

## TRAVEL GO : Smart Travel Booking and Information Platform

### Project Description:

Travel Go is a cloud-based travel solution that simplifies planning and booking through real-time destination information and seamless booking services. Built using Flask, AWS EC2, DynamoDB, and SNS, this system provides a scalable, user-friendly travel experience.

Travel Go addresses the issues faced by travelers who struggle with fragmented travel services, delayed information, and manual booking procedures. With a growing demand for digital, unified travel systems, Travel Go brings everything under one cloud-powered roof. It allows users to register, login, browse travel packages, and make bookings, all while receiving real-time email updates about their trip status.

### Scenario 1: Efficient Travel Assistance and Information System for Tourists

In the **Smart Travel System** at Greenfield Tourism Hub, **AWS EC2** provides a scalable and reliable infrastructure to support thousands of tourists accessing the platform simultaneously. For example, a traveler can log in, navigate to the itinerary planning page, and easily submit a request for information on unavailable or less-known destinations. **Flask** powers the backend, efficiently retrieving live data such as transportation schedules, local weather, and nearby attractions in real-time.

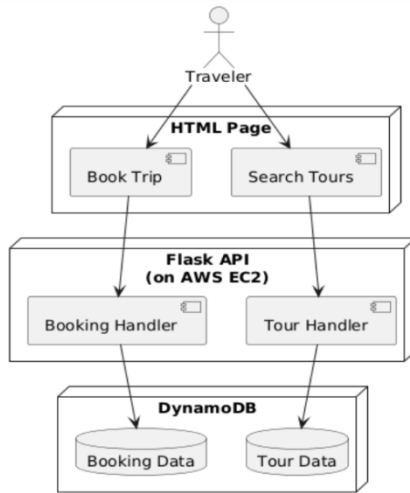
### Scenario 2: Seamless Travel Inquiry Notifications for Support Staff

When travelers inquire about unavailable services—such as guided tours, accommodation, or transport options—the **Smart Travel System** uses **AWS SNS** to notify both the user and the travel support team. For example, a tourist requests a local cultural tour that is currently unavailable. **Flask** handles the backend processing of the request, and **SNS** sends an instant email to both the traveler (confirming their inquiry) and the support team (notifying them of the request). All inquiries are securely stored in **AWS DynamoDB**, allowing staff to track and resolve requests efficiently, ensuring no traveler is left without assistance.

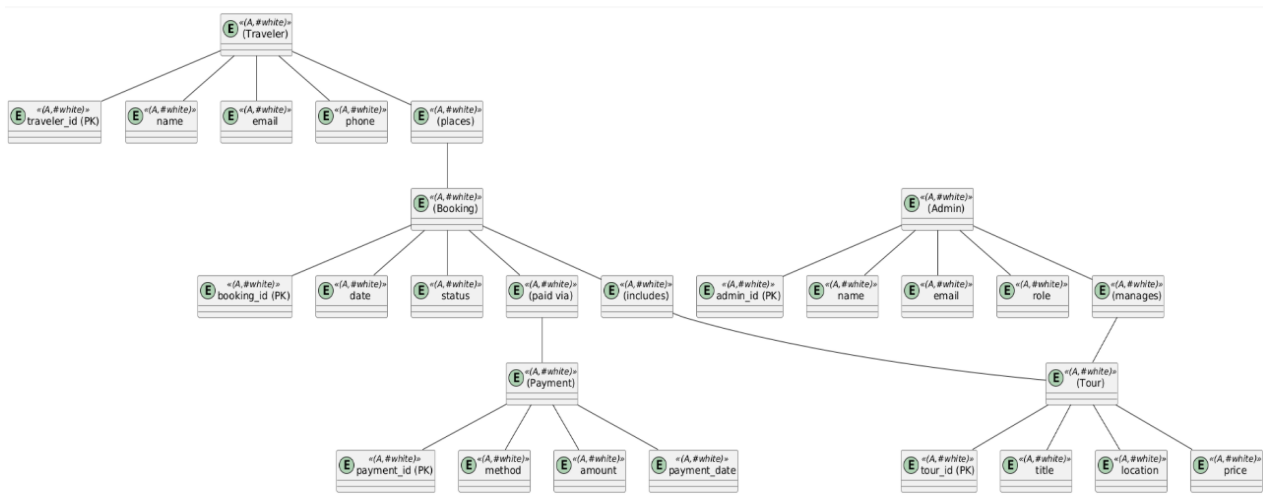
### Scenario 3: Easy Access to Travel Resources and Services

The **Smart Travel System** offers users simple, real-time access to available travel services and information. For example, a traveler logs in and explores a curated list of available tours, local attractions, transportation options, and accommodations. They can immediately check availability or submit a request for services that are not currently listed. **Flask** handles real-time data retrieval from **DynamoDB**, and the platform is hosted on **AWS EC2**, ensuring smooth performance even during holiday or festival seasons when user traffic peaks.

## AWS ARCHITECTURE



## Entity Relationship (ER)Diagram:



## Pre-requisites:

1. **AWS Account Setup:** [AWS Account Setup](#)
2. **Understanding IAM:** [IAM Overview](#)
3. **Amazon EC2 Basics:** [EC2 Tutorial](#)
4. **DynamoDB Basics:** [DynamoDB Introduction](#)
5. **SNS Overview:** [SNS Documentation](#)
6. **Git Version Control:** [Git Documentation](#)

## **Project WorkFlow:**

### **1. AWS Account Setup and Login**

**Activity 1.1:** Set up an AWS account if not already done.

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

### **2. DynamoDB Database Creation and Setup**

**Activity 2.1:** Create a DynamoDB Table.

**Activity 2.2:** Configure Attributes for User Data and Travel Requests.

### **3. SNS Notification Setup**

**Activity 3.1:** Create SNS topics for travel booking notifications.

**Activity 3.2:** Subscribe users and travel agents to SNS email notifications.

### **4. Backend Development and Application Setup**

**Activity 4.1:** Develop the Backend Using Flask.

**Activity 4.2:** Integrate AWS Services Using boto3.

### **5. IAM Role Setup**

**Activity 5.1:** Create IAM Role

**Activity 5.2:** Attach Policies

### **6. EC2 Instance Setup**

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

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

### **7. Deployment on EC2**

**Activity 7.1:** Upload Flask Files

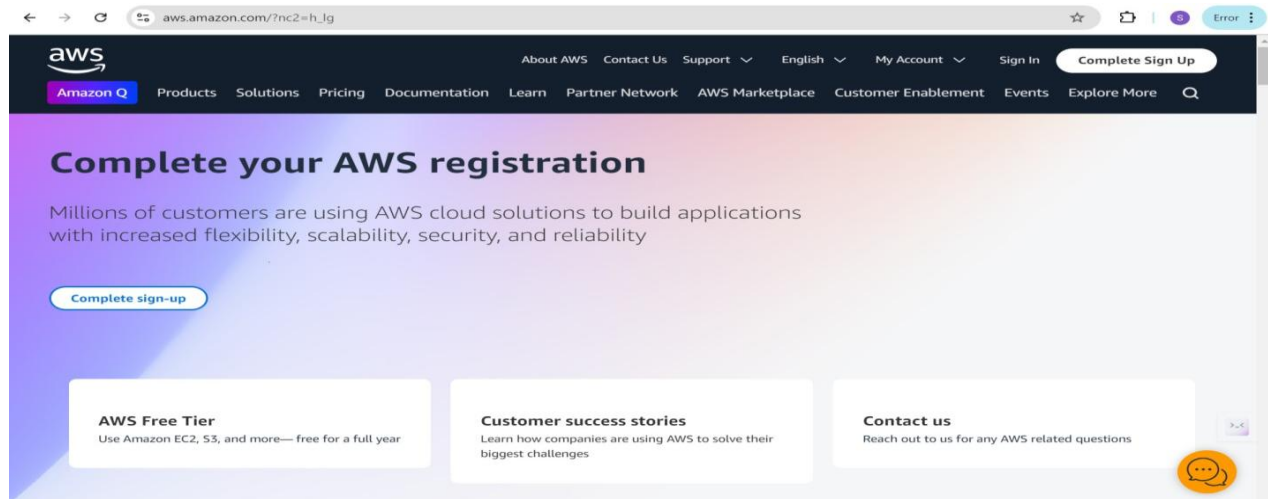
**Activity 7.2:** Run the Flask App

## 8. Testing and Deployment

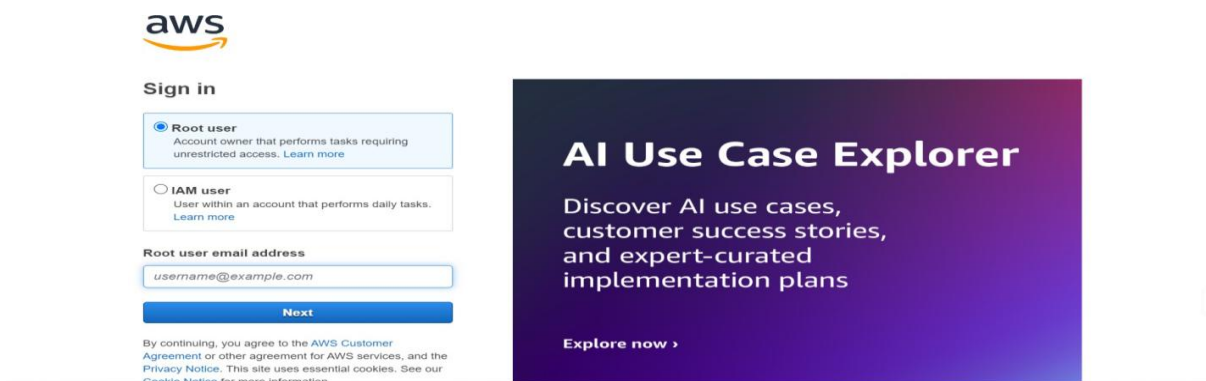
**Activity 8.1:** Conduct functional testing to verify user registration, login, travel bookings, and notifications.

### 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.

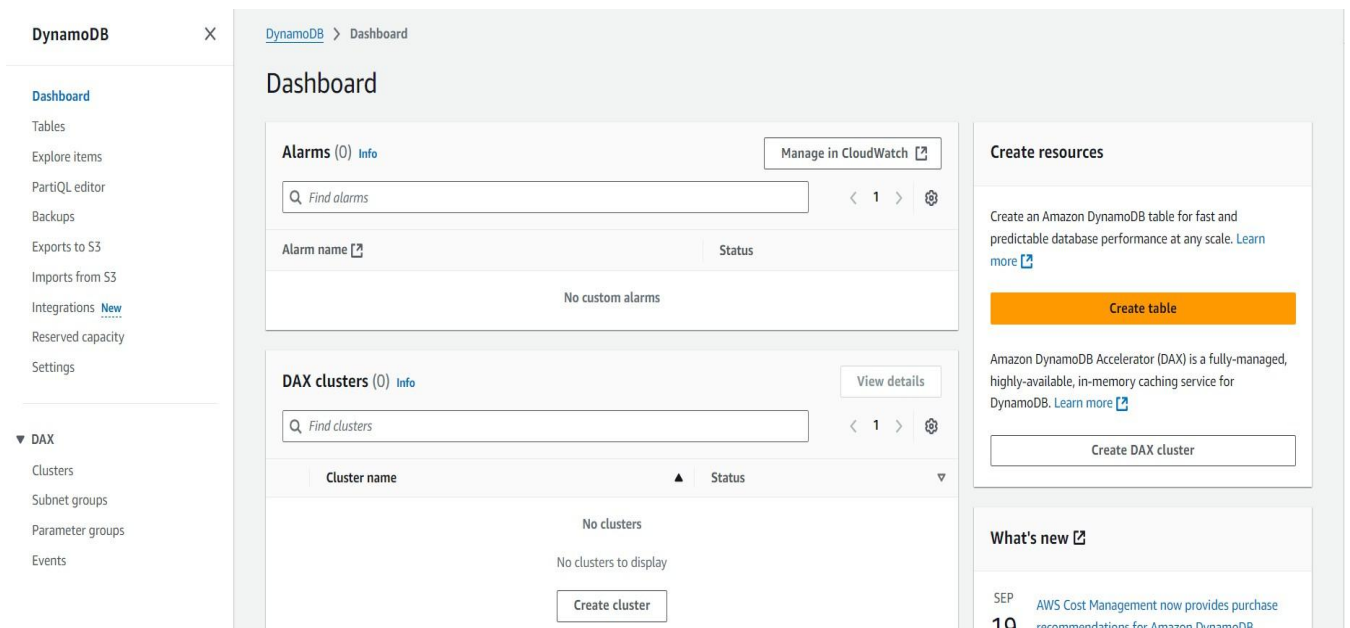
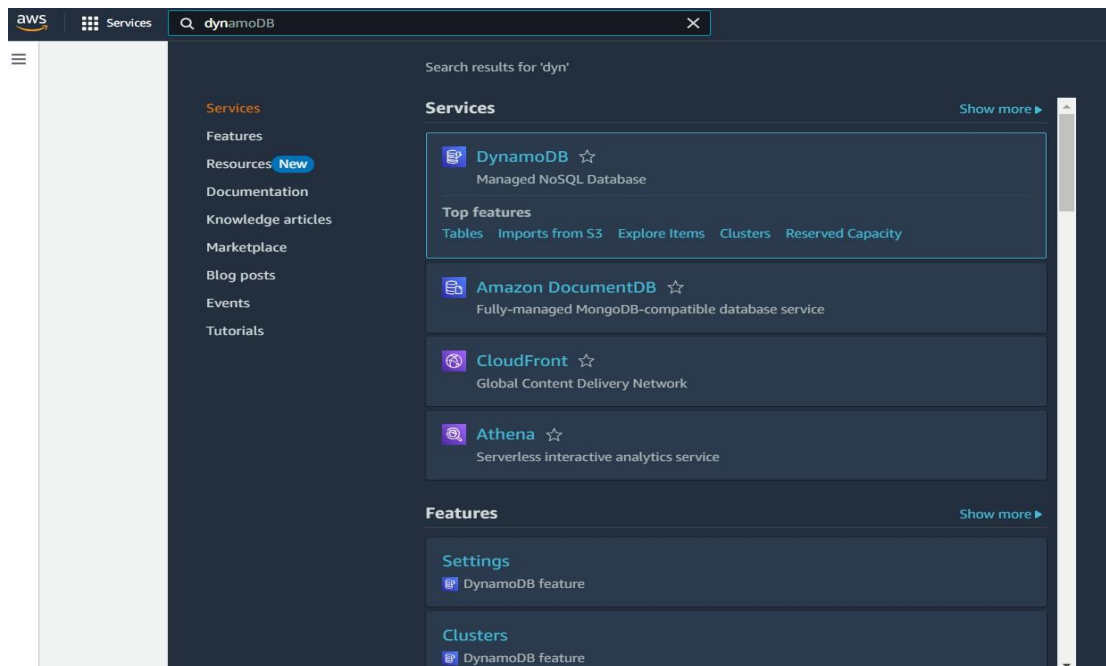


