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**.**

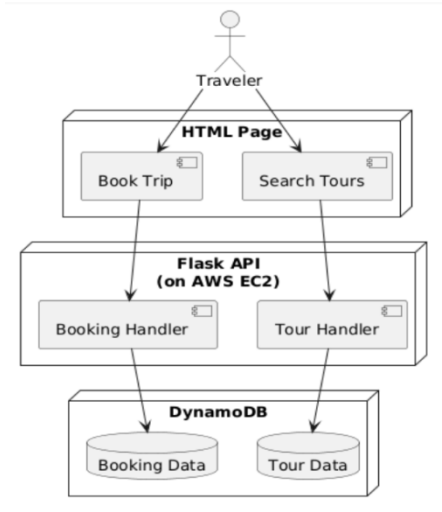
Scenario 2: Seamless Travel Inquiry Notifications for Support Staff

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.

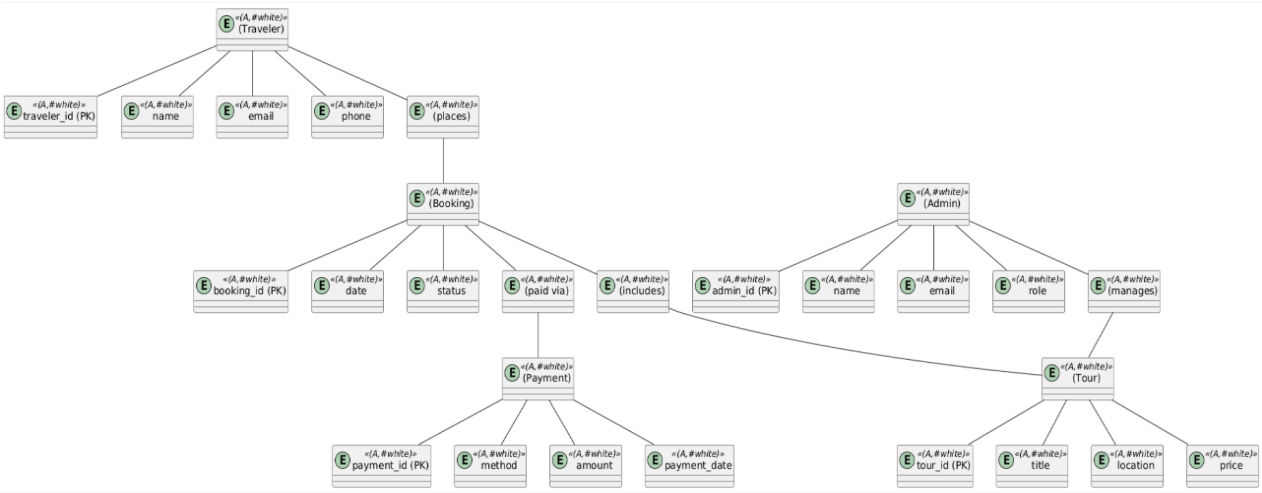
Scenario 3: Easy Access to Travel Resources and Services

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

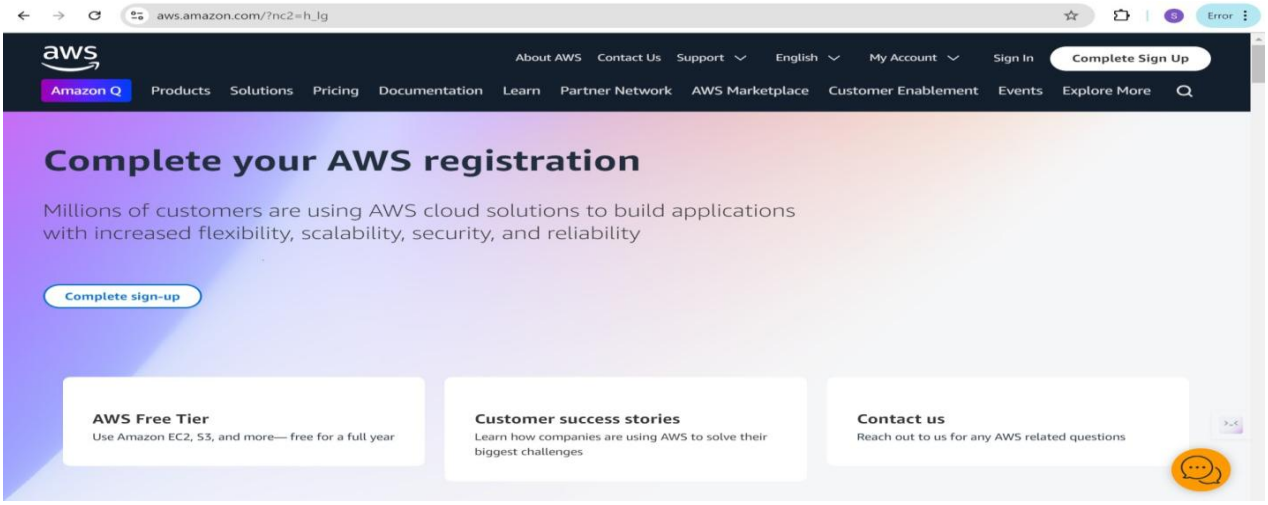
8. Testing and Deployment

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

Milestone 1: AWS Account Setup and Login

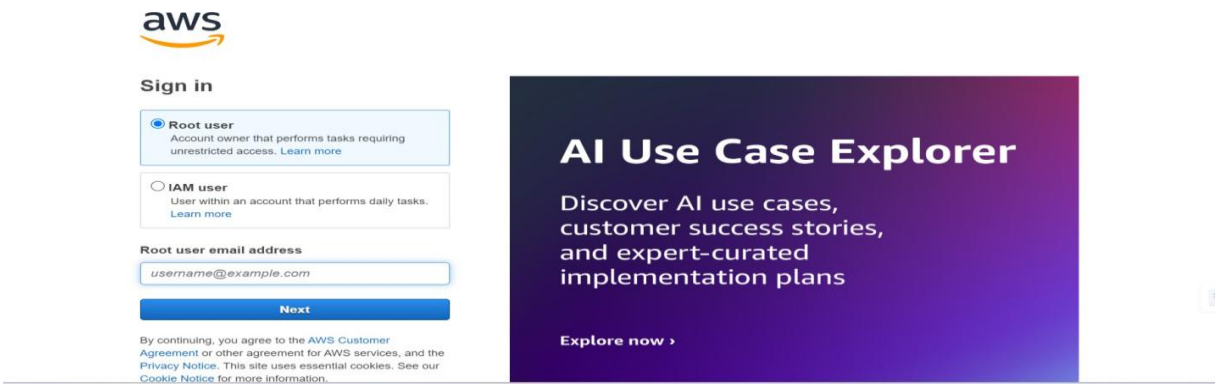
- **Activity 1.1: Set up an AWS account if not already done**

  - Sign up for an AWS account and configure billing settings.



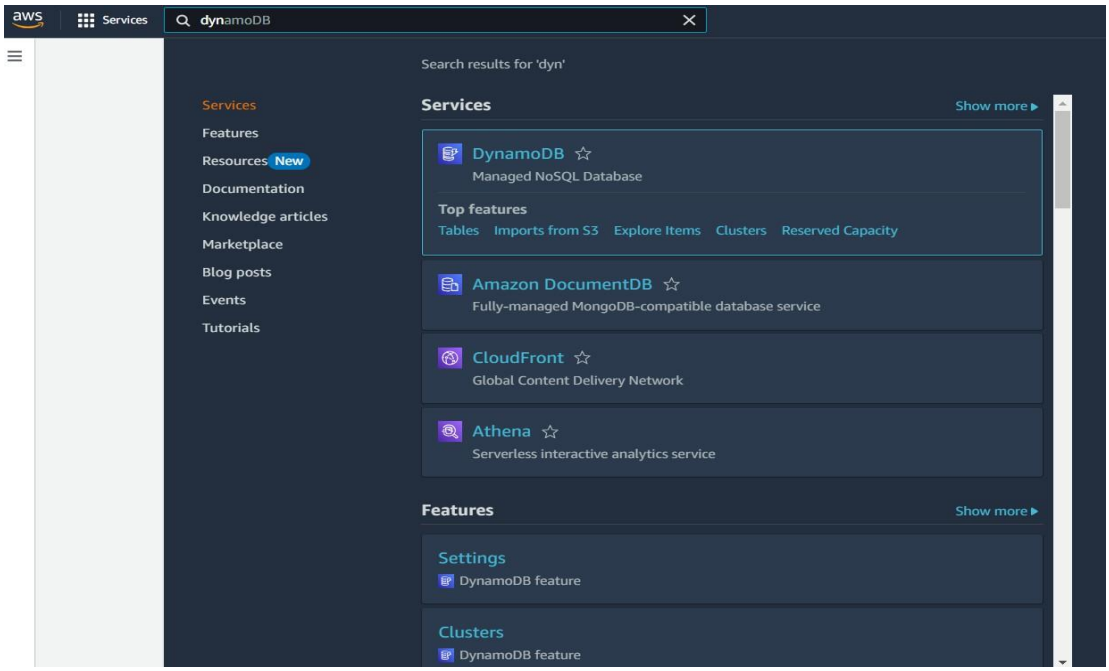
- **Activity 1.2: Log in to the AWS Management Console**

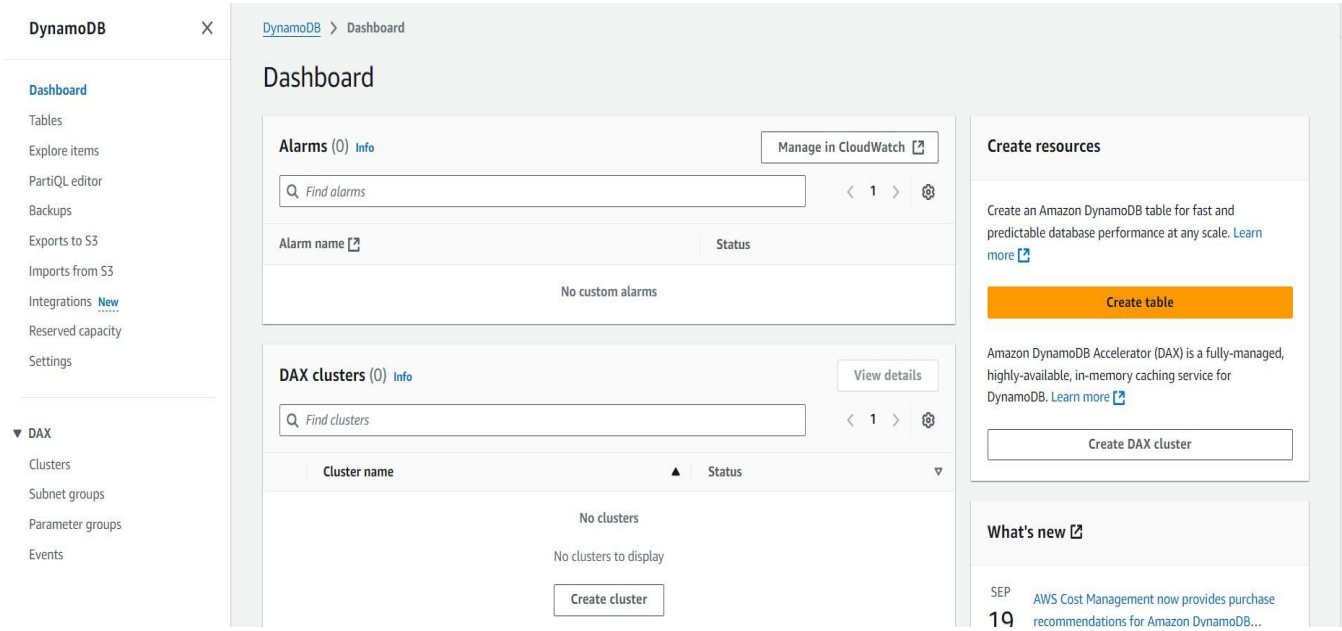
  - After setting up your account, log in to the AWS Management Console.

