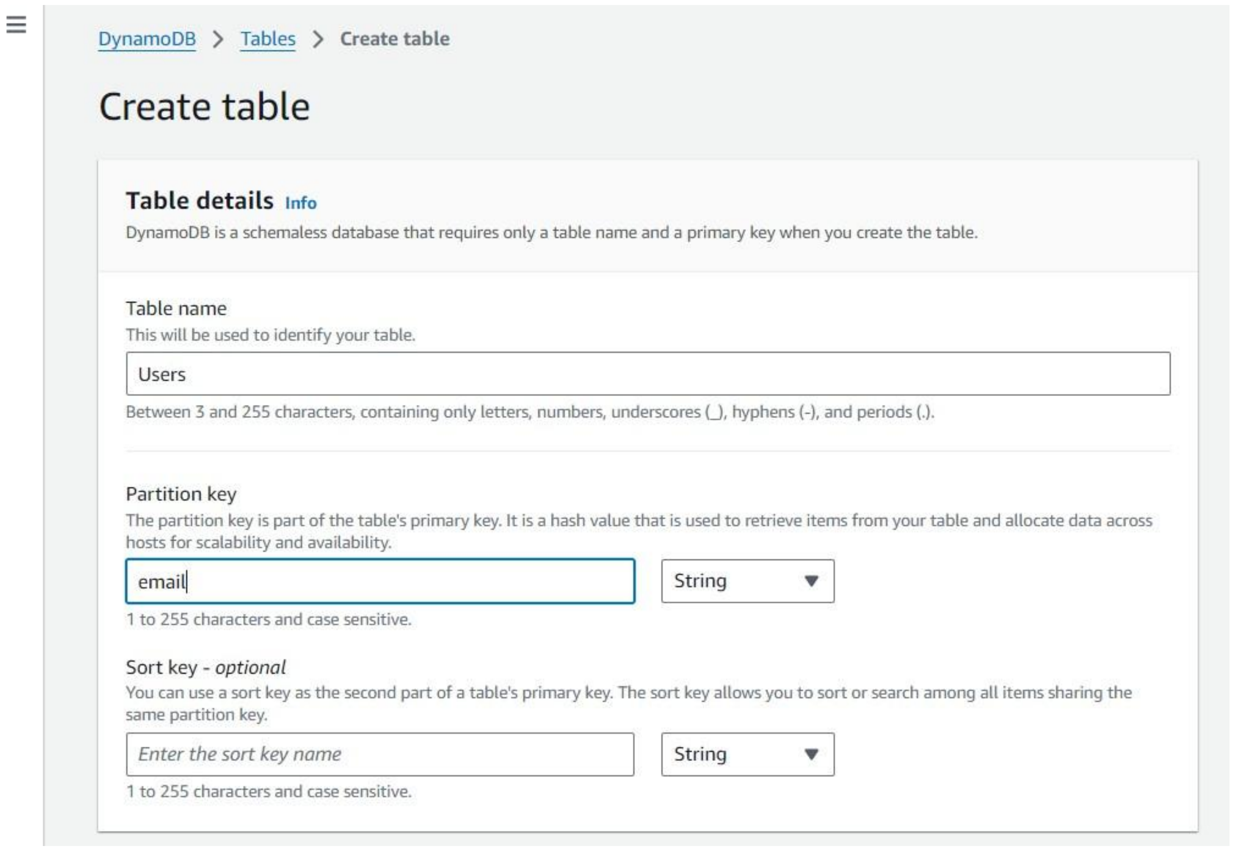




You have no tables in this account in this AWS Region.

Create table

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



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: Users

This will be used to identify your table.

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

Partition key: email — String

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

1 to 255 characters and case sensitive.

Sort key - *optional*: *Enter the sort key name* — String

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

1 to 255 characters and case sensitive.

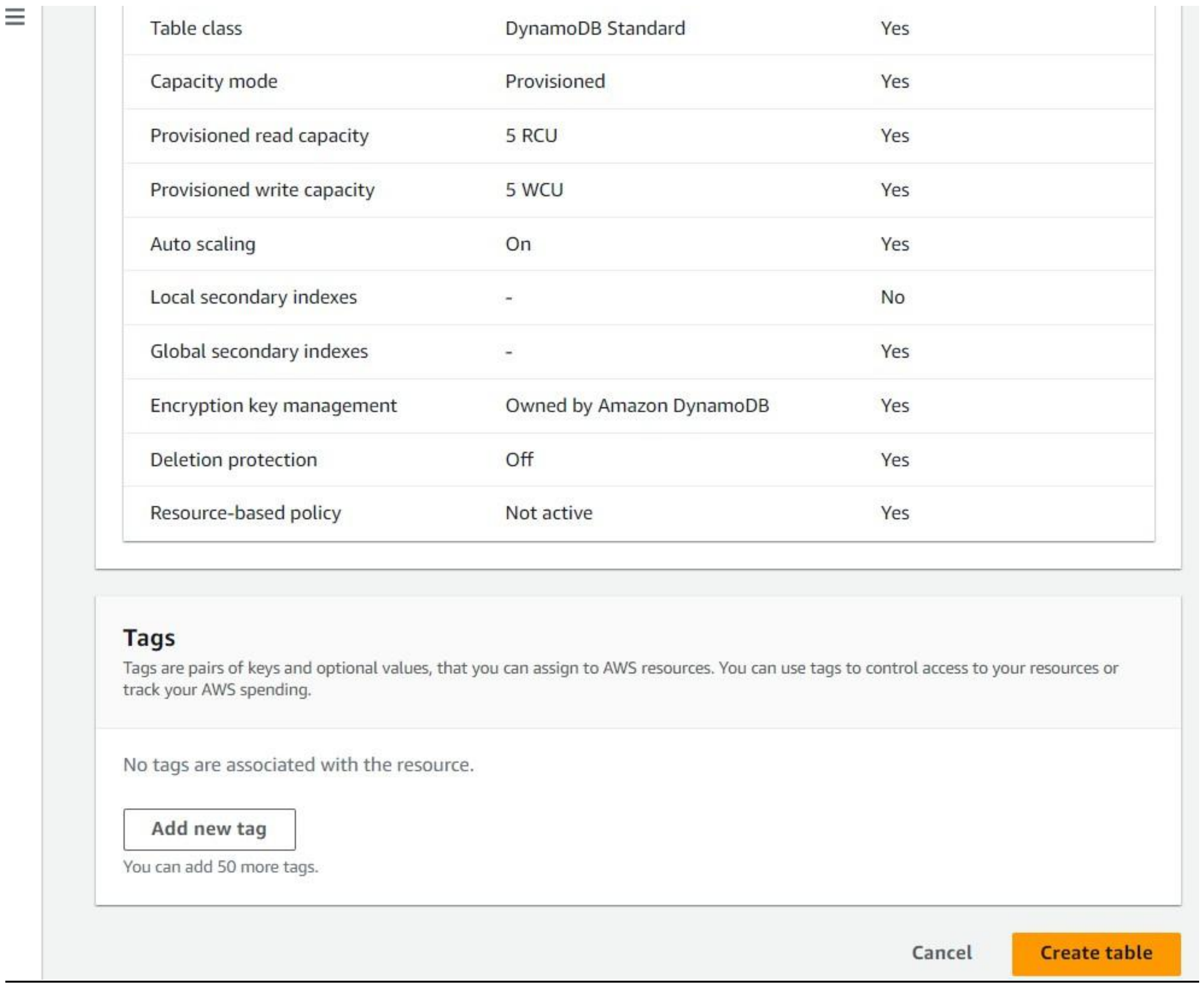


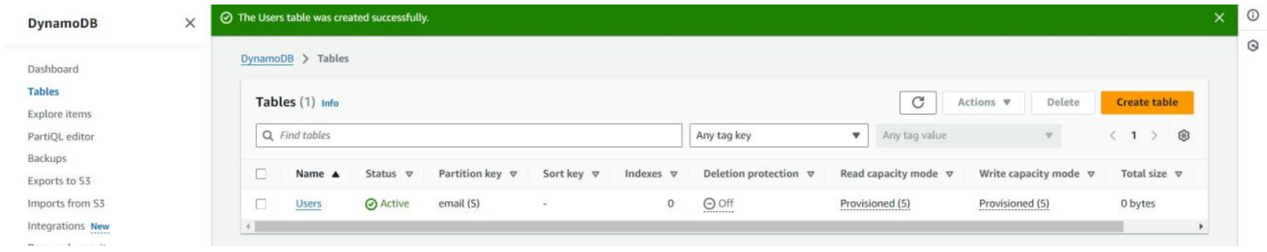
Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based 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.

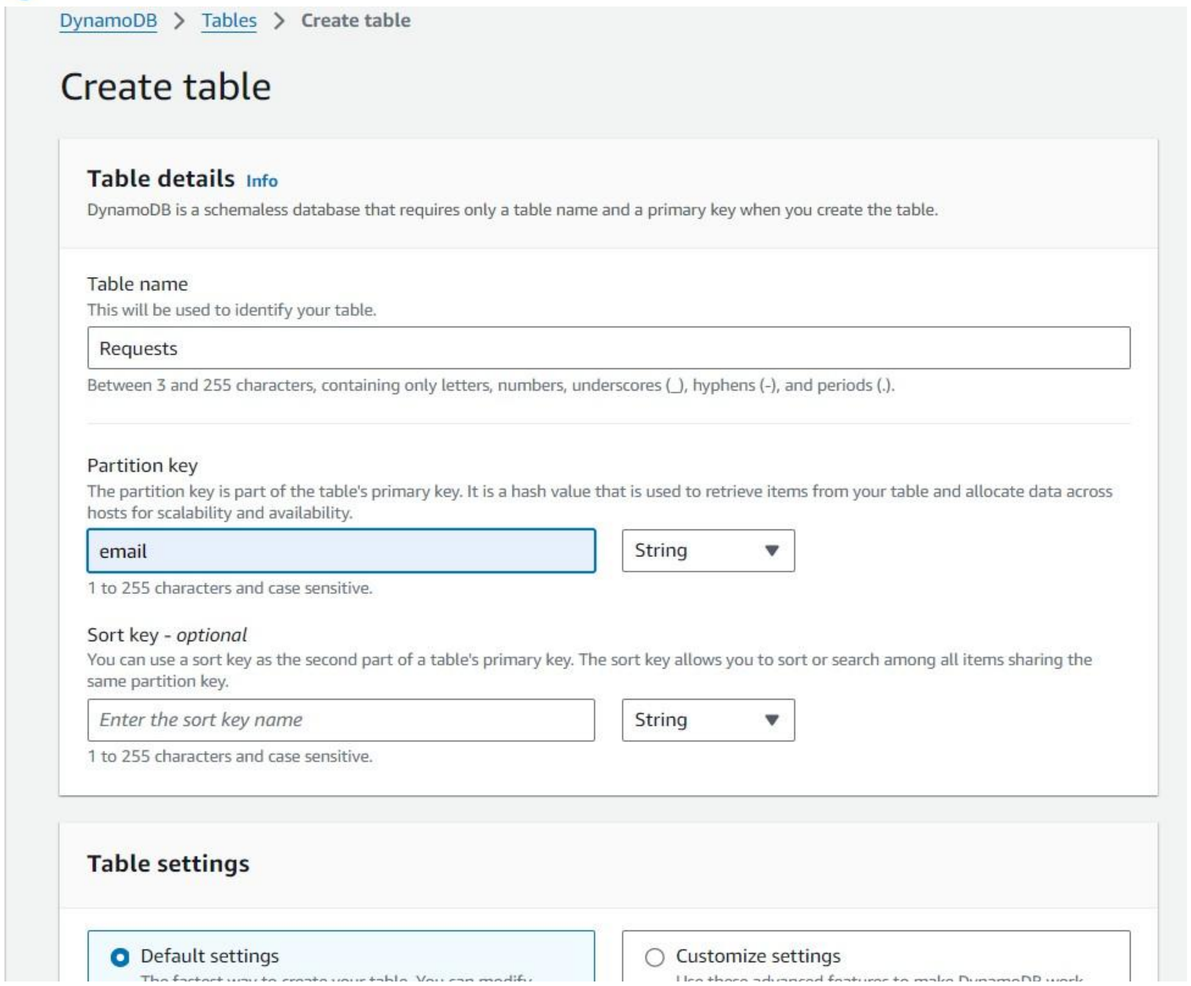
You can add 50 more tags.



The Users table was created successfully.

	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

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



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.

Requests

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.

email — String

1 to 255 characters and case sensitive.

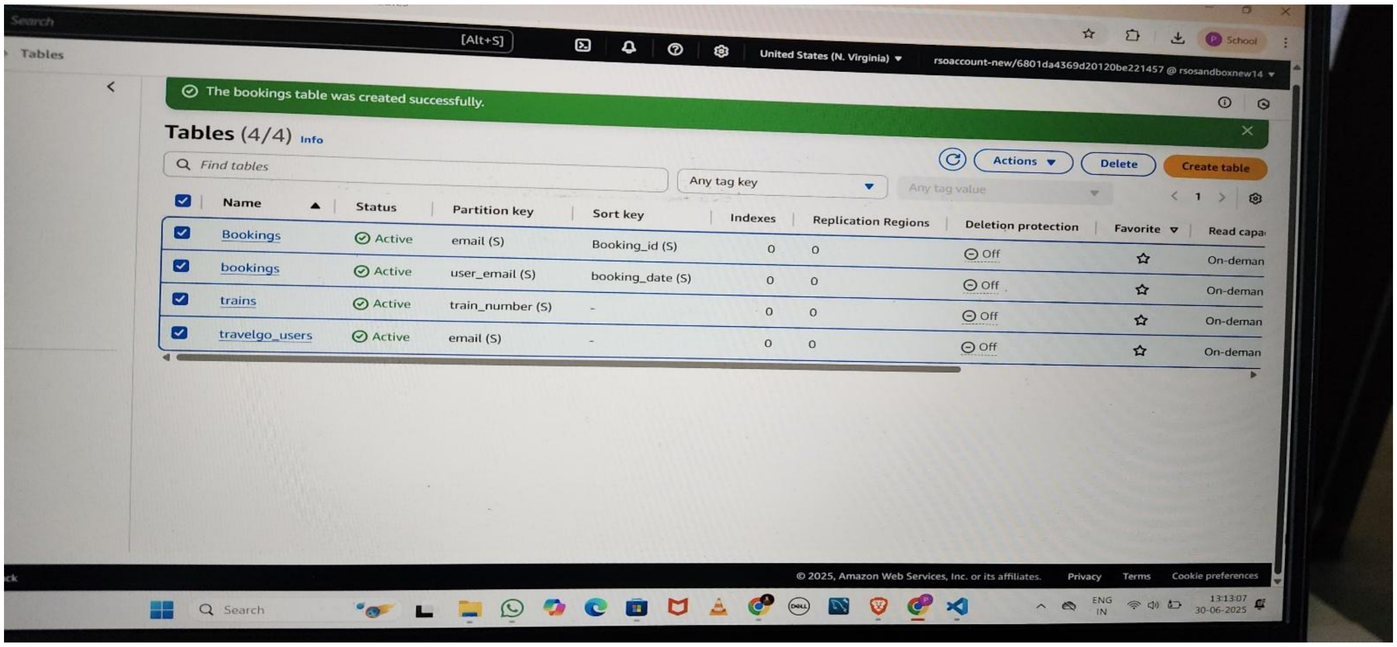
Sort key - *optional*
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name — String

1 to 255 characters and case sensitive.

Table settings

[✓ Default settings] Customize settings



The bookings table was created successfully.

Tables (4/4) Info

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

Tables
[Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew14
Find tables
Any tag key
Any tag value
Actions
Delete
Create table
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
13:13:07
30-06-2025

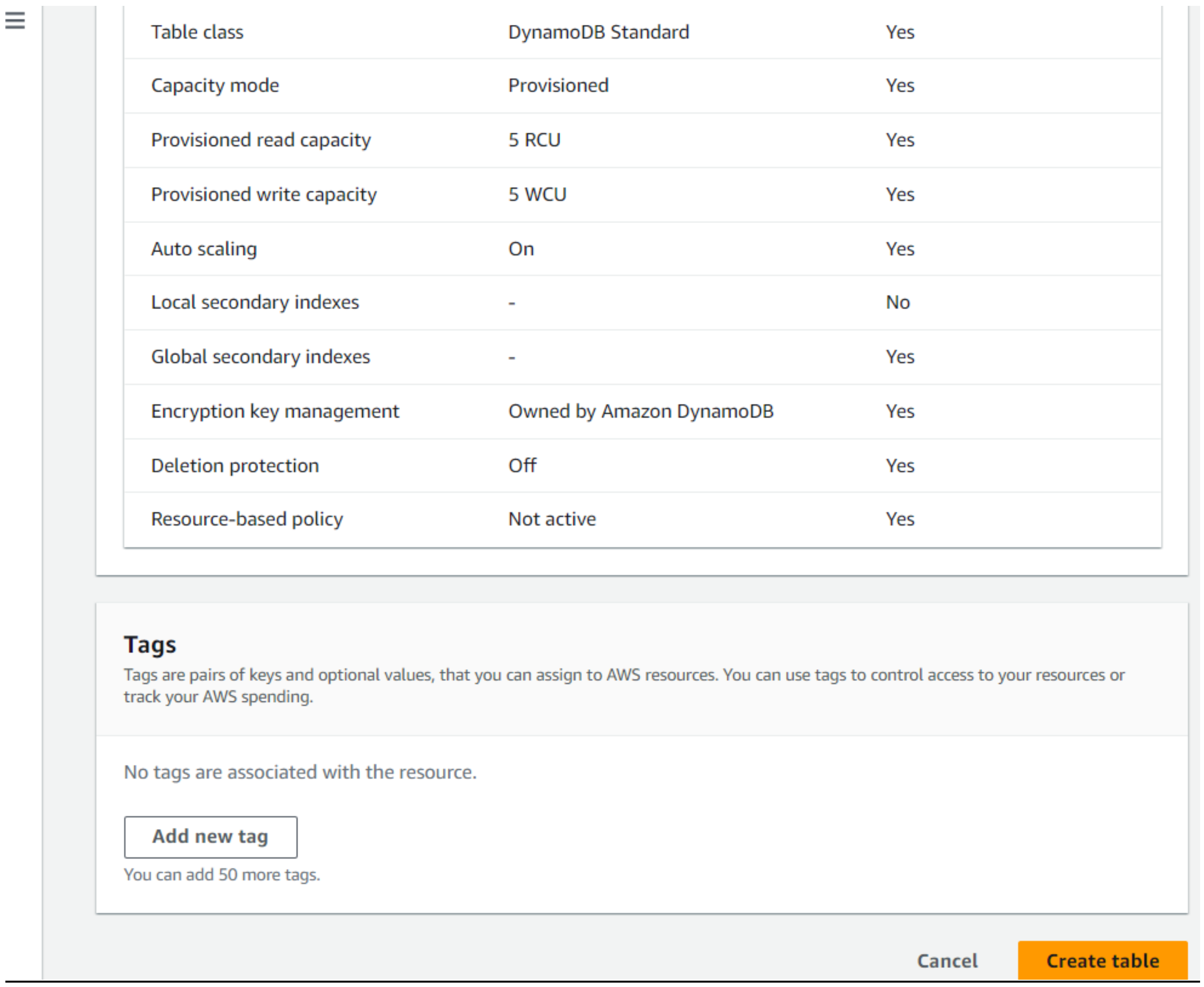


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based 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.

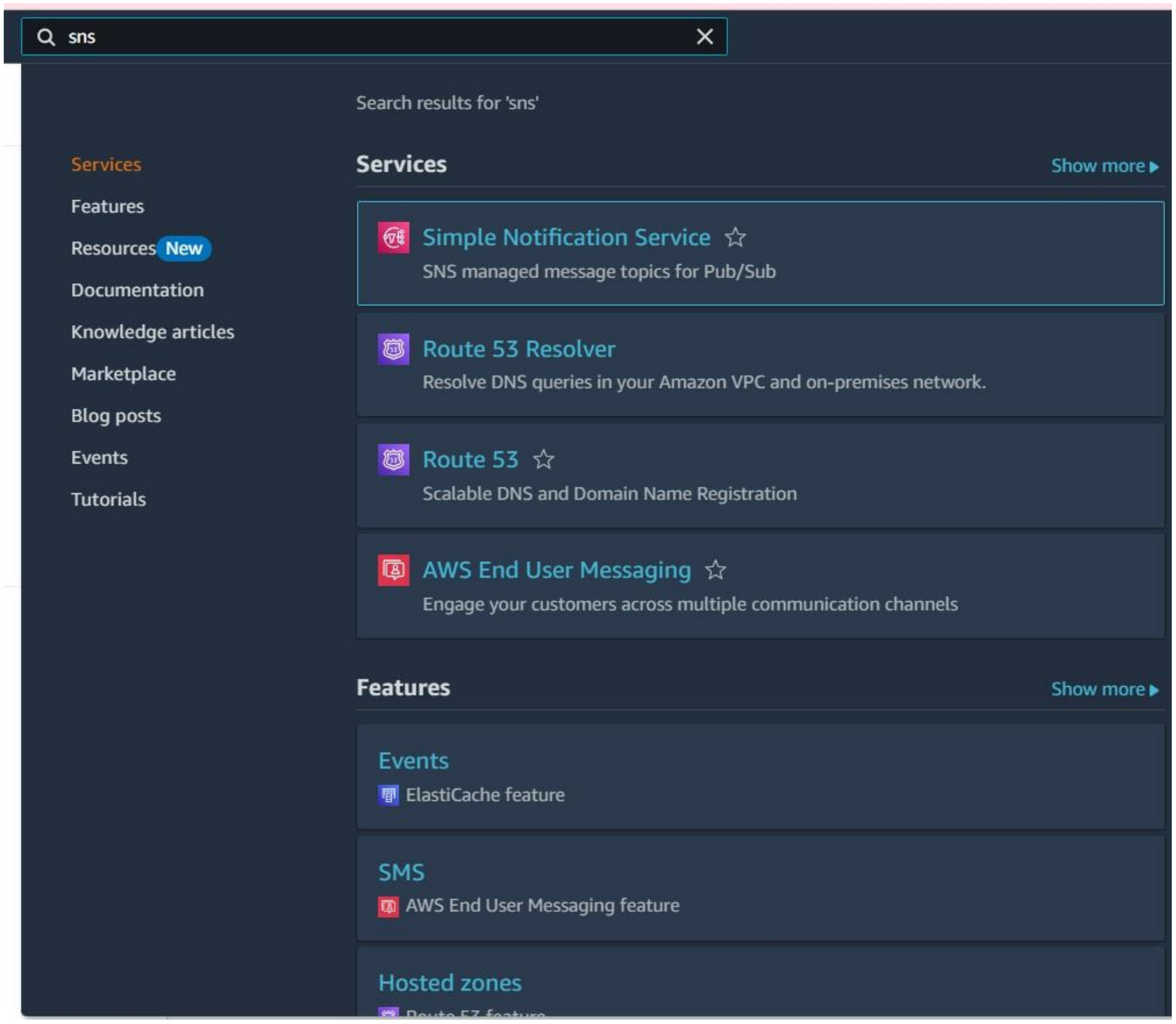
No tags are associated with the resource.