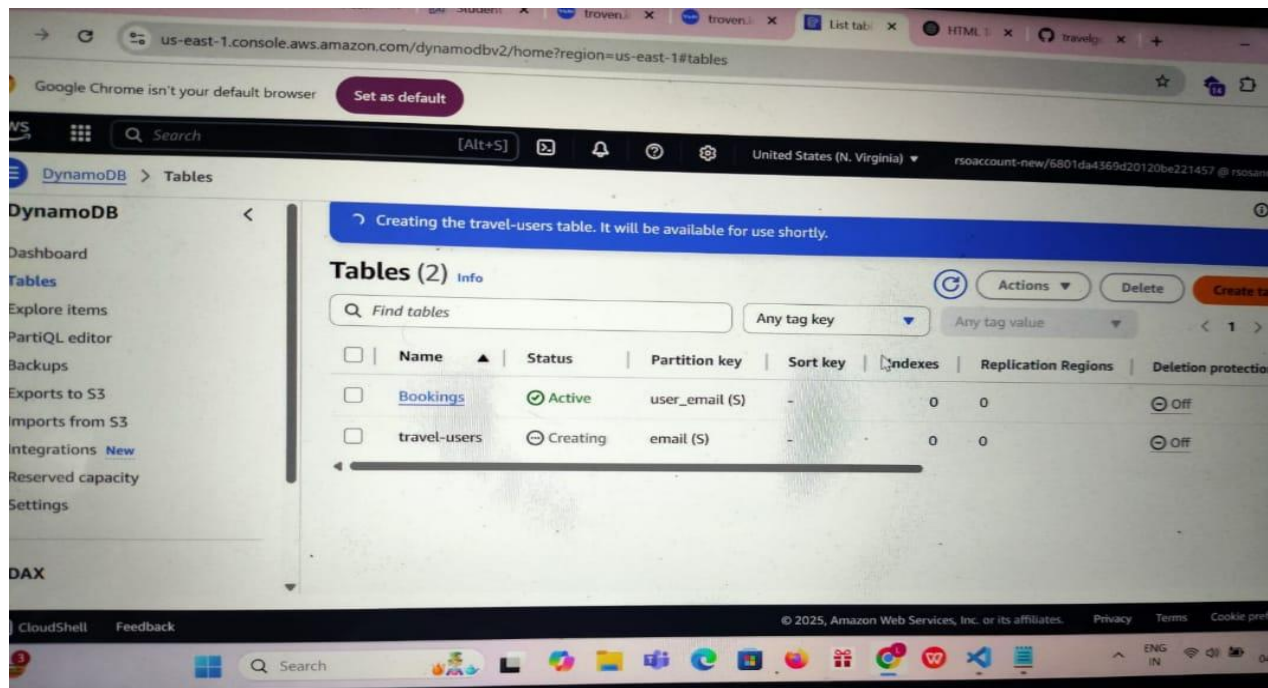
- **Activity 1.2: Log in to the AWS Management Console**
  - After setting up your account, log in to the [AWS Management Console](#).



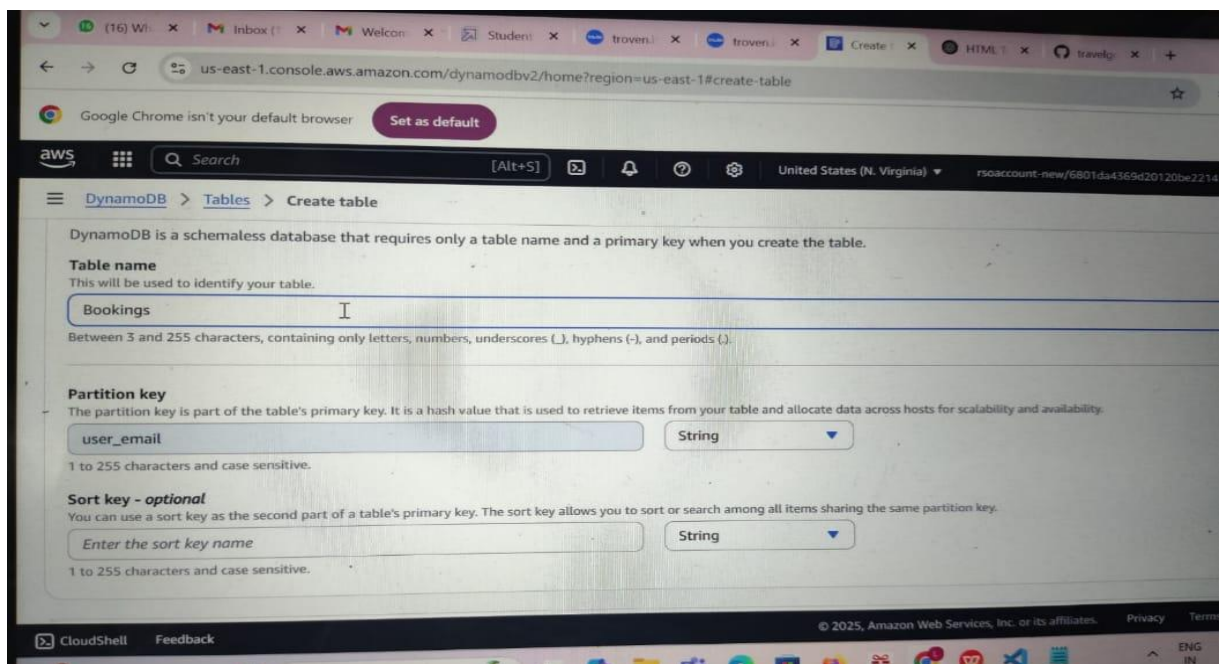
## Milestone 2: DynamoDB Database Creation and Setup

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

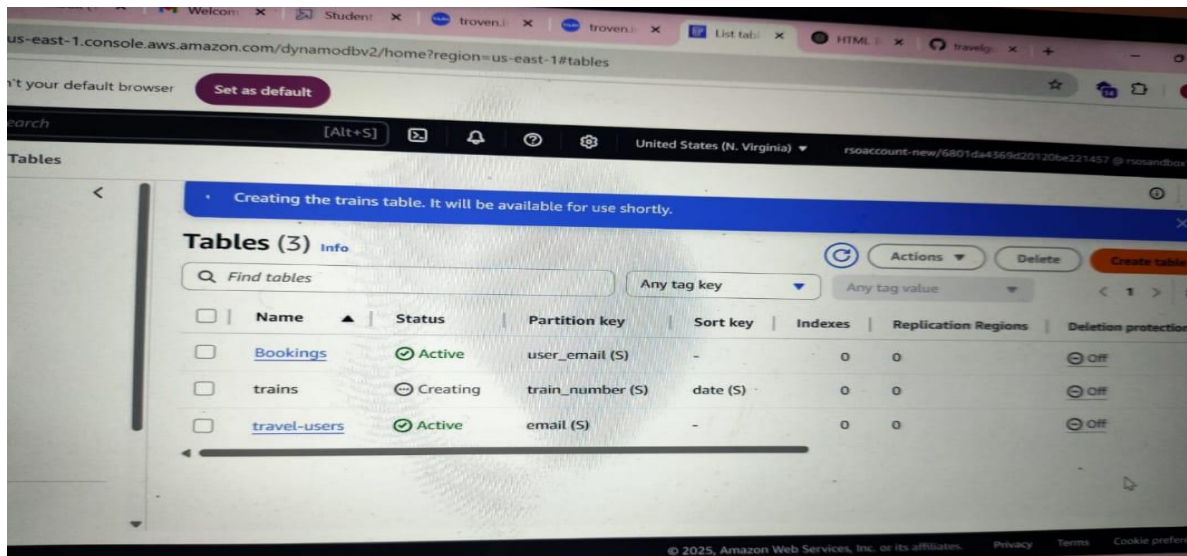




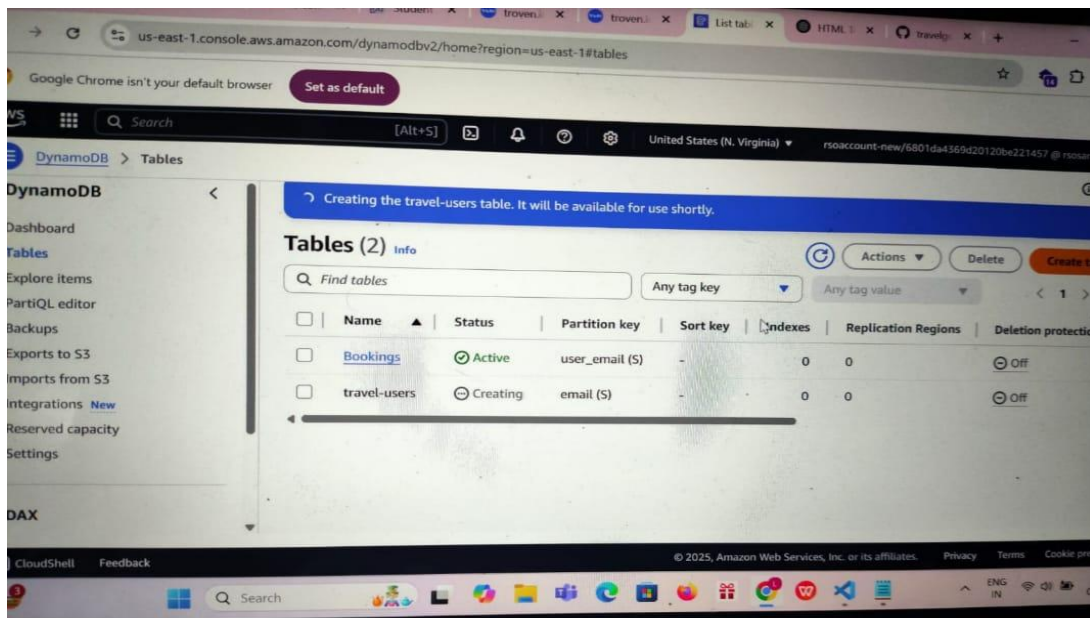
- **Activity 2.2: Create a DynamoDB table for storing registration details and book requests.**
  - Create Users table with partition key “Email” with type String and click on create tables.





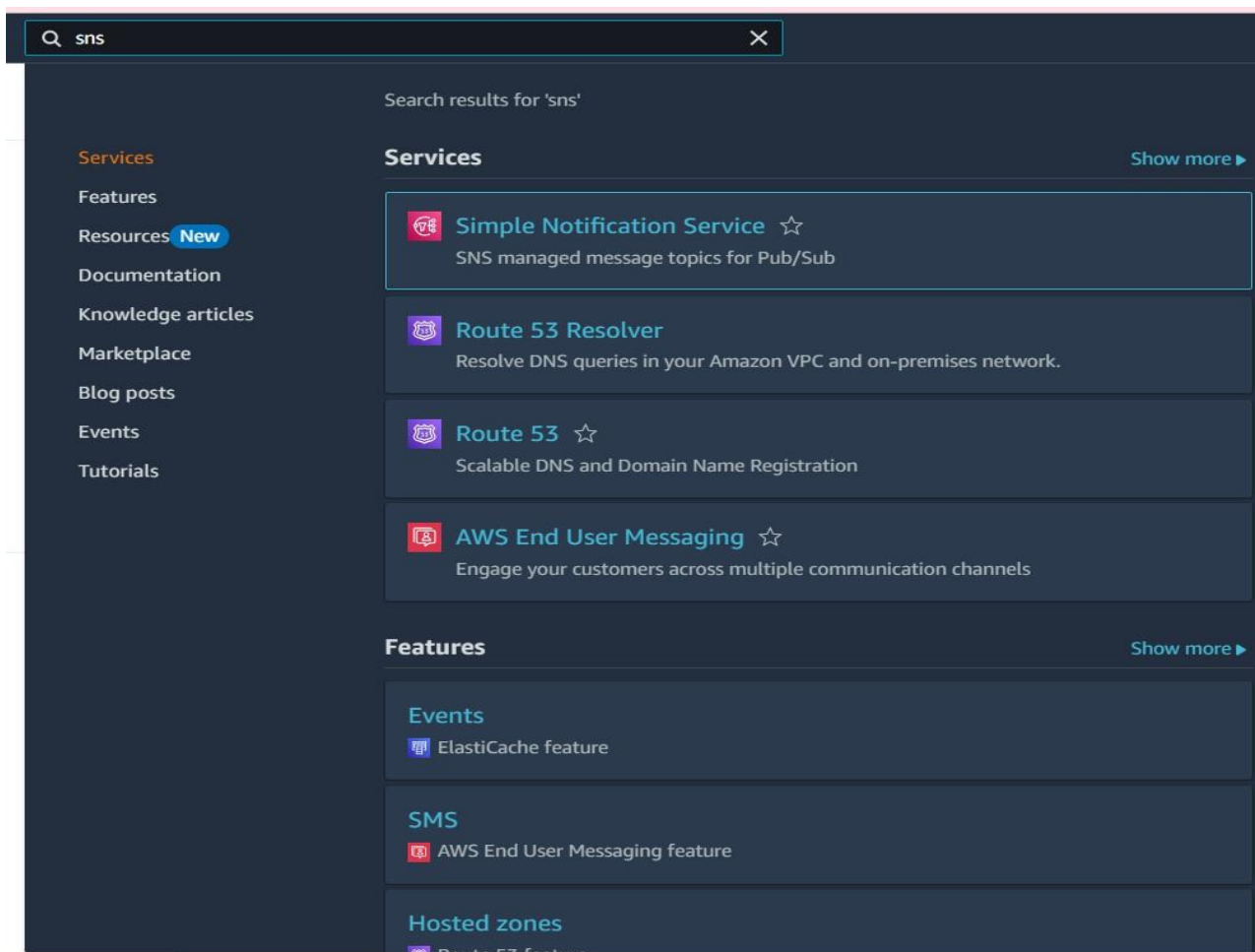


- Follow the same steps to create a requests table with Email as the primary key for book requests data.
- Choose Standard type for general notification use cases and Click on Create Topic.