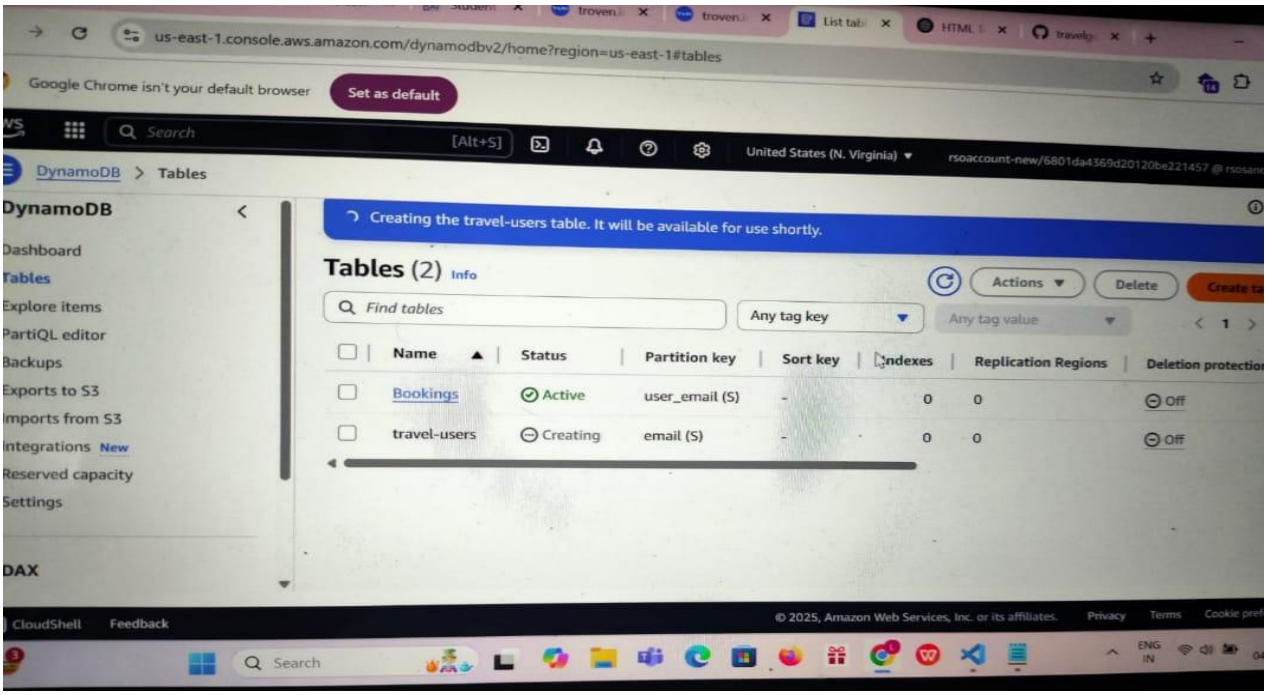


Milestone 2: DynamoDB Database Creation and Setup

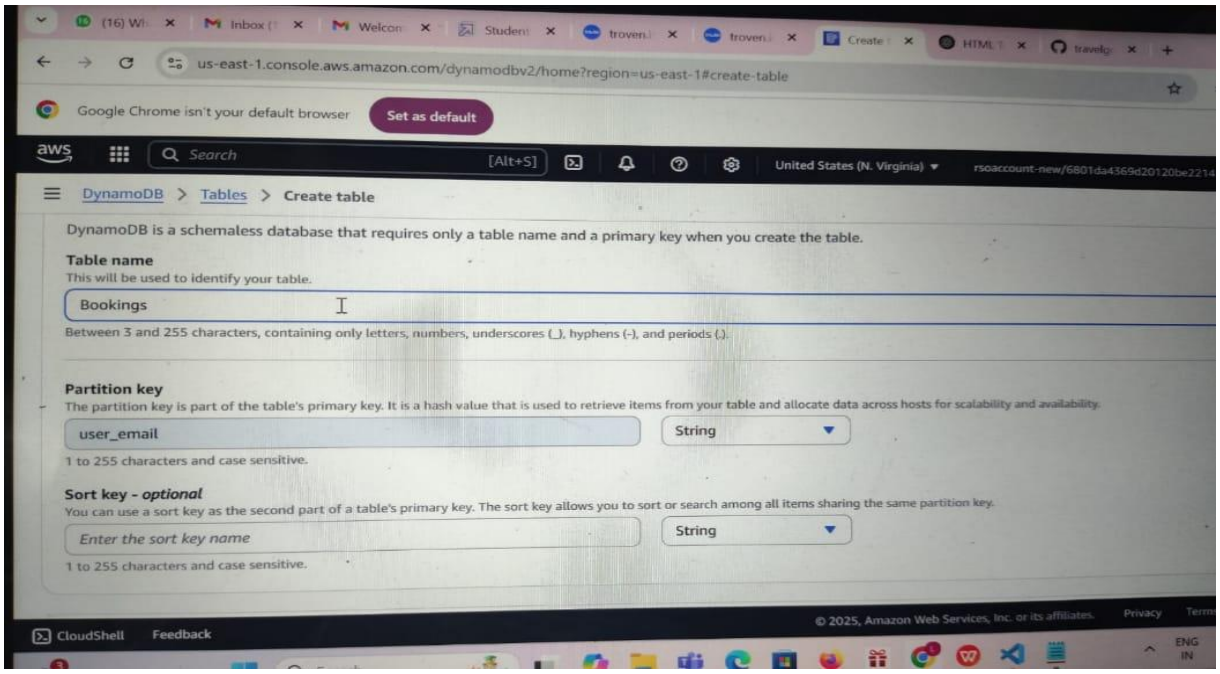
- **Activity 2.1:Navigate to the DynamoDB**
  - In the AWS Console, navigate to DynamoDB and click on create tables.

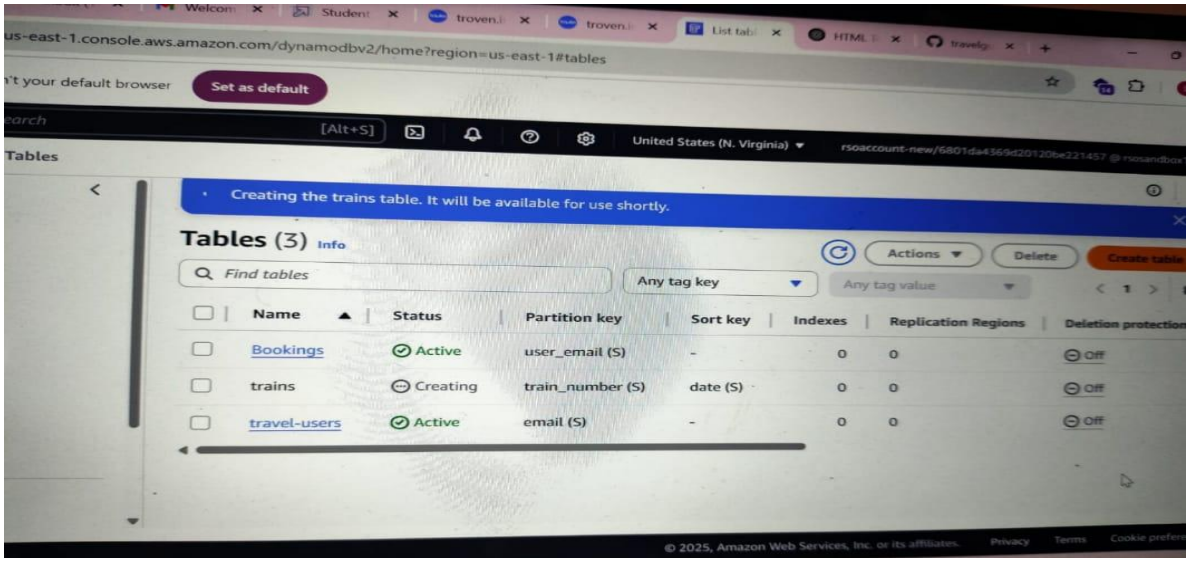




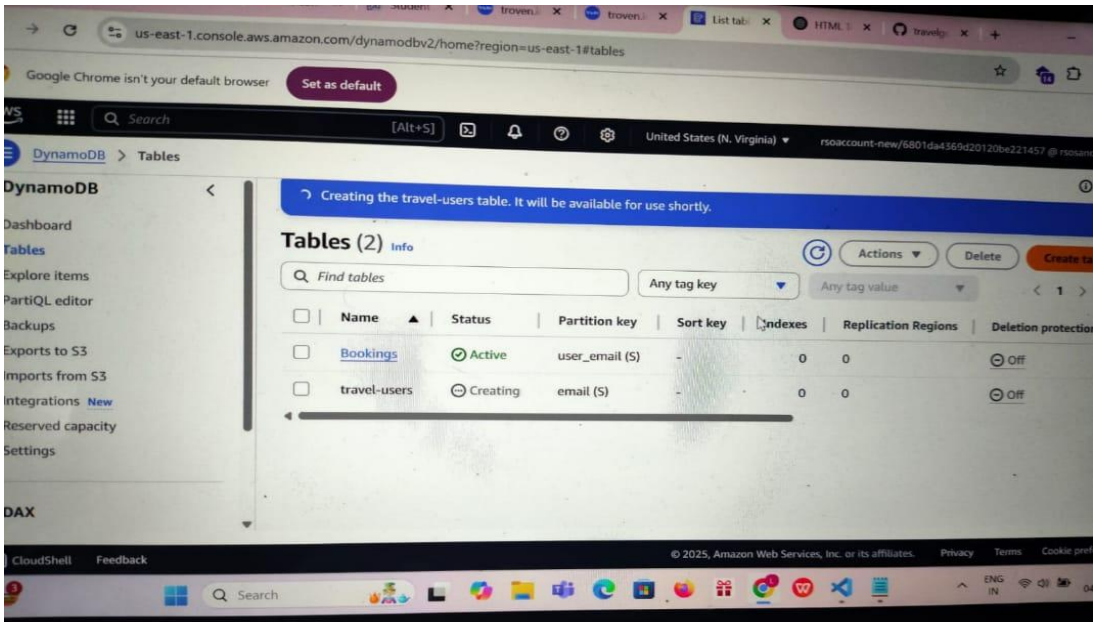


- **Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**
    - Create Users table with partition key “Email” with type String and click on create tables.