You can add 50 more tags.

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.**



sns

Search results for 'sns'

Services — Show more

Simple Notification Service
SNS managed message topics for Pub/Sub

Route 53 Resolver
Resolve DNS queries in your Amazon VPC and on-premises network.

Route 53
Scalable DNS and Domain Name Registration

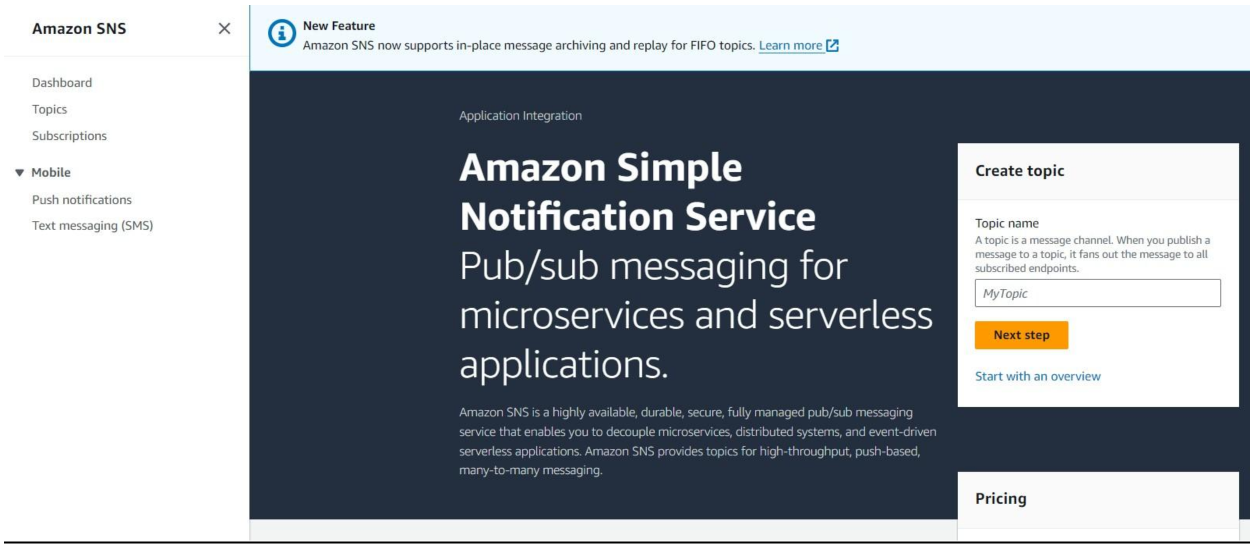
AWS End User Messaging
Engage your customers across multiple communication channels

Features — Show more

Events
ElastiCache feature

SMS
AWS End User Messaging feature

Hosted zones



Amazon SNS

Dashboard
Topics
Subscriptions
▼ Mobile
Push notifications
Text messaging (SMS)

New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more

Application Integration

Amazon Simple Notification Service
Pub/sub messaging for microservices and serverless applications.

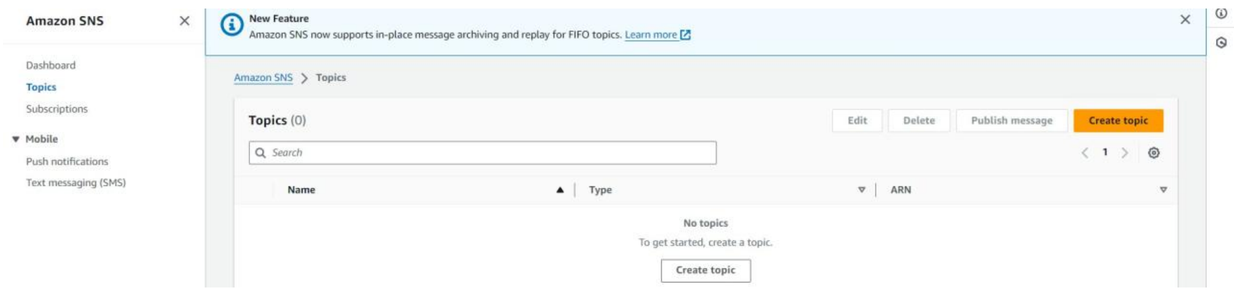
Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.

Create topic

Topic name
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.
MyTopic
Next step
Start with an overview

Pricing

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



Amazon SNS

Dashboard
Topics
Subscriptions
▼ Mobile
Push notifications
Text messaging (SMS)

New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more

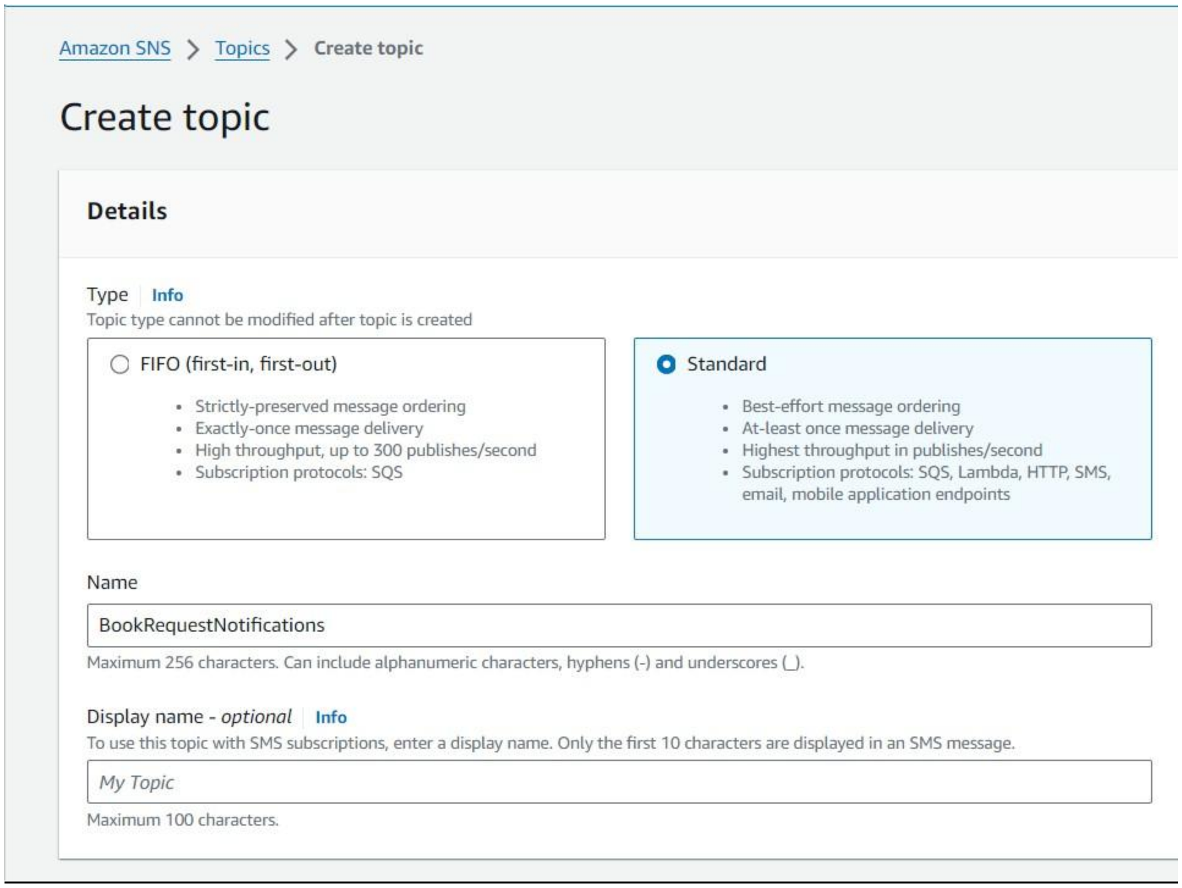
Amazon SNS > Topics

Topics (0)
Edit | Delete | Publish message | Create topic
Search

Name	Type	ARN

No topics
To get started, create a topic.
Create topic

- **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

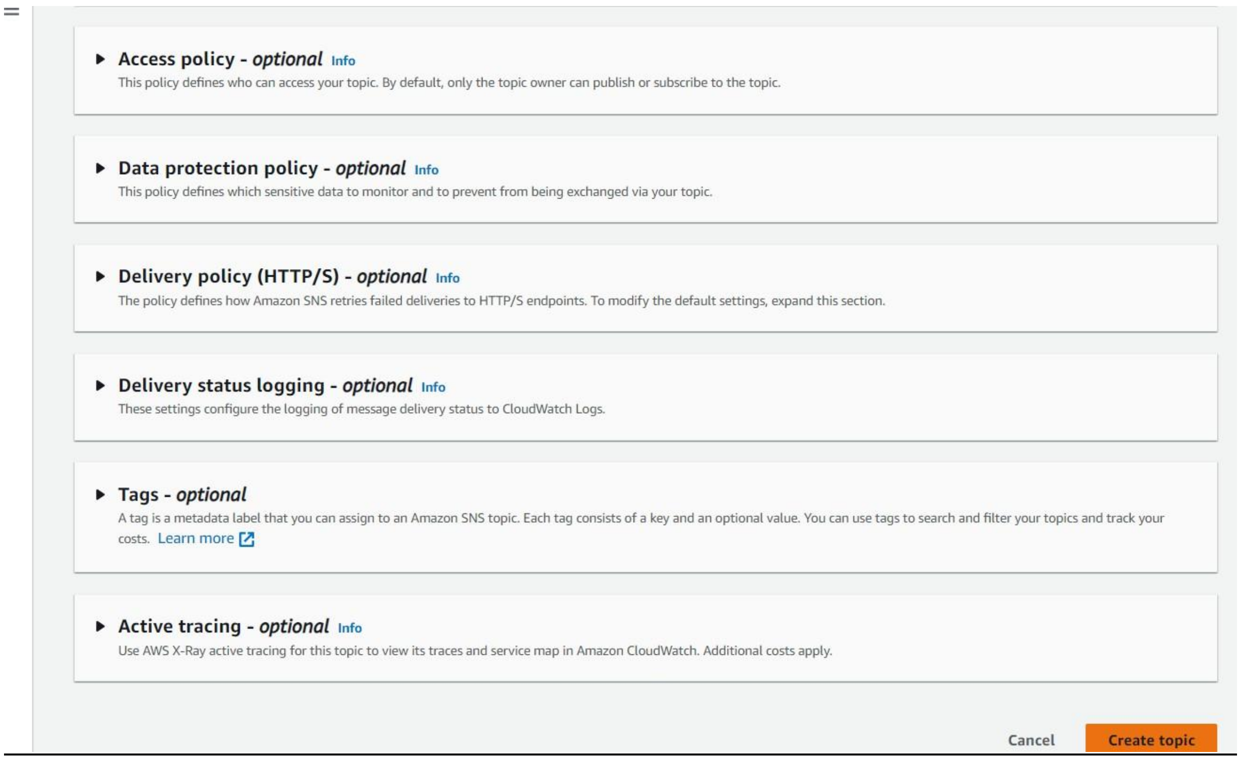
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

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.

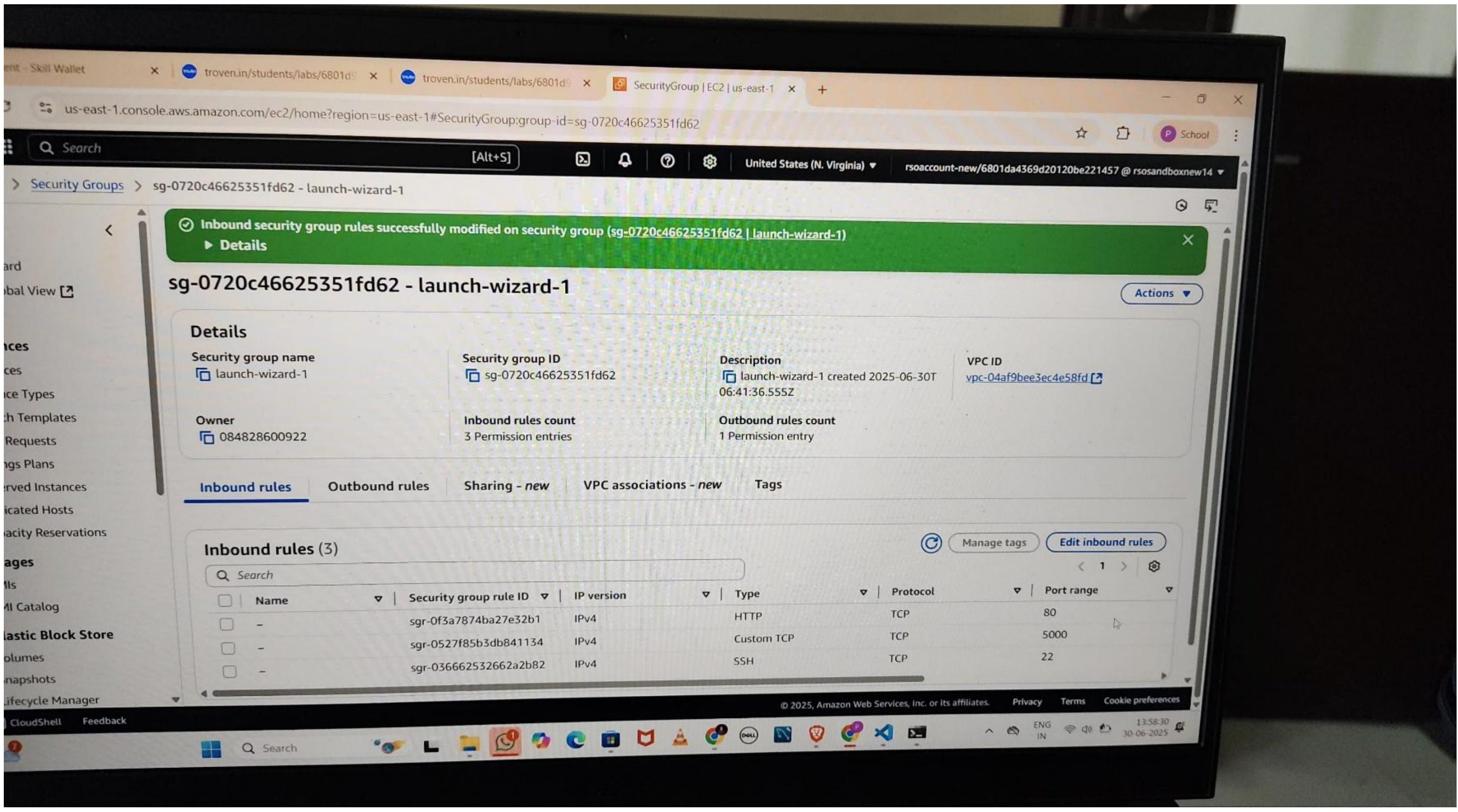
▸ **Delivery policy (HTTP/S) - *optional*** Info
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

▸ **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.

- **Configure the SNS topic and note down the Topic ARN.**



us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#SecurityGroup:group-id=sg-0720c46625351fd62

United States (N. Virginia) rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew14

Security Groups > sg-0720c46625351fd62 - launch-wizard-1

Inbound security group rules successfully modified on security group (sg-0720c46625351fd62 | launch-wizard-1)

sg-0720c46625351fd62 - launch-wizard-1

Details

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

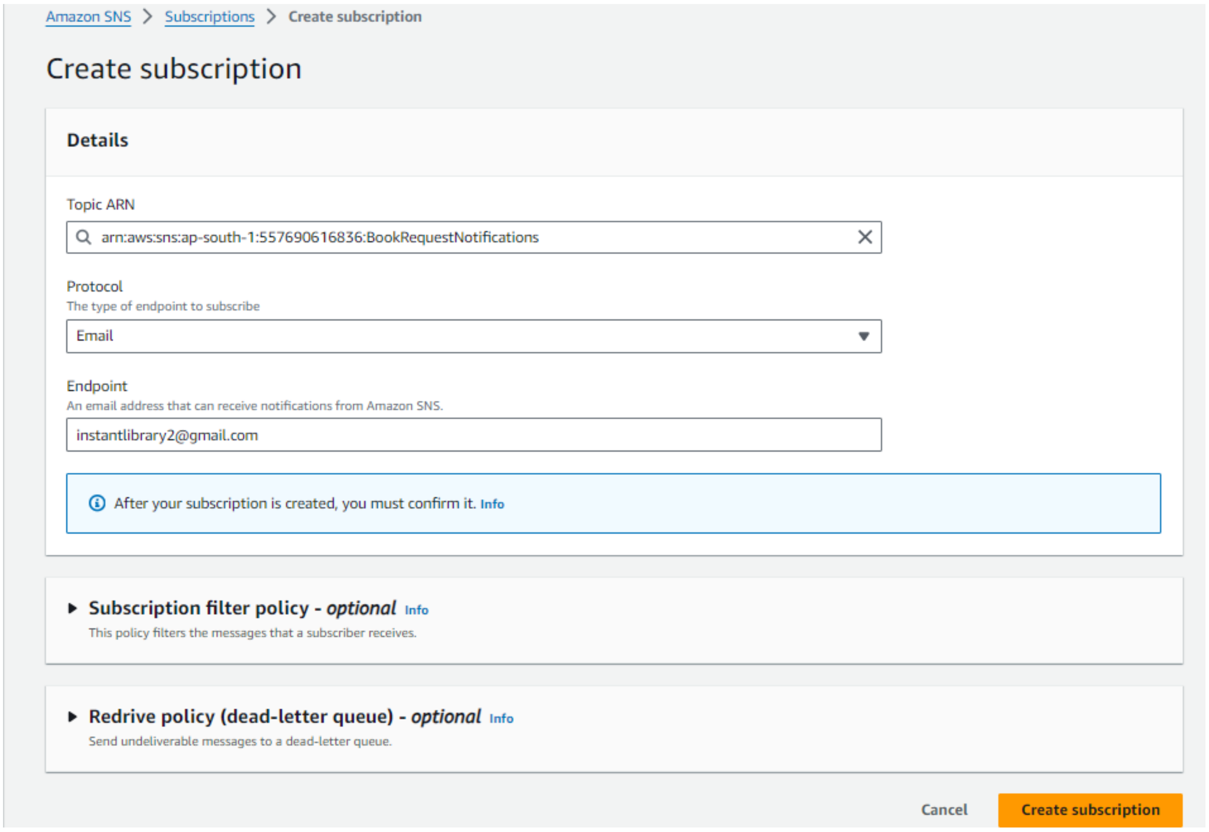
Inbound rules | Outbound rules | Sharing - *new* | VPC associations - *new* | Tags

Inbound rules (3)

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

Protocol
The type of endpoint to subscribe
Email

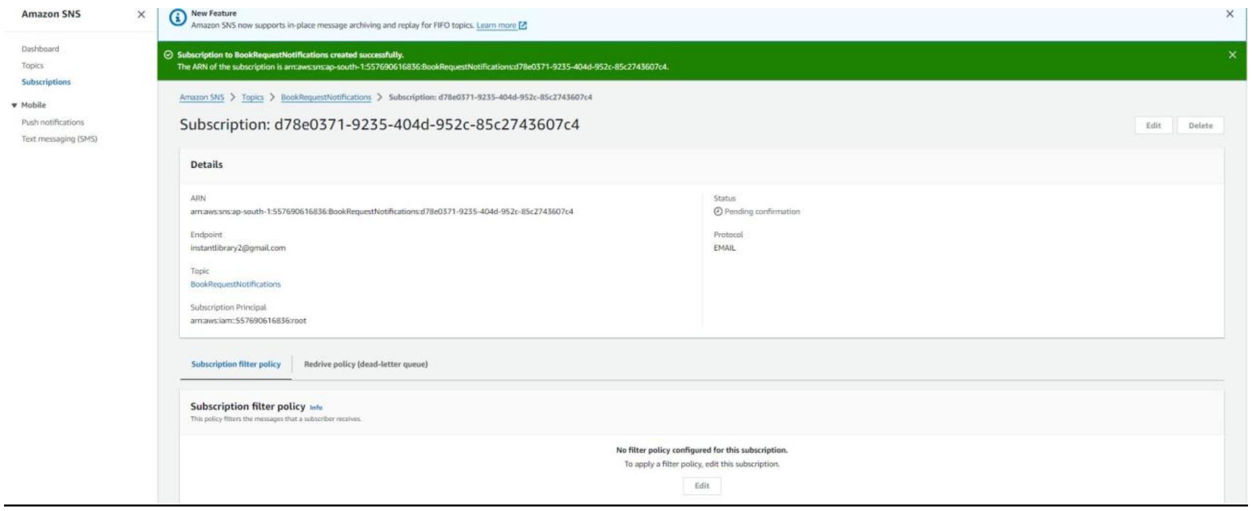
Endpoint
An email address that can receive notifications from Amazon SNS.
instantlibrary2@gmail.com

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


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.