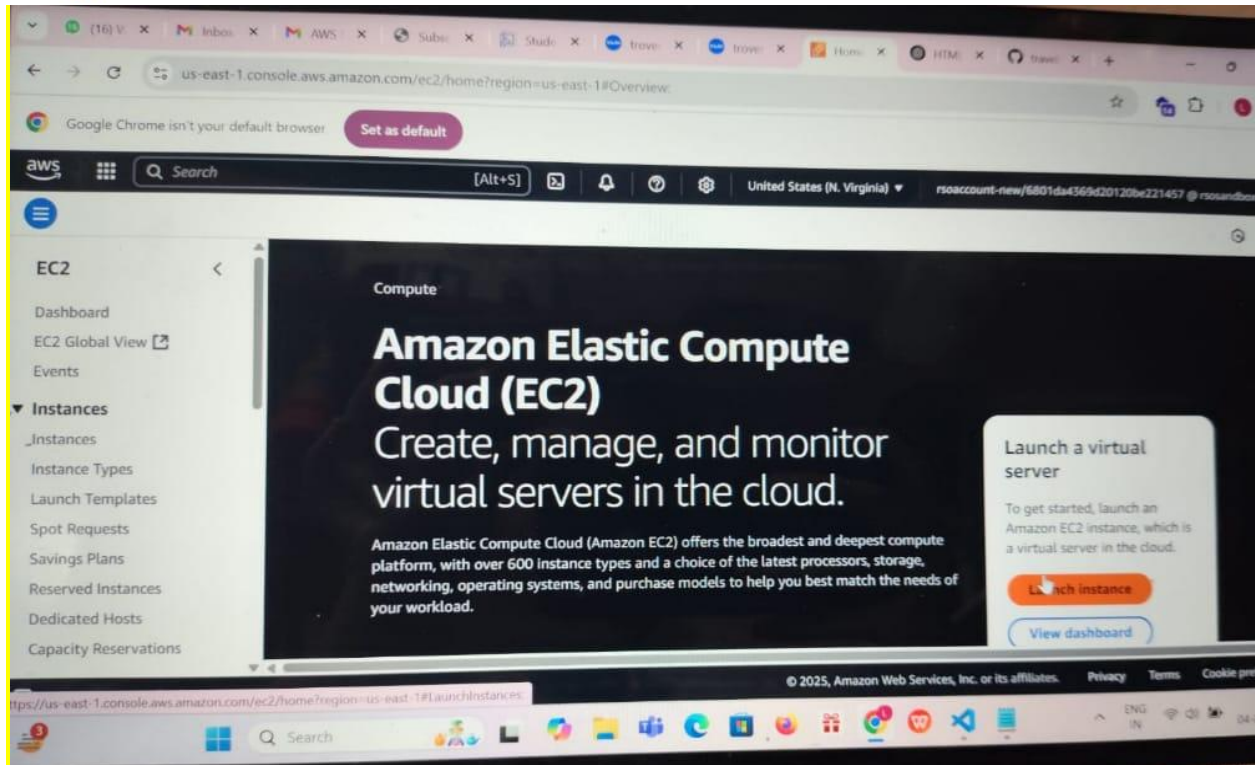


## Milestone 3: SNS Notification Setup

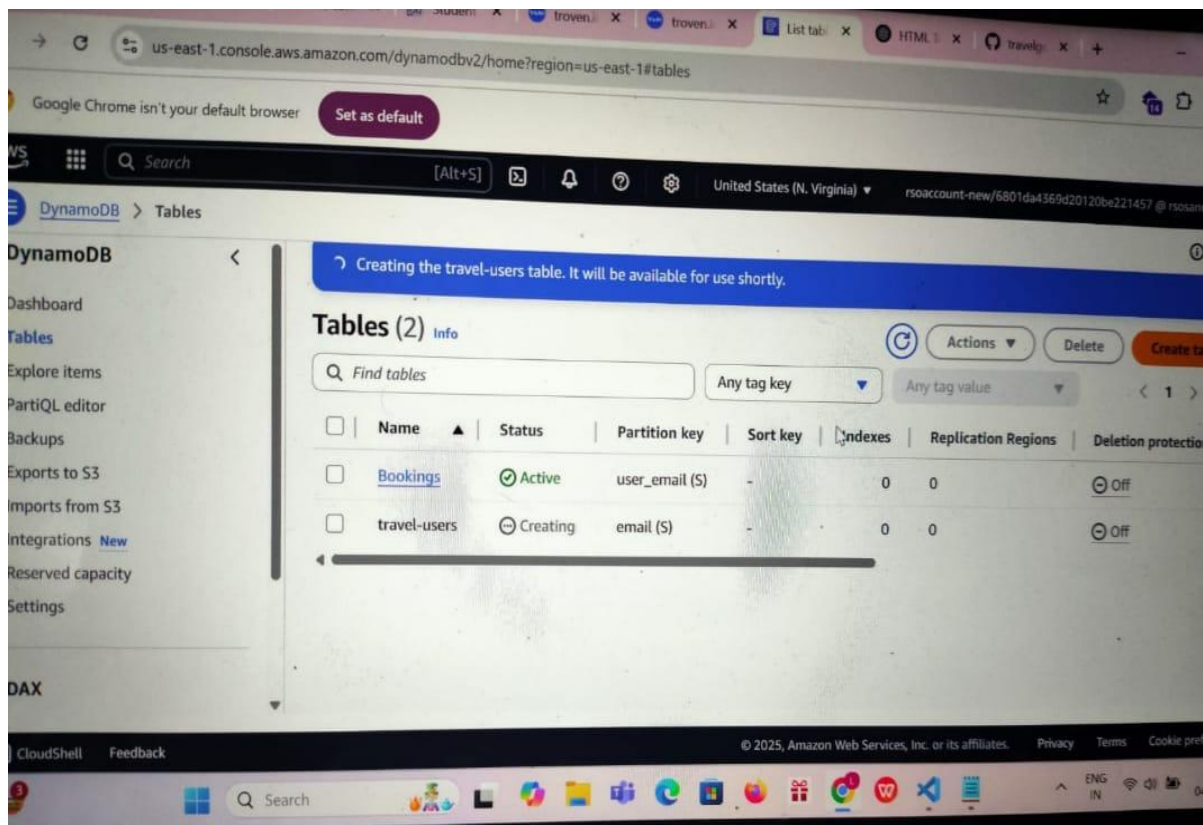
- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff**
- In the AWS Console, search for SNS and navigate to the SNS Dashboard.

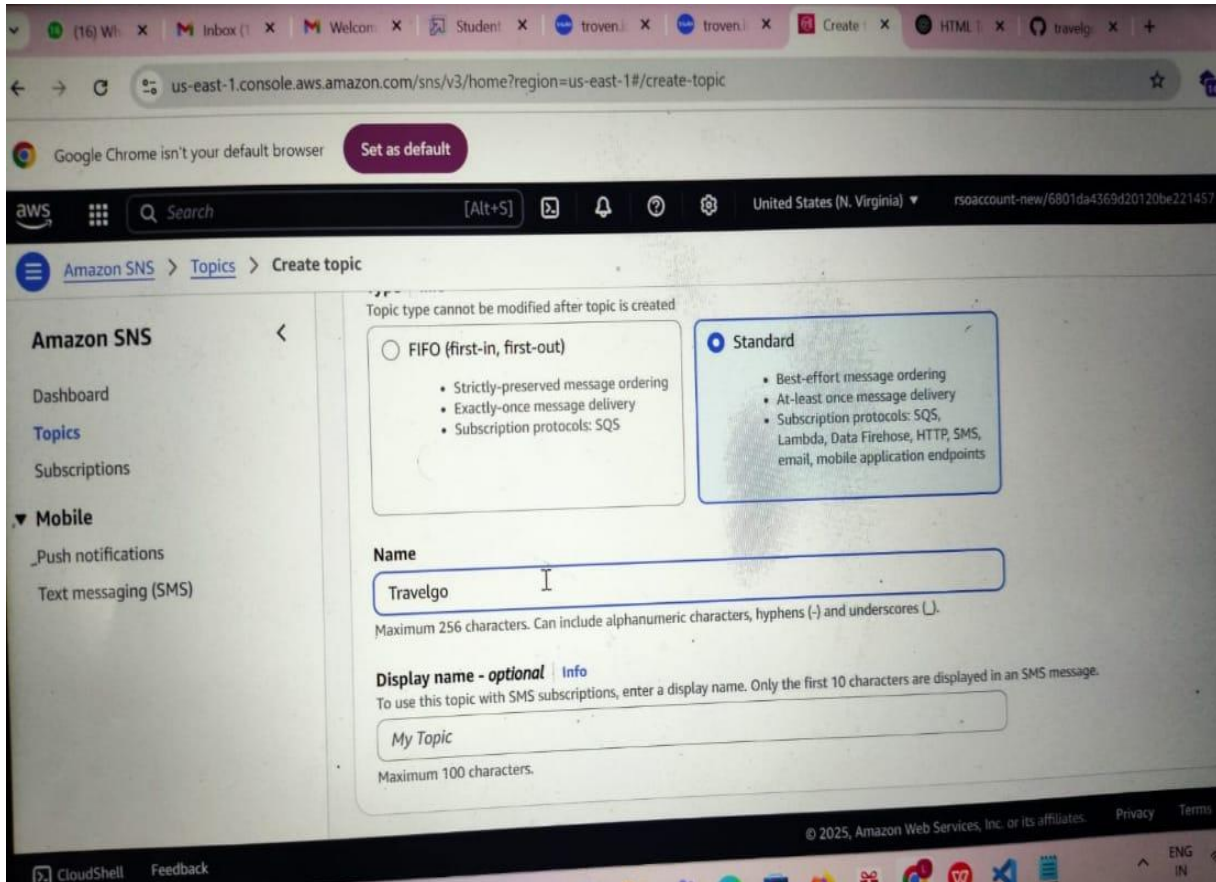






- Click on **Create Table** and choose a name





us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/create-topic

Google Chrome isn't your default browser [Set as default](#)

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

Amazon SNS > Topics > Create topic

**Amazon SNS**

- Dashboard
- Topics
- Subscriptions

▼ Mobile

- Push notifications
- Text messaging (SMS)

Topic type cannot be modified after topic is created

☐ FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- Subscription protocols: SQS

☒ Standard

- Best-effort message ordering
- At-least once message delivery
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

**Name**

Travelgo

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

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

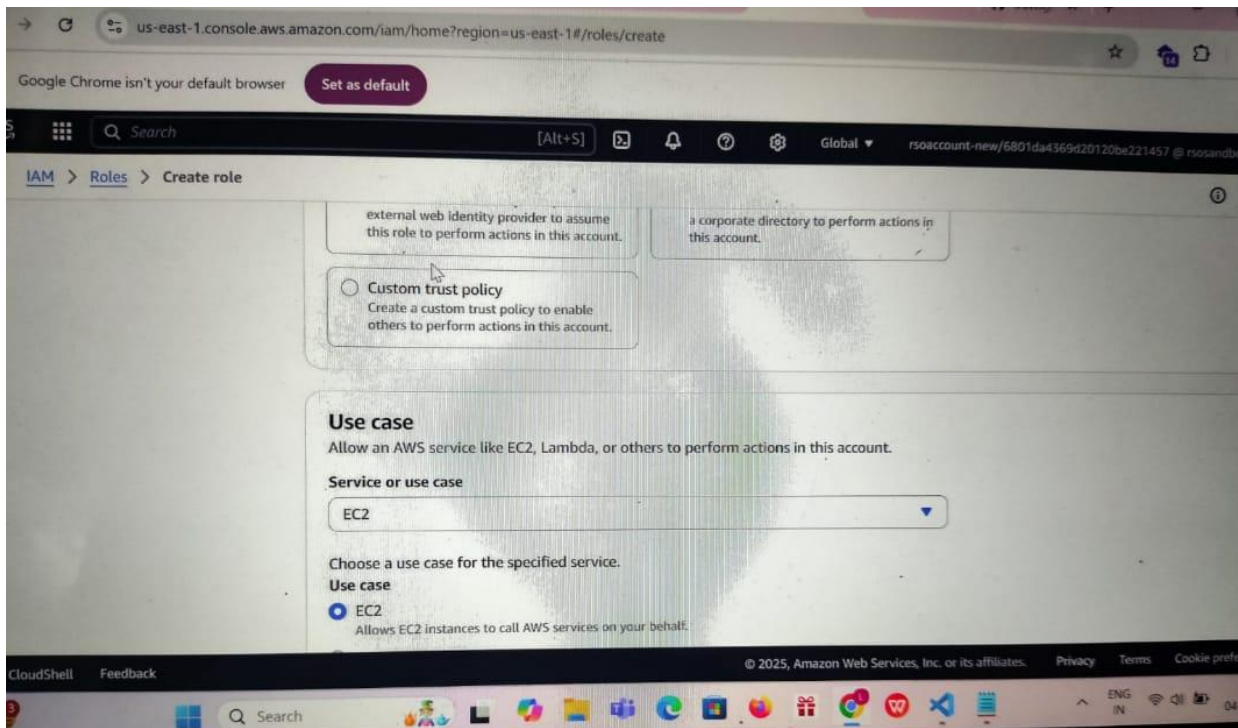
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

My Topic

Maximum 100 characters.

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

CloudShell Feedback



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

Google Chrome isn't your default browser [Set as default](#)

Search [Alt+S] Global rsoaccount-new/6801da4369d20120be221457 @ rsoaccount

IAM > Roles > Create role

external web identity provider to assume this role to perform actions in this account.

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

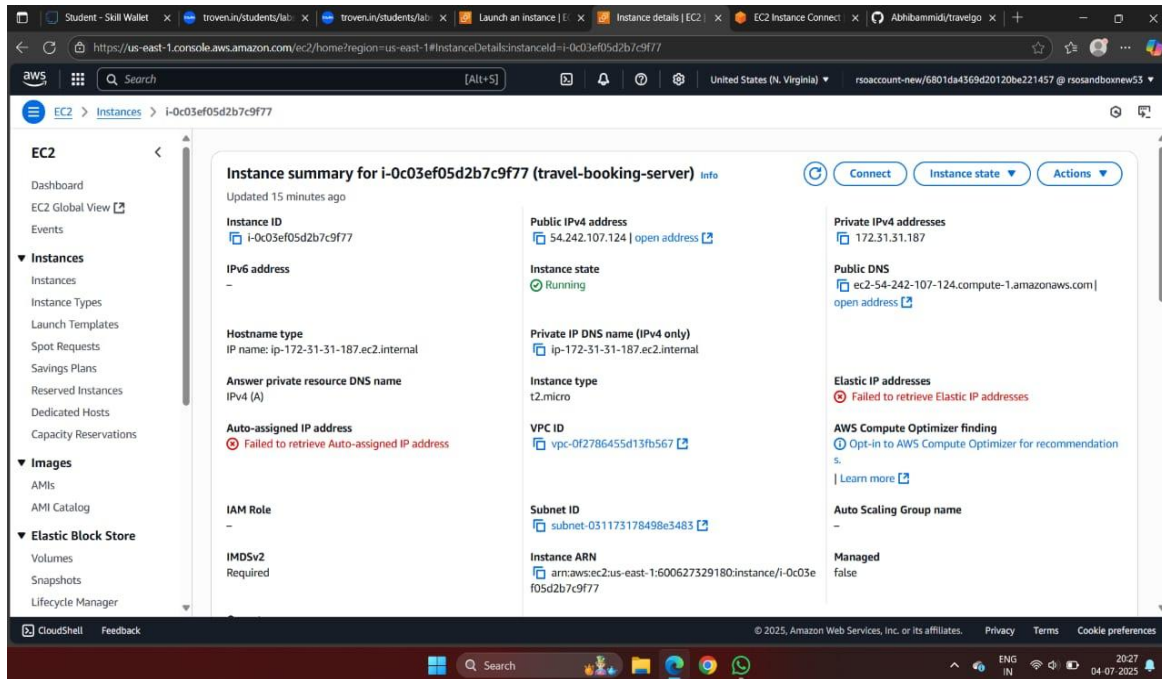
Allows EC2 instances to call AWS services on your behalf.