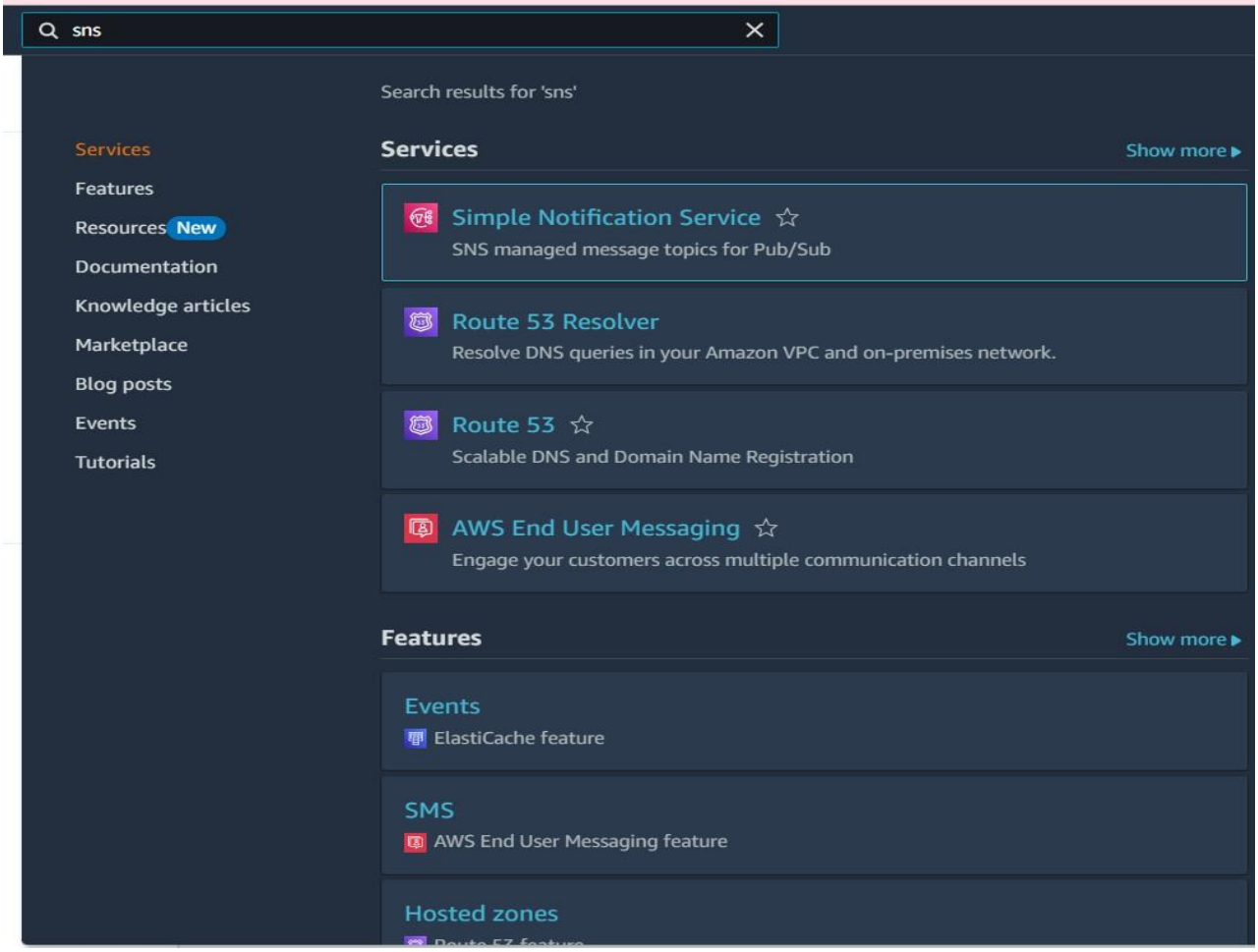


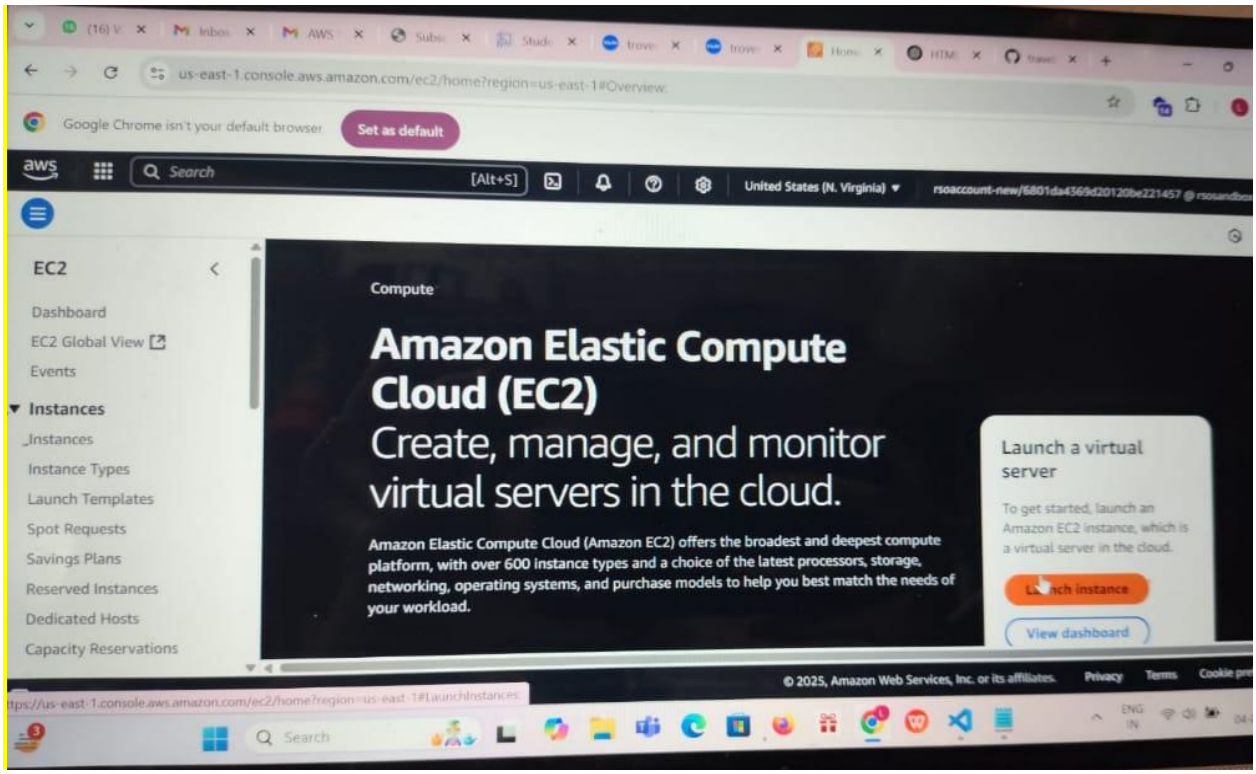
- Follow the same steps to create a requests table with Email as the primary key for book requests data.
- Choose Standard type for general notification use cases and Click on Create Topic.



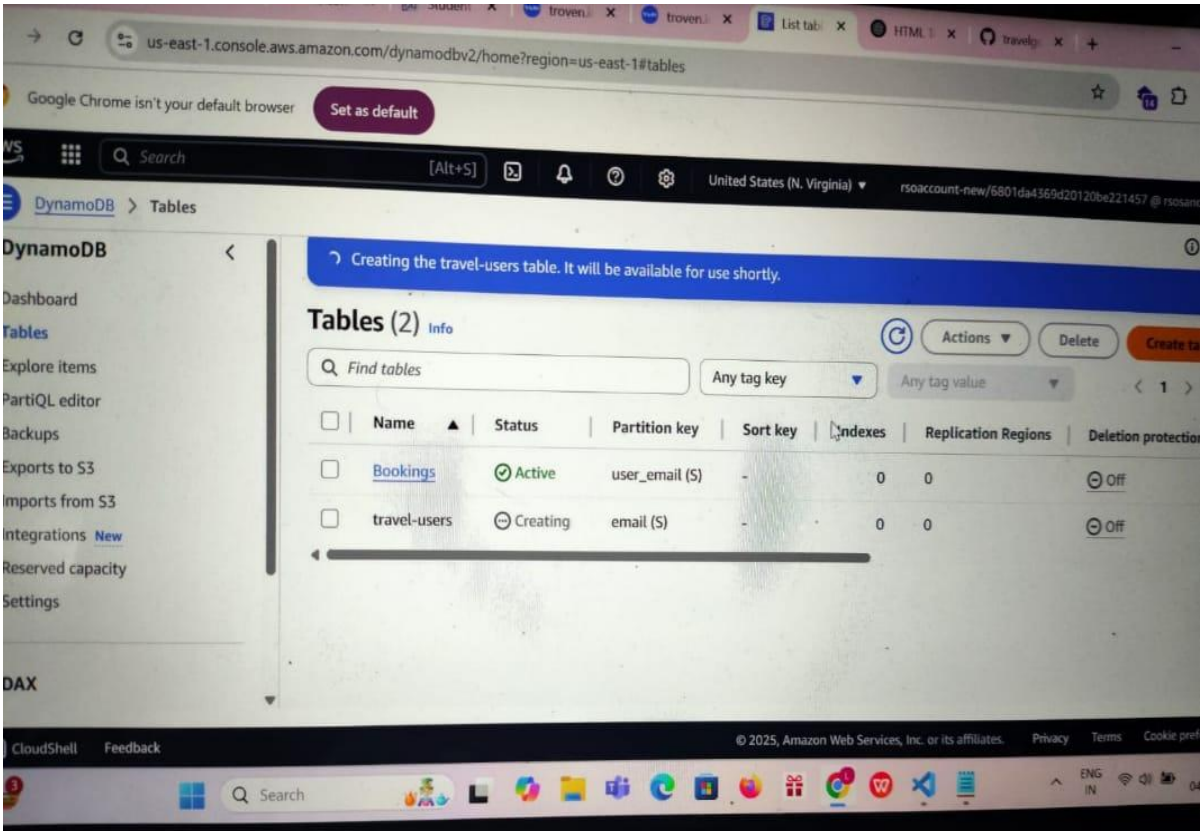
Milestone 3: SNS Notification Setup

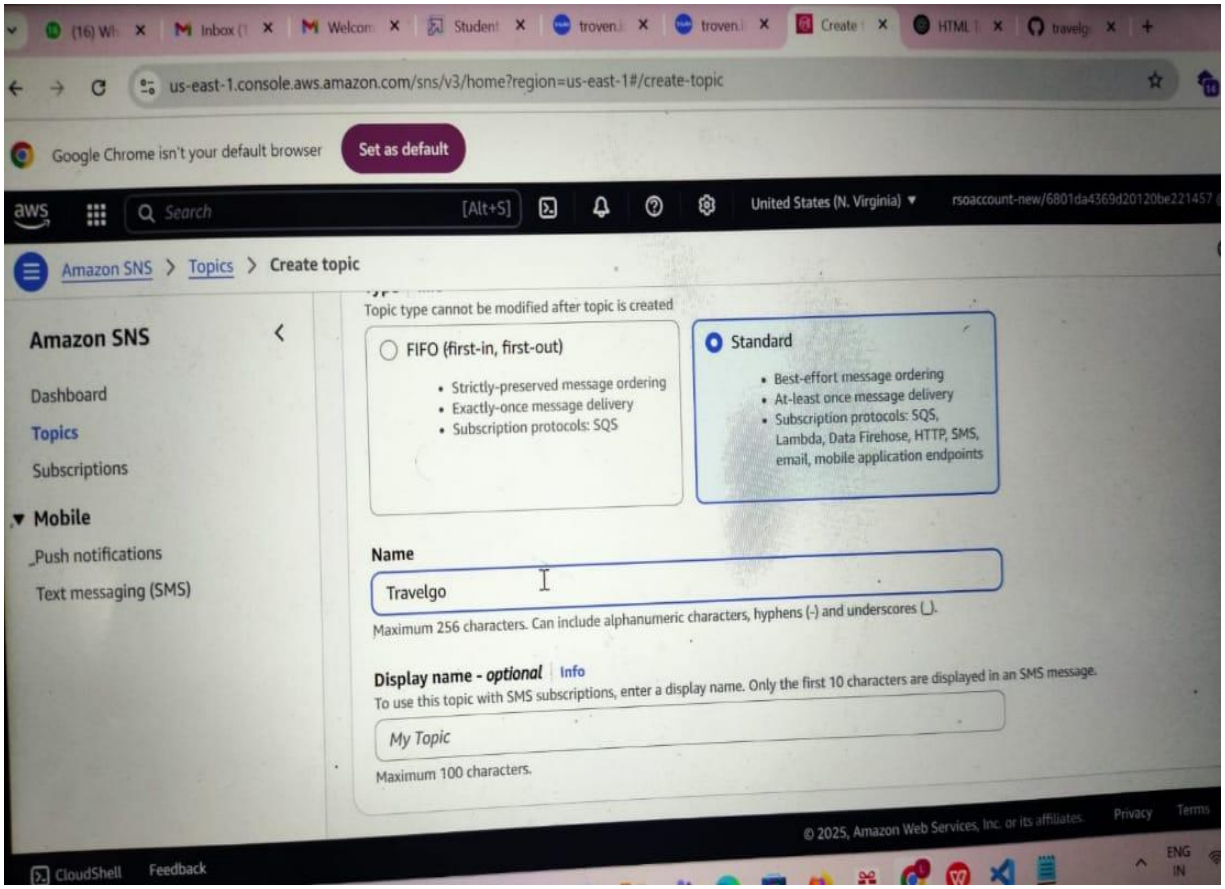
- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff**
- In the AWS Console, search for SNS and navigate to the SNS Dashboard.