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

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

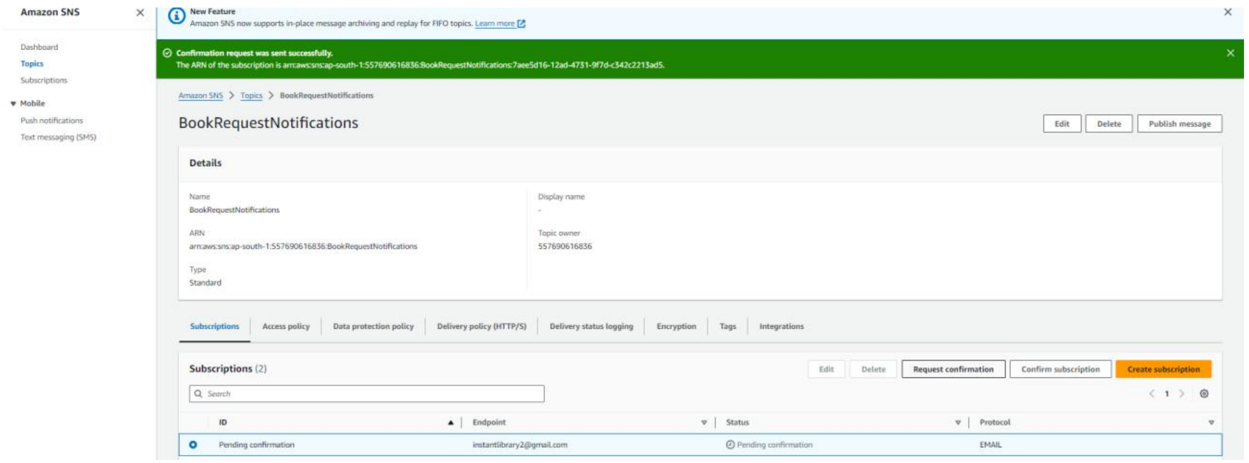
Protocol
EMAIL

Subscription filter policy
This policy filters the messages that a subscriber receives.

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

Amazon SNS
Dashboard
Topics
Subscriptions
Mobile
Push notifications
Text messaging (SMS)
Amazon SNS > Topics > BookRequestNotifications > Subscription: d78e0371-9235-404d-952c-85c2743607c4
Edit
Delete
Subscription filter policy
Redrive policy (dead-letter queue)
Info
Edit

- **<u>After subscription request for the mail confirmation</u>**


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:7aee5d16-12ad-4731-9f7d-c342c2213ad5.

BookRequestNotifications

Details

Name
BookRequestNotifications

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

Type
Standard

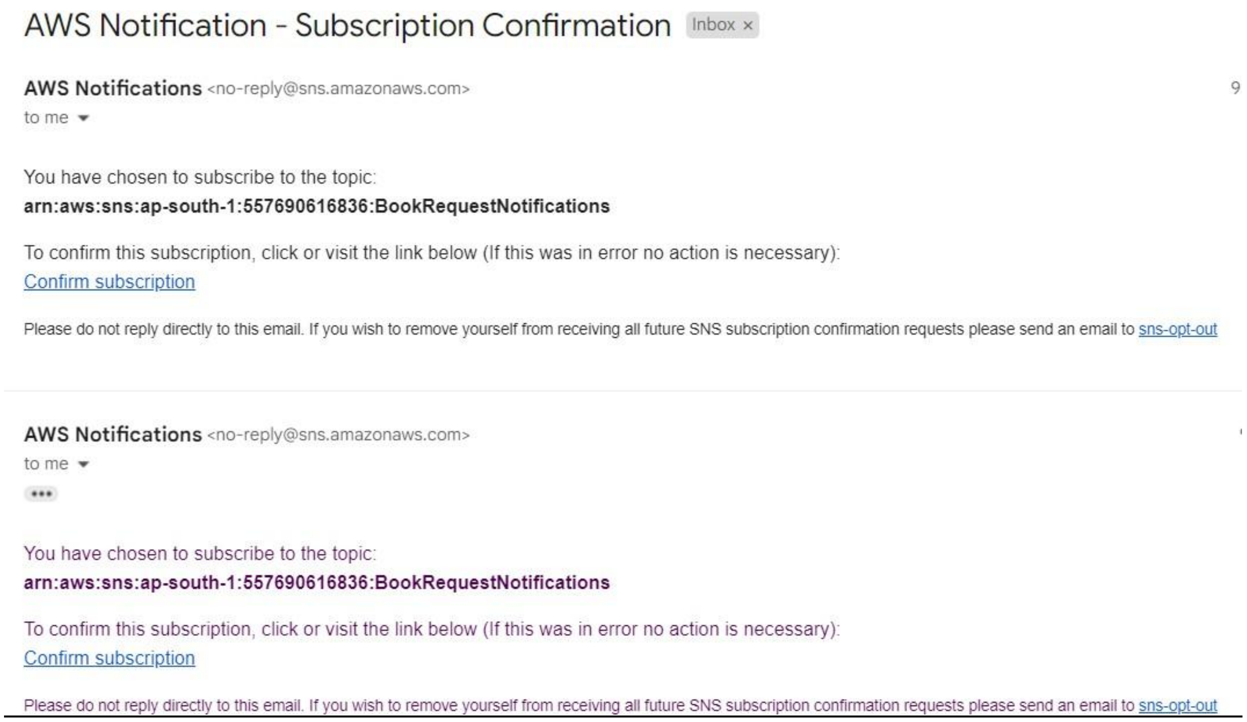
Display name
-

Topic owner
557690616836

Subscriptions (2)

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

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



AWS Notification - Subscription Confirmation Inbox ×

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

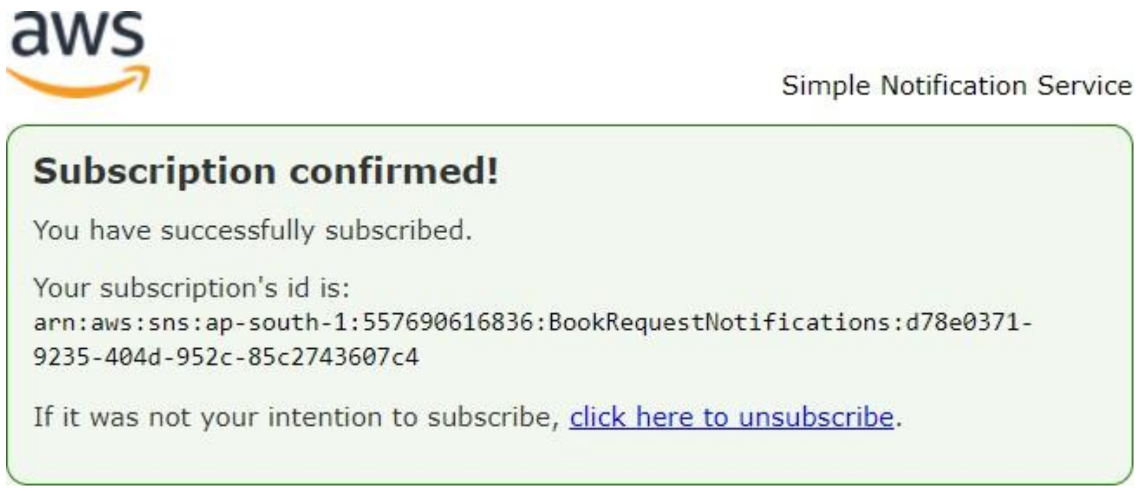
Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to sns-opt-out

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

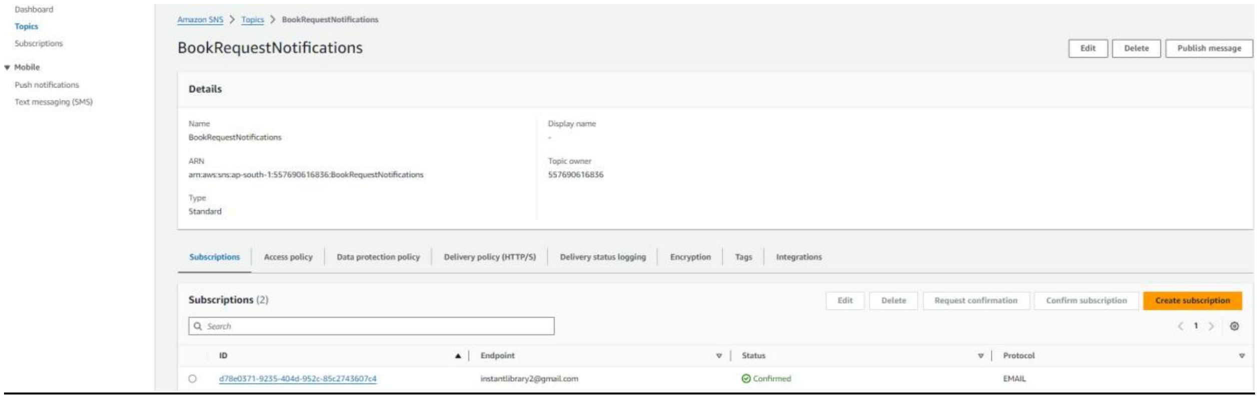
Subscription confirmed!

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.

- <u>**Successfully done with the SNS mail subscription and setup, now store the ARN link.**</u>



BookRequestNotifications

Details

Name
BookRequestNotifications

Display name
-

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

Topic owner
557690616836

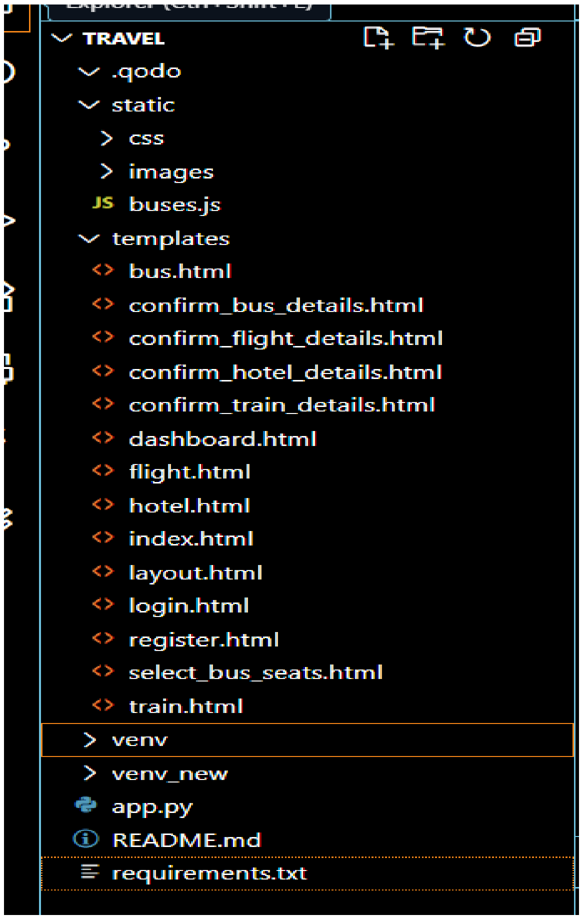
Type
Standard

Subscriptions (2)

	ID	Endpoint	Status	Protocol
○	d78e0371-9235-404d-952c-85c2743607c4	instantlibrary2@gmail.com	Confirmed	EMAIL

Milestone 4:Backend Development and Application Setup

- **Activity 4.1: Develop the backend using Flask**
    - **File Explorer Structure**

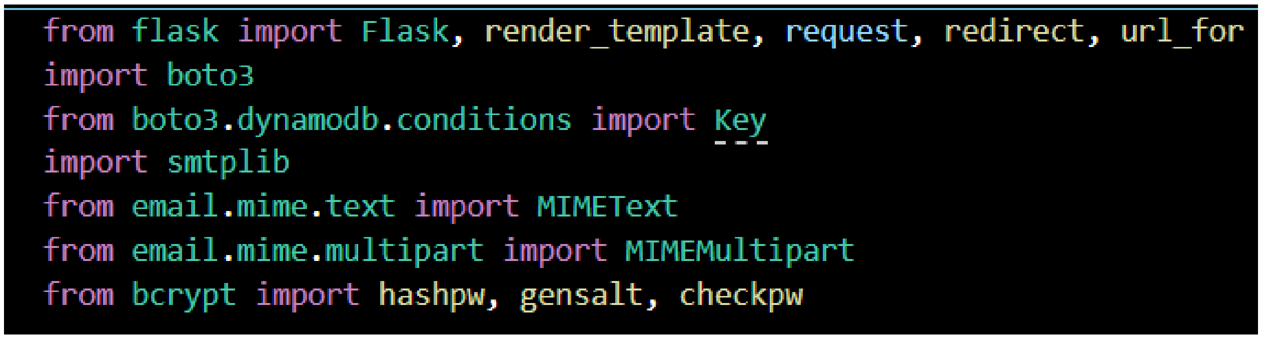


```
TRAVEL
  .qodo
  static
    css
    images
    buses.js
  templates
    bus.html
    confirm_bus_details.html
    confirm_flight_details.html
    confirm_hotel_details.html
    confirm_train_details.html
    dashboard.html
    flight.html
    hotel.html
    index.html
    layout.html
    login.html
    register.html
    select_bus_seats.html
    train.html
  venv
  venv_new
  app.py
  README.md
  requirements.txt
```

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).

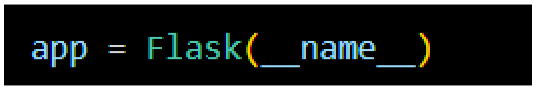
Description of the code :

- **Flask App Initialization**



```python
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
```

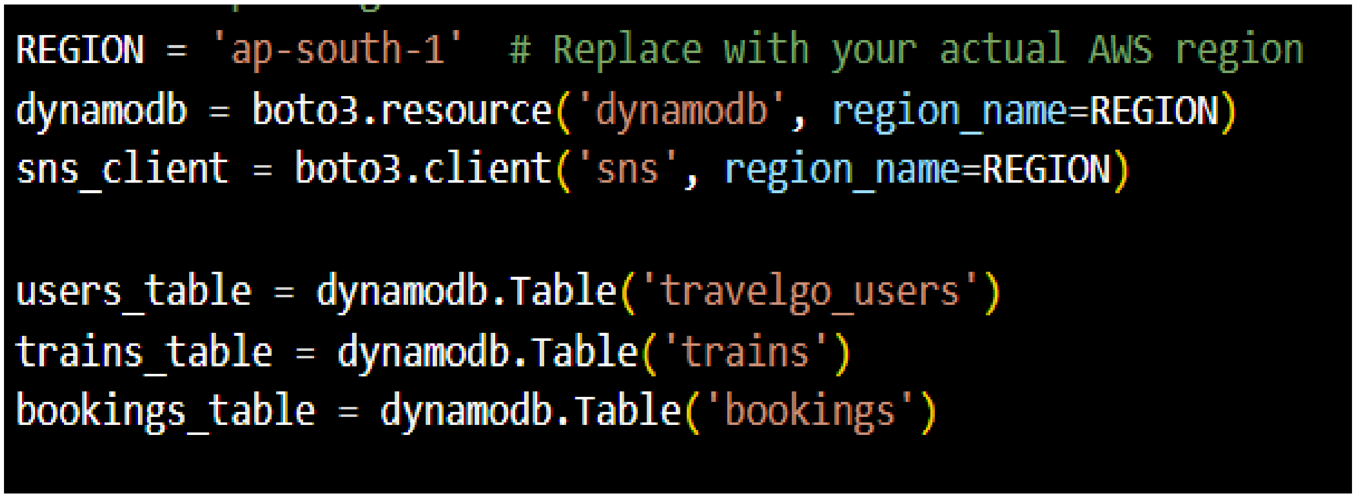
Description: import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification



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

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

- **Dynamodb Setup:**



```python
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')
```

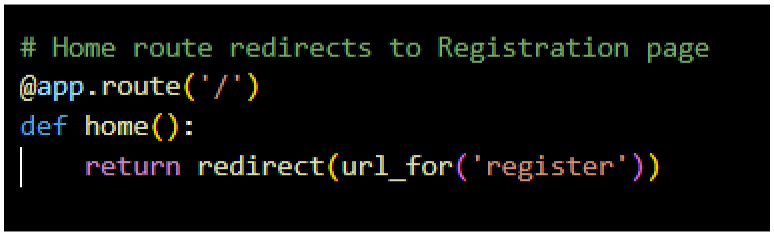
Description: initialize the DynamoDB resource for the ap-south-1 region and set up access to the Users and Requests tables for storing user details and book requests.

- **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.

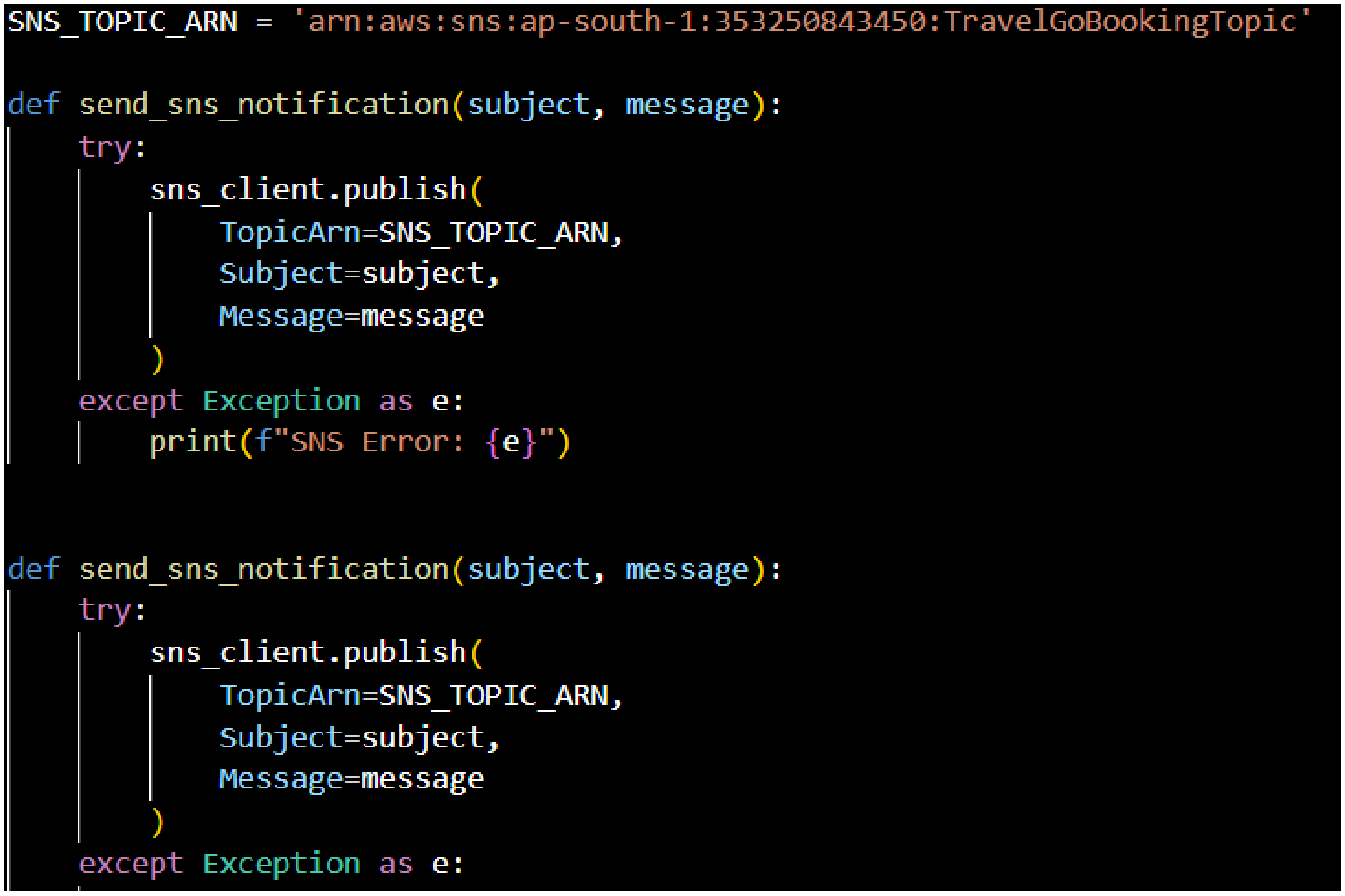
- **Routes for Web Pages**

    - **Home Route:**



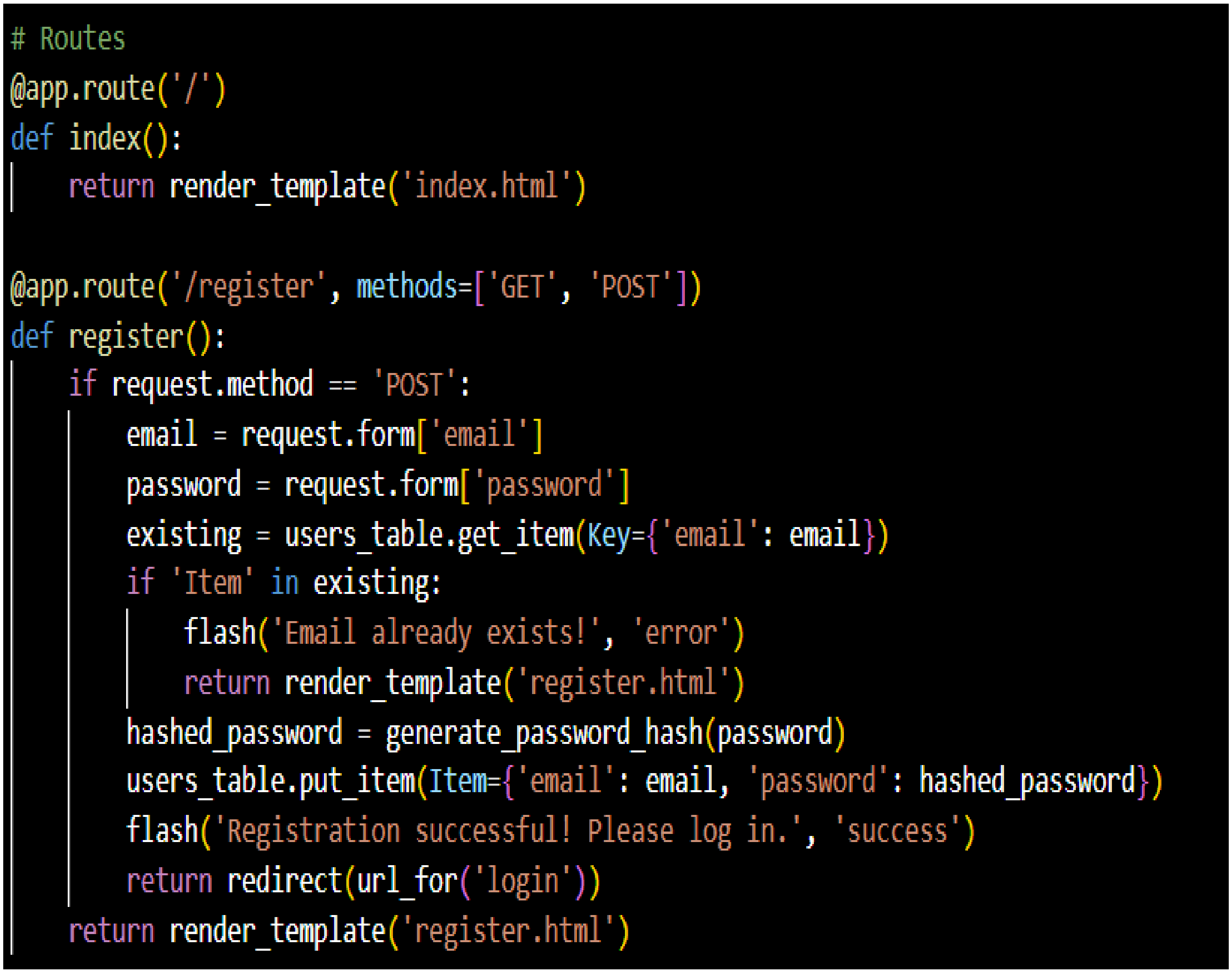
```python
# Home route redirects to Registration page
@app.route('/')
def home():
    return redirect(url_for('register'))
```