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

CloudShell Feedback

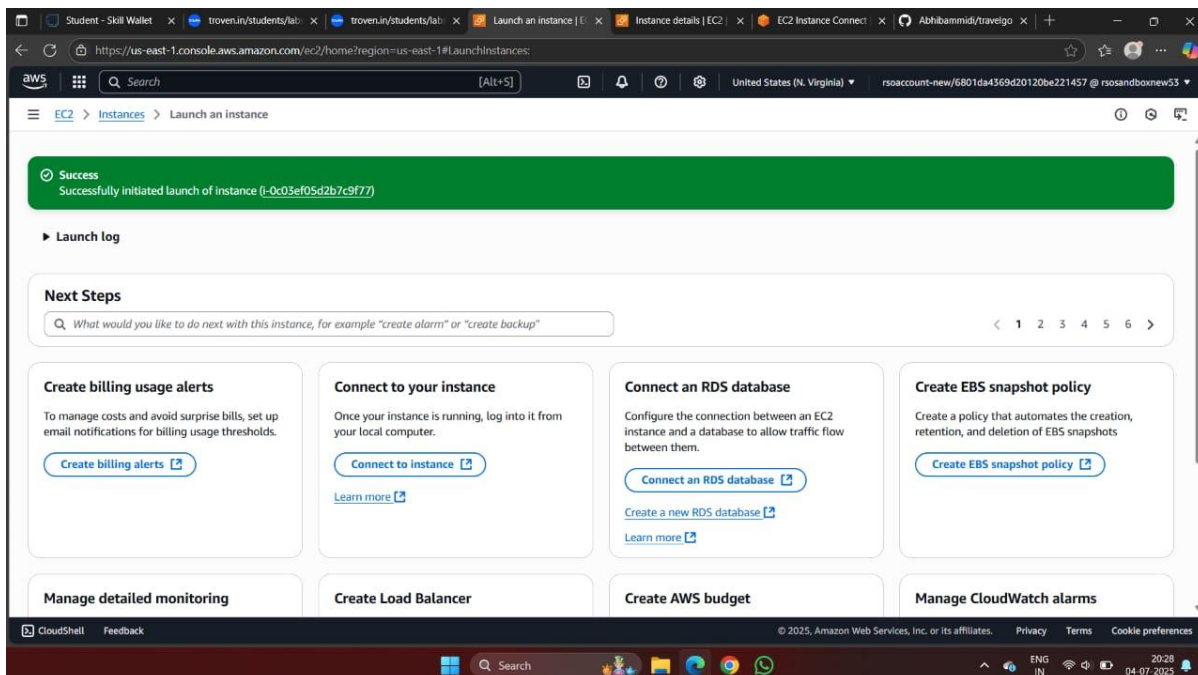
- Configure the SNS topic and note down the **Topic ARN**

- **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.



The screenshot shows the AWS Management Console for an EC2 instance named 'travel-booking-server' with ID 'i-0c03ef05d2b7c9f77'. The instance is in the 'Running' state. Key details include:

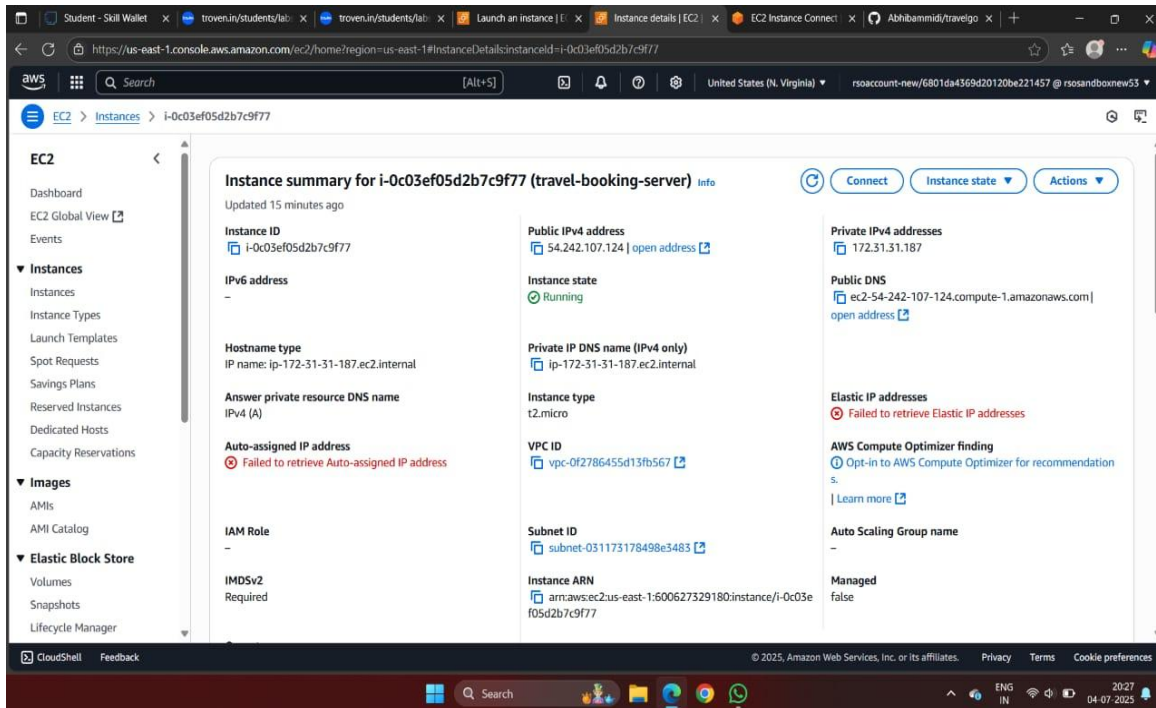
- Instance ID:** i-0c03ef05d2b7c9f77
- Public IPv4 address:** 54.242.107.124
- Private IPv4 addresses:** 172.31.31.187
- Public DNS:** ec2-54-242-107-124.compute-1.amazonaws.com
- Instance state:** Running
- Private DNS name (IPv4 only):** ip-172-31-31-187.ec2.internal
- Instance type:** t2.micro
- VPC ID:** vpc-0f2786455d13fb567
- Subnet ID:** subnet-031173178498e3483
- Instance ARN:** arn:aws:ec2:us-east-1:600627329180:instance/i-0c03ef05d2b7c9f77
- Auto-assigned IP address:** Failed to retrieve Auto-assigned IP address
- Answer private resource DNS name:** IPv4 (A)
- IAM Role:** -
- IMDSv2:** Required
- Managed:** false



The screenshot shows the 'Launch an instance' page in the AWS Management Console. A green success banner at the top states: 'Successfully initiated launch of instance (i-0c03ef05d2b7c9f77)'. Below this, the 'Next Steps' section provides several actionable items:

- Create billing usage alerts:** To manage costs and avoid surprise bills, set up email notifications for billing usage thresholds. [Create billing alerts](#)
- Connect to your instance:** Once your instance is running, log into it from your local computer. [Connect to instance](#)
- 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 EBS snapshot policy:** Create a policy that automates the creation, retention, and deletion of EBS snapshots. [Create EBS snapshot policy](#)
- Manage detailed monitoring**
- Create Load Balancer**
- Create AWS budget**
- Manage CloudWatch alarms**



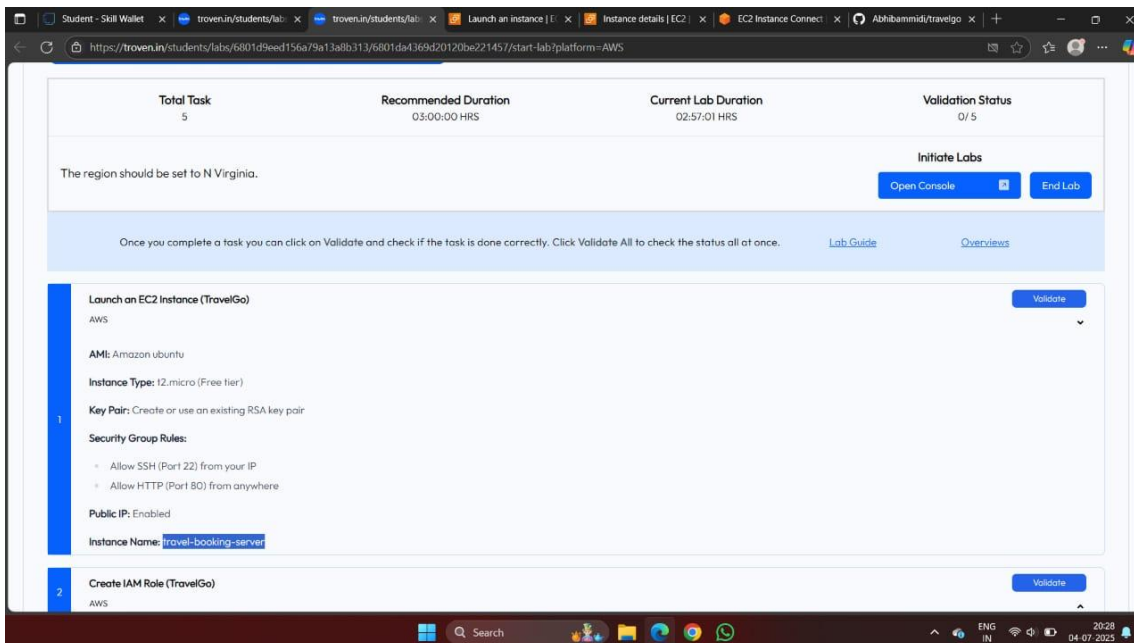


The screenshot shows the AWS Management Console interface for an EC2 instance. The instance ID is `i-0c03ef05d2b7c9f77` and it is named `travel-booking-server`. The instance is in the `Running` state. Key details include:

- Public IPv4 address:** 54.242.107.124
- Private IPv4 address:** 172.31.31.187
- Public DNS:** ec2-54-242-107-124.compute-1.amazonaws.com
- Private IP DNS name (IPv4 only):** ip-172-31-31-187.ec2.internal
- Instance type:** t2.micro
- VPC ID:** vpc-0f278645d13fb567
- Subnet ID:** subnet-031173178498e3483
- Instance ARN:** arn:aws:ec2:us-east-1:600627329180:instance/i-0c03ef05d2b7c9f77

There are some error messages indicating failures to retrieve the Auto-assigned IP address and Elastic IP addresses.

- After subscription request for the mail confirmation

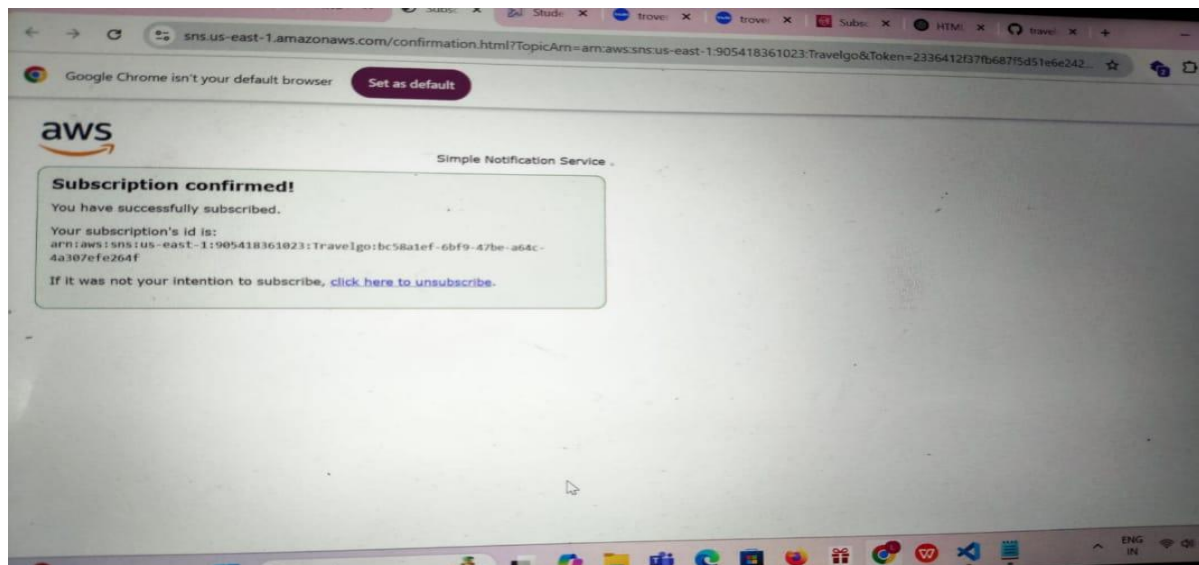
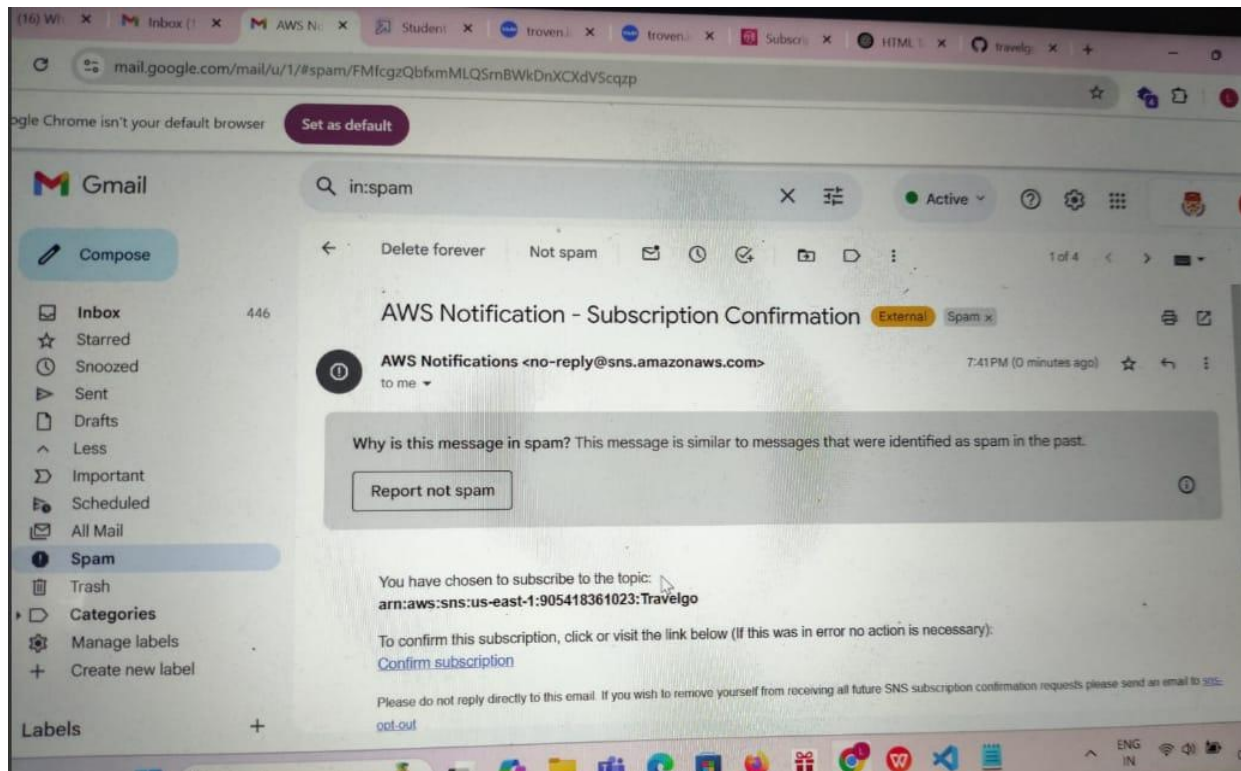