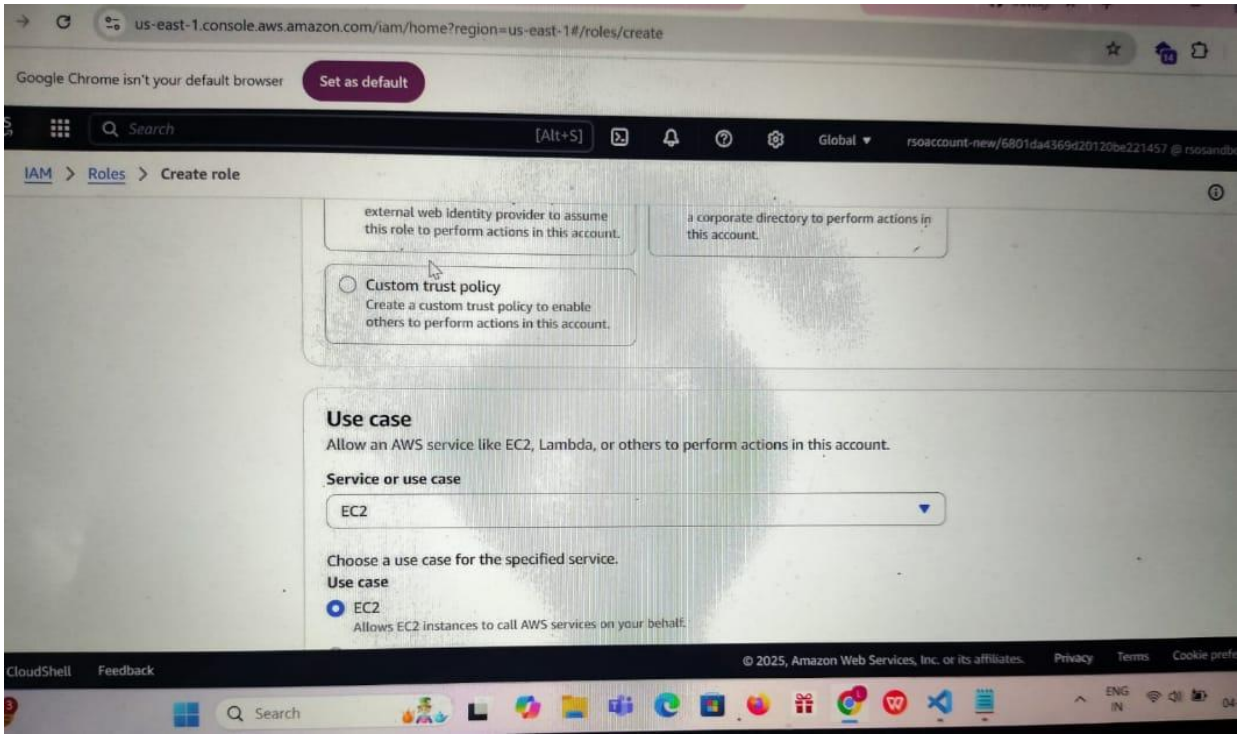




- Click on **Create Table** and choose a name



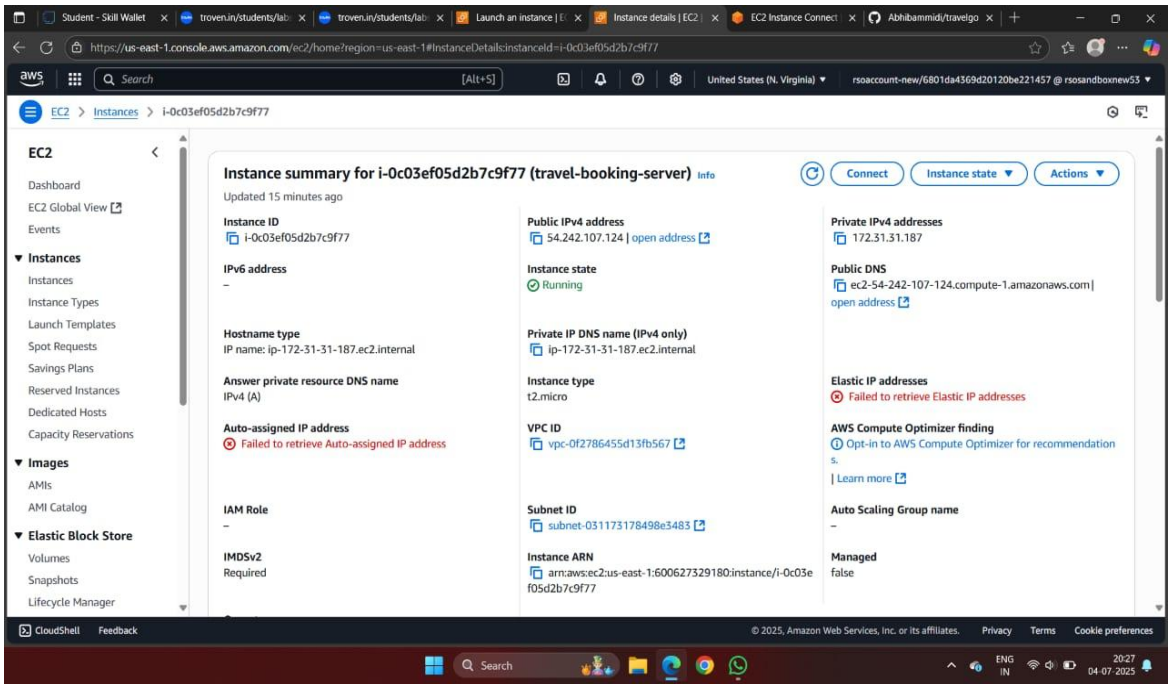


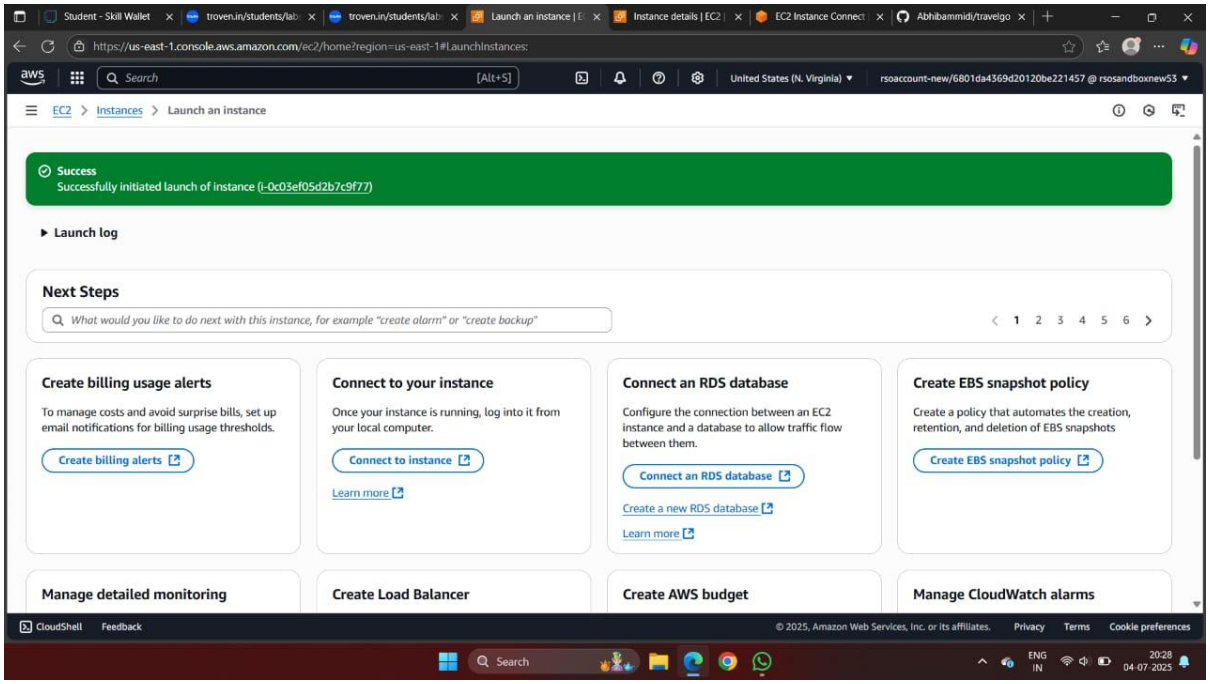


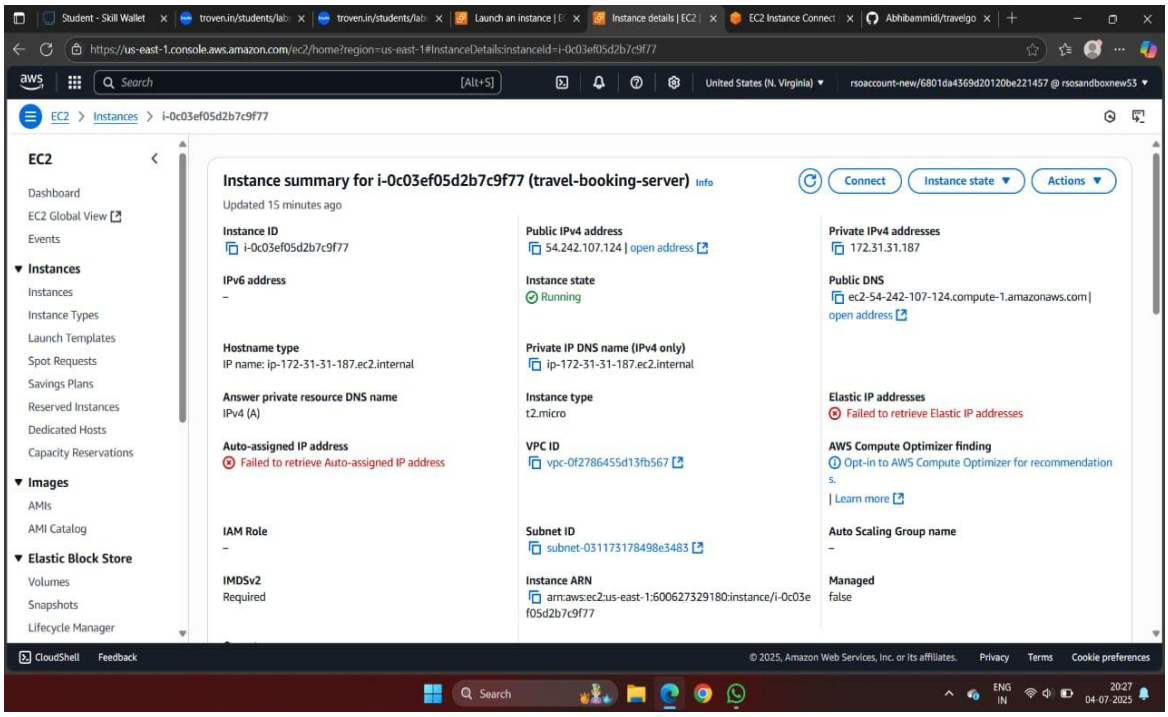
- Configure the SNS topic and note down the **Topic ARN**

- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

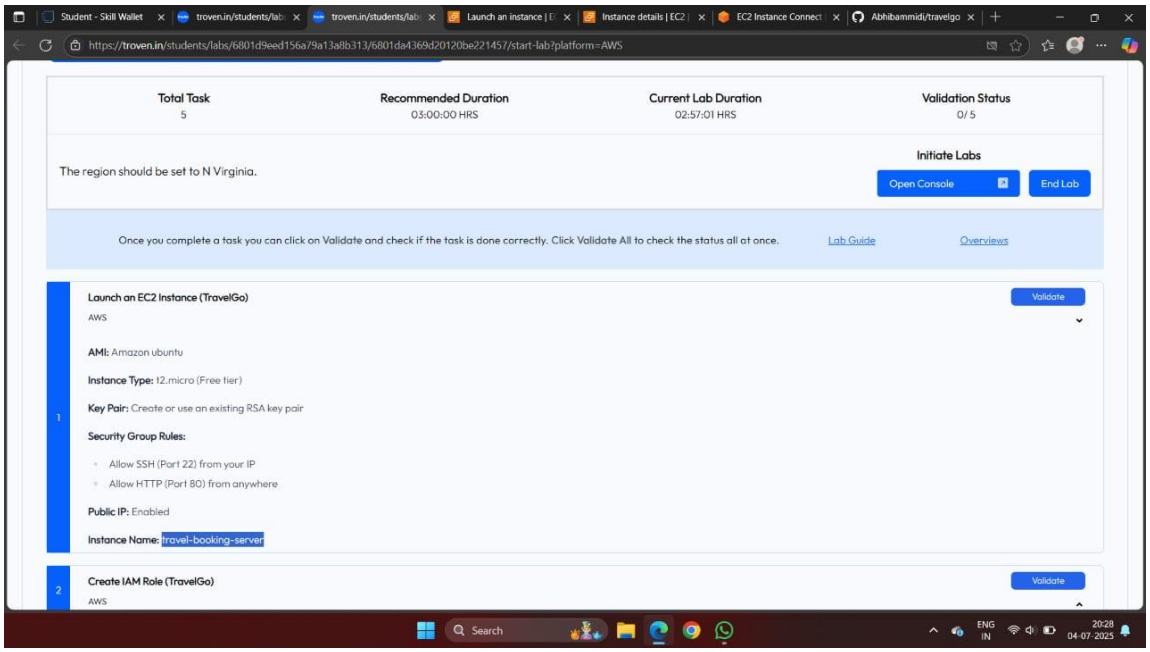
  - Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.