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



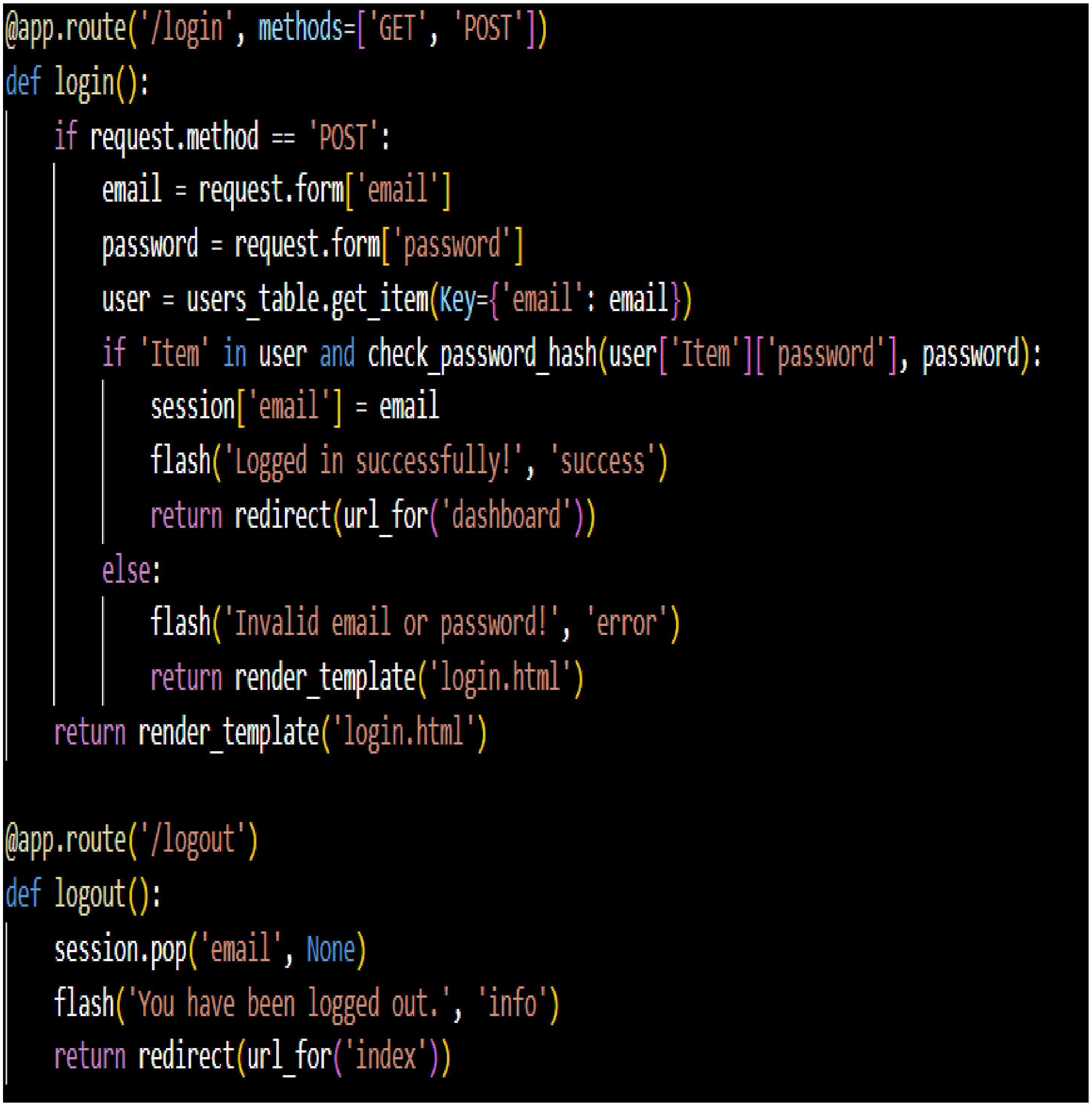
```python
SNS_TOPIC_ARN = 'arn:aws:sns:ap-south-1:353250843450:TravelGoBookingTopic'

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:**



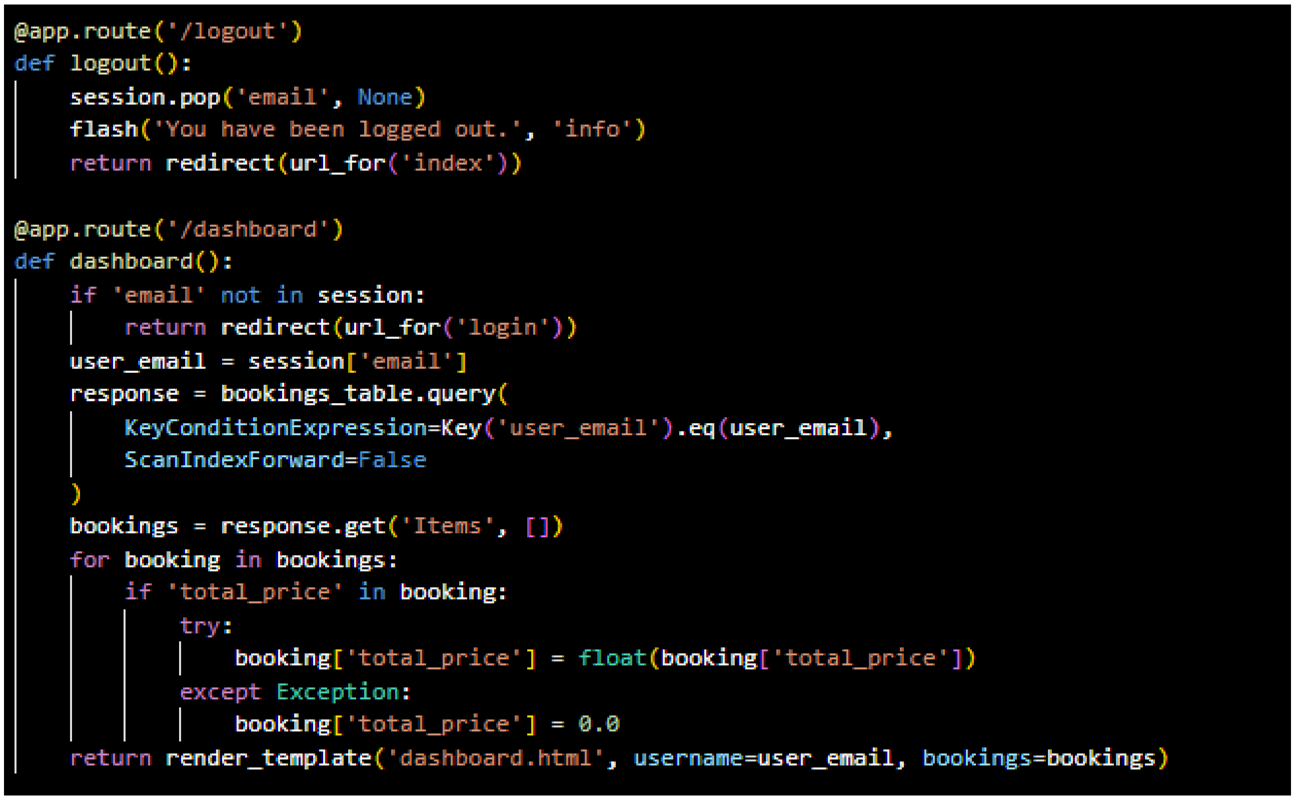
```python
# Routes
@app.route('/')
def index():
    return render_template('index.html')

@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):**



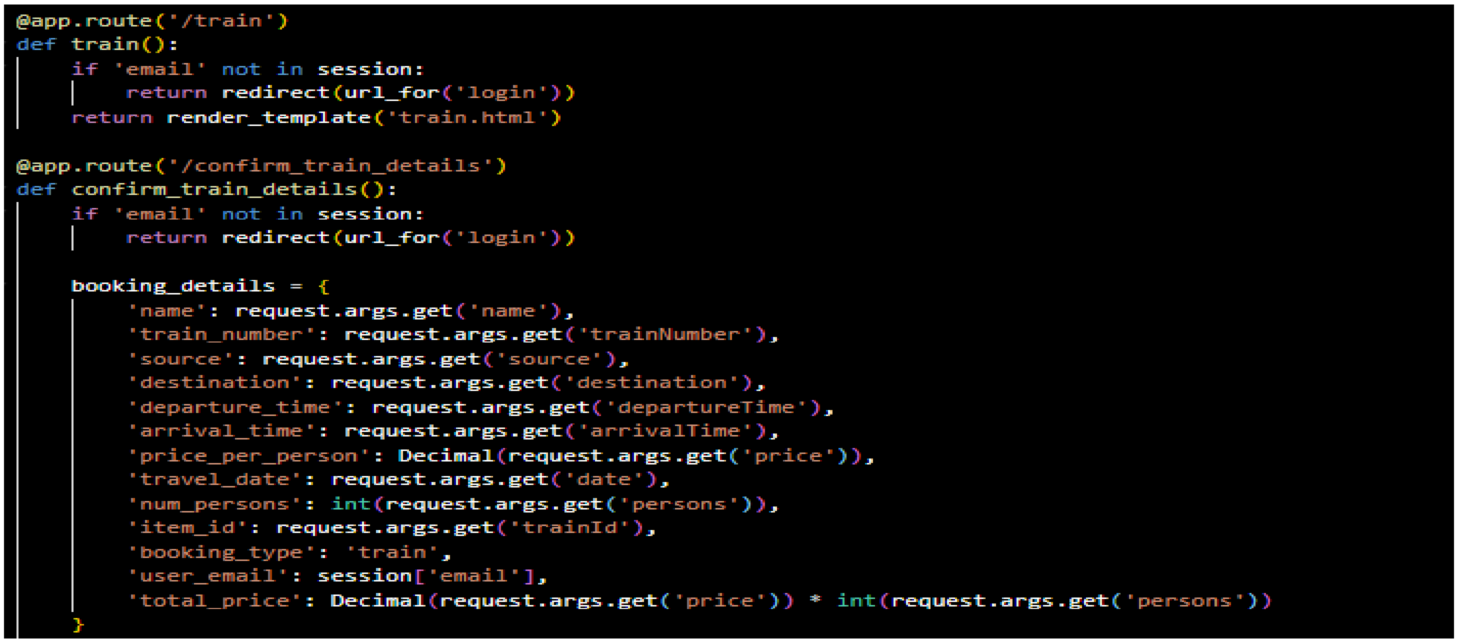
```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = users_table.get_item(Key={'email': email})
        if 'Item' in user and check_password_hash(user['Item']['password'],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

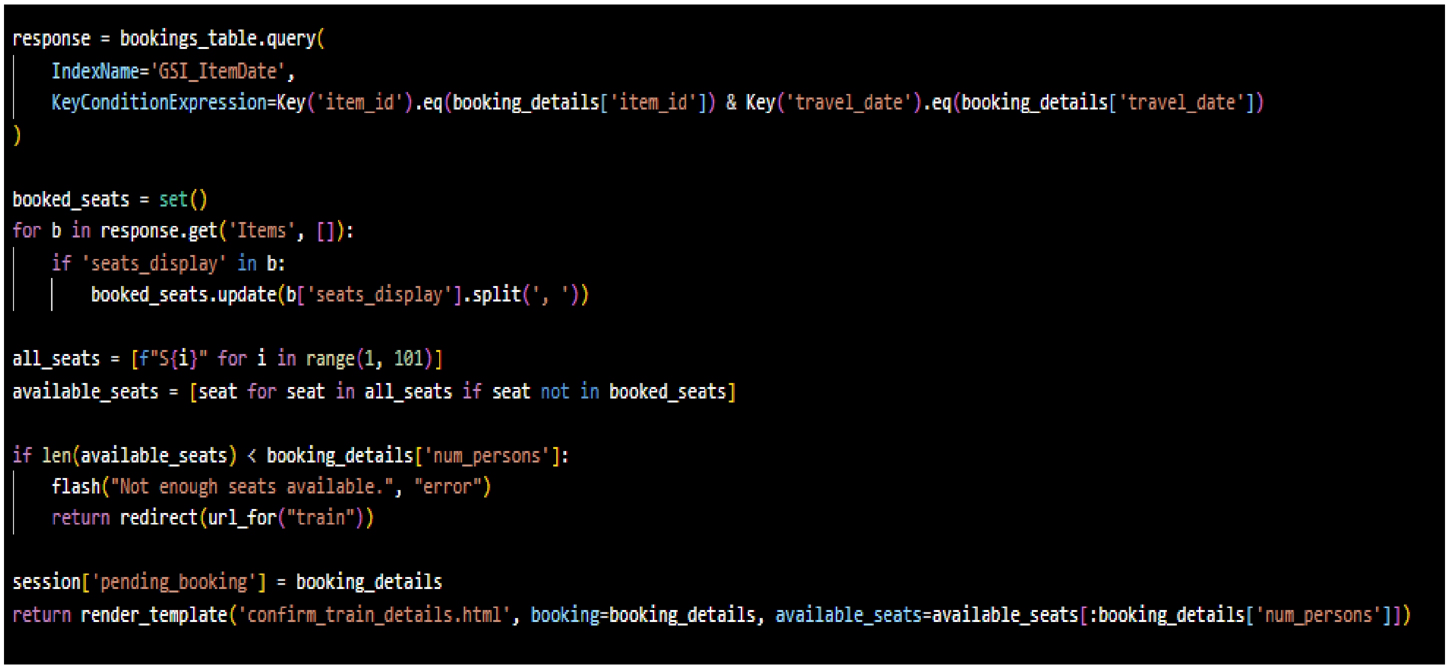


```
@app.route('/logout')
def logout():
    session.pop('email', None)
    flash('You have been logged out.', 'info')
    return redirect(url_for('index'))

@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 :-

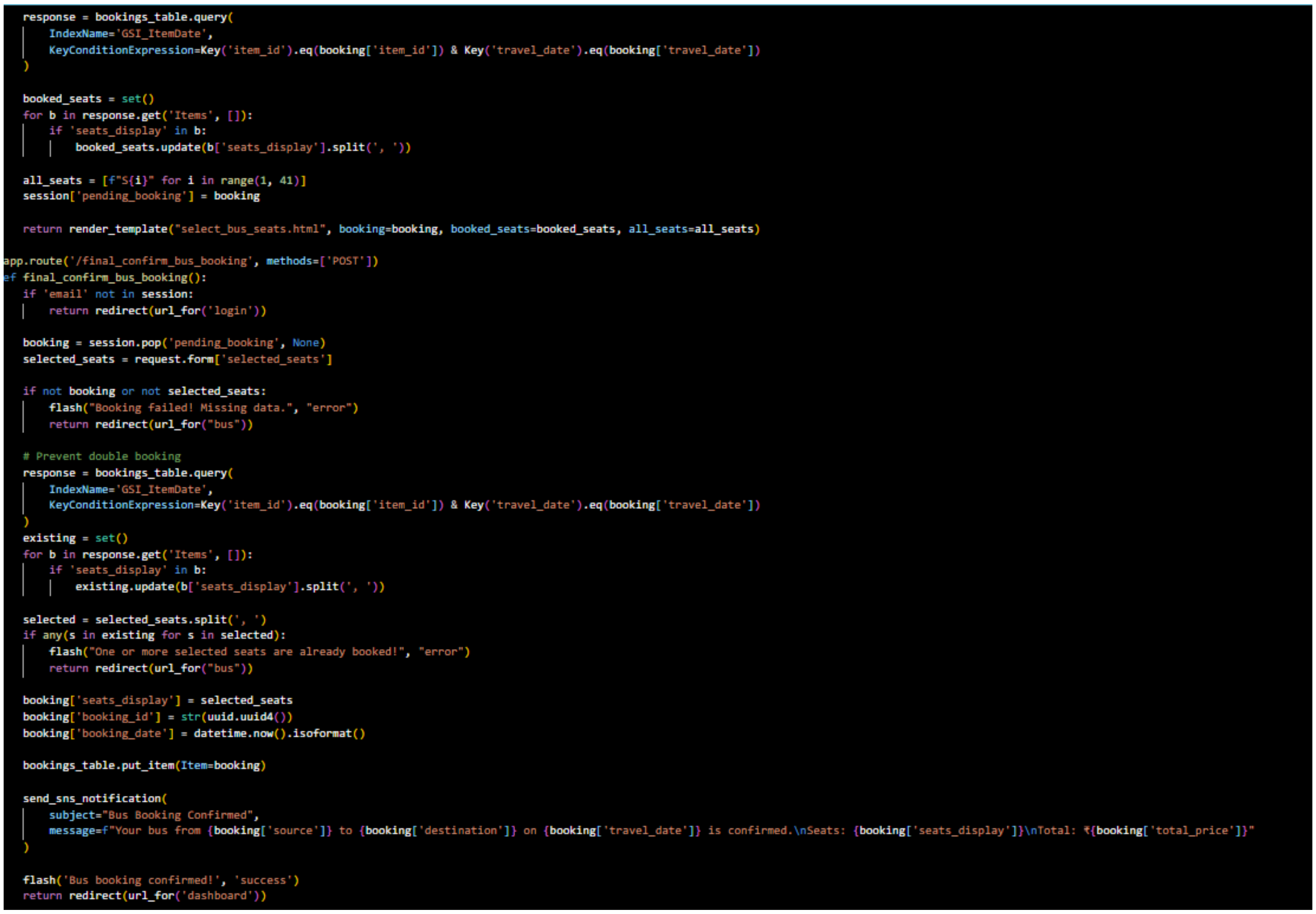


```python
@app.route('/train')
def train():
    if 'email' not in session:
        return redirect(url_for('login'))
    return render_template('train.html')

@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'))
    }
```