The screenshot shows the AWS Lab interface with a task titled "Launch an EC2 Instance (TravelGo)". The task is part of a sequence of 5 tasks, and the current duration is 02:57:01 HRS. The validation status is 0/5. The task instructions are as follows:

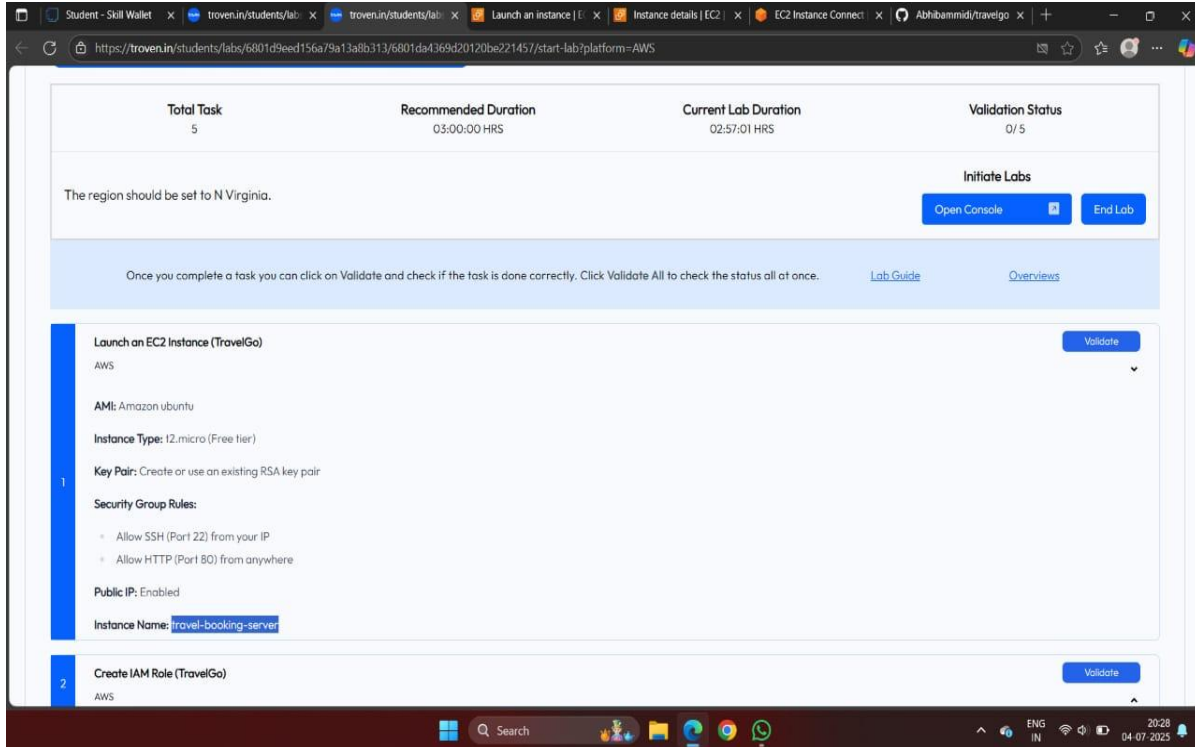
- Launch an EC2 Instance (TravelGo)**
  - AMI:** Amazon ubuntu
  - Instance Type:** t2.micro (Free tier)
  - Key Pair:** Create or use an existing RSA key pair
  - Security Group Rules:**
    - Allow SSH (Port 22) from your IP
    - Allow HTTP (Port 80) from anywhere
  - Public IP:** Enabled
  - Instance Name:** travel-booking-server
- Create IAM Role (TravelGo)**

Buttons for "Initiate Labs", "Open Console", "End Lab", and "Validate" are visible.

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



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



The screenshot shows a web browser window with multiple tabs. The active tab is 'Launch an instance | F...', displaying the 'troven.in' lab interface. The URL is 'https://troven.in/students/labs/6801d9eed156a79a13a8b313/6801da4369d20120be221457/start-lab?platform=AWS'.

At the top, a summary bar shows:

- Total Task:** 5
- Recommended Duration:** 03:00:00 HRS
- Current Lab Duration:** 02:57:01 HRS
- Validation Status:** 0/5

Below this, a message states: 'The region should be set to N Virginia.' To the right are buttons for 'Initiate Labs', 'Open Console', and 'End Lab'.

A blue banner below the message reads: 'Once you complete a task you can click on Validate and check if the task is done correctly. Click Validate All to check the status all at once.' with links for 'Lab Guide' and 'Overviews'.

The main content area lists tasks:

- 1. Launch an EC2 Instance (TravelGo)** (AWS)
  - AMI: Amazon ubuntu
  - Instance Type: t2.micro (Free tier)
  - Key Pair: Create or use an existing RSA key pair
  - Security Group Rules:
    - Allow SSH (Port 22) from your IP
    - Allow HTTP (Port 80) from anywhere
  - Public IP: Enabled
  - Instance Name: travel-booking-server
- 2. Create IAM Role (TravelGo)** (AWS)

Each task has a 'Validate' button.

The Windows taskbar at the bottom shows the time as 20:28 on 04-07-2023, with system icons for network, volume, and battery.

## Milestone 4: Backend Development and Application Setup

- **Activity 4.1: Develop the backend using Flask**

**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.



## 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:

```
# Initialize DynamoDB resource
dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')

# DynamoDB Tables
users_table = dynamodb.Table('Users') # Ensure the 'Users' table
requests_table = dynamodb.Table('Requests') # Ensure the 'Requests' table
```

**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**

```
# SNS Topic ARN (create the SNS topic in AWS and provide the ARN here)
sns = boto3.client('sns', region_name='ap-south-1')
sns_topic_arn = 'arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications'

# Email settings (for sending emails)
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587
SENDER_EMAIL = "instantlibrary2@gmail.com"
SENDER_PASSWORD = "luut dsih nyvq dgzv" # Your app password

# Function to send email
def send_email(to_email, subject, body):
    msg = MIMEText(body, 'plain')
    msg['From'] = SENDER_EMAIL
    msg['To'] = to_email
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'plain'))

    try:
        server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        server.starttls()
        server.login(SENDER_EMAIL, SENDER_PASSWORD)
        text = msg.as_string()
        server.sendmail(SENDER_EMAIL, to_email, text)
        server.quit()
        print("Email sent successfully")
    except Exception as e:
        print(f"Failed to send email: {e}")
```

**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.

- Register Route:

```
# Registration Page
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        password = request.form['password']
        confirm_password = request.form['confirm_password']

        # Basic Validation: Ensure all fields are filled
        if not name or not email or not password or not confirm_password:
            return "All fields are mandatory! Please fill out the entire form."
        if password != confirm_password:
            return "Passwords do not match! Please try again."

        # Check if user already exists
        response = users_table.get_item(Key={'email': email})
        if 'Item' in response:
            return "User already exists! Please log in."

        # Hash the password
        hashed_password = hashpw(password.encode('utf-8'), gensalt()).decode('utf-8')

        # Store user in DynamoDB with login_count initialized to 0
        users_table.put_item(
            Item={
                'email': email,
                'name': name,
                'password': hashed_password,
                'login_count': 0
            }
        )

        # Send SNS notification for new registration
        sns.publish(
            TopicArn=sns_topic_arn,
            Message=f'New user registered: {name} ({email})',
            Subject='New User Registration'
        )

        return redirect(url_for('login'))
    return render_template('register.html')
```

**Description:** define /register route to validate registration form fields, hash the user password using Bcrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

- **login Route (GET/POST):**

```
# Login Page
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        # Basic Validation: Ensure both fields are filled
        if not email or not password:
            return "Please enter both email and password."

        # Fetch user data from DynamoDB
        response = users_table.get_item(Key={'email': email})
        user = response.get('Item')

        if not user or not checkpw(password.encode('utf-8'), user['password'].encode('utf-8')):
            return "Incorrect email or password! Please try again."

        # Update login count
        users_table.update_item(
            Key={'email': email},
            UpdateExpression='SET login_count = login_count + :inc',
            ExpressionAttributeValues={':inc': 1}
        )

        # Successful login
        return redirect(url_for('home_page'))
    return render_template('login.html')
```

**Description:** define /login route to validate user credentials against DynamoDB, check the password using Bcrypt, update the login count on successful authentication, and redirect users to the home page

- **Home, E- book buttons and subject routes:**

```
# Home Page with E-Books, Request Books, and Exit
@app.route('/home-page')
def home_page():
    return render_template('home.html')

# E-Books Page (Dropdown Selection for Course and Subject)
@app.route('/ebook-buttons', methods=['GET', 'POST'])
def ebook_buttons():
    if request.method == 'POST':
        subject = request.form['subject']
        return redirect(url_for('subject_page', subject=subject))
    return render_template('ebook-buttons.html')

# Subject Page (Example with Mathematics)
@app.route('/<subject>.html')
def subject_page(subject):
    return render_template(f'{subject}.html')
```

**Description:** define /home-page to render the main homepage, /ebook-buttons to handle subject selection and redirection, and /<subject>.html dynamic route to render subject-specific pages like Mathematics or English.

- **Request Routes:**

```
# Book Request Form Page
@app.route('/request-form', methods=['GET', 'POST'])
def request_form():
    if request.method == 'POST':
        # Retrieve form data from the POST request
        email = request.form['email'] # Capture email to send thank-you note
        name = request.form['name']
        year = request.form['year']
        semester = request.form['semester']
        roll_no = request.form['roll-no']
        subject = request.form['subject']
        book_name = request.form['book-name']
        description = request.form['description']

        # Store book request in DynamoDB along with the user email
        requests_table.put_item(
            Item={
                'email': email,
                'roll_no': roll_no,
                'name': name,
                'year': year,
                'semester': semester,
                'subject': subject,
                'book_name': book_name,
                'description': description
            }
        )

        # Send a thank-you email to the requesting user
        thank_you_message = f"Dear {name},\n\nThank you for submitting a book request for '{book_name}'. We will send_email(email, "Thank You for Your Book Request", thank_you_message)

        # Send an email to the Instant Library admin with the book request details
        admin_message = f"User {name} ({email}) has requested the book '{book_name}'.\n\nDetails:\nYear: {year}\nSubject: {subject}\nBook Name: {book_name}\nDescription: {description}"
        send_email("instantlibrary2@gmail.com", "New Book Request", admin_message)

        return "<h3>Book request submitted successfully! We will get back to you soon.</h3>"

    # Render the request form for GET requests
    return render_template('request-form.html')
```

**Description:** define /request-form route to capture book request details from users, store the request in DynamoDB, send a thank-you email to the user, notify the admin, and confirm submission with a success message.

**Exit Route:**

```
# Exit Page
@app.route('/exit')
def exit_page():
    return render_template('exit.html')
```

**Description:** define /exit route to render the exit.html page when the user chooses to leave or close the application.



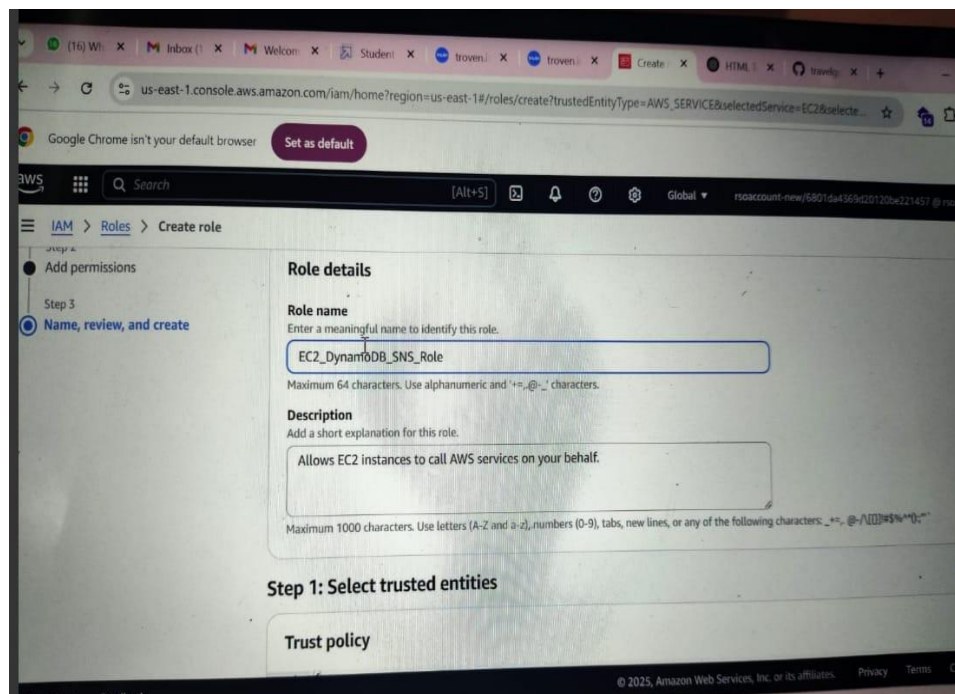
### Deployment Code:

```
if __name__ == "__main__":  
    app.run(host='0.0.0.0', port=80, debug=True)
```

**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.

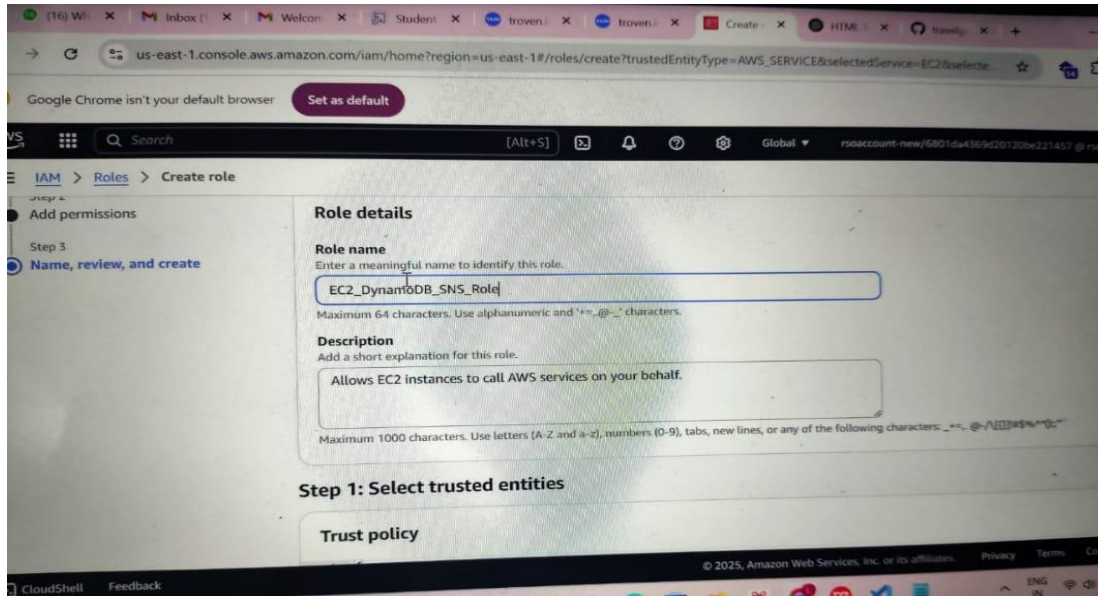




### ○ Activity 5.2: Attach Policies.




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.



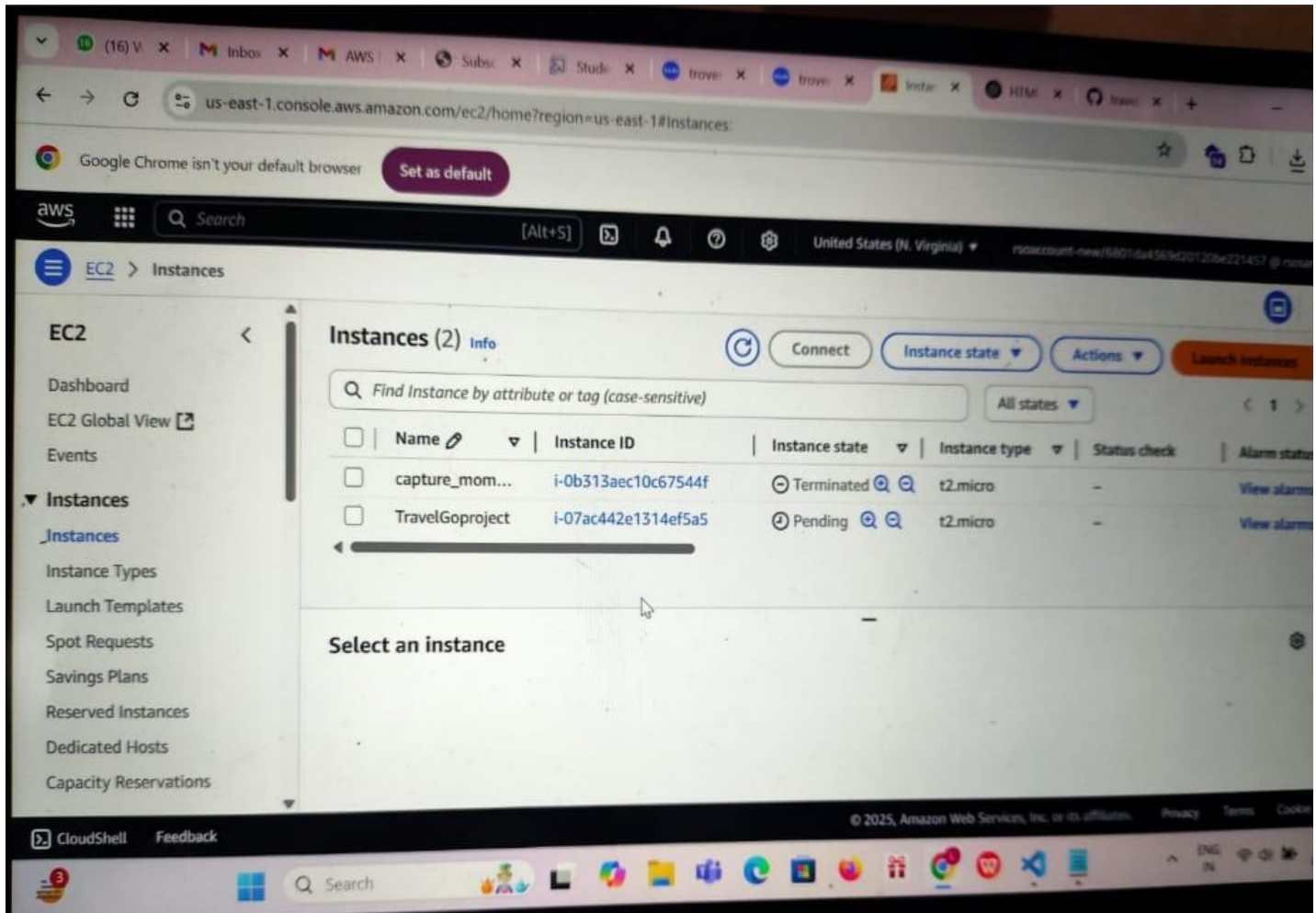
## Milestone 6: EC2 Instance Setup

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

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

### 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**



- 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

64-bit (x86)

### Boot mode

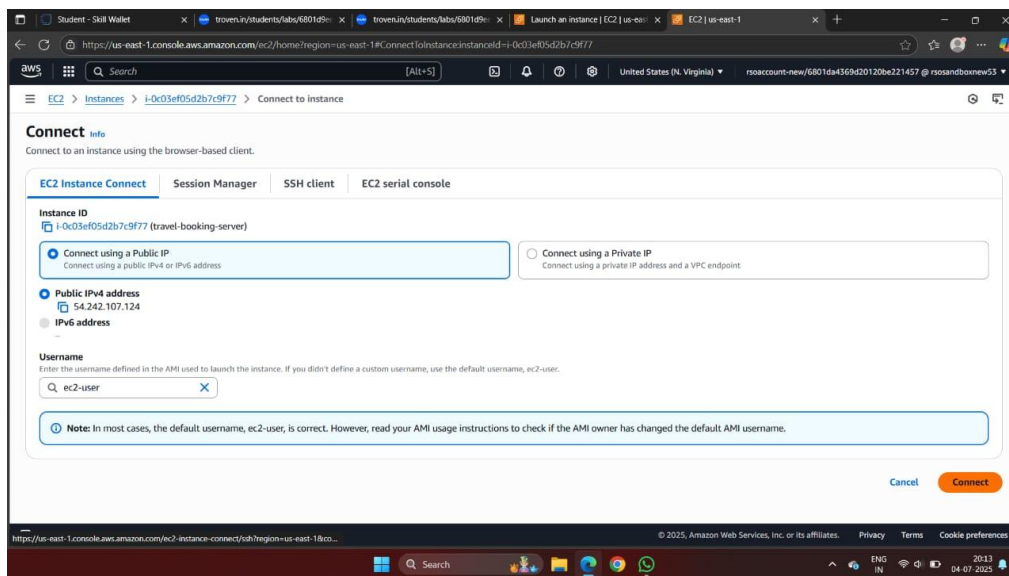
uefi-preferred

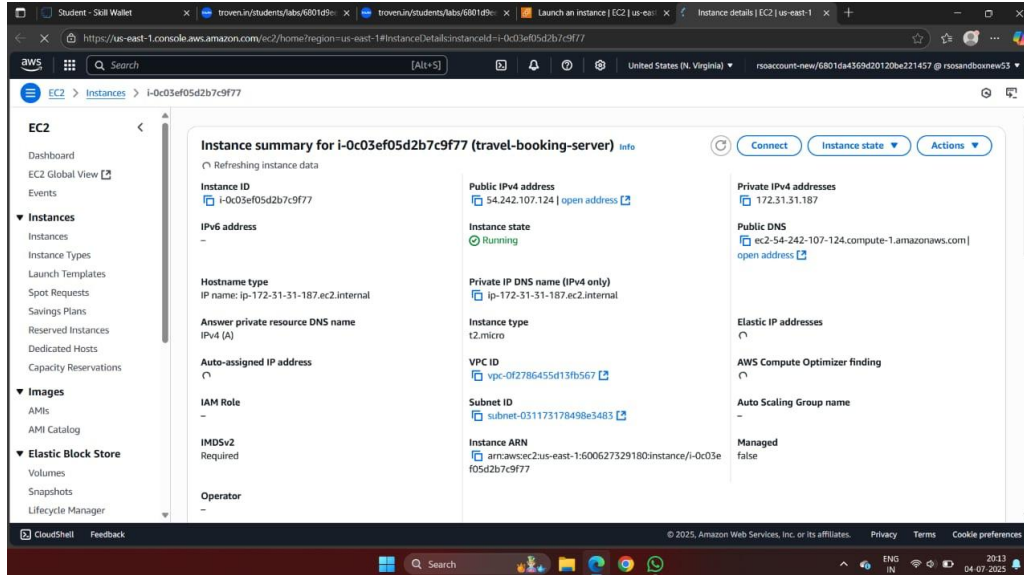
### AMI ID

ami-02b49a24cfb95941c

Verified provider

- Create and download the key pair for Server access.



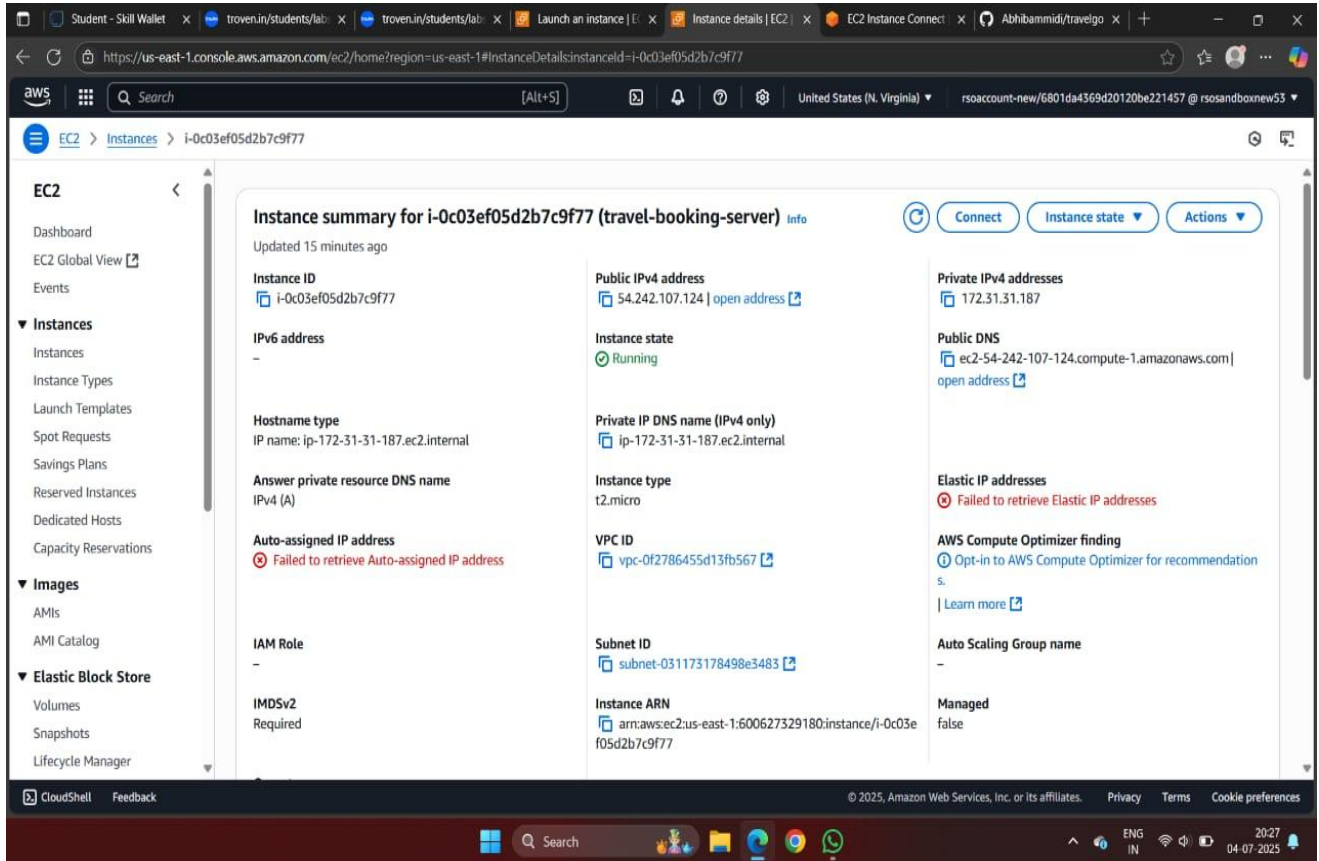


The screenshot displays the AWS Management Console interface for an EC2 instance. The left sidebar shows the navigation menu with categories like EC2, Images, and Elastic Block Store. The main content area shows the 'Instance summary for i-0c03ef05d2b7c9f77 (travel-bookings-server)'. The instance is in a 'Running' state. Key details include:

- Instance ID:** i-0c03ef05d2b7c9f77
- Public IPv4 address:** 54.242.107.124
- Private IPv4 addresses:** 172.31.31.187
- Instance state:** Running
- Private IP DNS name (IPv4 only):** ip-172-31-31-187.ec2.internal
- Instance type:** t2.micro
- VPC ID:** vpc-0f278645d13fb567
- Subnet ID:** subnet-031173178498e3483
- Instance ARN:** arn:aws:ec2:us-east-1:600627329180:instance/i-0c03ef05d2b7c9f77
- Auto-assigned IP address:** -
- IAM Role:** -
- IMDSv2:** Required
- Operator:** -



InstantLibrary.pem



The screenshot displays the AWS Management Console interface for an EC2 instance. The browser address bar shows the URL: `https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#InstanceDetails:instanceId=i-0c03ef05d2b7c9f77`. The console header includes the AWS logo, a search bar, and navigation links for EC2, Instances, and the specific instance ID.

**EC2**

- Dashboard
- EC2 Global View
- Events
- Instances**
  - Instances
  - Instance Types
  - Launch Templates
  - Spot Requests
  - Savings Plans
  - Reserved Instances
  - Dedicated Hosts
  - Capacity Reservations
- Images**
  - AMIs
  - AMI Catalog
- Elastic Block Store**
  - Volumes
  - Snapshots
  - Lifecycle Manager

**Instance summary for i-0c03ef05d2b7c9f77 (travel-bookings-server)** [Info](#)

Updated 15 minutes ago

**Instance ID**  
[i-0c03ef05d2b7c9f77](#)

**IPv6 address**  
-

**Hostnames**  
IP name: ip-172-31-31-187.ec2.internal

**Answer private resource DNS name**  
IPv4 (A)

**Auto-assigned IP address**  
Failed to retrieve Auto-assigned IP address

**IAM Role**  
-

**IMDSv2**  
Required

**Public IPv4 address**  
[54.242.107.124](#) | [open address](#)

**Instance state**  
Running

**Private IP DNS name (IPv4 only)**  
[ip-172-31-31-187.ec2.internal](#)

**Instance type**  
t2.micro

**VPC ID**  
[vpc-0f2786455d13fb567](#)

**Subnet ID**  
[subnet-031173178498e3483](#)

**Instance ARN**  
[arn:aws:ec2:us-east-1:600627329180:instance/i-0c03ef05d2b7c9f77](#)

**Private IPv4 addresses**  
[172.31.31.187](#)

**Public DNS**  
[ec2-54-242-107-124.compute-1.amazonaws.com](#) | [open address](#)

**Elastic IP addresses**  
Failed to retrieve Elastic IP addresses

**AWS Compute Optimizer finding**  
Opt-in to AWS Compute Optimizer for recommendation s. | [Learn more](#)

**Auto Scaling Group name**  
-

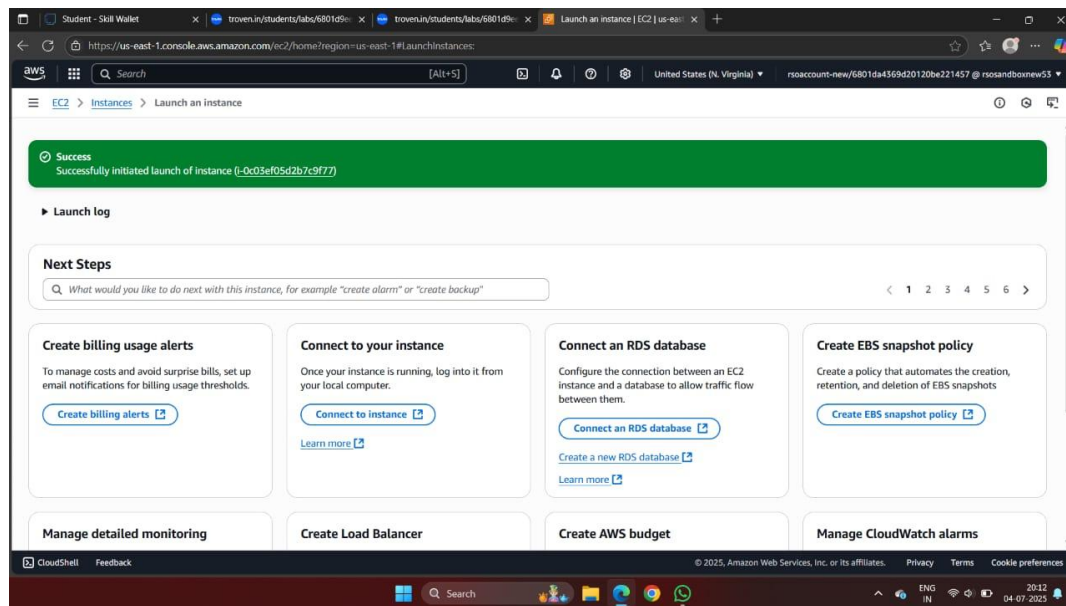
**Managed**  
false

© 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

CloudShell Feedback

Search

ENG IN 20:27 04-07-2025



- 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.



Student - Skill Wallet x troven.in/students/lab: x troven.in/students/lab: x Launch an instance | EC2 x Instance details | EC2 x EC2 Instance Connect x Abhibammi/travelgo x

https://troven.in/students/labs/6801d9eed156a79a13a8b313/6801da4369d20120be221457/start-lab?platform=AWS

Total Task	Recommended Duration	Current Lab Duration	Validation Status
5	03:00:00 HRS	02:57:01 HRS	0/5

The region should be set to N Virginia.

Initiate Labs

Open Console End Lab

Once you complete a task you can click on Validate and check if the task is done correctly. Click Validate All to check the status all at once. [Lab Guide](#) [Overviews](#)

- Launch an EC2 Instance (TravelGo)**

AWS

AMI: Amazon ubuntu

Instance Type: t2.micro (Free tier)

Key Pair: Create or use an existing RSA key pair

Security Group Rules:

  - Allow SSH (Port 22) from your IP
  - Allow HTTP (Port 80) from anywhere

Public IP: Enabled

Instance Name: travel-booking-server

Validate
- Create IAM Role (TravelGo)**

AWS

Validate

Student - Skill Wallet x troven.in/students/lab: x troven.in/students/lab: x Launch an instance | EC2 x Instance details | EC2 x EC2 Instance Connect x Abhibammi/travelgo x

https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#InstanceDetails:instanceId=i-0c03ef05d2b7c9f77

aws [Alt+S] United States (N. Virginia) rsoaccount-new/6801da4369d20120be221457 @ rsoandboxnew53

EC2 > Instances > i-0c03ef05d2b7c9f77

EC2

- Dashboard
- EC2 Global View
- Events
- ▼ Instances
  - Instances
  - Instance Types
  - Launch Templates
  - Spot Requests
  - Savings Plans
  - Reserved Instances
  - Dedicated Hosts
  - Capacity Reservations
- ▼ Images
  - AMIs
  - AMI Catalog
- ▼ Elastic Block Store
  - Volumes
  - Snapshots
  - Lifecycle Manager

CloudShell Feedback

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

Instance summary for i-0c03ef05d2b7c9f77 (travel-booking-server) info

Updated 15 minutes ago

Instance ID [i-0c03ef05d2b7c9f77](#)

Public IPv4 address [54.242.107.124](#) | [open address](#)

Private IPv4 addresses [172.31.31.187](#)

IPv6 address -

Instance state [Running](#)

Public DNS [ec2-54-242-107-124.compute-1.amazonaws.com](#) | [open address](#)

Hostname type IP name: ip-172-31-31-187.ec2.internal

Private IP DNS name (IPv4 only) [ip-172-31-31-187.ec2.internal](#)

Answer private resource DNS name IPv4 (A)

Instance type t2.micro

Auto-assigned IP address [Failed to retrieve Auto-assigned IP address](#)

VPC ID [vpc-0f2786455d13fb567](#)

Elastic IP addresses [Failed to retrieve Elastic IP addresses](#)

IAM Role -

Subnet ID [subnet-031173178498e3483](#)

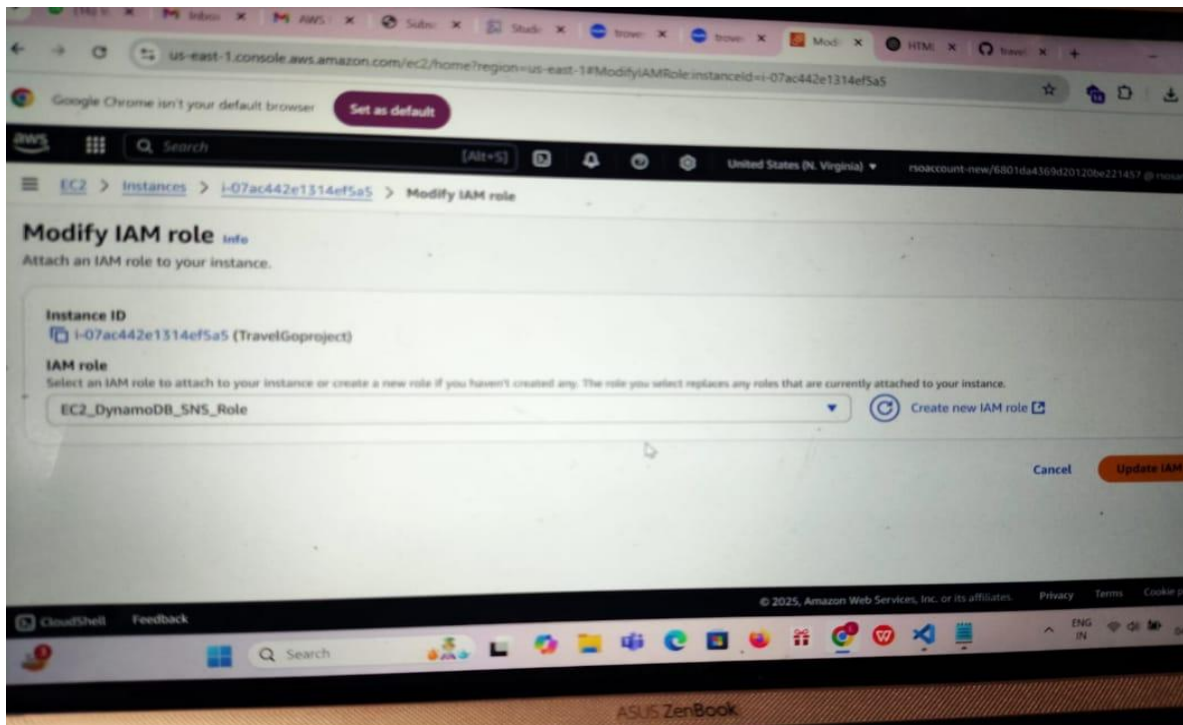
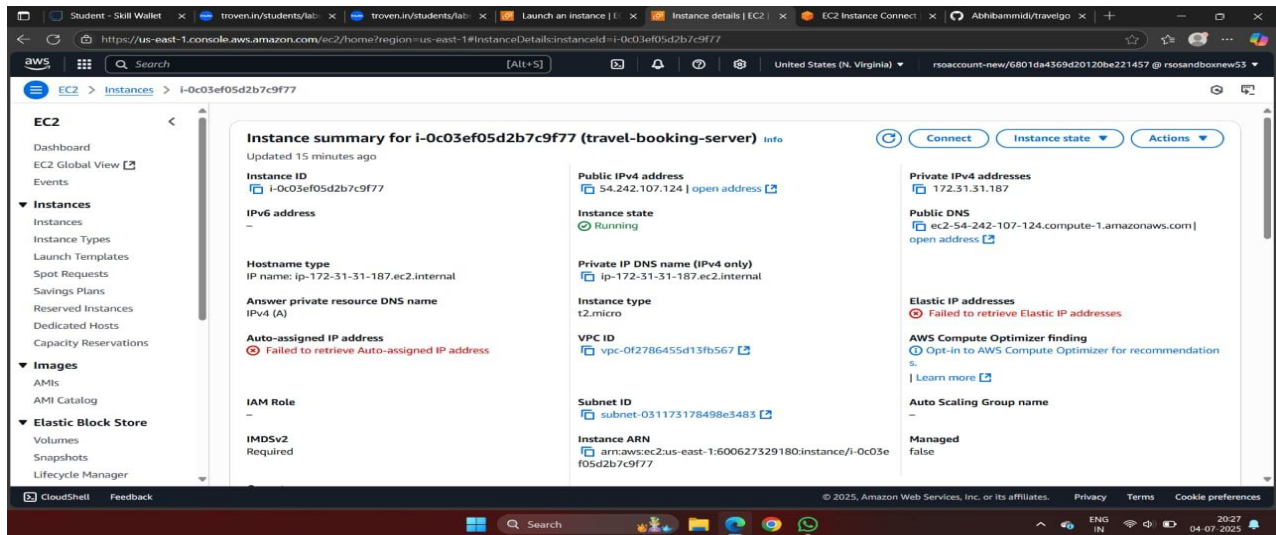
AWS Compute Optimizer finding [Opt-in to AWS Compute Optimizer for recommendation s.](#) | [Learn more](#)

IMDSv2 Required

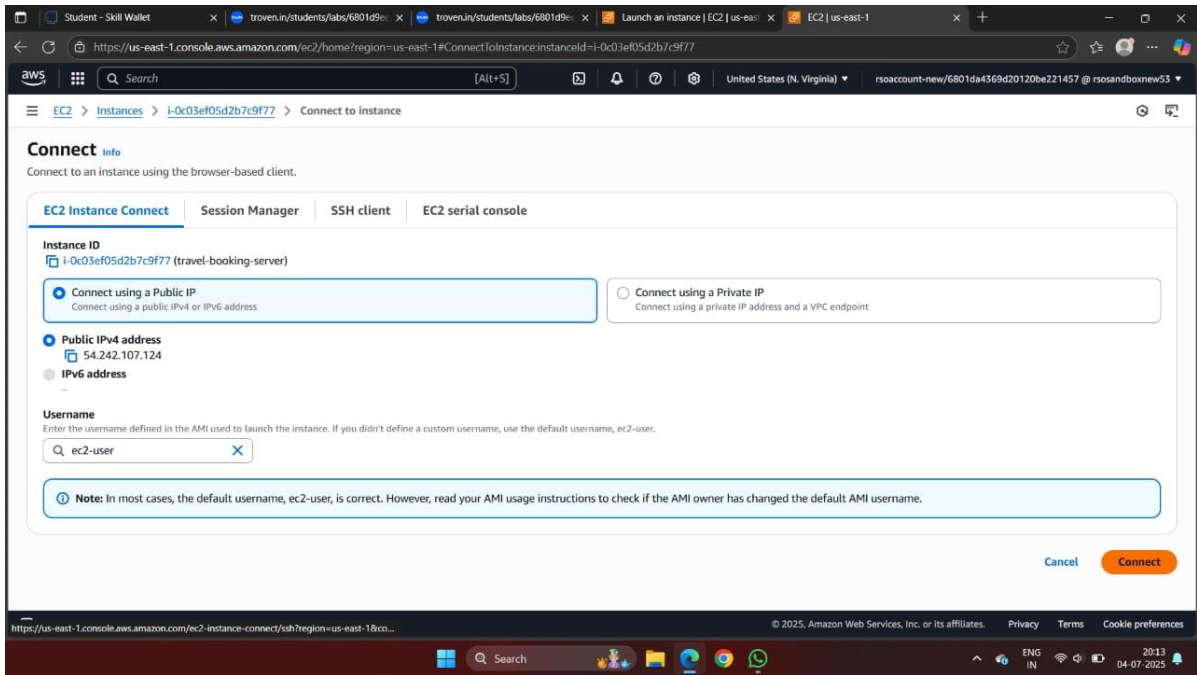
Instance ARN [arn:aws:ec2:us-east-1:600627329180:instance/i-0c03ef05d2b7c9f77](#)

Auto Scaling Group name -

Managed false



- Now connect the EC2 with the files



**Connect** info

Connect to an instance using the browser-based client.

**EC2 Instance Connect** | Session Manager | SSH client | EC2 serial console

**Instance ID**  
i-0c03ef05d2b7c9f77 (travel-booking-server)

☒ **Connect using a Public IP**  
Connect using a public IPv4 or IPv6 address

☐ **Connect using a Private IP**  
Connect using a private IP address and a VPC endpoint

**Public IPv4 address**  
54.242.107.124

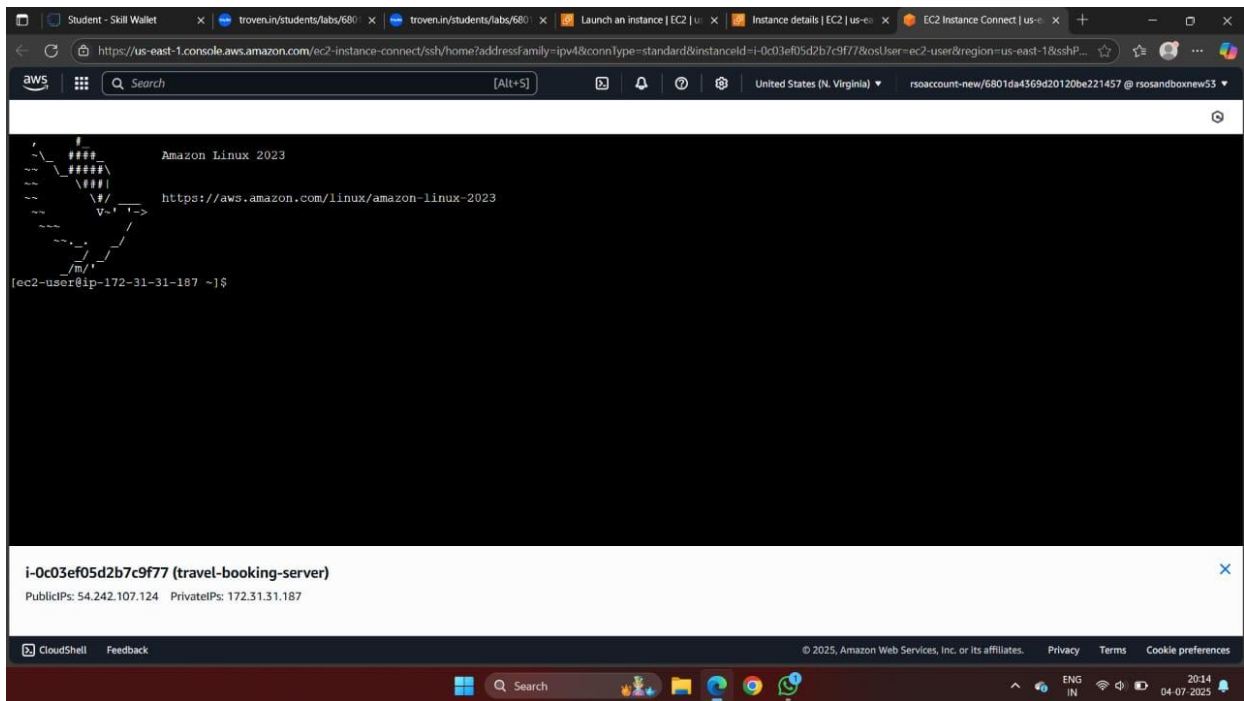
☐ **IPv6 address**

**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](#)



Amazon Linux 2023

<https://aws.amazon.com/linux/amazon-linux-2023>

[ec2-user@ip-172-31-187 ~]\$

**i-0c03ef05d2b7c9f77 (travel-booking-server)**

PublicIPs: 54.242.107.124 PrivateIPs: 172.31.31.187

## 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/Alekhyapenubakula/InstantLibrary.git'

- This will download your project to the EC2 instance.

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

```
cd InstantLibrary
```

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

#### Run the Flask Application

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

```

Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

Last login: Fri Jul 4 14:44:03 2025 from 18.206.107.27
[ec2-user@ip-172-31-31-187 ~]$ sudo yum install git -y
Amazon Linux 2023 Kernel livepatch repository                               166 kB/s | 17 kB    00:00
Dependencies resolved.
Package                           Architecture      Version            Repository          Size
Installing:
git                               x86_64            2.47.1-1.amzn2023.0.3  amazonlinux         52 k
Installing dependencies:
git-core                         x86_64            2.47.1-1.amzn2023.0.3  amazonlinux         4.5 M
git-core-doc                    noarch            2.47.1-1.amzn2023.0.3  amazonlinux         2.8 M
perl-Error                      noarch            1:0.17029-5.amzn2023.0.2  amazonlinux         41 k
perl-File-Find                  noarch            1.37-477.amzn2023.0.7    amazonlinux         25 k
perl-git                        noarch            2.47.1-1.amzn2023.0.3    amazonlinux         40 k
perl-TermReadKey                x86_64            2.38-9.amzn2023.0.2     amazonlinux         36 k
perl-lib                        x86_64            0.65-477.amzn2023.0.7    amazonlinux         15 k

Transaction Summary
i-0c03ef05d2b7c9f77 (travel-bookings-server)
PublicIPs: 54.242.107.124 PrivateIPs: 172.31.31.187
  
```

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 : git-core-2.47.1-1.amzn2023.0.3.x86_64 1/1
Installing : git-core-2.47.1-1.amzn2023.0.3.noarch 1/8
Installing : perl-lib-0.65-477.amzn2023.0.7.x86_64 2/8
Installing : perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64 3/8
Installing : perl-File-Find-1.37-477.amzn2023.0.7.noarch 4/8
Installing : perl-Error-1.0.17029-5.amzn2023.0.2.noarch 5/8
Installing : perl-git-2.47.1-1.amzn2023.0.3.noarch 6/8
Installing : git-2.47.1-1.amzn2023.0.3.x86_64 7/8
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.noarch 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-172-31-31-187 ~]$
  
```

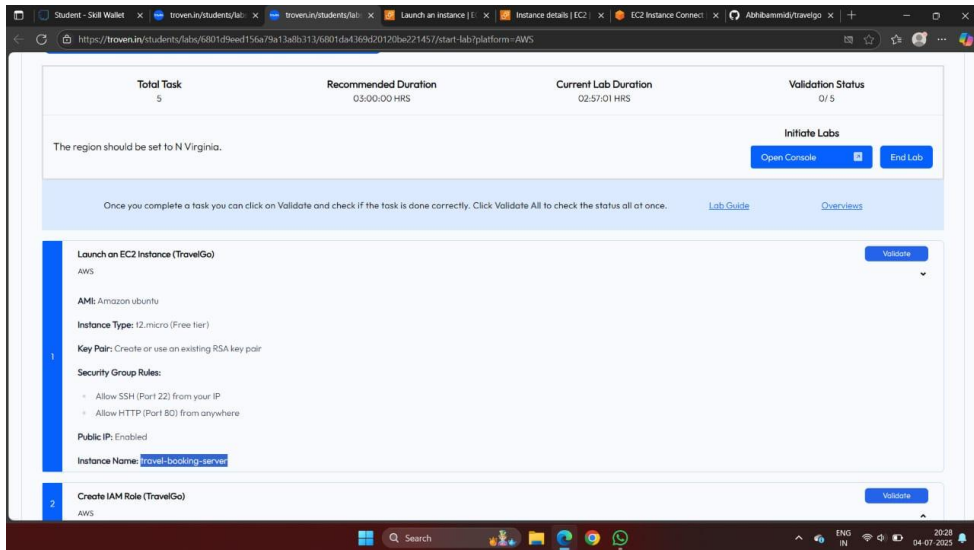
Access the website through:

PublicIPs: <https://13.201.74.42/>

## Milestone 8: Testing and Deployment

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

Login Page:



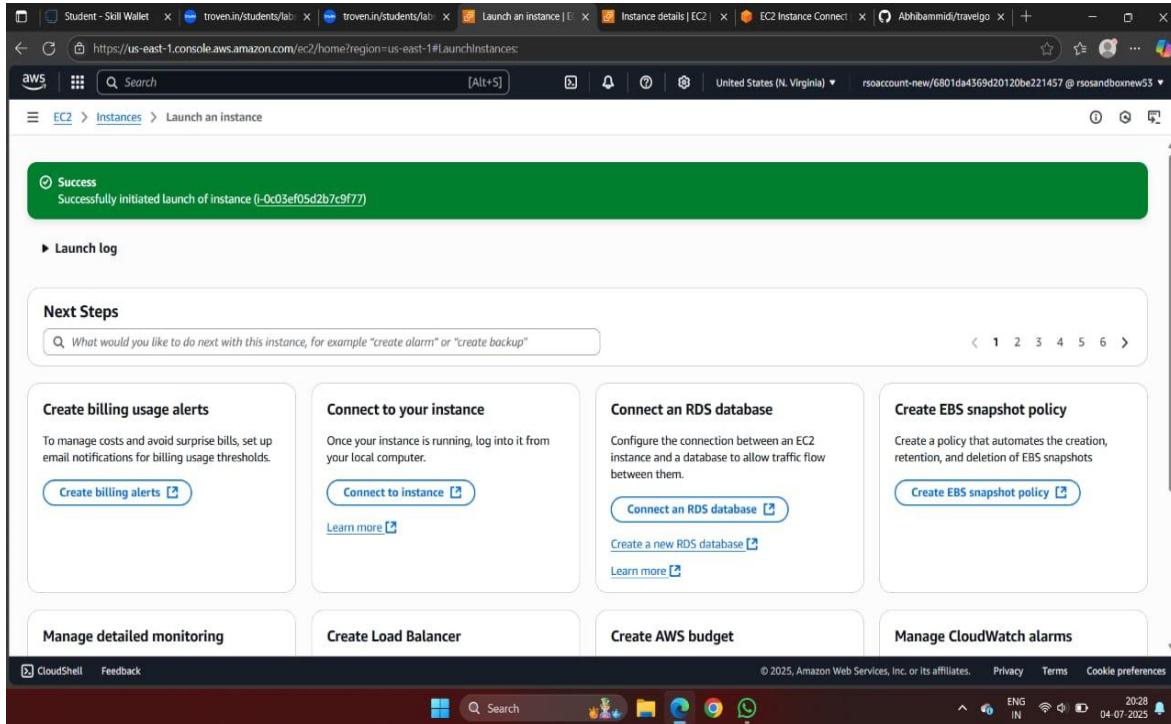
The screenshot displays a web application interface with a task list. At the top, there are four columns: 'Total Task' (5), 'Recommended Duration' (03:00:00 HRS), 'Current Lab Duration' (02:57:01 HRS), and 'Validation Status' (0/5). Below these columns, a message states 'The region should be set to N Virginia.' with buttons for 'Open Console' and 'End Lab'. A blue banner below the message reads: 'Once you complete a task you can click on Validate and check if the task is done correctly. Click Validate All to check the status all at once.' with links for 'Lab Guide' and 'Overview'. The task list contains two items:

- Launch on EC2 Instance (TravelGo)**  
AWS  
AMI: Amazon ubuntu  
Instance Type: t2.micro (Free tier)  
Key Pair: Create or use an existing RSA key pair  
Security Group Rules:
  - Allow SSH (Port 22) from your IP
  - Allow HTTP (Port 80) from anywherePublic IP: Enabled  
Instance Name: `travel-bookings-server`  
Validate
- Create IAM Role (TravelGo)**  
AWS  
Validate

The Windows taskbar at the bottom shows the date and time as 04-07-2025, 20:28.



## Register Page:



The screenshot displays the AWS Management Console interface for launching an EC2 instance. The browser tabs include 'Student - Skill Wallet', 'troven.in/students/ai...', 'Launch an instance | EC2', 'Instance details | EC2', 'EC2 Instance Connect', and 'Abhikamriddi/travelgo'. The address bar shows the URL 'https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances'. The AWS header indicates the user is logged in as 'rsoaccount-new/6801da4369d20120be221457 @ rso.sandboxnew53' in the 'United States (N. Virginia)' region.

The main content area shows a green success message: 'Success Successfully initiated launch of instance (i-0c03ef05d2b7c9f77)'. Below this is a 'Launch log' section. The 'Next Steps' section includes a search bar with the placeholder text 'What would you like to do next with this instance, for example "create alarm" or "create backup"'. There are six recommended actions, each with a description and a button:

- Create billing usage alerts**: To manage costs and avoid surprise bills, set up email notifications for billing usage thresholds. Button: [Create billing alerts](#)
- Connect to your instance**: Once your instance is running, log into it from your local computer. Button: [Connect to instance](#). Link: [Learn more](#)
- Connect an RDS database**: Configure the connection between an EC2 instance and a database to allow traffic flow between them. Button: [Connect an RDS database](#). Link: [Create a new RDS database](#). Link: [Learn more](#)
- Create EBS snapshot policy**: Create a policy that automates the creation, retention, and deletion of EBS snapshots. Button: [Create EBS snapshot policy](#)
- Manage detailed monitoring**
- Create Load Balancer**
- Create AWS budget**
- Manage CloudWatch alarms**

The footer of the console shows '© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences'. The Windows taskbar at the bottom shows the time as 20:28 on 04-07-2025.

## Conclusion:

The **Travel Go** platform has been successfully developed and deployed using a robust cloud-based architecture. By leveraging AWS services such as **EC2** for hosting, **DynamoDB** for data management, and **SNS** for real-time notifications, the platform ensures reliable and scalable access to essential travel booking services. This system addresses the challenges of fragmented travel planning by providing users with a convenient way to book trips and receive timely updates, while administrators can efficiently manage and track bookings.

The cloud-native approach allows for seamless scalability, ensuring that as user demand increases, the platform can handle the load without compromising performance. The integration of **Flask** with AWS ensures that backend processes, including user authentication and travel bookings, run efficiently. The platform's testing phase has ensured that all functionalities—from user registration to trip confirmation notifications—work smoothly.

In conclusion, **Travel Go** offers a modern, efficient solution for managing travel services, enhancing the overall user experience, and improving communication between travelers and service providers. This project demonstrates the potential of cloud-based systems in solving real-world challenges in the travel industry.

