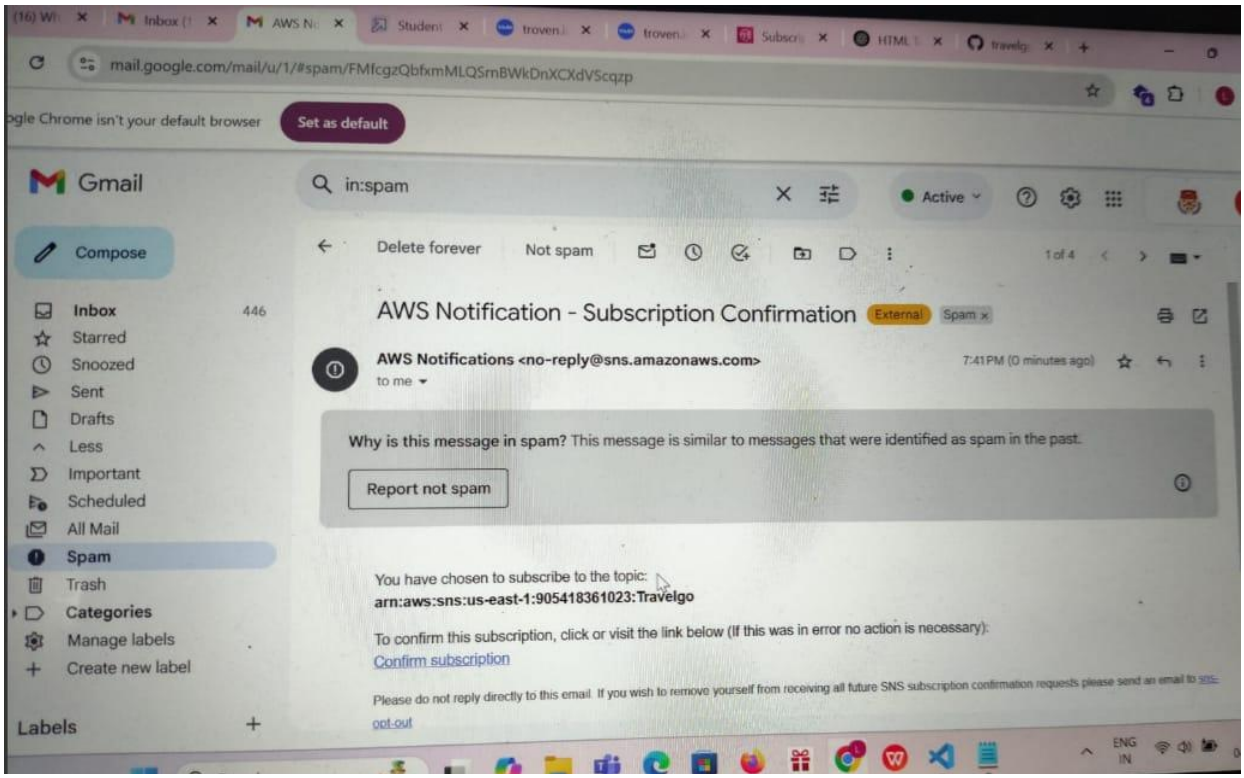


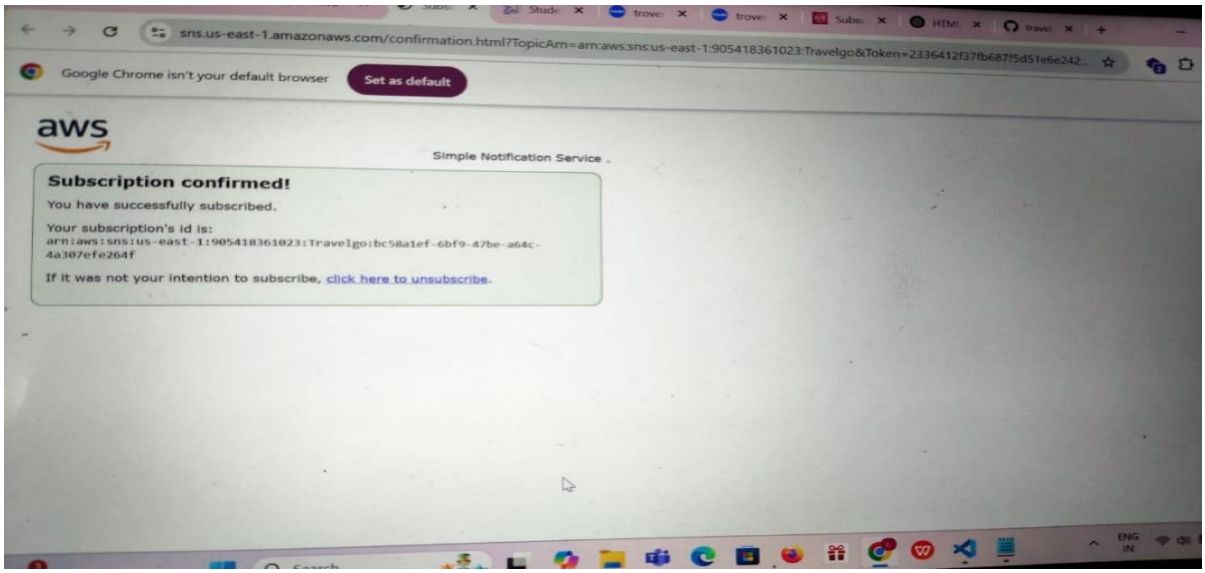


- After subscription request for the mail confirmation

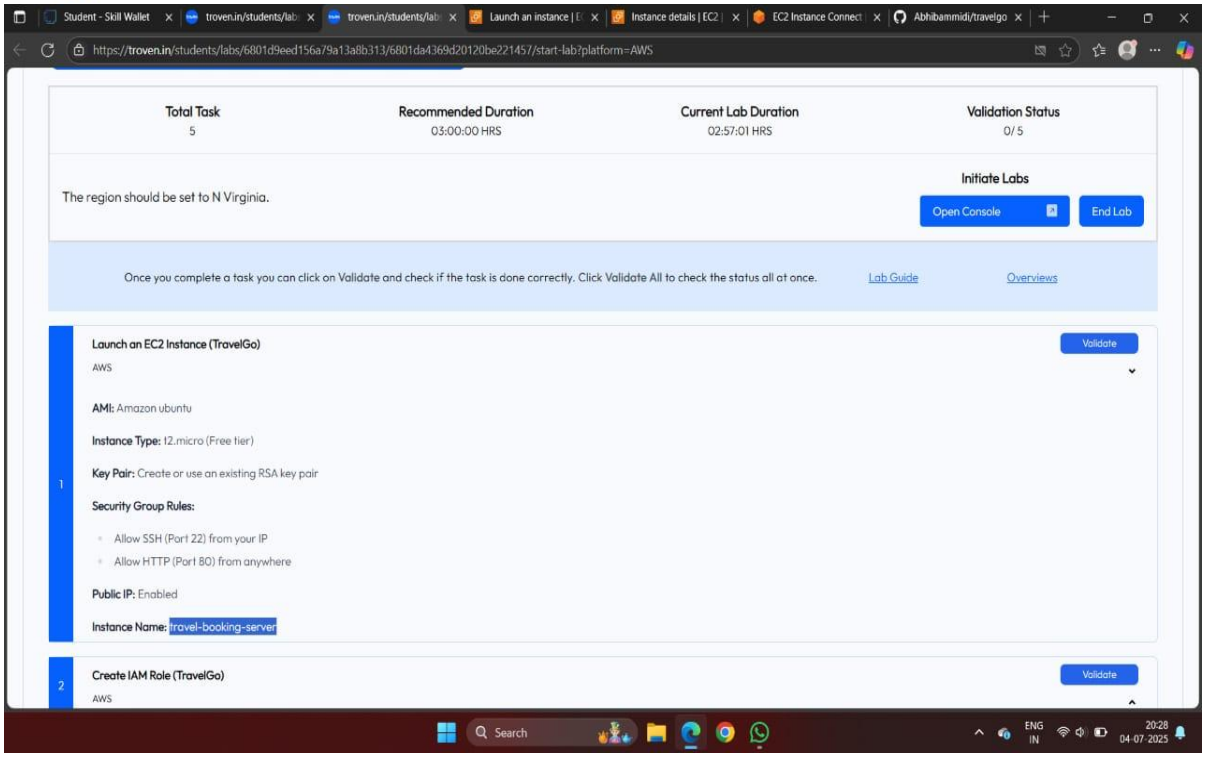


- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.





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



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 'Reques
```

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 = MIMEMultipart()
    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:**

```python
# 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)**:

```python
# 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:**

```python
# 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:**

```python
# 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}\
        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:

```python
# 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.