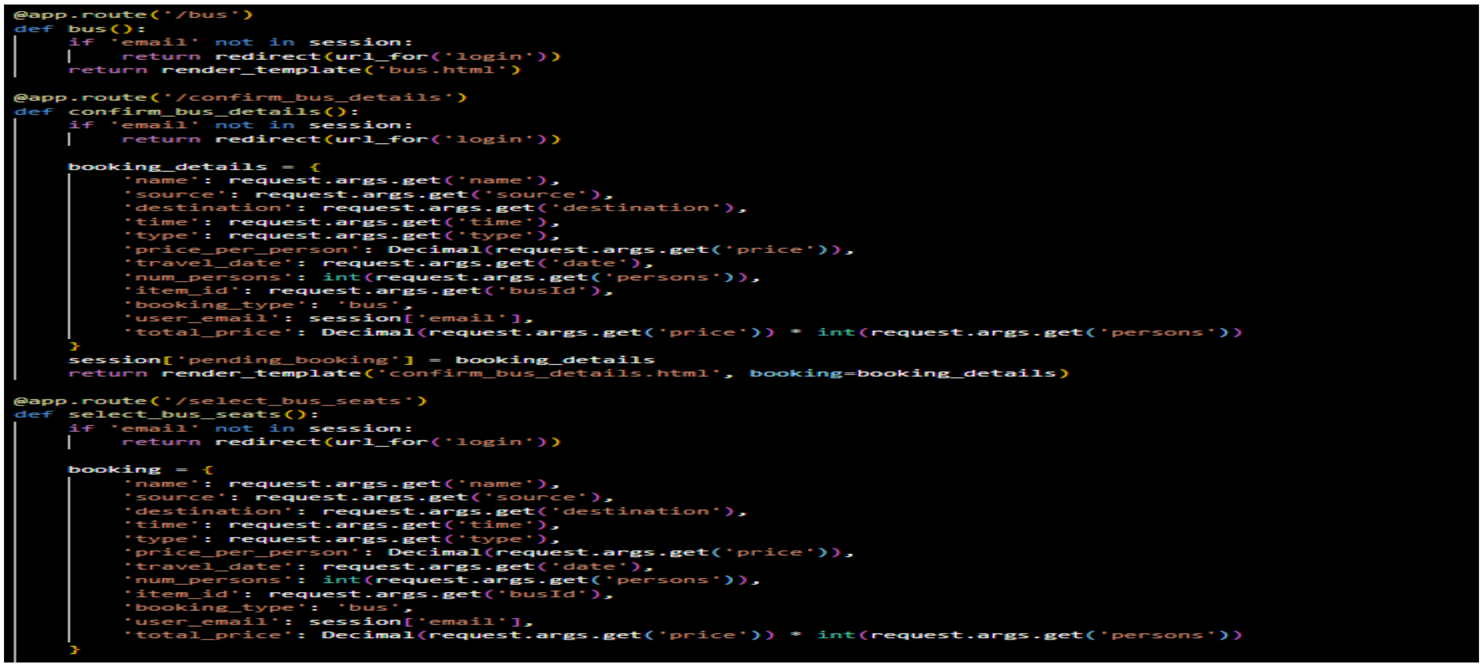


```python
response = bookings_table.query(
    IndexName='GSI_ItemDate',
    KeyConditionExpression=Key('item_id').eq(booking_details['item_id']) & Key('travel_date').eq(booking_details['travel_date'])
)

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:-

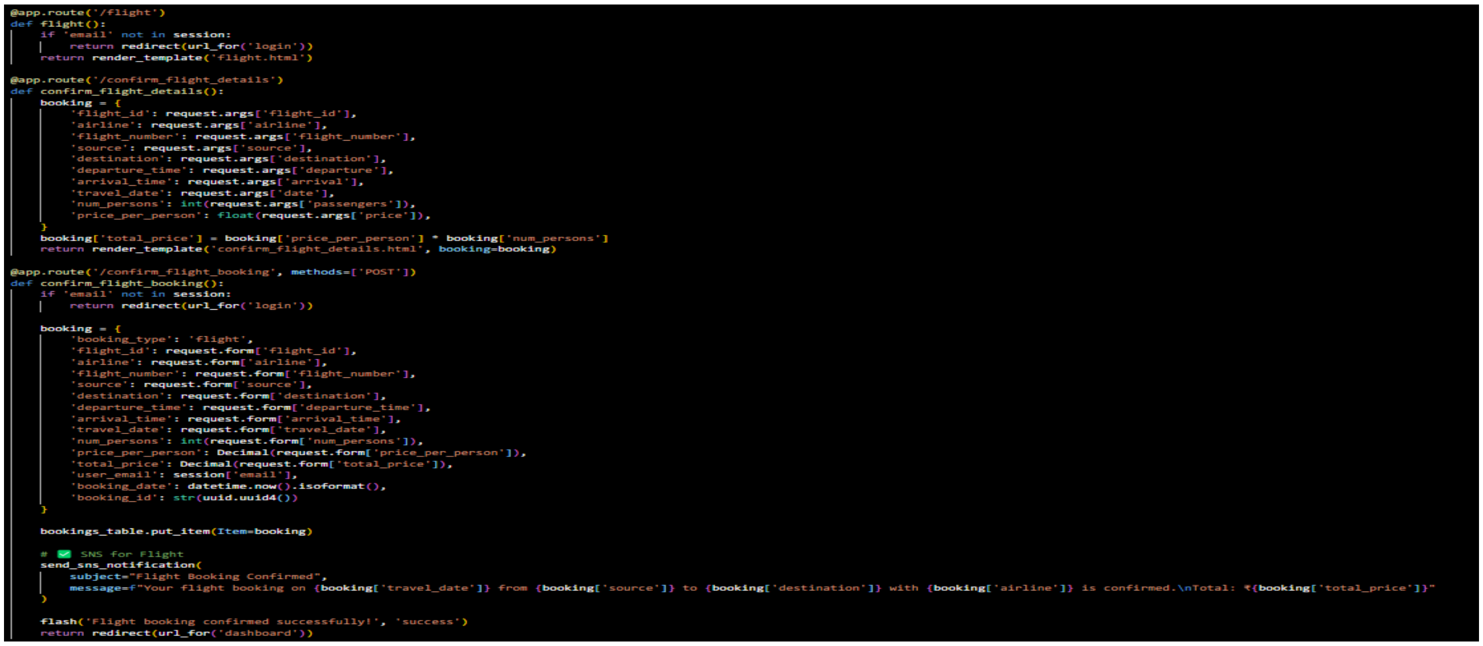


```
    response = bookings_table.query(
        IndexName='GSI_ItemDate',
        KeyConditionExpression=Key('item_id').eq(booking['item_id']) & Key('travel_date').eq(booking['travel_date'])
    )

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

    all_seats = [f"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'])
ef 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'))
```

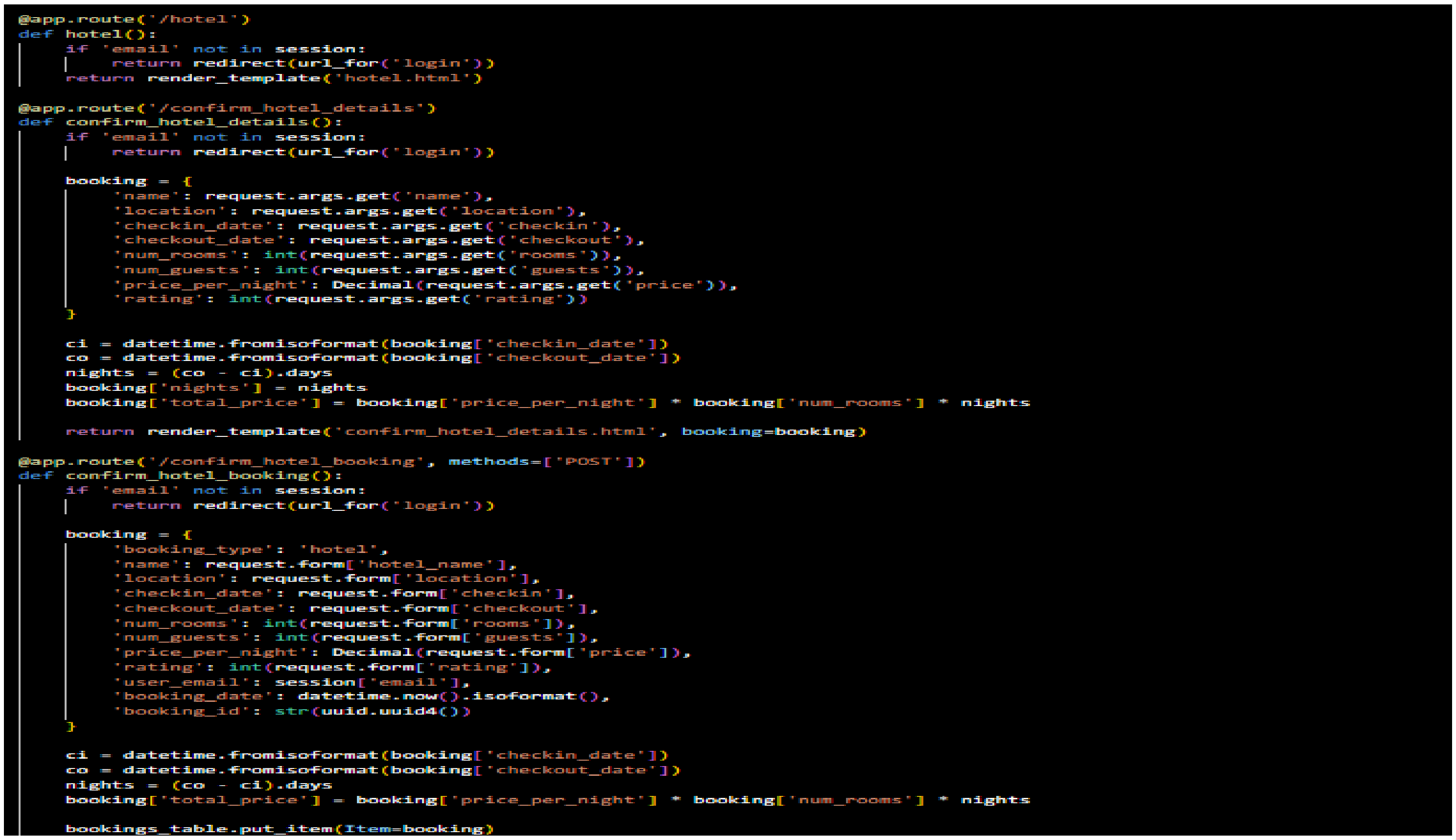


```
@app.route('/bus')
def bus():
    if 'email' not in session:
        return redirect(url_for('login'))
    return render_template('bus.html')

@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 :

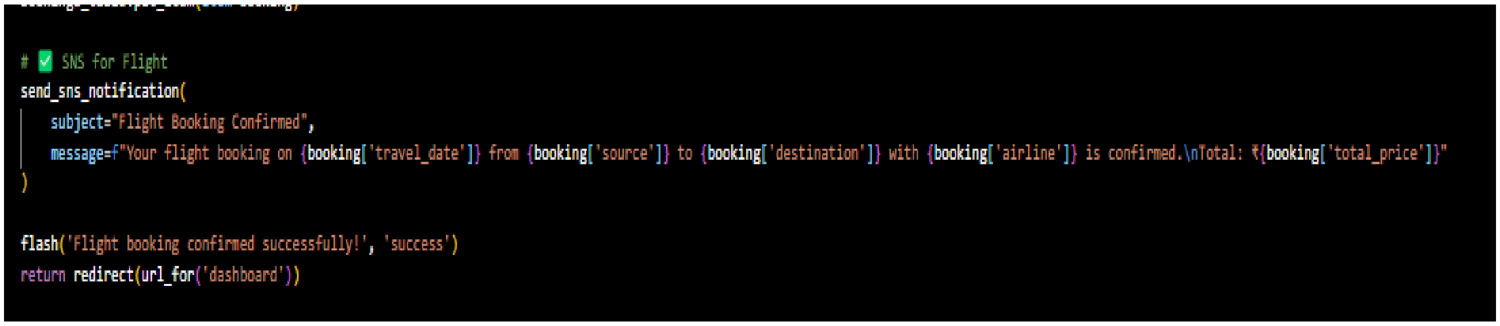


```python
@app.route('/flight')
def flight():
    if 'email' not in session:
        return redirect(url_for('login'))
    return render_template('flight.html')

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

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

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

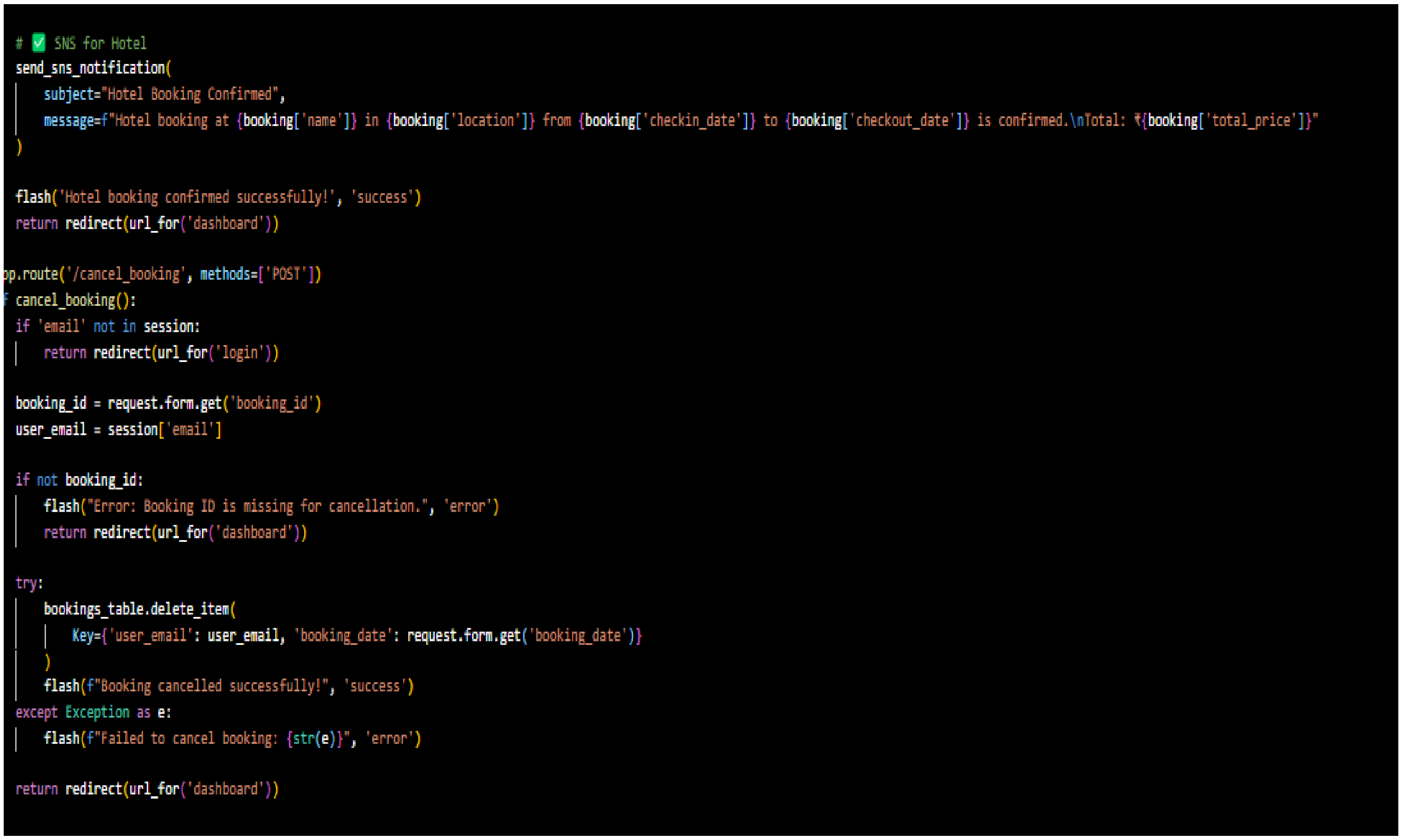


```python
@app.route('/hotel')
def hotel():
    if 'email' not in session:
        return redirect(url_for('login'))
    return render_template('hotel.html')

@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 :-

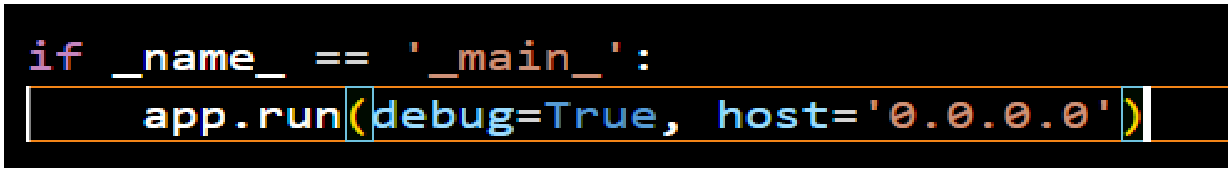


```
# ✅ SNS for Flight
send_sns_notification(
    subject="Flight Booking Confirmed",
    message=f"Your flight booking on {booking['travel_date']} from {booking['source']} to {booking['destination']} with {booking['airline']} is confirmed.\nTotal: ₹{booking['total_price']}"
)

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'))

pp.route('/cancel_booking', methods=['POST'])
f 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:

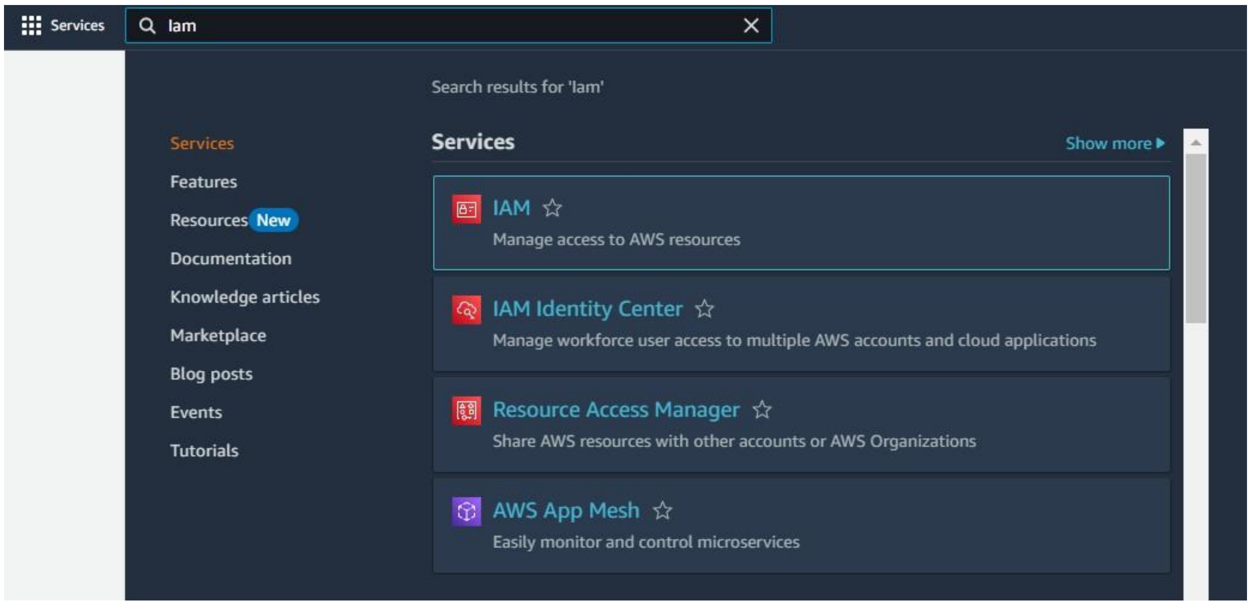


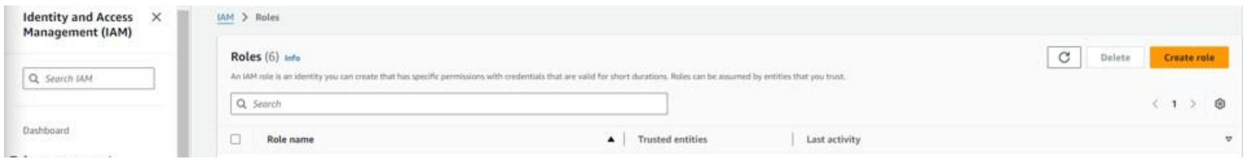
```
if _name_ == '_main_':
    app.run(debug=True, host='0.0.0.0')
```

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.

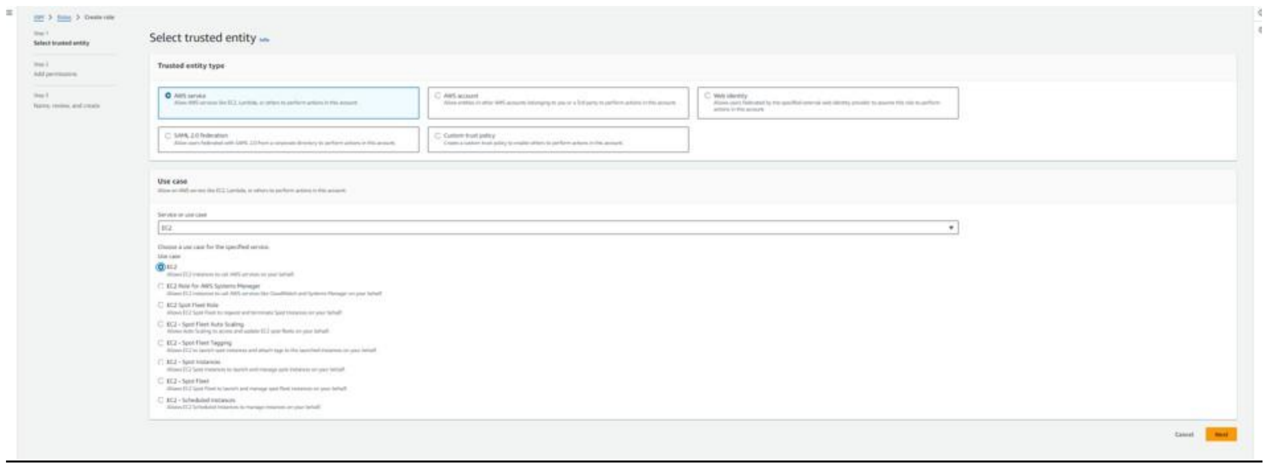
Milestone 5: IAM Role Setup

- **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.**





An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.



IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity

Trusted entity type

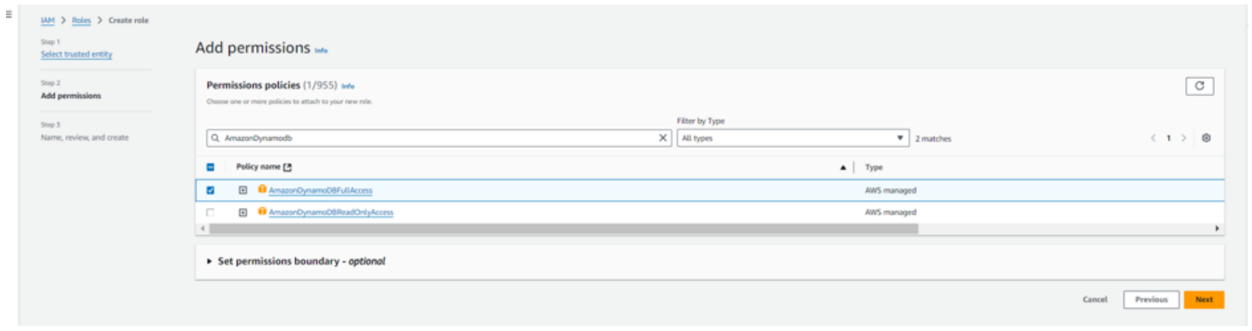
- [✓ AWS service] Allows AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account: Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity: Allows users federated by the specified external web identity provider 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.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case: EC2

Choose a use case for the specified service.
Use case

- [✓ 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.



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.

Search: AmazonDynamodb — Filter by Type: All types — 2 matches

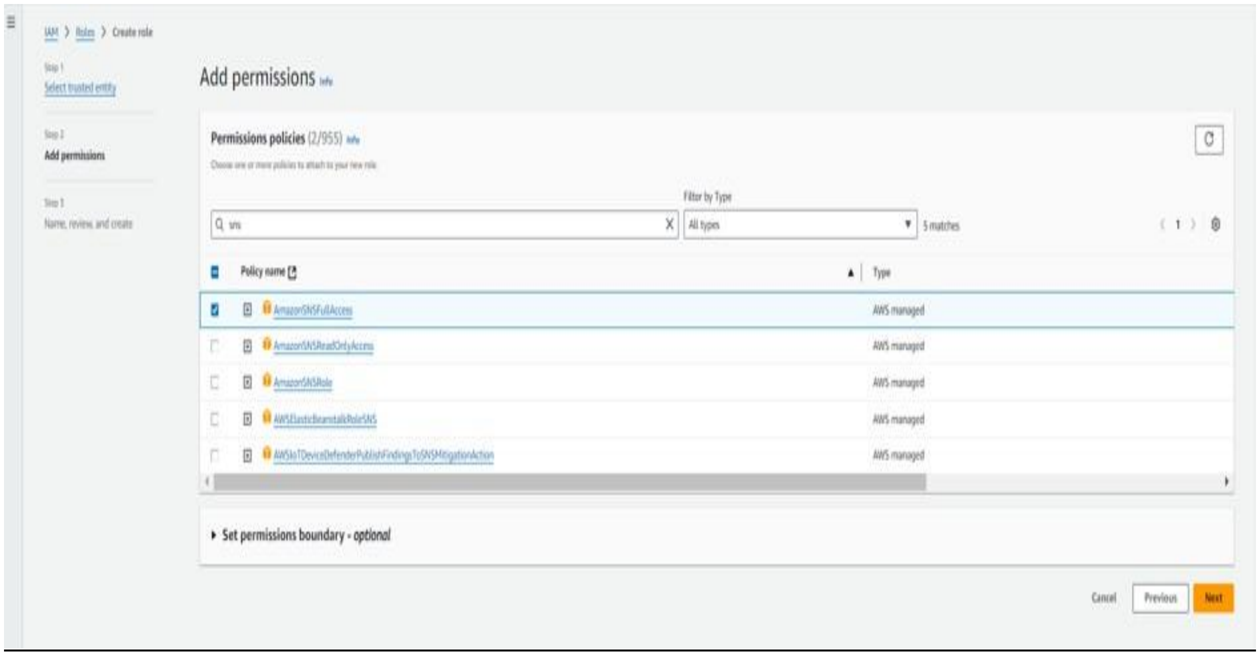
	Policy name	Type
✓	AmazonDynamoDBFullAccess	AWS managed
	AmazonDynamoDBReadOnlyAccess	AWS managed

▸ Set permissions boundary - *optional*

- **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.**



IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions Info

Permissions policies (2/955) Info

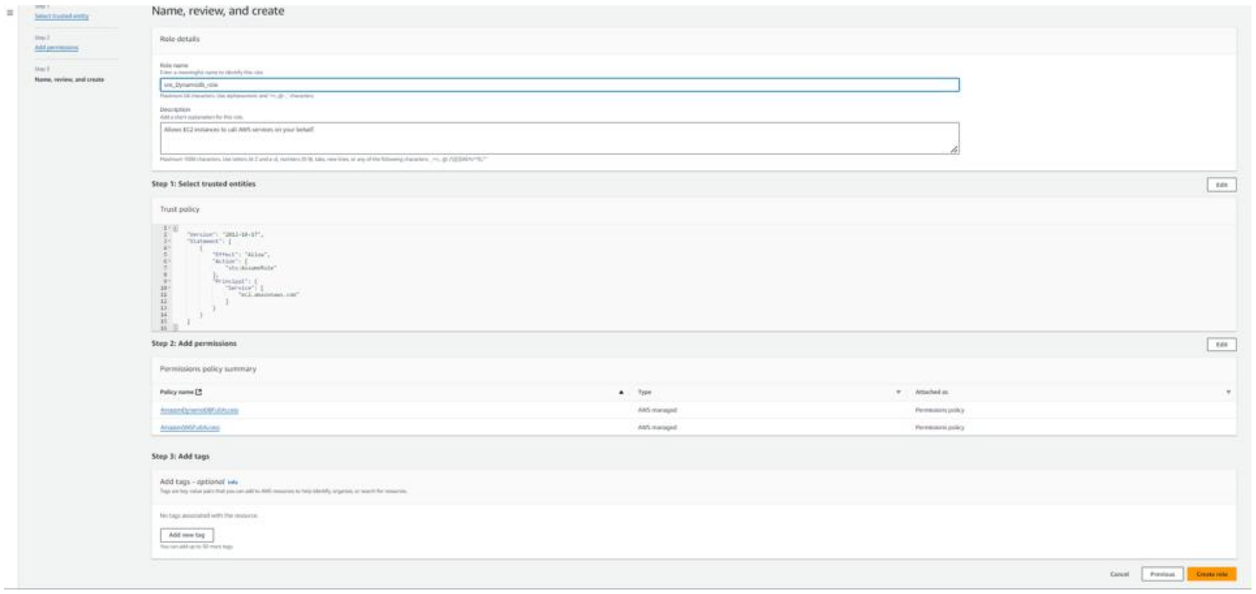
Choose one or more policies to attach to your new role.

Filter by Type

Search: sns | All types | 5 matches

	Policy name	Type
✓	AmazonSNSFullAccess	AWS managed
	AmazonSNSReadOnlyAccess	AWS managed
	AmazonSNSRole	AWS managed
	AWSElasticBeanstalkRoleSNS	AWS managed
	AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction	AWS managed

▸ **Set permissions boundary - *optional***



Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.

sns_Dynamodb_role

Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description
Add a short explanation for this role.

Allows EC2 instances to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

Step 2: Add permissions

Permissions policy summary

Policy name	Type	Attached as
AmazonDynamoDBFullAccess	AWS managed	Permissions policy
AmazonSNSFullAccess	AWS managed	Permissions policy

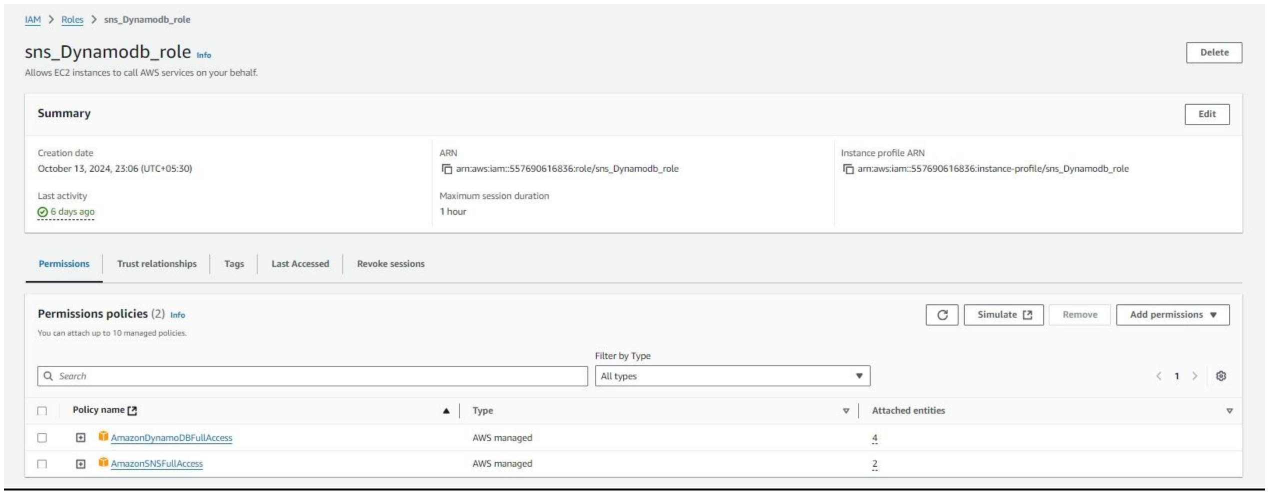
Step 3: Add tags

Add tags - *optional* Info

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

You can add up to 50 more tags.


IAM > Roles > sns_Dynamodb_role

sns_Dynamodb_role Info

Allows EC2 instances to call AWS services on your behalf.

Summary

Creation date: October 13, 2024, 23:06 (UTC+05:30)

ARN: arn:aws:iam::557690616836:role/sns_Dynamodb_role

Instance profile ARN: arn:aws:iam::557690616836:instance-profile/sns_Dynamodb_role

Last activity: 6 days ago

Maximum session duration: 1 hour

Permissions policies (2) Info

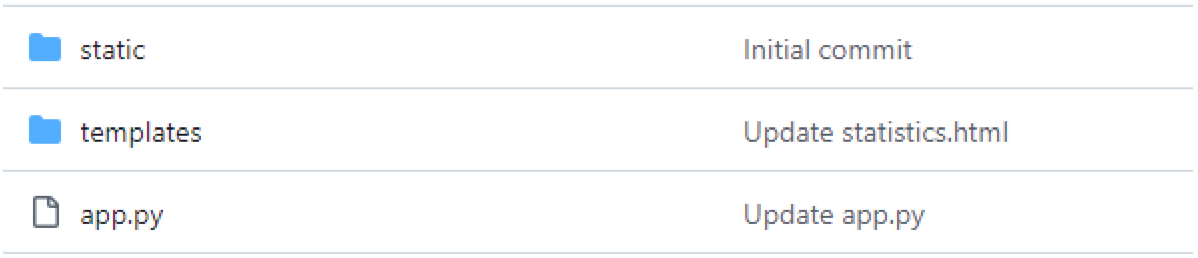
You can attach up to 10 managed policies.

Filter by Type: All types

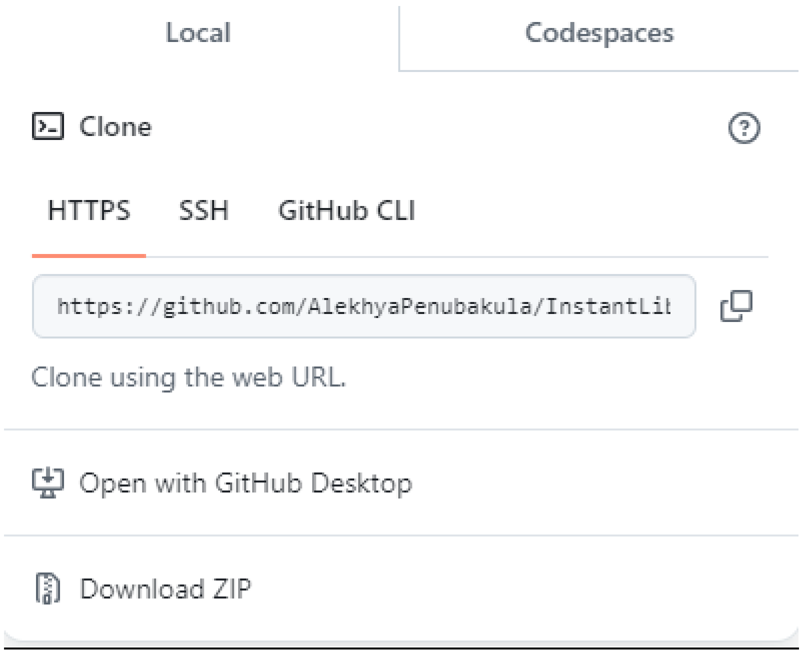
Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	4
AmazonSNSFullAccess	AWS managed	2

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

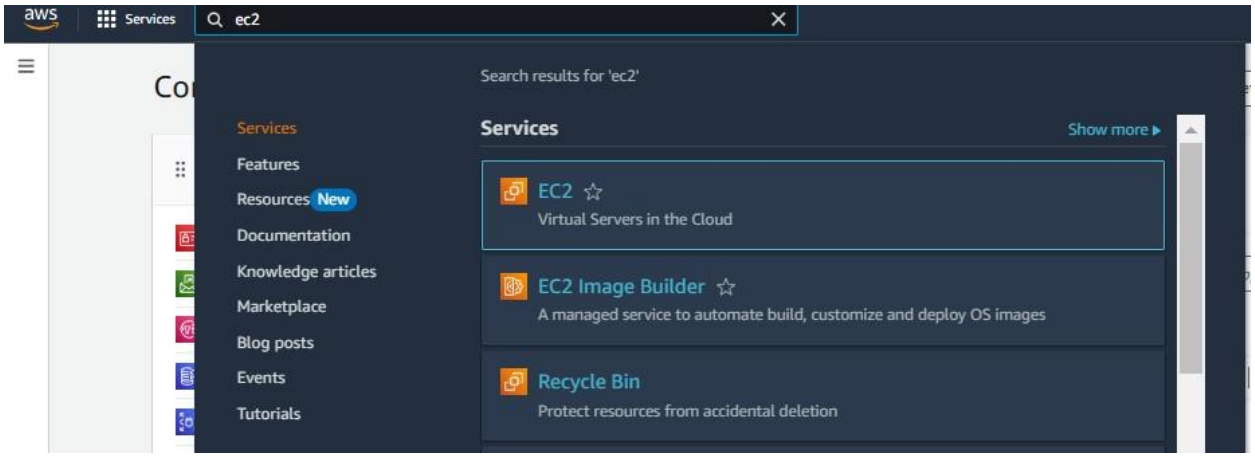

Local
Codespaces
Clone
HTTPS
SSH
GitHub CLI

```
https://github.com/AlekhyaPenubakula/InstantLit
```

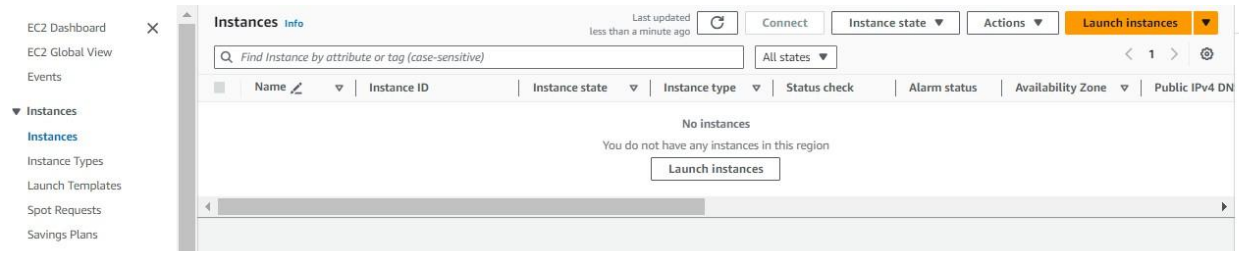
Clone using the web URL.
Open with GitHub Desktop
Download ZIP

- <u>**Activity 6.1: Launch an EC2 instance to host the Flask application.**</u>

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



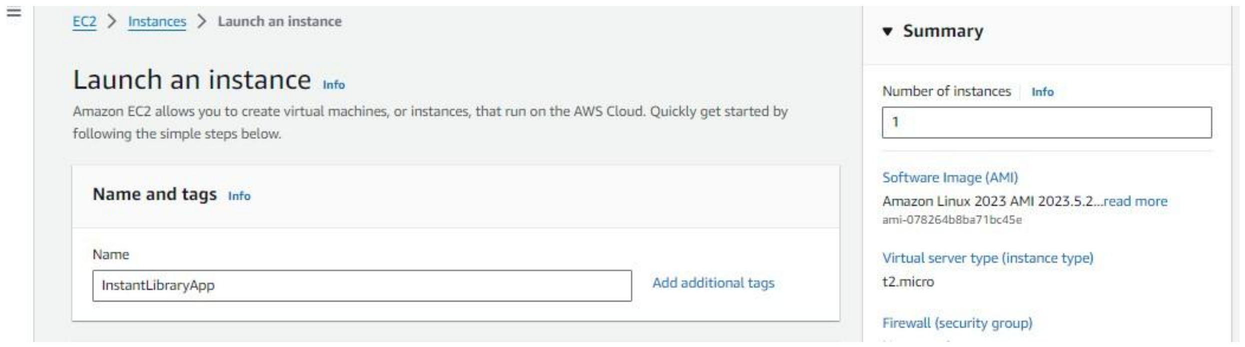
- **Click on Launch instance to launch EC2 instance**



No instances

You do not have any instances in this region

Launch instances



Launch an instance

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.

Name and tags

Name: InstantLibraryApp

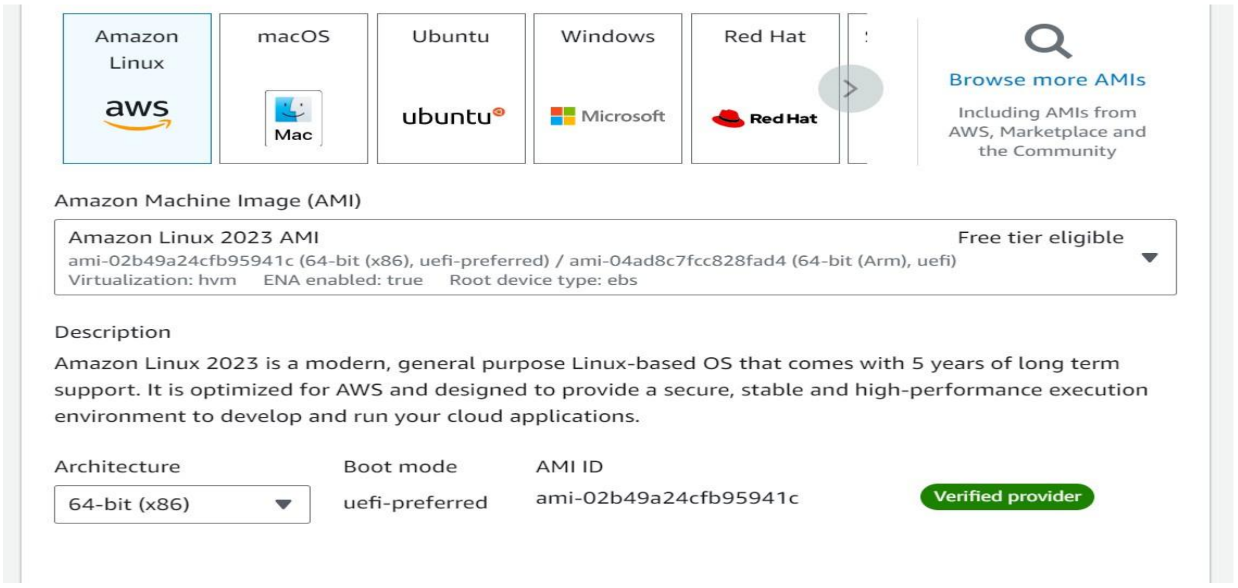
Number of instances: 1

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

Virtual server type (instance type): t2.micro

Firewall (security group)

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



Amazon Machine Image (AMI)

Amazon Linux 2023 AMI — Free tier eligible

ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)

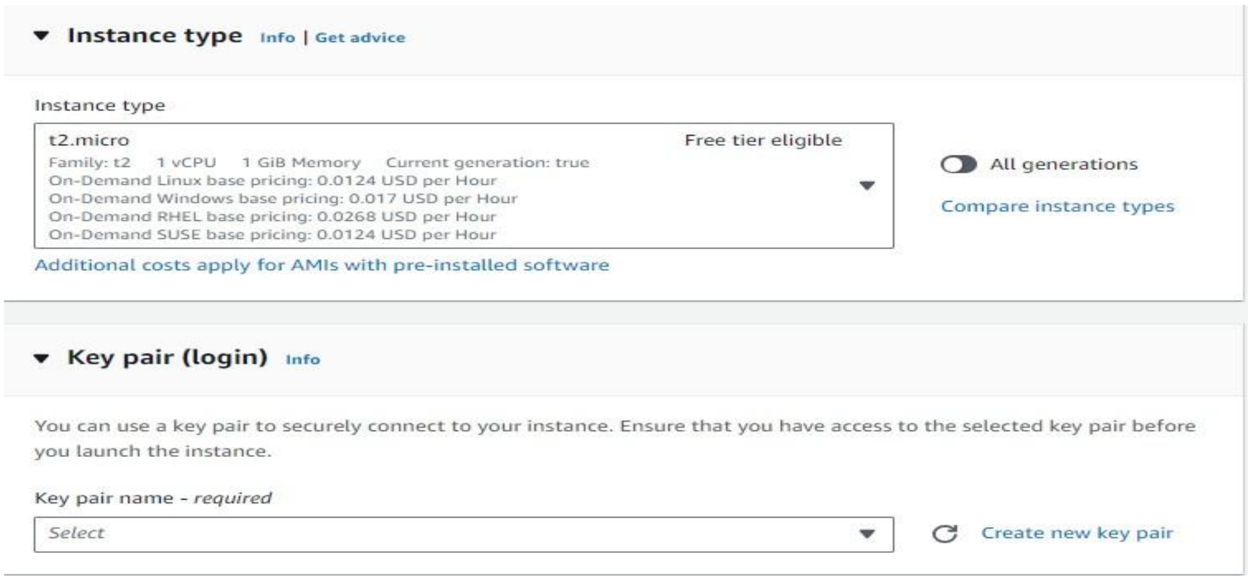
Virtualization: hvm ENA enabled: true Root device type: ebs

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
64-bit (x86)	uefi-preferred	ami-02b49a24cfb95941c

- **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

Additional costs apply for AMIs with pre-installed software

All generations

Compare instance types

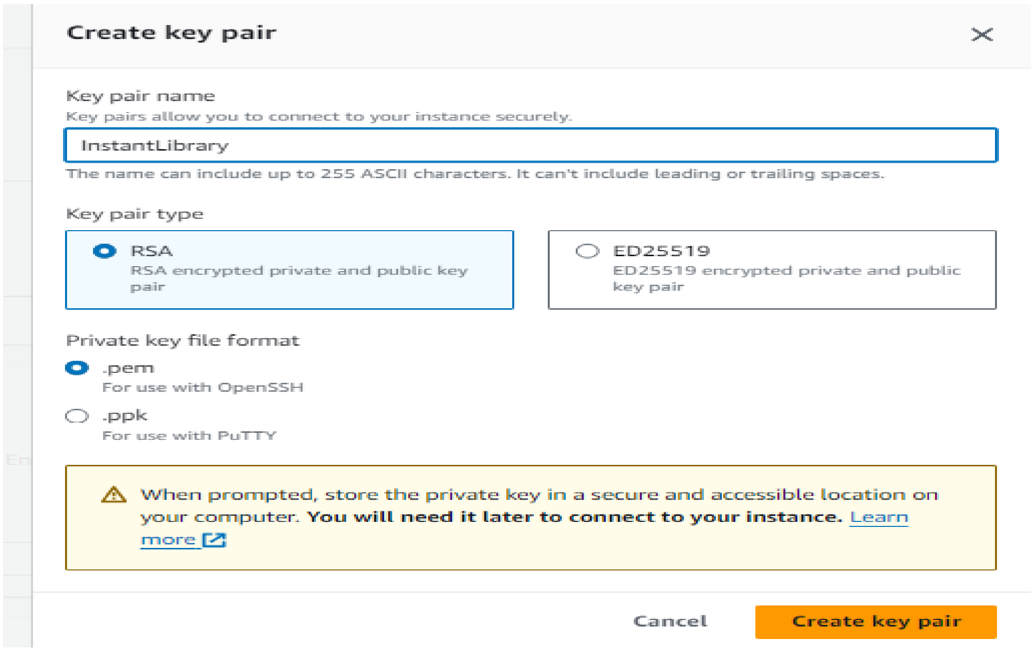
Key pair (login) Info

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.

InstantLibrary

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

[✓ RSA] RSA encrypted private and public key pair

ED25519 — ED25519 encrypted private and public key pair

Private key file format

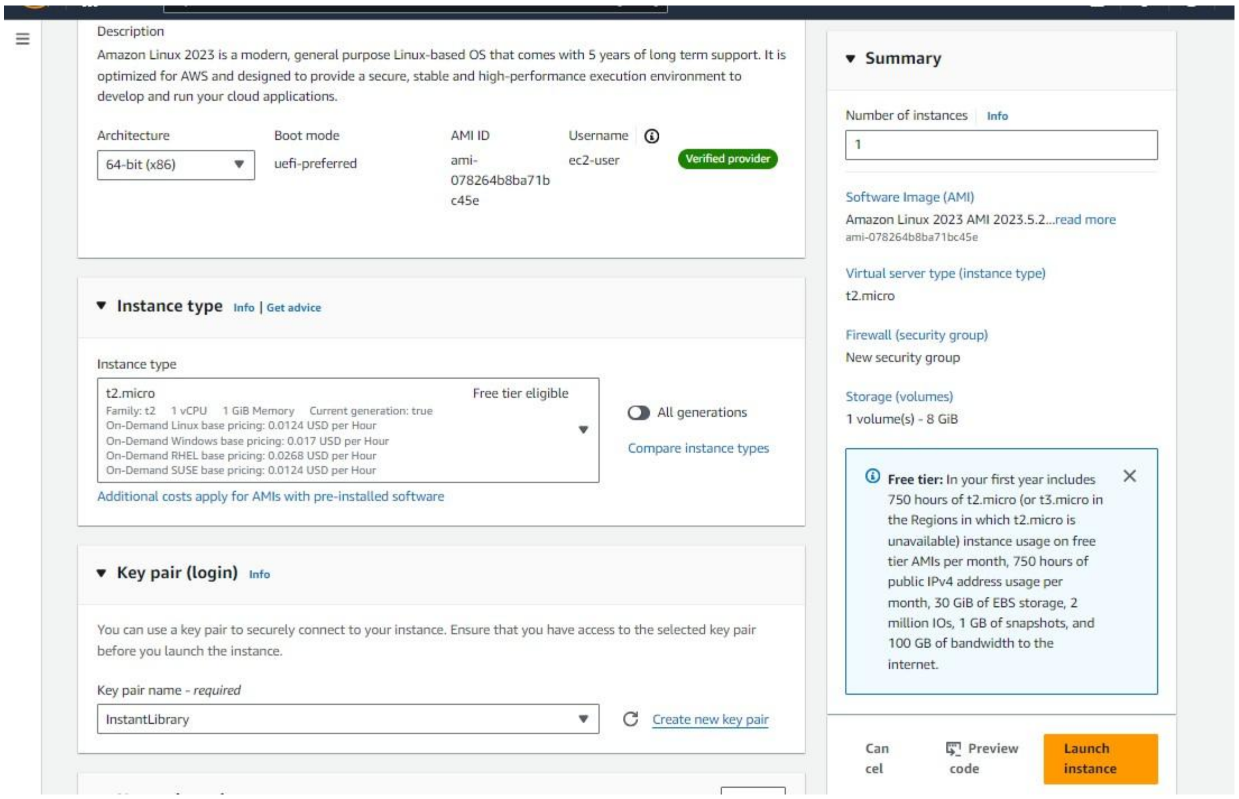
[✓ .pem] For use with OpenSSH

.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	Verified provider

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*

InstantLibrary

Create new key pair

Summary

Number of instances Info

1

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

Virtual server type (instance type)
t2.micro

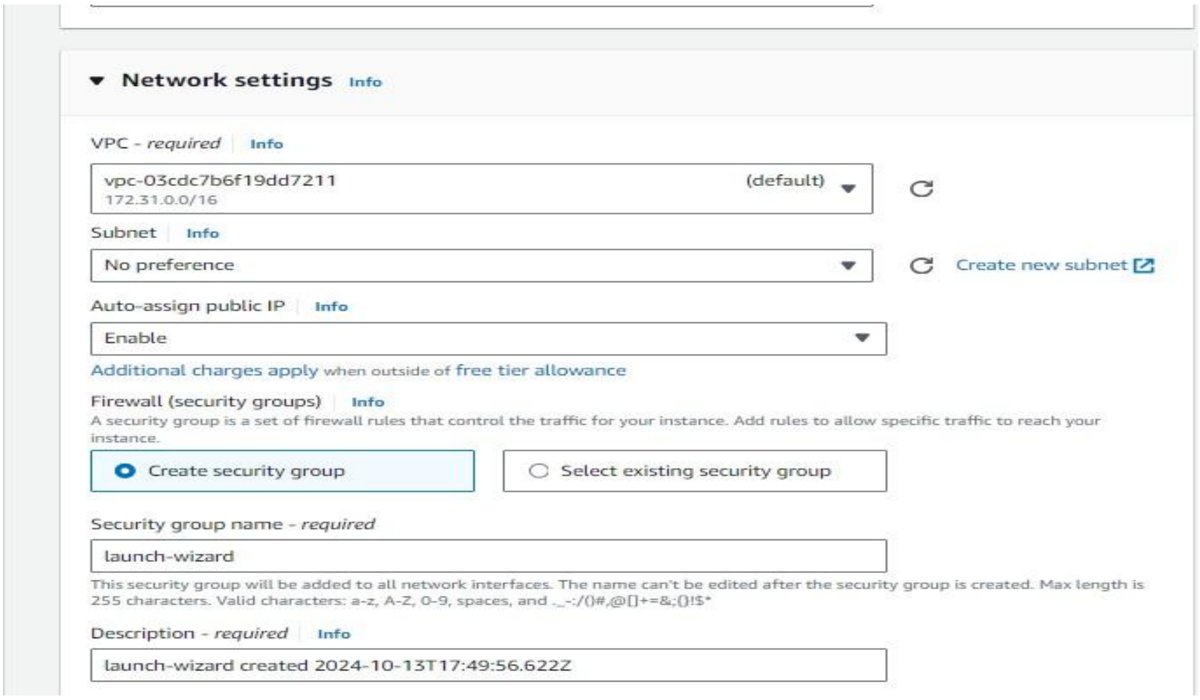
Firewall (security group)
New security group

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 IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Cancel | Preview code | Launch instance

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



Network settings

VPC - *required*

vpc-03cdc7b6f19dd7211 (default)
172.31.0.0/16

Subnet

No preference

Auto-assign public IP

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups)

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

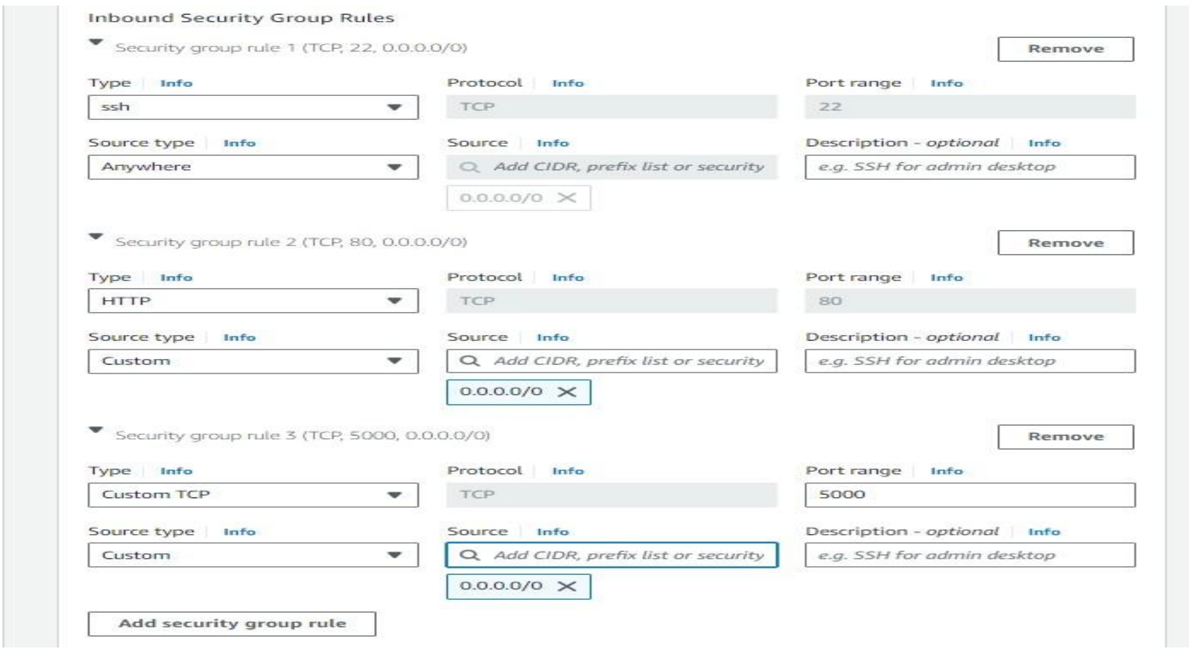
Security group name - *required*

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*

launch-wizard created 2024-10-13T17:49:56.622Z



Inbound Security Group Rules

Security group rule 1 (TCP, 22, 0.0.0.0/0)

Type	Protocol	Port range
ssh	TCP	22

Source type	Source	Description - *optional*
Anywhere	0.0.0.0/0	e.g. SSH for admin desktop

Security group rule 2 (TCP, 80, 0.0.0.0/0)

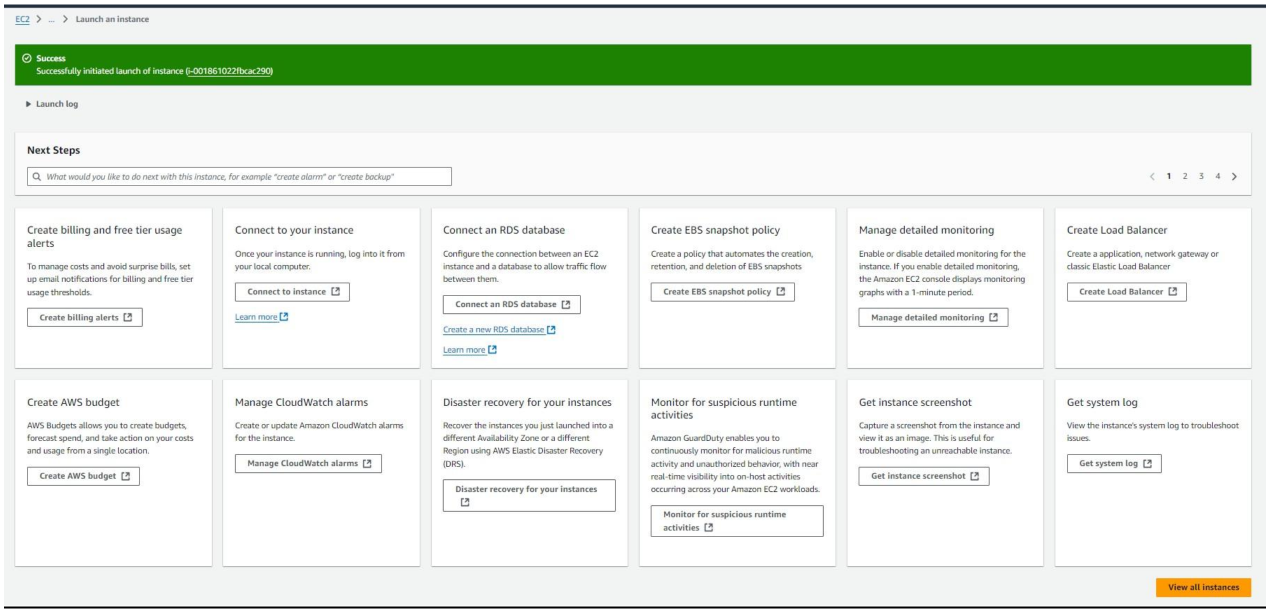
Type	Protocol	Port range
HTTP	TCP	80

Source type	Source	Description - *optional*
Custom	0.0.0.0/0	e.g. SSH for admin desktop

Security group rule 3 (TCP, 5000, 0.0.0.0/0)

Type	Protocol	Port range
Custom TCP	TCP	5000

Source type	Source	Description - *optional*
Custom	0.0.0.0/0	e.g. SSH for admin desktop


EC2 > ... > Launch an instance

Success
Successfully initiated launch of instance (i-001861022fbcac290)

Launch log

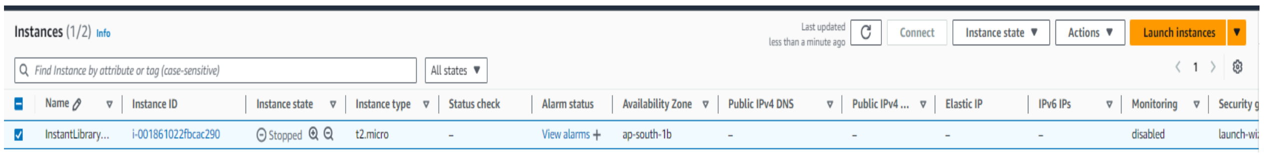
Next Steps

What would you like to do next with this instance, for example "create alarm" or "create backup"

Create billing and free tier usage alerts To manage costs and avoid surprise bills, set up email 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 EC2 console displays monitoring graphs with a 1-minute period. 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 instances you just launched into a different Availability Zone or a different Region using AWS Elastic Disaster Recovery (DRS). 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-host 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

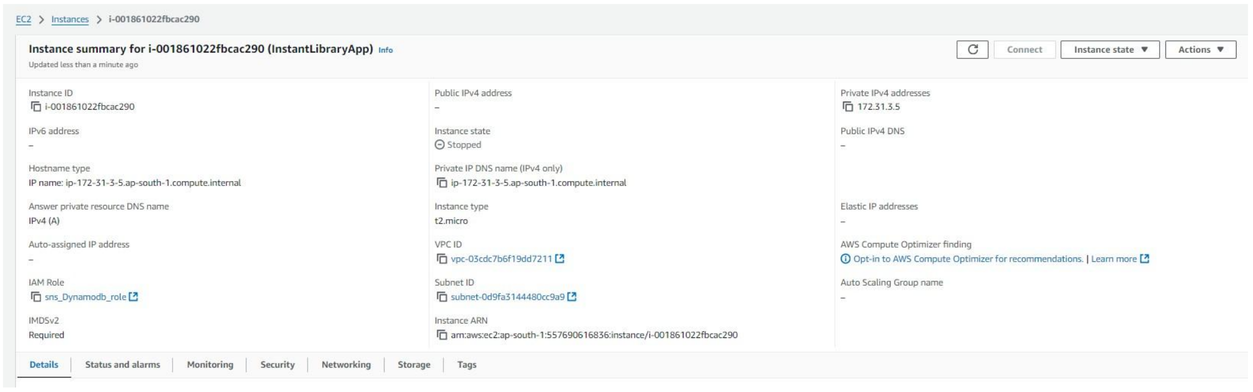
- **<u>To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected,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


Instances (1/2) Info

Last updated less than a minute ago | Connect | Instance state | Actions | Launch instances

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

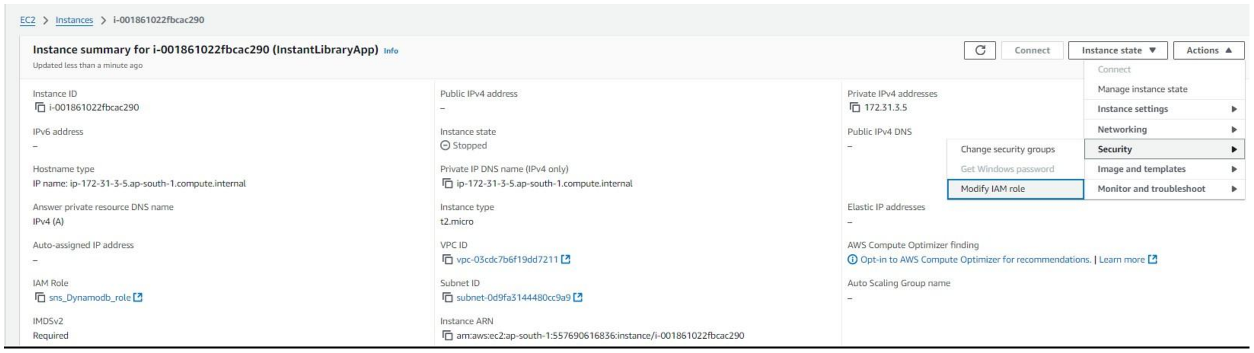

EC2 > Instances > i-001861022fbcac290

Instance summary for i-001861022fbcac290 (InstantLibraryApp) Info
Updated less than a minute ago

Connect | Instance state | Actions

Instance ID i-001861022fbcac290	Public IPv4 address –	Private IPv4 addresses 172.31.3.5
IPv6 address –	Instance state Stopped	Public IPv4 DNS –
Hostname type IP name: ip-172-31-3-5.ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-3-5.ap-south-1.compute.internal	
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	Elastic IP addresses –
Auto-assigned IP address –	VPC ID vpc-03cdc7b6f19dd7211	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role sns_Dynamodb_role	Subnet ID subnet-0d9fa3144480cc9a9	Auto Scaling Group name –
IMDSv2 Required	Instance ARN arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290	

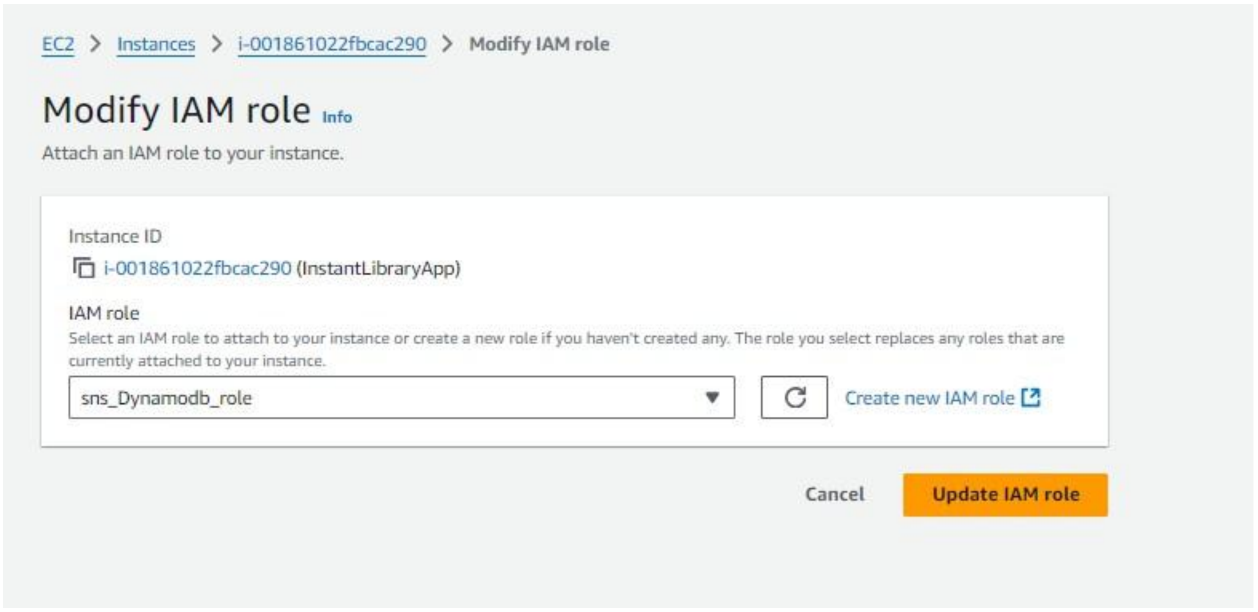
Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags


EC2 > Instances > i-001861022fbcac290

Instance summary for i-001861022fbcac290 (InstantLibraryApp) Info

Updated less than a minute ago

Instance ID i-001861022fbcac290	Public IPv4 address –	Private IPv4 addresses 172.31.3.5
IPv6 address –	Instance state Stopped	Public IPv4 DNS –
Hostname type IP name: ip-172-31-3-5.ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-3-5.ap-south-1.compute.internal	
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	Elastic IP addresses –
Auto-assigned IP address –	VPC ID vpc-03cdc7b6f19dd7211	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role sns_Dynamodb_role	Subnet ID subnet-0d9fa3144480cc9a9	Auto Scaling Group name –
IMDSv2 Required	Instance ARN arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290	


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

Modify IAM role Info

Attach an IAM role to your instance.

Instance ID

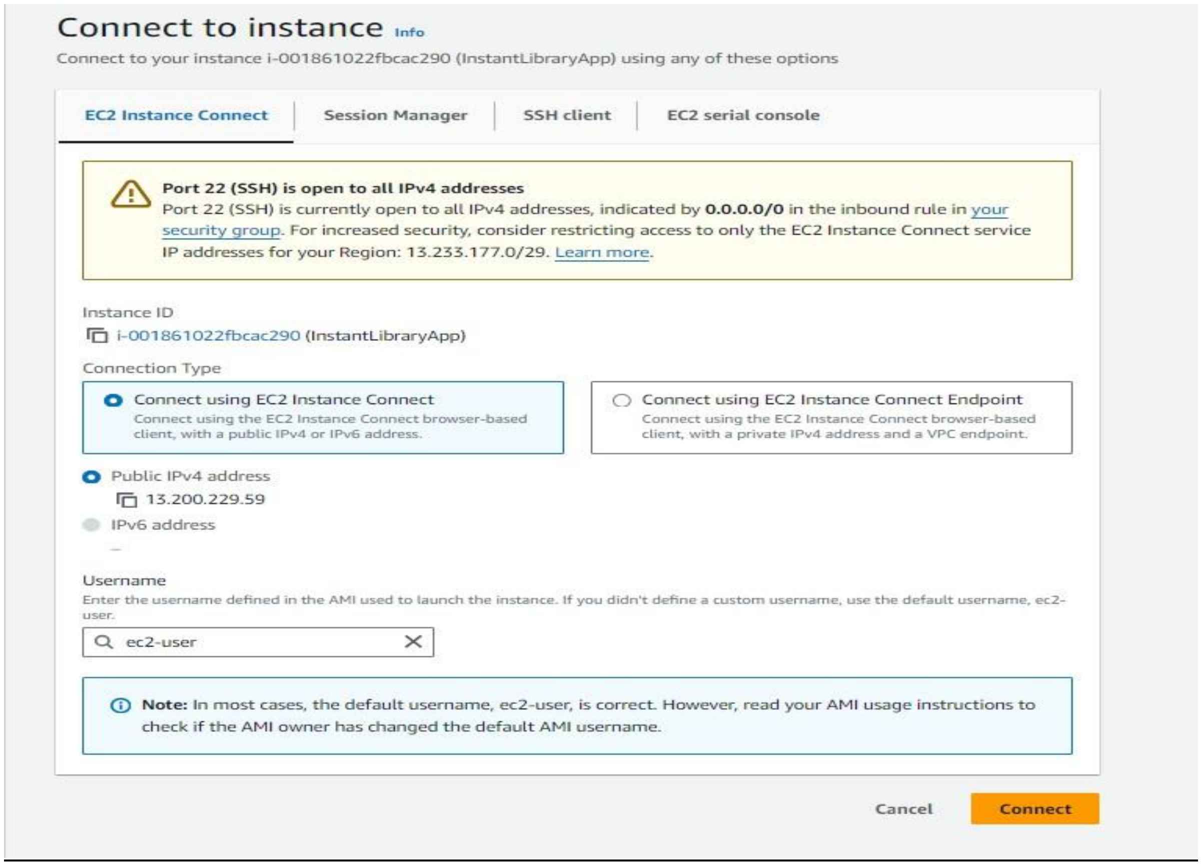
i-001861022fbcac290 (InstantLibraryApp)

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.

sns_Dynamodb_role

- **<u>Now connect the EC2 with the files</u>**



Connect to instance

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

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.

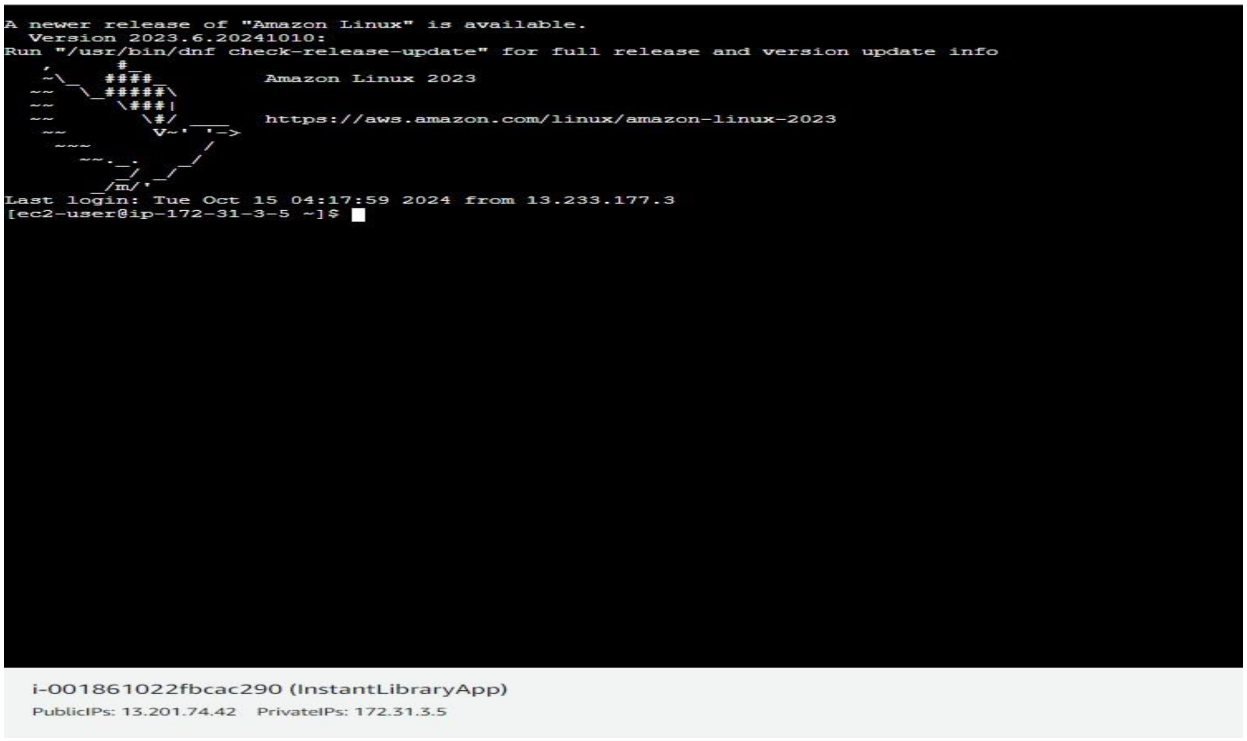
Connect using EC2 Instance Connect Endpoint
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

[✓ 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.



```
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
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[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/your-github-username/your-repository-name.git'

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

credentials here:

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

- This will download your project to the EC2 instance.

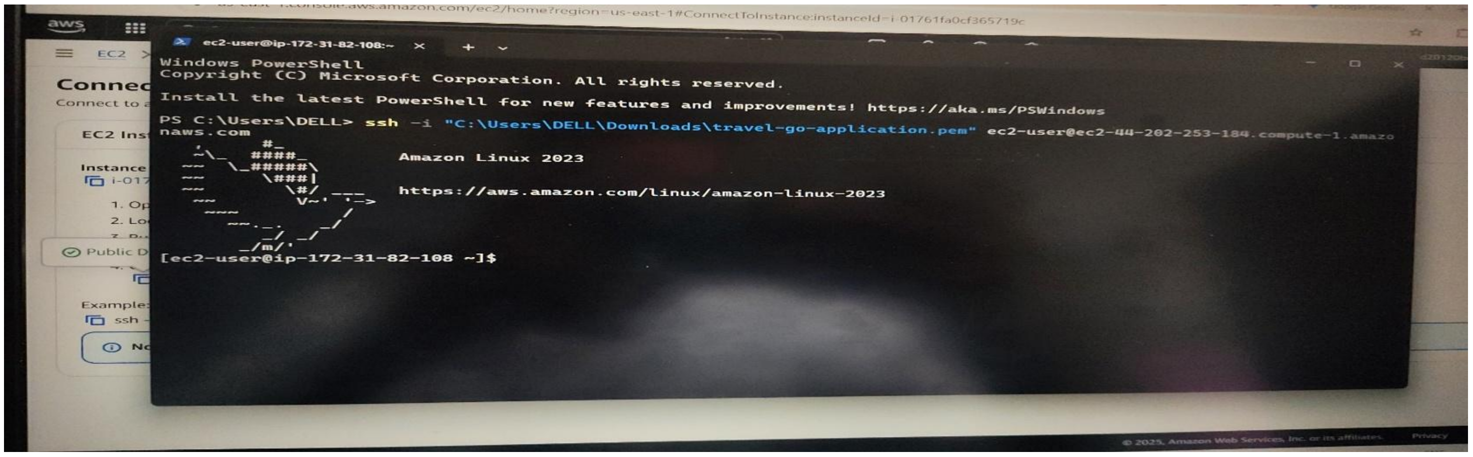
To navigate to the project directory, run the following command:

```
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
```



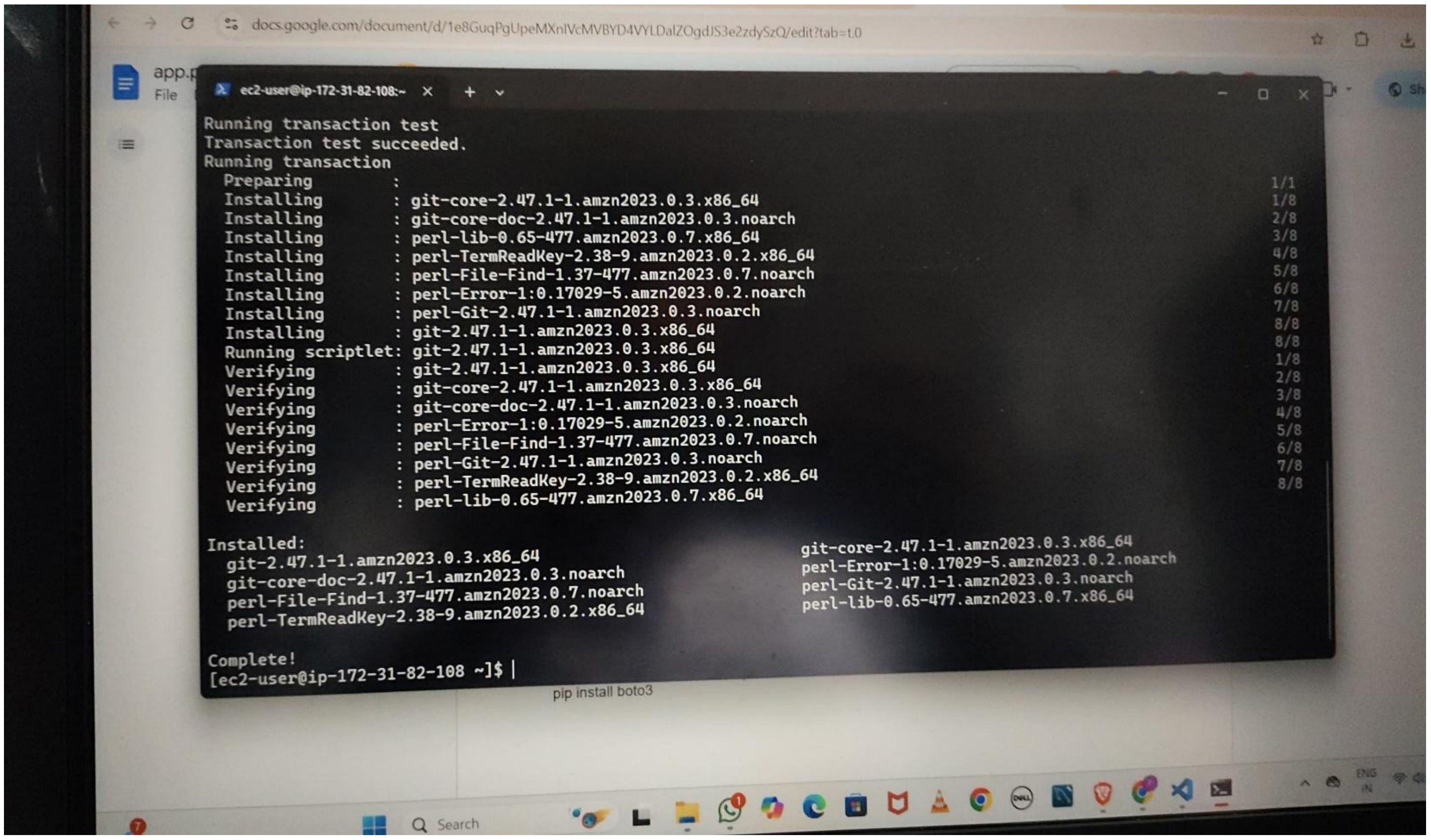
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

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

PS C:\Users\DELL> ssh -i "C:\Users\DELL\Downloads\travel-go-application.pem" ec2-user@ec2-44-202-253-184.compute-1.amazo
naws.com
        #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-82-108 ~]$
```

Verify the Flask app is running:

http://your-ec2-public-

ip

- **Run the Flask app on the EC2 instance**



```
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-2.47.1-1.amzn2023.0.3.x86_64
  git-core-doc-2.47.1-1.amzn2023.0.3.noarch           perl-Error-1:0.17029-5.amzn2023.0.2.noarch
  perl-File-Find-1.37-477.amzn2023.0.7.noarch         perl-Git-2.47.1-1.amzn2023.0.3.noarch
  perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64         perl-lib-0.65-477.amzn2023.0.7.x86_64

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

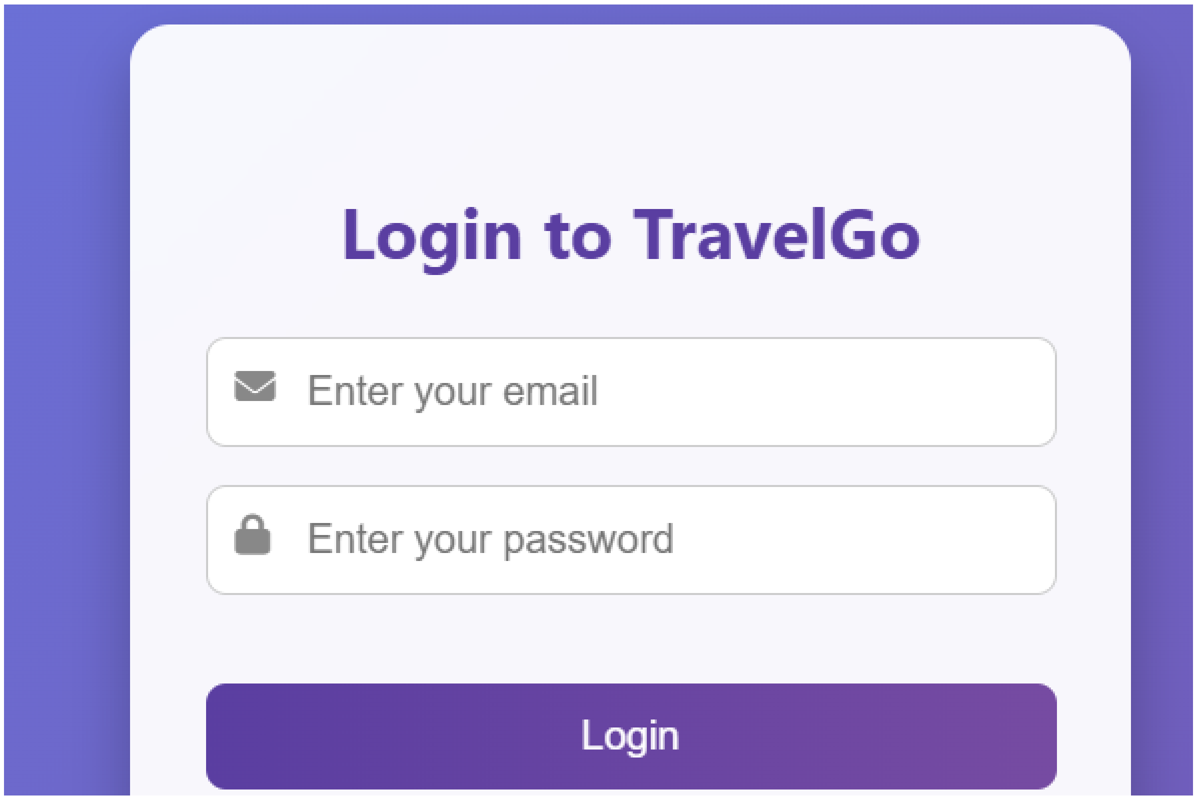
Access the website through:

http://44.202.253.184:5000/

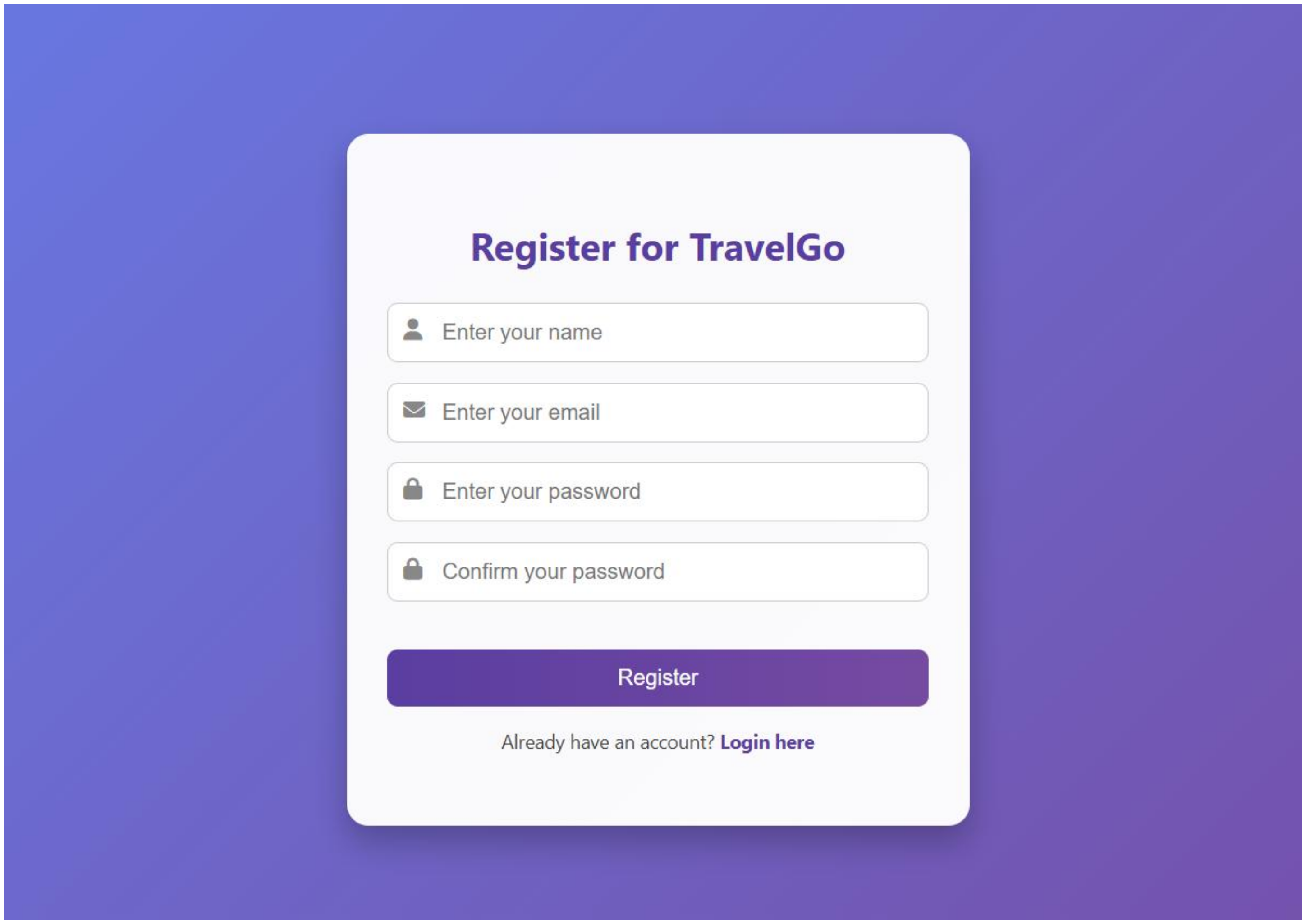
Milestone 8: Testing and Deployment

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

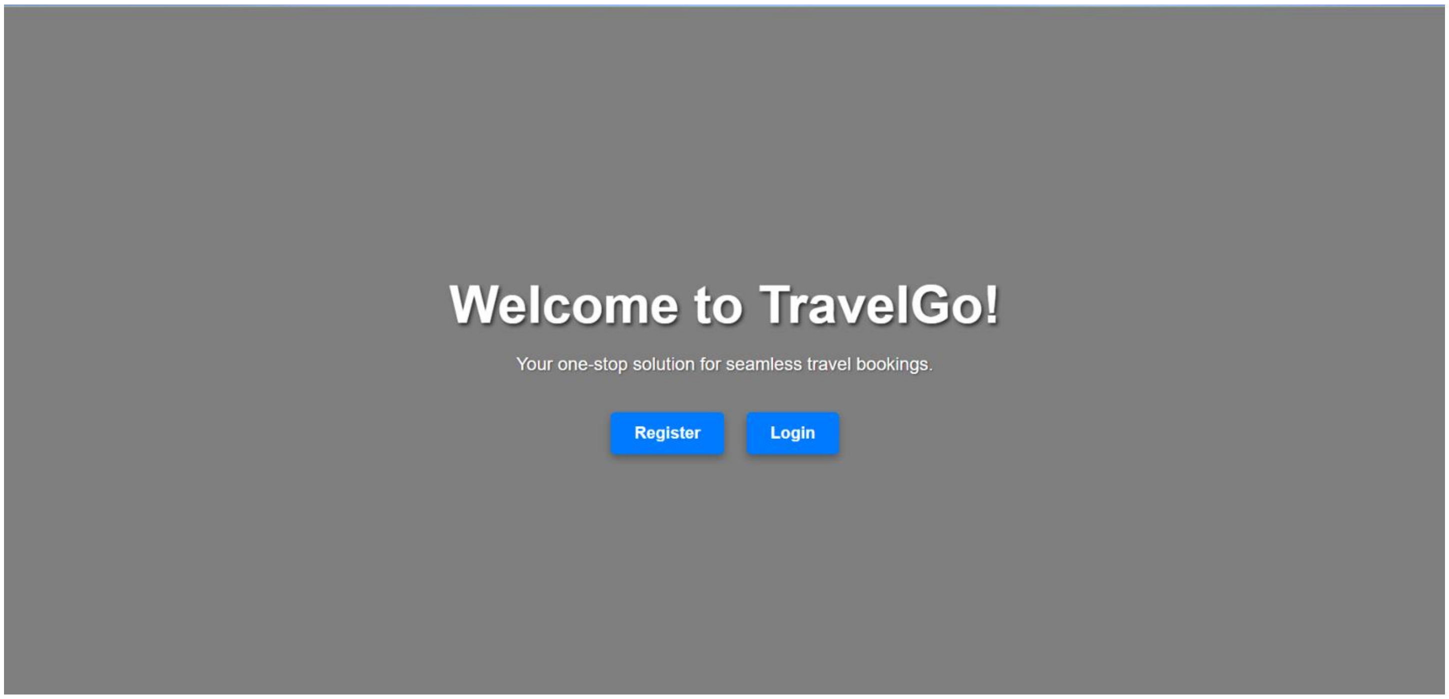
  Login Page:



Registration Page:



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.