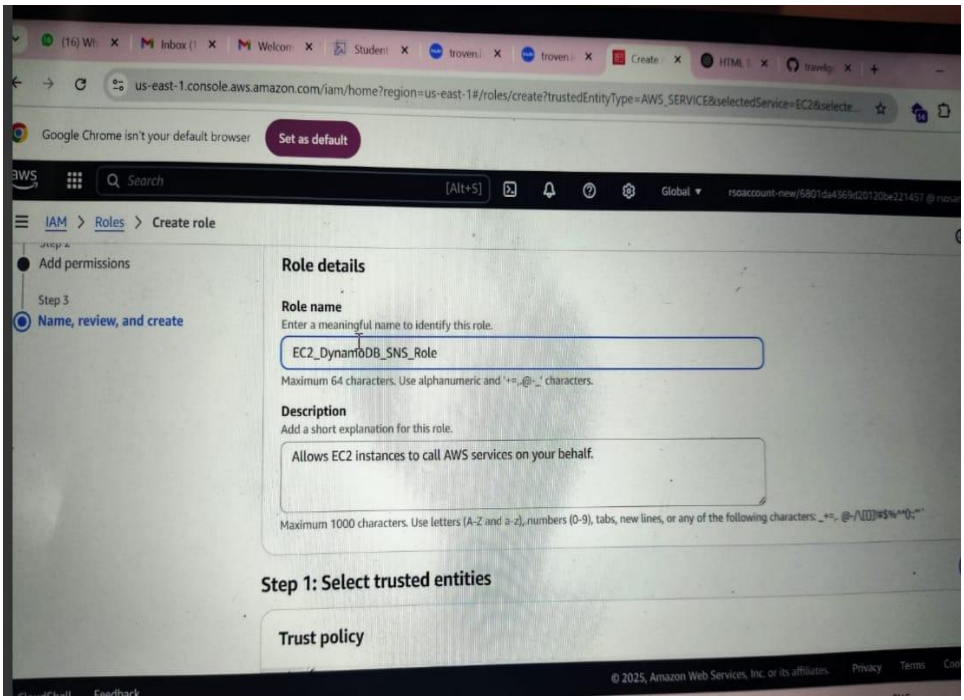
Deployment Code:

```
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

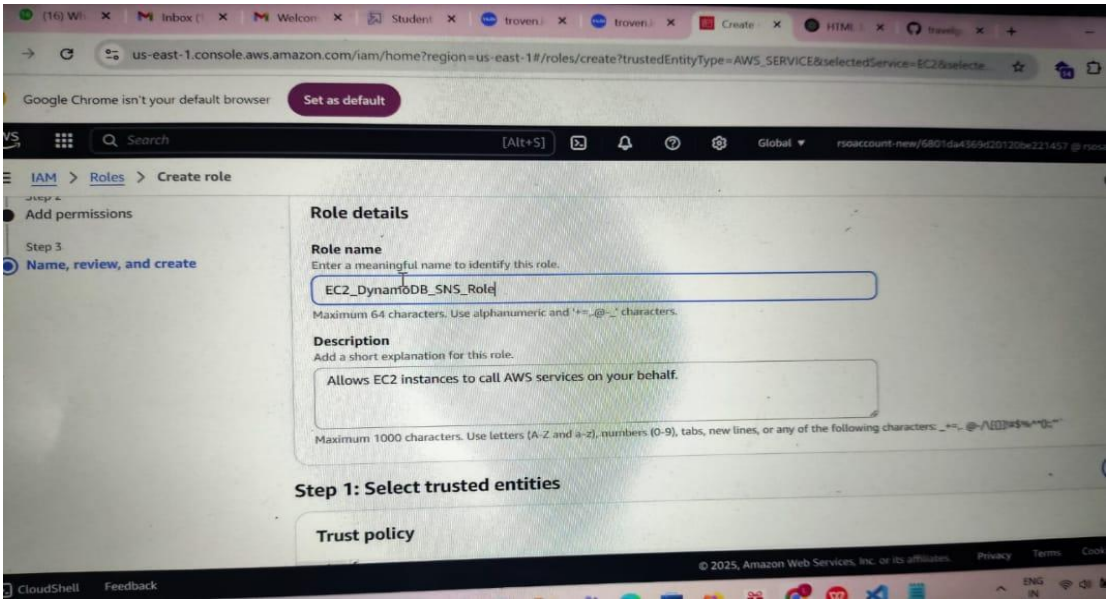
- **Activity 5.1:Create IAM Role.**
    - In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.



- **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.**

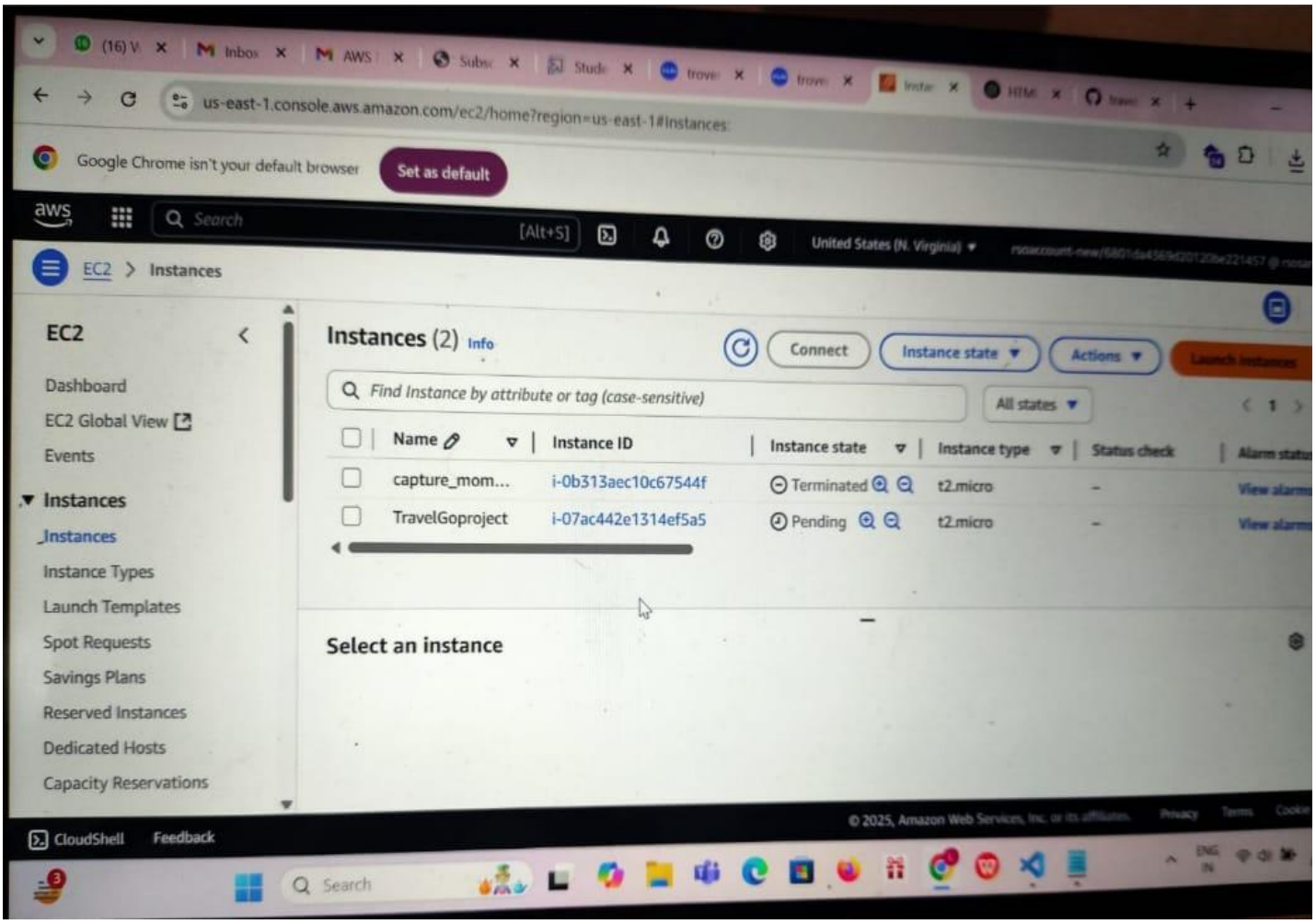
static	Initial commit
templates	Update statistics.html
app.py	Update app.py

A

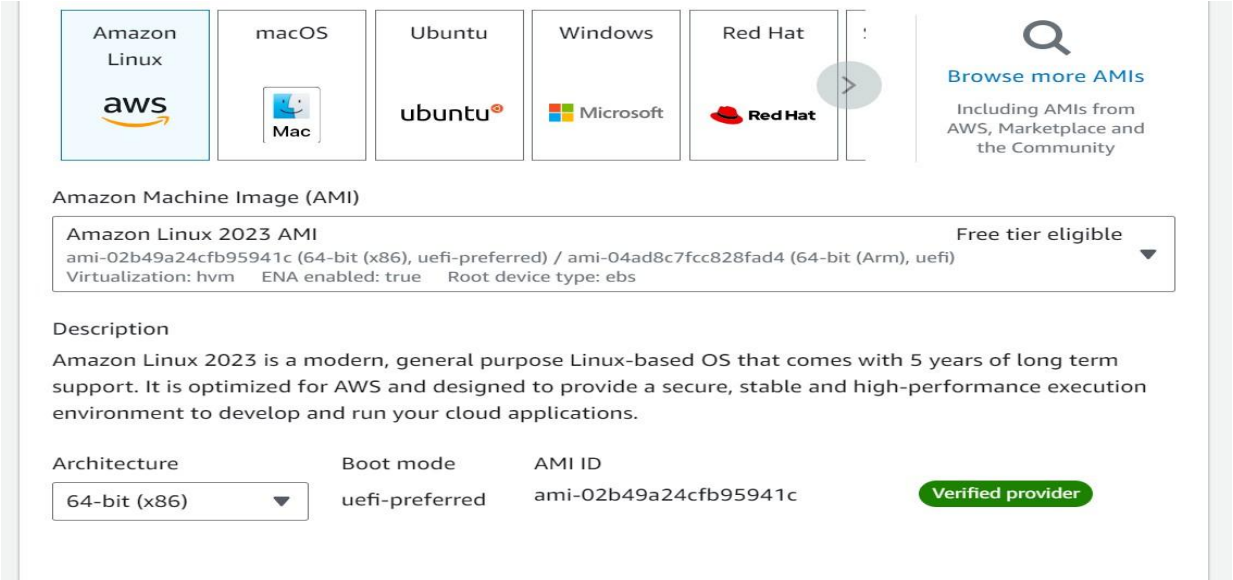
Activity 6.1: Launch an EC2 instance to host the Flask application.

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

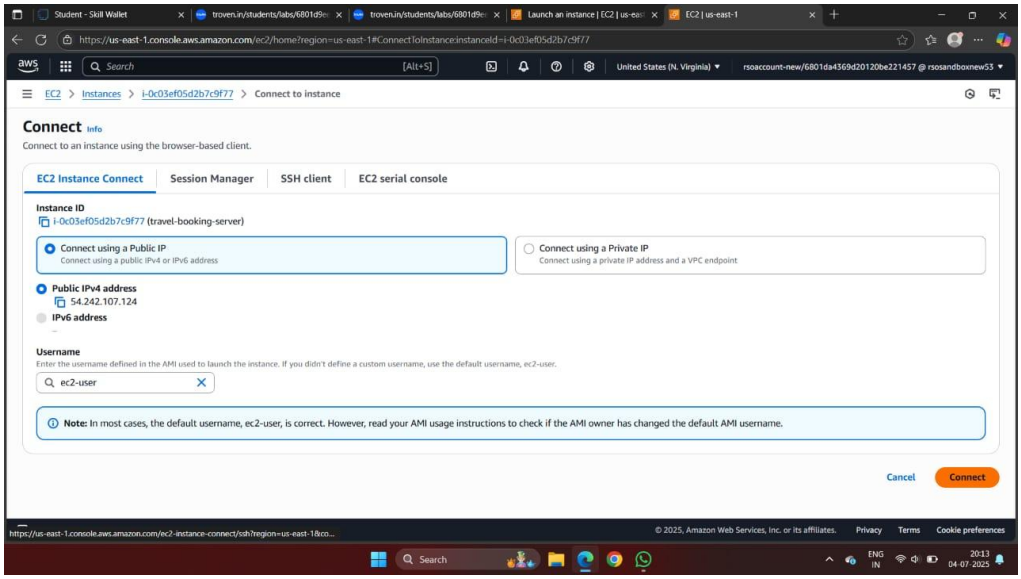
- **Click on Launch instance to launch EC2 instance**

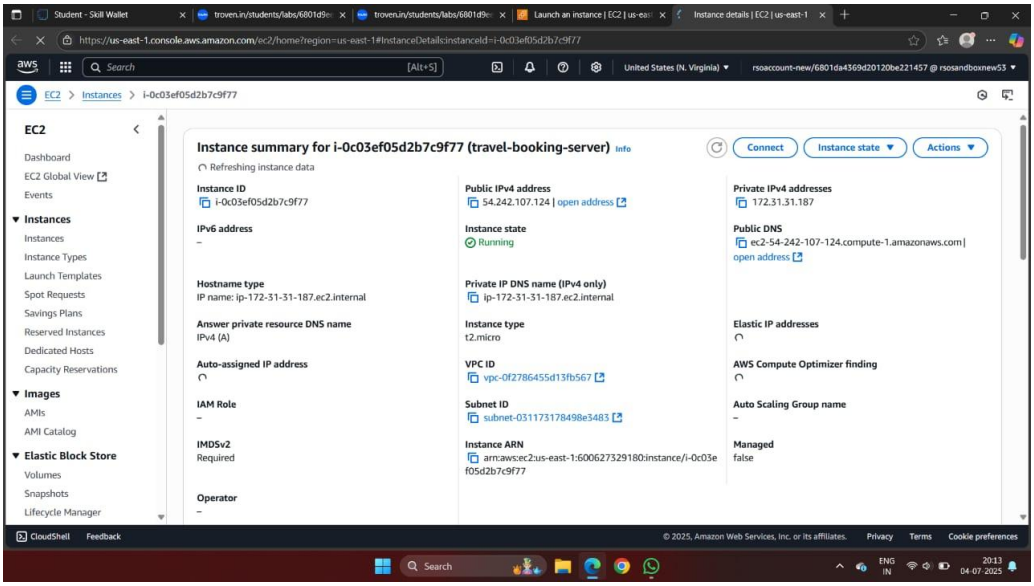


- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

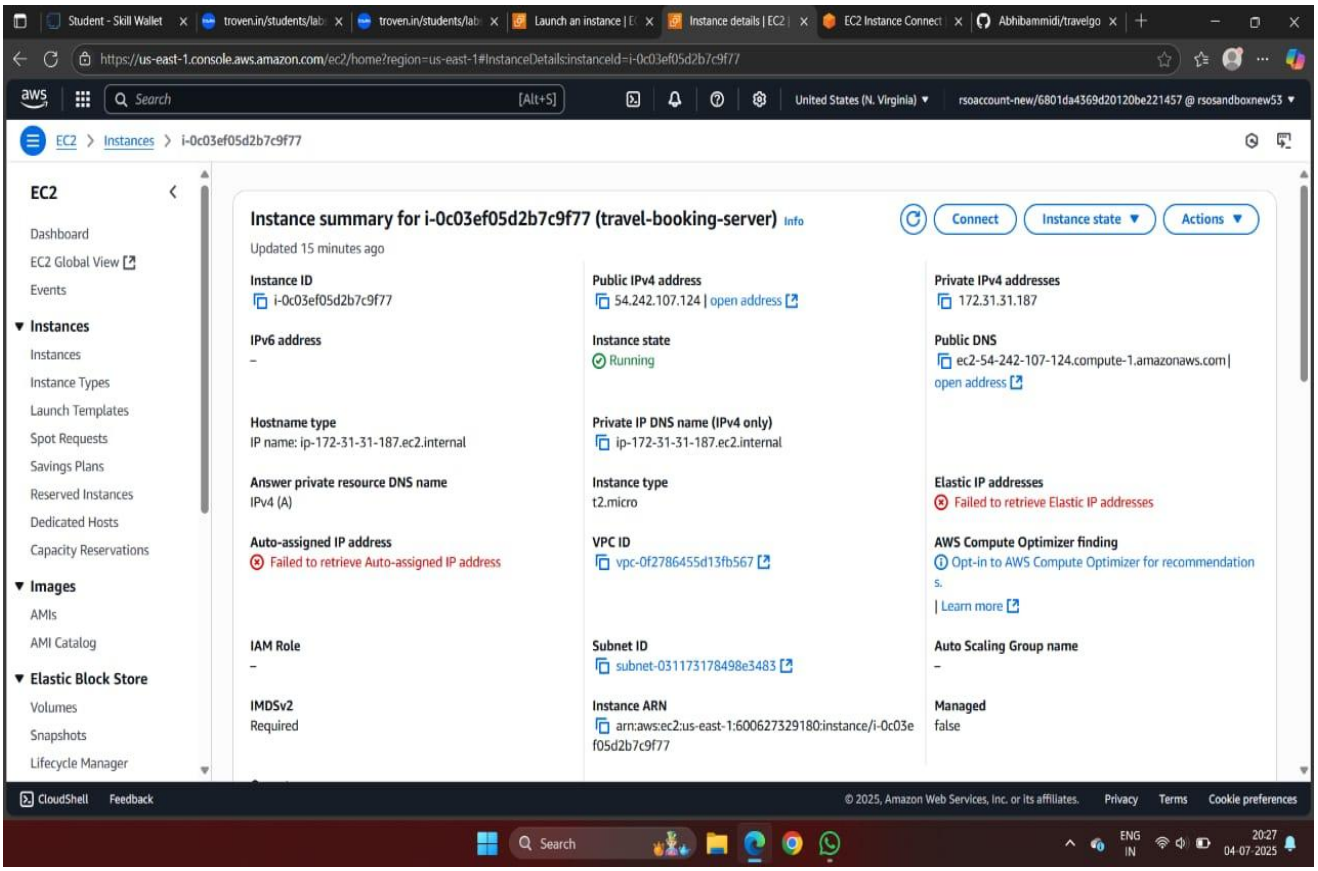


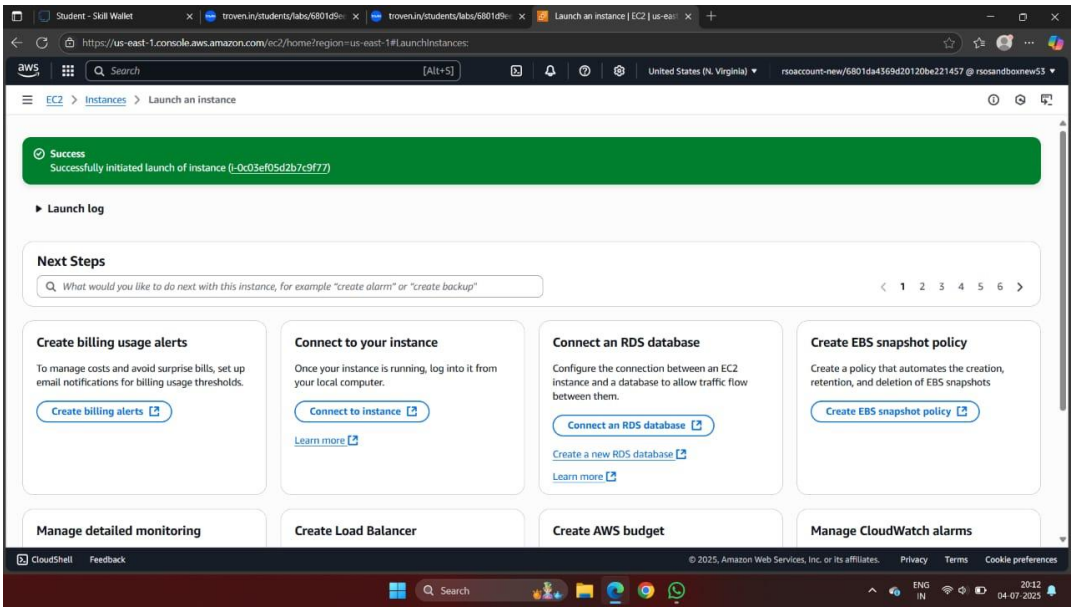
- Create and download the key pair for Server access.



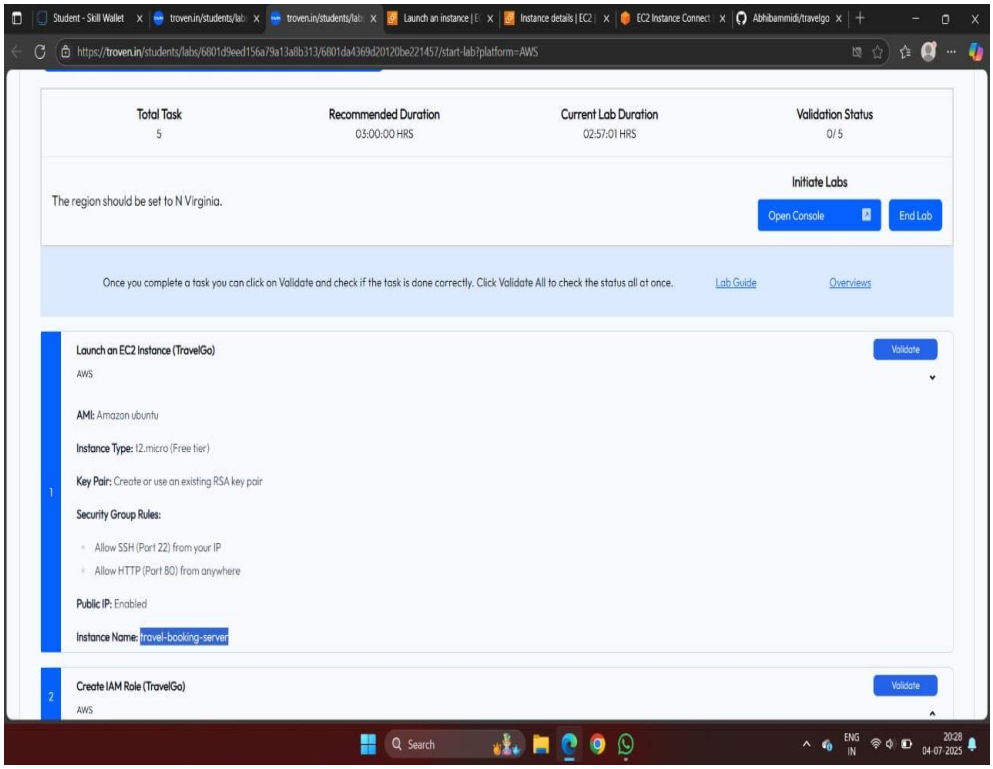


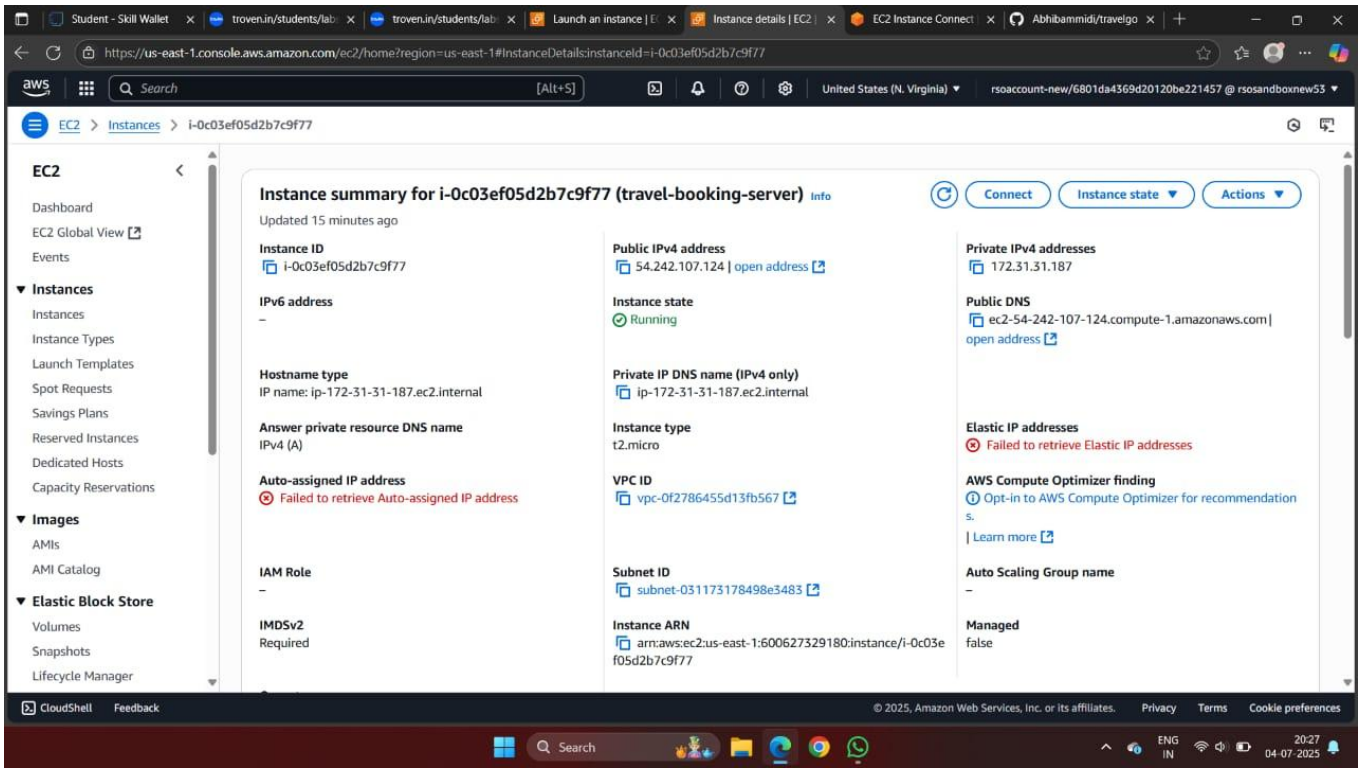
InstantLibrary.pem

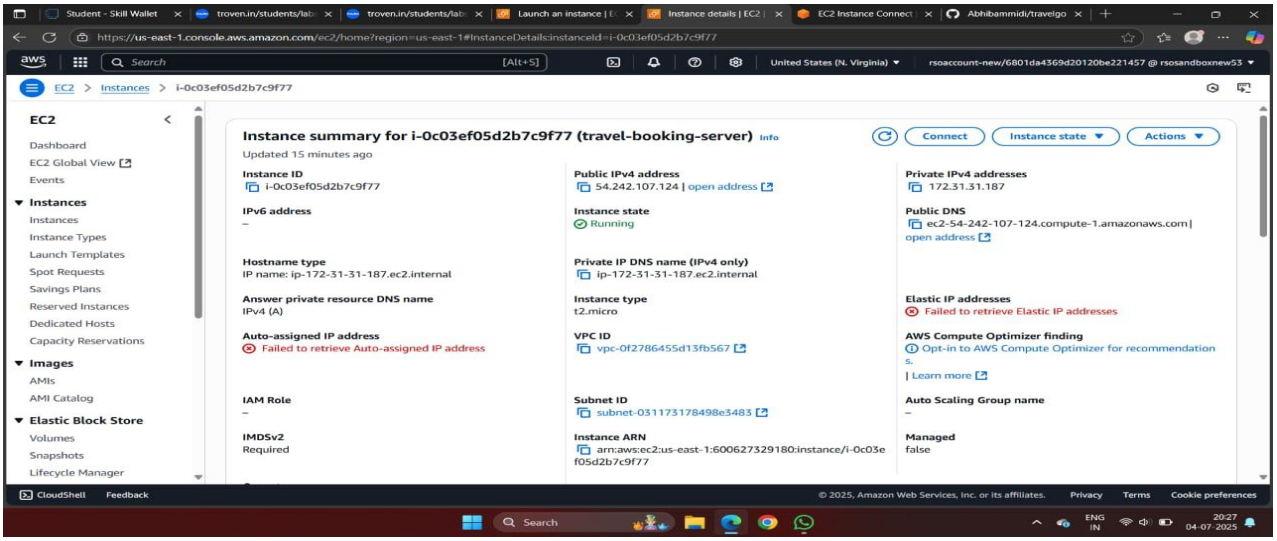

Student - Skill Wallet
troven.in/students/lab
troven.in/students/lab
Launch an instance | EC
Instance details | EC2
EC2 Instance Connect
Abhibammidi/travelgo
https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#InstanceDetails:instanceId=i-0c03ef05d2b7c9f77
aws
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew53
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
Instance summary for i-0c03ef05d2b7c9f77 (travel-booking-server) Info
Updated 15 minutes ago
Connect
Instance state
Actions
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
Elastic IP addresses
Failed to retrieve Elastic IP addresses
Auto-assigned IP address
Failed to retrieve Auto-assigned IP address
VPC ID
vpc-0f2786455d13fb567
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role
–
Subnet ID
subnet-031173178498e3483
Auto Scaling Group name
–
IMDSv2
Required
Instance ARN
arn:aws:ec2:us-east-1:600627329180:instance/i-0c03ef05d2b7c9f77
Managed
false
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Search
ENG
IN
20:27
04-07-2025

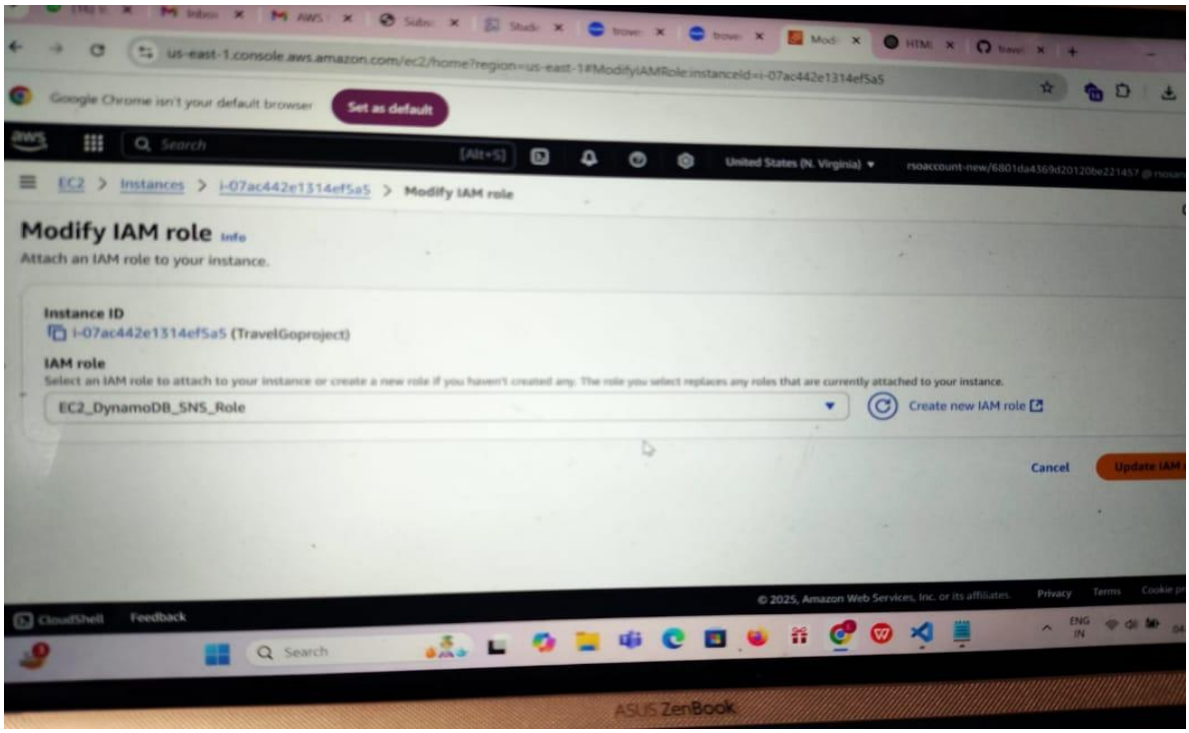


- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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

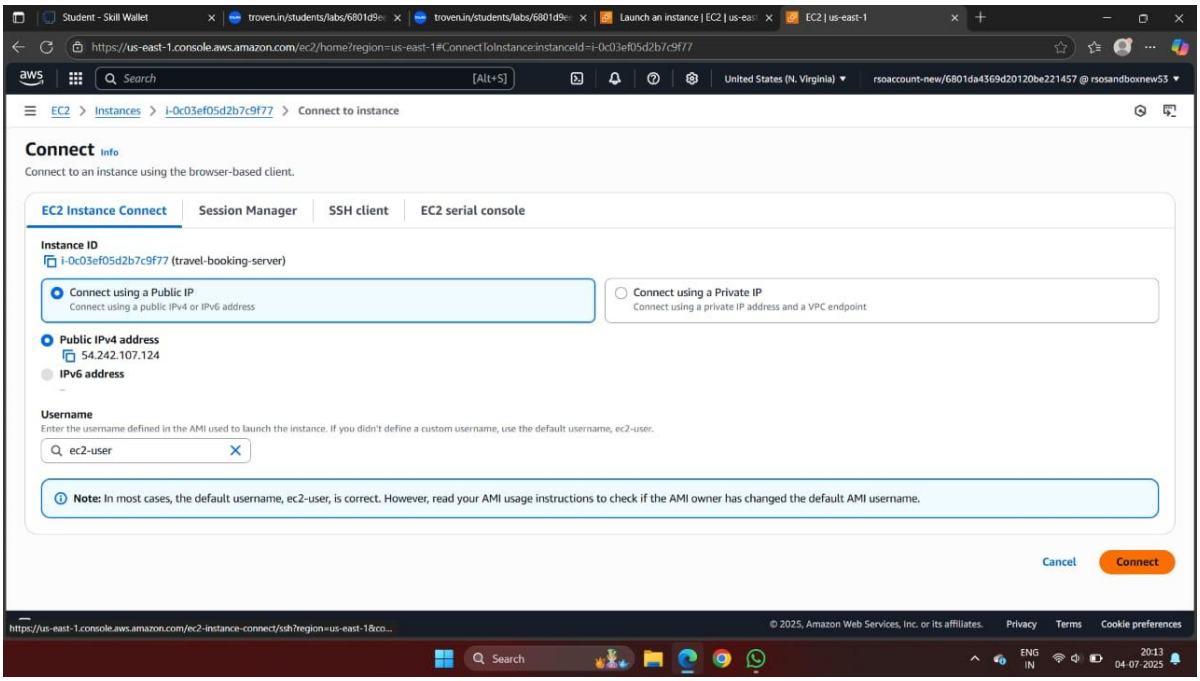

Total Task
5
Recommended Duration
03:00:00 HRS
Current Lab Duration
02:57:01 HRS
Validation Status
0/5
The region should be set to N Virginia.
Initiate Labs
Open Console
End Lab
Once you complete a task you can click on Validate and check if the task is done correctly. Click Validate All to check the status all at once.
Lab Guide
Overviews
Launch an EC2 Instance (TravelGo)
Validate
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
Create IAM Role (TravelGo)
Validate
AWS

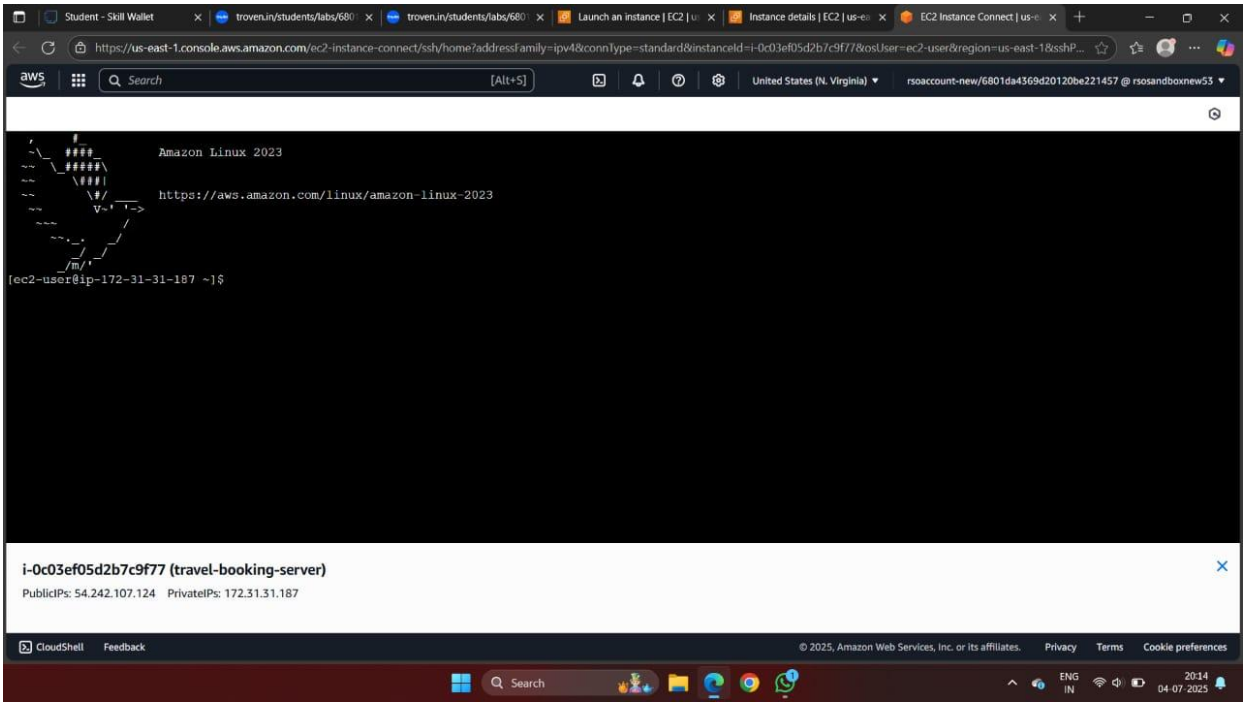

EC2 > Instances > i-0c03ef05d2b7c9f77
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Instance summary for i-0c03ef05d2b7c9f77 (travel-booking-server) Info
Updated 15 minutes ago
Connect
Instance state
Actions
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
Elastic IP addresses
Failed to retrieve Elastic IP addresses
Auto-assigned IP address
Failed to retrieve Auto-assigned IP address
VPC ID
vpc-0f2786455d13fb567
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role
–
Subnet ID
subnet-031173178498e3483
Auto Scaling Group name
–
IMDSv2
Required
Instance ARN
arn:aws:ec2:us-east-1:600627329180:instance/i-0c03ef05d2b7c9f77
Managed
false
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences





- Now connect the EC2 with the files


Student - Skill Wallet
Launch an instance | EC2 | us-east-1
EC2 | us-east-1
https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ConnectToInstance:instanceId=i-0c03ef05d2b7c9f77
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew53
EC2 > Instances > i-0c03ef05d2b7c9f77 > Connect to instance
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
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Student - Skill Wallet
Launch an instance | EC2 | us-ea
Instance details | EC2 | us-ea
EC2 Instance Connect | us-e
https://us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-0c03ef05d2b7c9f77&osUser=ec2-user&region=us-east-1&sshP...
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew53
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
[ec2-user@ip-172-31-31-187 ~]$
i-0c03ef05d2b7c9f77 (travel-booking-server)
PublicIPs: 54.242.107.124 PrivateIPs: 172.31.31.187
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

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.

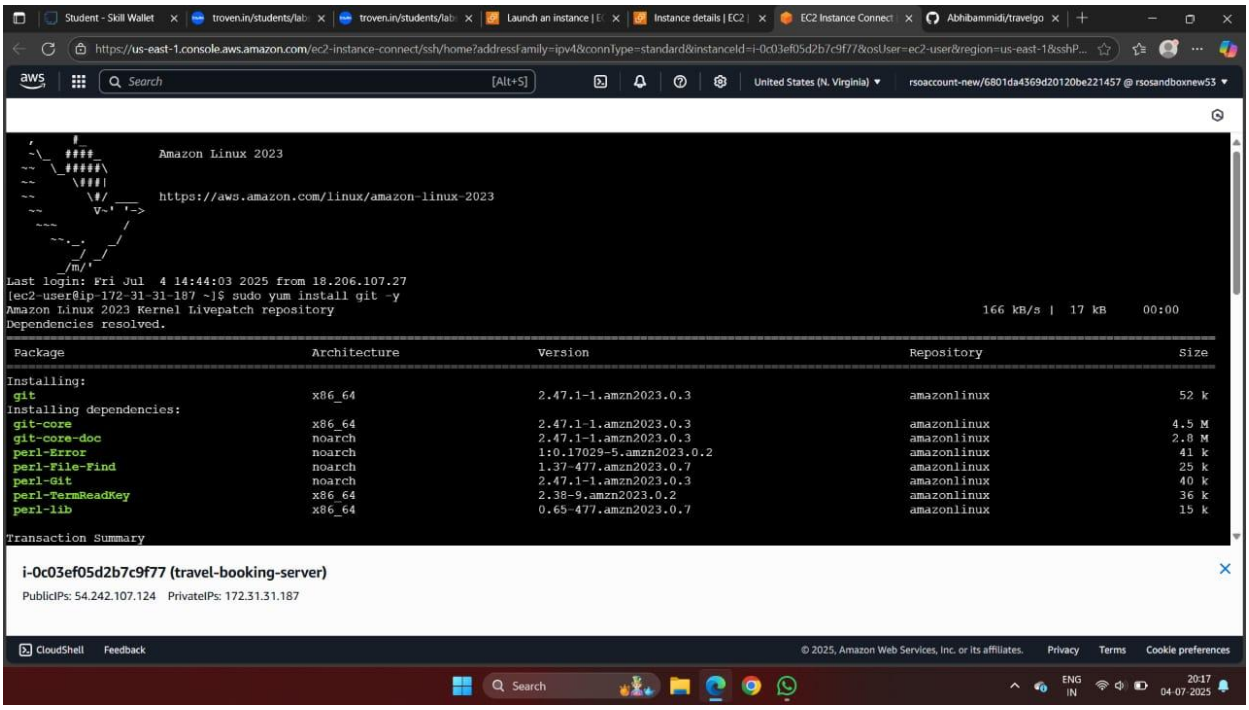
To navigate to the project directory, run the following command:

```
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

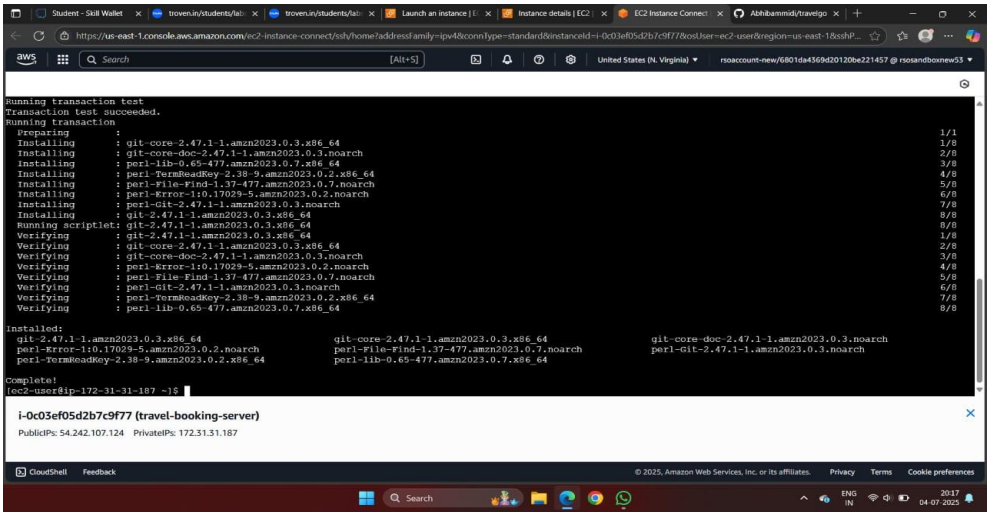
```
sudo flask run --host=0.0.0.0 --port=80
```



Verify the Flask app is running:

http://your-ec2-public-ip

- Run the Flask app on the EC2 instance



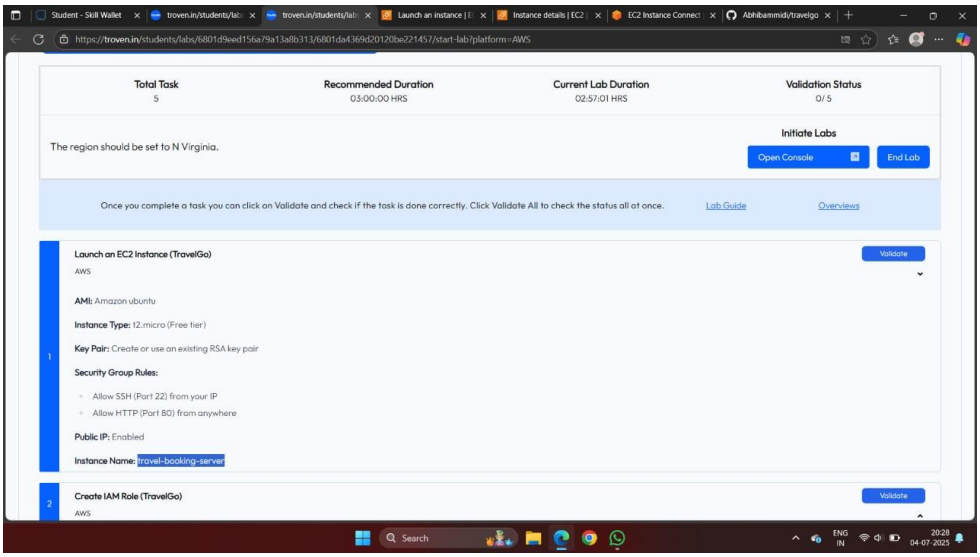
Access the website through:

PublicIPs: https://13.201.74.42/

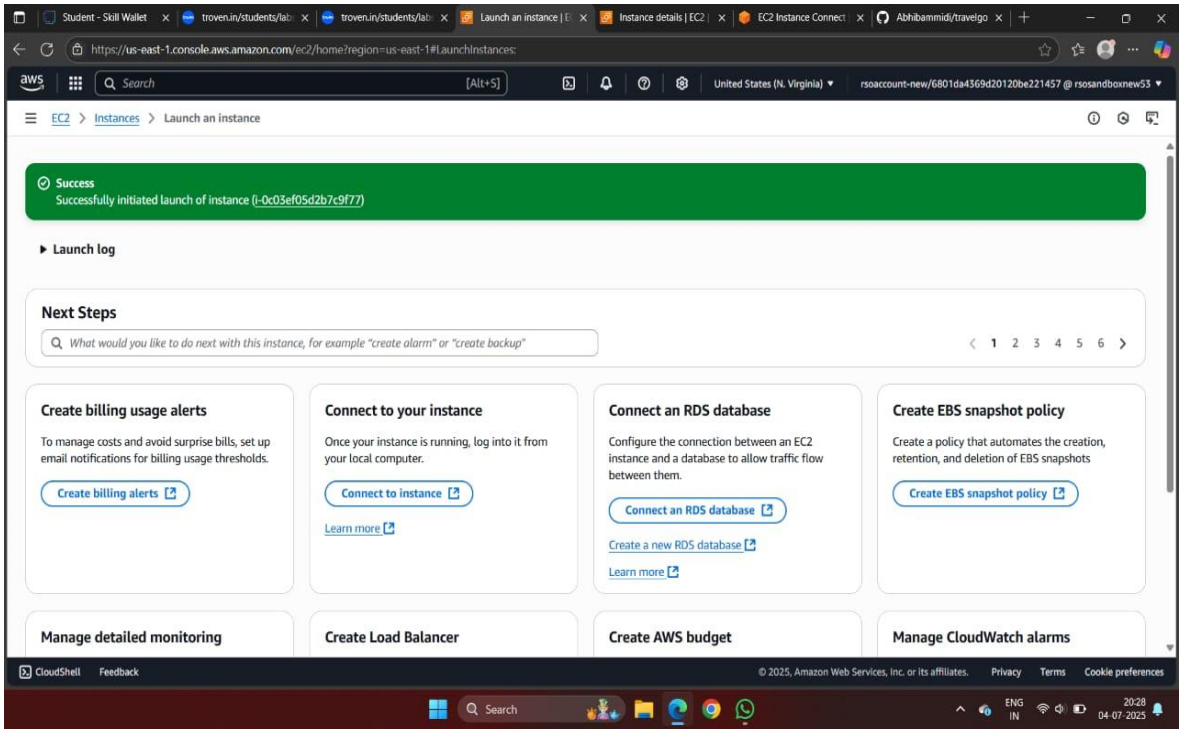
Milestone 8: Testing and Deployment

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

Login Page:



Register Page:



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.

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz