



TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using AWS

Project Description:

TravelGo is a full-stack, cloud-based travel booking platform designed to simplify the process of reserving buses, trains, flights, and hotels through a unified interface. Built using Flask as the backend framework, the application is deployed on Amazon EC2 and leverages DynamoDB for efficient storage of user data and bookings. TravelGo allows users to register, log in, search for transportation and accommodation options, and book their travel with ease. Once a booking is confirmed or cancelled, users receive real-time email notifications powered by AWS Simple Notification Service (SNS), keeping them informed throughout their journey.

The platform's user-friendly interface supports dynamic seat selection for buses, hotel filtering based on preferences such as luxury or budget, and provides booking summaries along with centralized cancellation management. By combining cloud scalability, responsive design, and secure session handling, TravelGo delivers a seamless and real-time travel planning experience for users.

Scenario 1: Hassle-Free Multi-Mode Travel Booking Experience

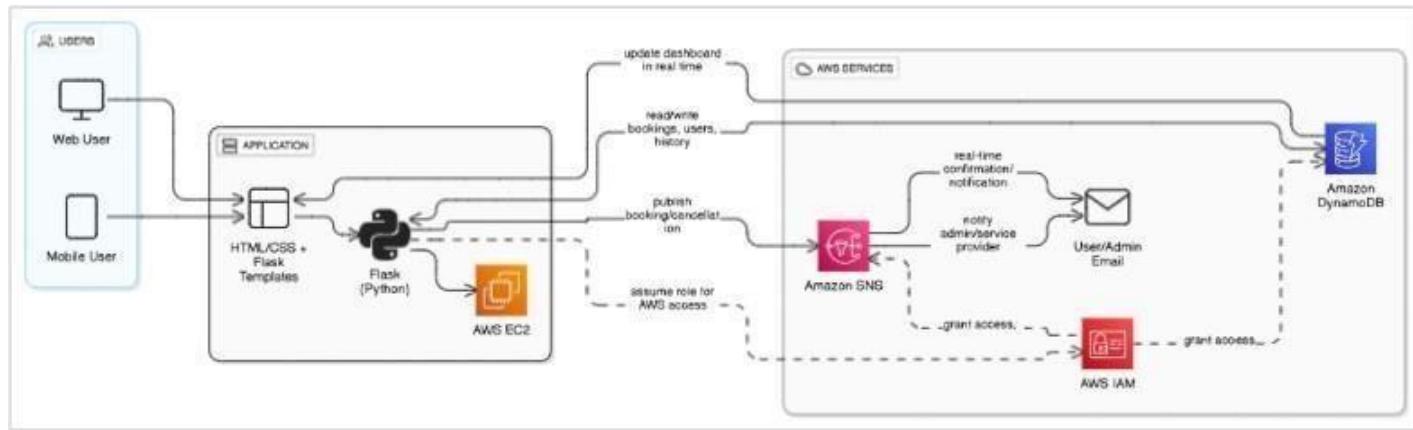
TravelGo offers users a unified platform to search and book buses, trains, flights, and hotels all in one place. For instance, a user planning a trip from Hyderabad to Bangalore can log in, select their preferred mode of transport, choose from available options, and proceed to booking. Flask manages the backend operations such as retrieving travel listings and processing user input in real-time. Hosted on AWS EC2, the platform remains responsive even during high-traffic hours like weekends or holiday seasons, allowing multiple users to browse and book without delay. **Scenario 2: RealTime Booking Confirmation with AWS SNS**

Once a booking is made—whether it's a train ticket or a hotel stay—TravelGo uses AWS SNS to instantly notify the user. For example, after a student books a hotel in Chennai, SNS sends a real-time email notification confirming the booking with all the relevant details. This notification is triggered from the Flask backend after the booking is successfully recorded in DynamoDB. Additionally, SNS can alert admin or service providers, ensuring transparency and real-time updates on every transaction.

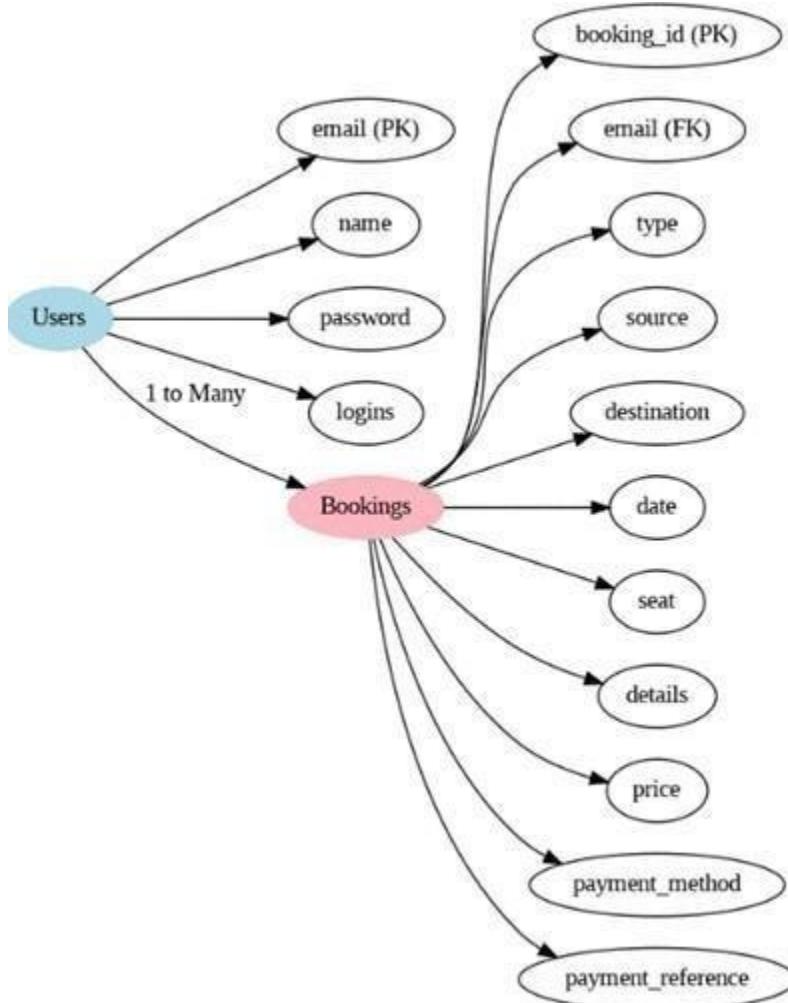
Scenario 3: Dynamic Dashboard with Personal Travel History

TravelGo features a dynamic user dashboard that displays all past and upcoming bookings for the logged-in user. For example, a user who has booked a flight and a hotel can view these bookings categorized by type, along with dates, price, and cancellation options. Flask fetches this data from AWS DynamoDB, which persistently stores all user bookings. The dashboard UI, powered by responsive HTML/CSS and Flask templates, ensures users can review or manage bookings anytime, from any device, with real-time updates and quick cancellation workflows supported.

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](#)

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 Book Requests.

3. SNS Notification Setup

Activity 3.1: Create SNS topics for book request notifications.

Activity 3.2: Subscribe users and library staff 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

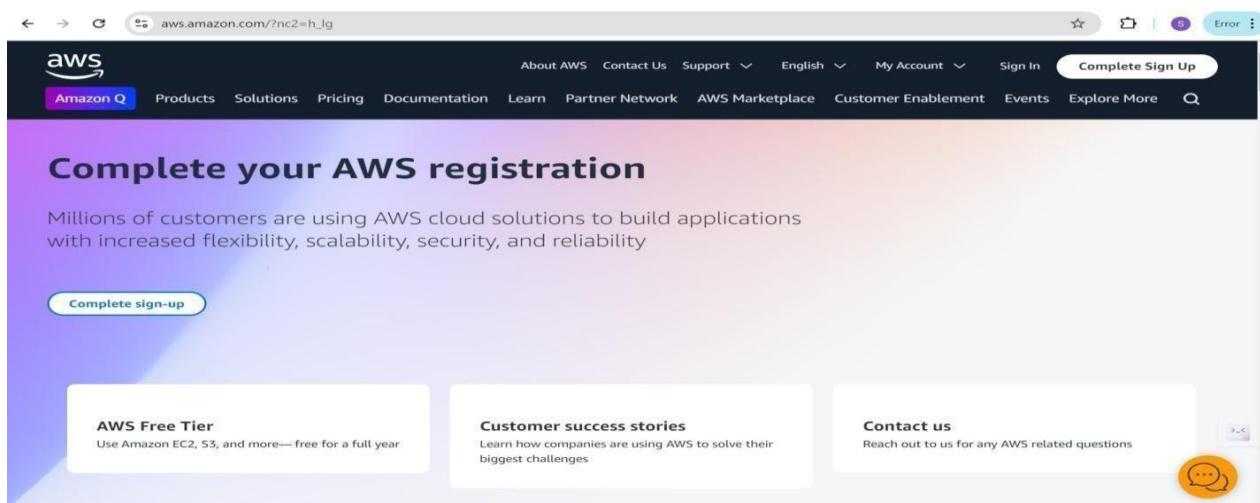
Activity 7.2: Run the Flask App

8. Testing and Deployment

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

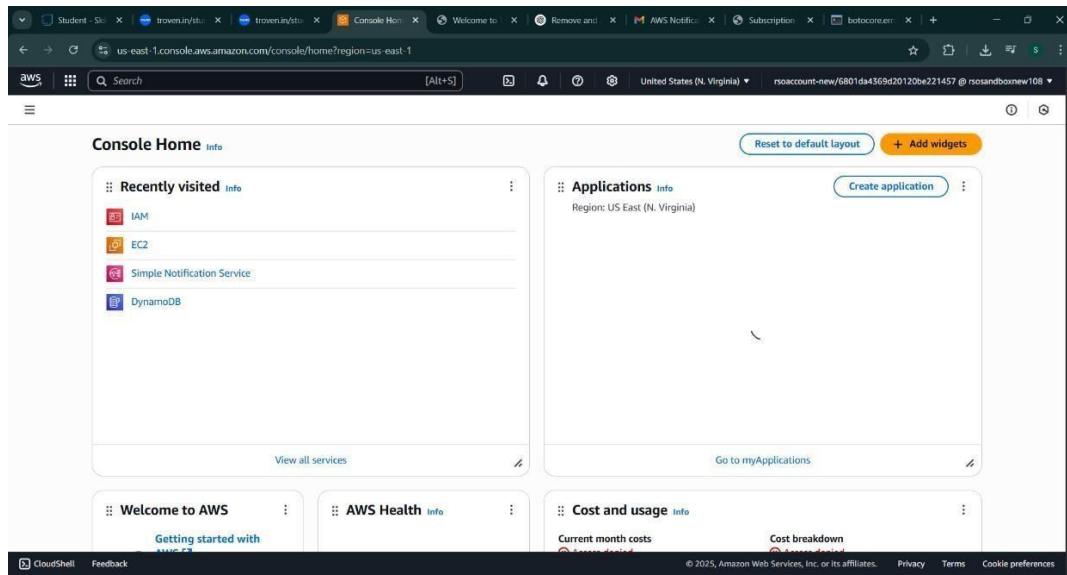
Milestone 1: AWS Account Setup and Login

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

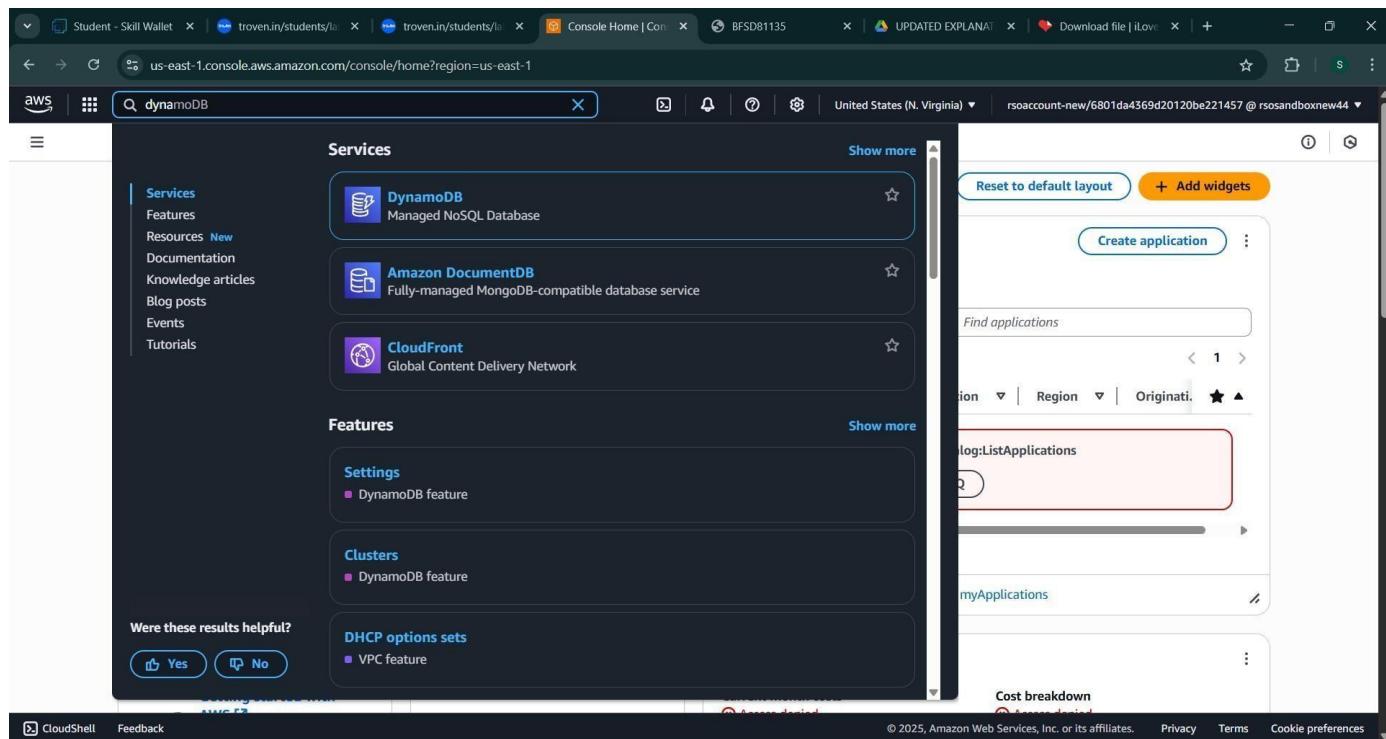


- **Activity 1.2: Log in to the AWS Management Console**

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



The screenshot shows the AWS Console Home page. At the top, there are links for IAM, EC2, Simple Notification Service, and DynamoDB under 'Recently visited'. Below this is a 'Welcome to AWS' section with a 'Getting started with AWS' link. To the right, there's an 'Applications' section showing a single application entry for 'Simple Notification Service'. A 'Cost and usage' summary is also present. The bottom of the page includes links for CloudShell, Feedback, and various AWS services.

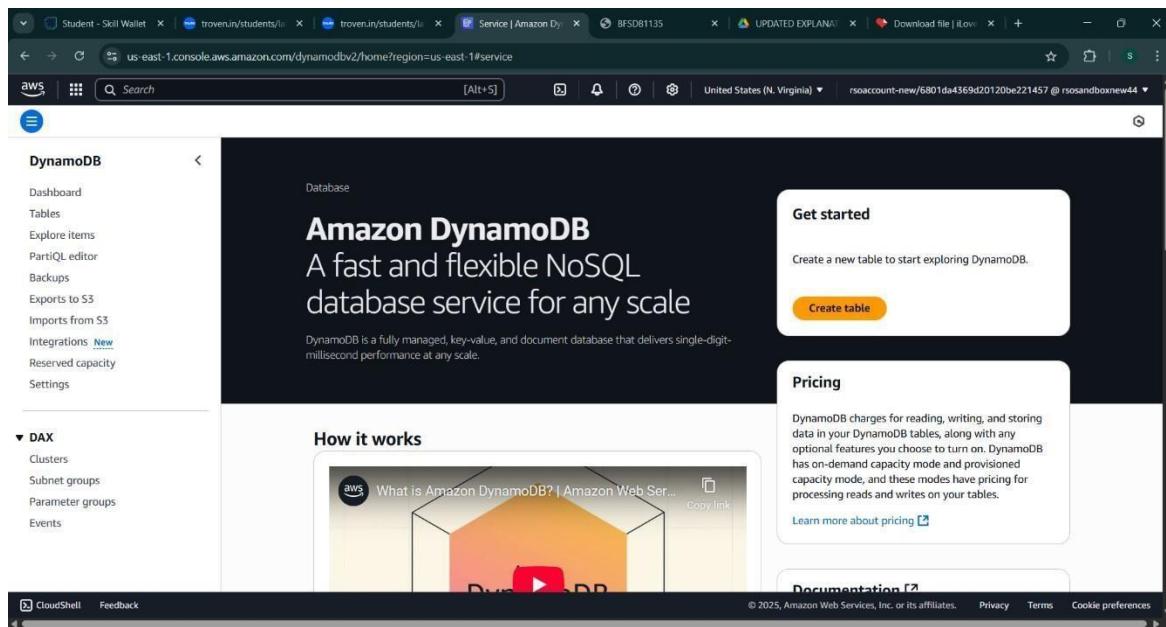


The screenshot shows the AWS search results for 'dynamoDB'. The search bar at the top has 'dynamoDB' typed in. The main results list includes 'DynamoDB' (Managed NoSQL Database), 'Amazon DocumentDB' (Fully-managed MongoDB-compatible database service), and 'CloudFront' (Global Content Delivery Network). On the left, a sidebar lists 'Services' like Features, Resources, Documentation, Knowledge articles, Blog posts, Events, and Tutorials. On the right, a sidebar titled 'Find applications' shows a list of applications including 'log:ListApplications' and 'myApplications'. A 'Cost breakdown' section is also visible.

Milestone 2: DynamoDB Database Creation and Setup

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

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



The screenshot shows the Amazon DynamoDB console homepage. On the left, there is a navigation sidebar with sections for 'DynamoDB' (Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, Settings) and 'DAX' (Clusters, Subnet groups, Parameter groups, Events). The main content area has a dark header 'Database' and a large title 'Amazon DynamoDB' with the subtitle 'A fast and flexible NoSQL database service for any scale'. Below this, a paragraph describes DynamoDB as a fully managed, key-value, and document database that delivers single-digit-millisecond performance at any scale. There are three callout boxes: 'Get started' (Create a new table to start exploring DynamoDB, with a 'Create table' button), 'Pricing' (DynamoDB charges for reading, writing, and storing data in your DynamoDB tables, along with any optional features you choose to turn on. DynamoDB has on-demand capacity mode and provisioned capacity mode, and these modes have pricing for processing reads and writes on your tables. A 'Learn more about pricing' link is provided), and 'Documentation' (with a 'View documentation' button). At the bottom of the page, there are links for CloudShell, Feedback, and copyright information: © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences.

9.
10.



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

11.

Student - x trover.in/x trover.in/x Create tab Welcome x Remove all x AWS Notif x Subscript x Subscript x botcore... x + - _ S ...

us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#create-table

DynamoDB > Tables > Create table

Table details Info

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

Table name

This will be used to identify your table.

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

Partition key

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

String

1 to 255 characters and case sensitive.

Sort key - optional

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

String

1 to 255 characters and case sensitive.

Table settings

Default settings
 The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

Customize settings
 Use these advanced features to make DynamoDB work better for your needs.

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



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

Tags

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

No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

[Cancel](#)
[Create table](#)

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

us-east-1.console.aws.amazon.com/dynamodbv2/home#create-table

DynamoDB > Tables > Create table

Table details Info

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

Table name

This will be used to identify your table.

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

Partition key

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

String

1 to 255 characters and case sensitive.

Sort key - optional

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

String

1 to 255 characters and case sensitive.

Table settings

Default settings

The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

Customize settings

Use these advanced features to make DynamoDB work better for your needs.

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

us-east-1.console.aws.amazon.com/dynamodbv2/home#create-table

DynamoDB > Tables > Create table

Create table

Table details Info

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

Table name

This will be used to identify your table.

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

Partition key

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

String

1 to 255 characters and case sensitive.

Sort key - optional

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

String

1 to 255 characters and case sensitive.

Table settings

Default settings

The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

Customize settings

Use these advanced features to make DynamoDB work better for your needs.

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

Screenshot of the AWS DynamoDB 'Create table' wizard.

The table configuration is as follows:

| | | |
|------------------------------|---------------|-----|
| Maximum write capacity units | - | Yes |
| Local secondary indexes | - | No |
| Global secondary indexes | - | Yes |
| Encryption key management | AWS owned key | Yes |
| Deletion protection | Off | Yes |
| Resource-based policy | Not active | Yes |

Tags

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

No tags are associated with the resource.

[Add new tag](#)

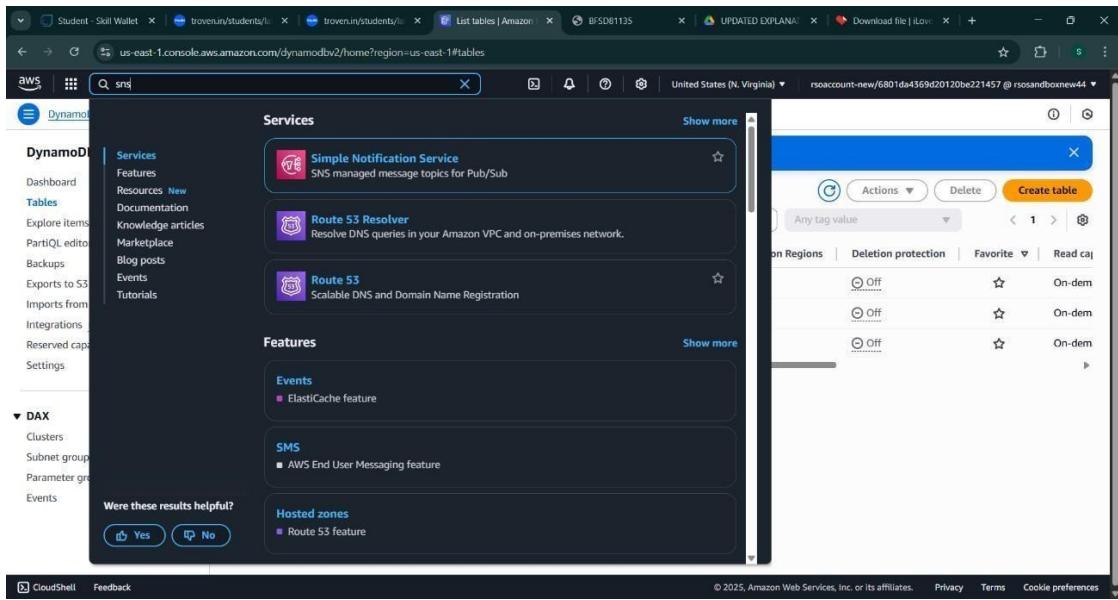
You can add 50 more tags.

ⓘ This table will be created with auto scaling deactivated. You do not have permissions to turn on auto scaling.

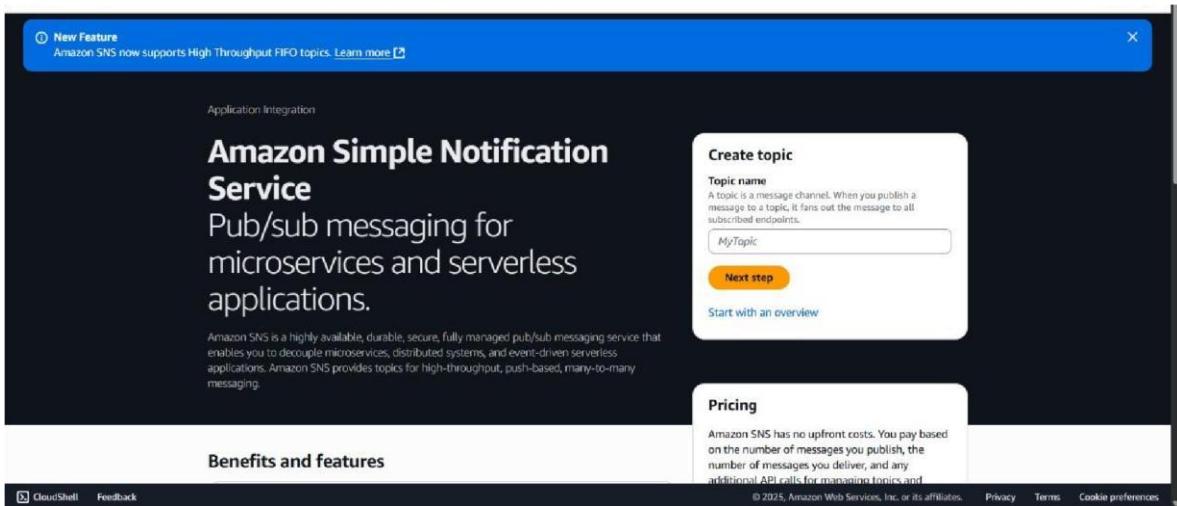
[Cancel](#) [Create table](#)

Milestone 3: SNS Notification Setup

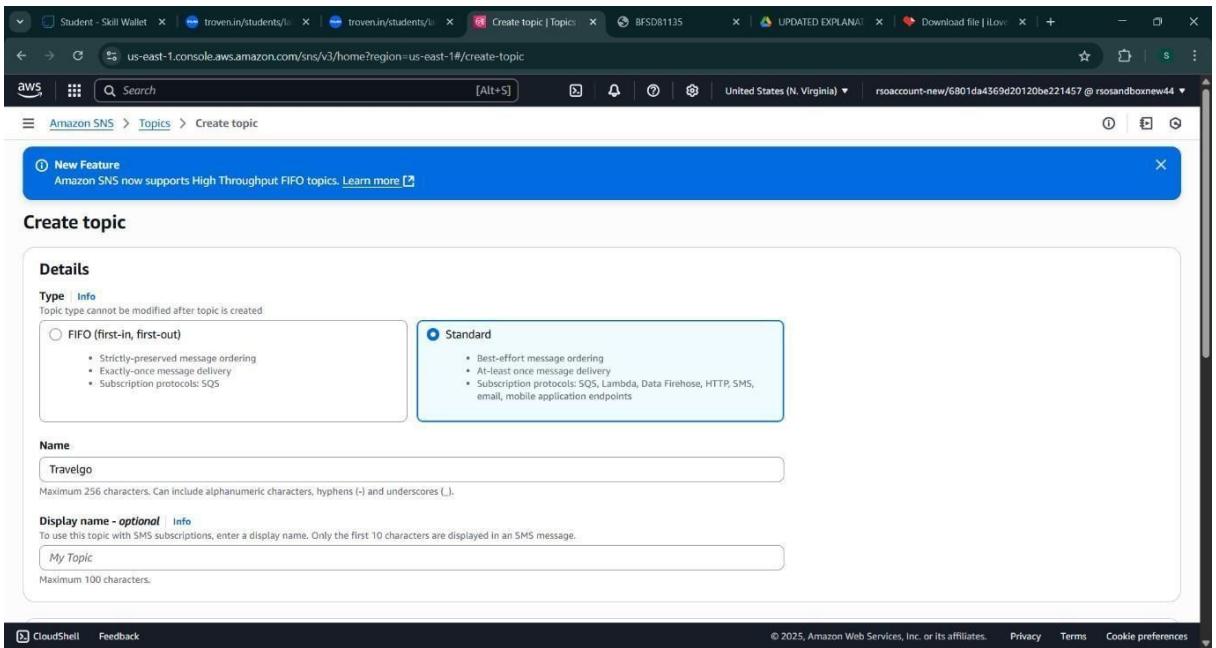
- **Activity 3.1: Create SNS topics for sending email notifications to users**



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



- Click on **Create Topic** and choose a name for the 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
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Standard

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

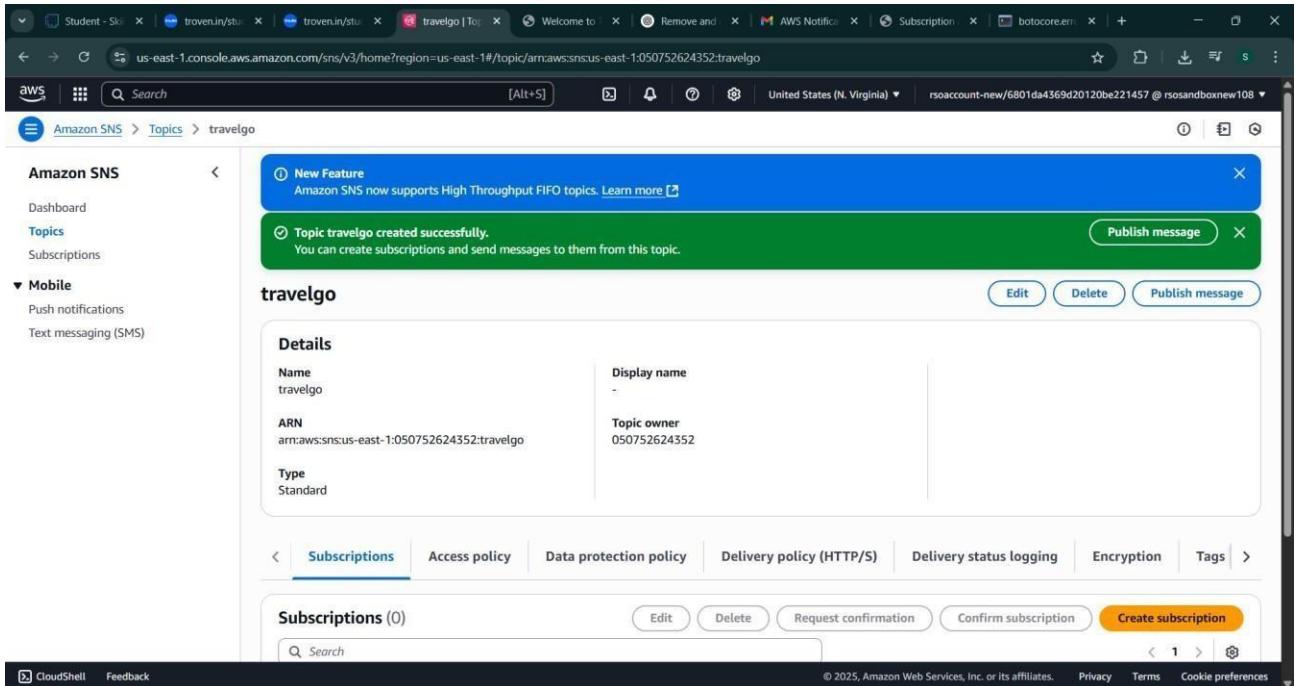
Name: Travelgo
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

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.

Description: My Topic
Maximum 100 characters.

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

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



Welcome to AWS Notifications

Amazon SNS > Topics > travelgo

travelgo

Details

Name: travelgo
Display name: -
ARN: arn:aws:sns:us-east-1:050752624352:travelgo
Topic owner: 050752624352
Type: Standard

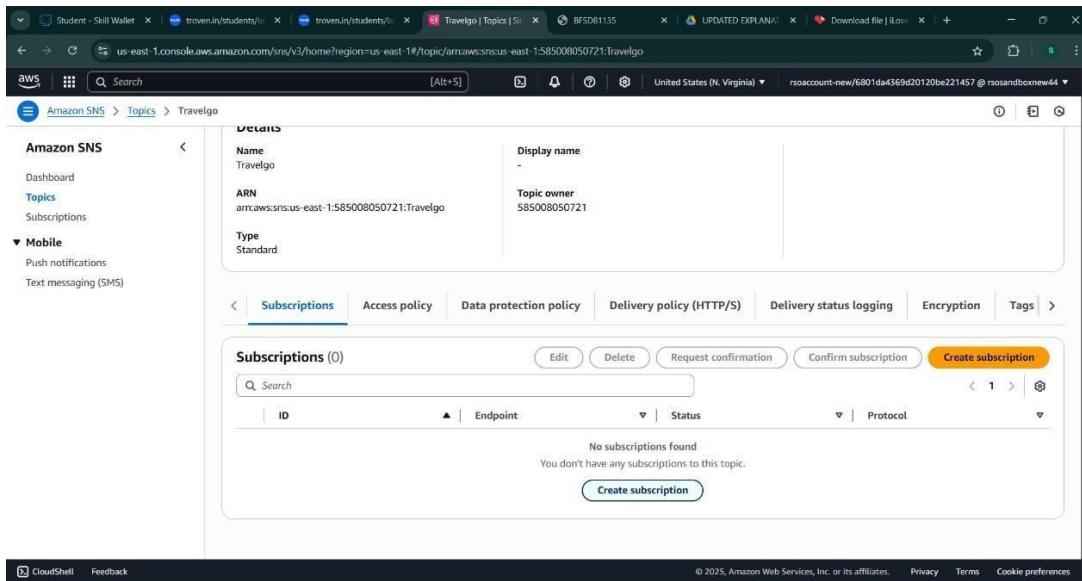
Subscriptions (0)

Edit Delete Request confirmation Confirm subscription Create subscription

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

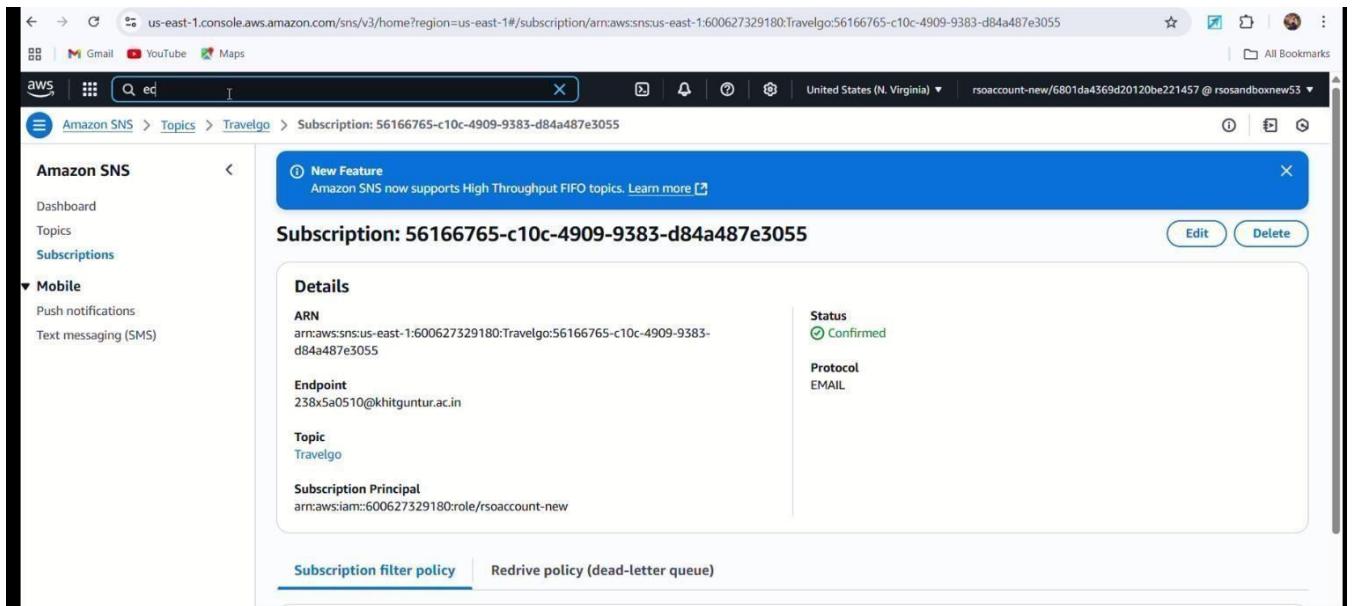
- Configure the SNS topic and note down the **Topic ARN**.

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



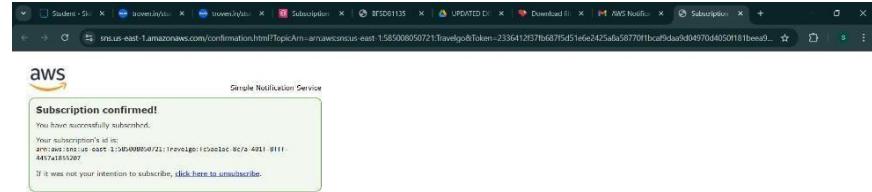
The screenshot shows the AWS SNS Topics page for the 'Travelgo' topic. The 'Details' section displays the topic name ('Travelgo'), ARN ('arn:aws:sns:us-east-1:585008050721:Travelgo'), Topic owner ('585008050721'), and Type ('Standard'). Below this, the 'Subscriptions' tab is selected, showing a table with one row: 'No subscriptions found'. A message indicates 'You don't have any subscriptions to this topic.' A 'Create subscription' button is visible at the bottom of the table.

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



The screenshot shows the AWS SNS Subscription details page for the 'Subscription: 56166765-c10c-4909-9383-d84a487e3055'. The 'Details' section shows the ARN ('arn:aws:sns:us-east-1:600627329180:Travelgo:56166765-c10c-4909-9383-d84a487e3055'), Endpoint ('238x5a0510@khitguntur.ac.in'), Topic ('Travelgo'), and Subscription Principal ('arn:aws:iam::600627329180:role/rsoaccount-new'). The 'Status' is listed as 'Confirmed' with a green checkmark. The 'Protocol' is set to 'EMAIL'. At the top right, there are 'Edit' and 'Delete' buttons.

After subscription request for the mail confirmation

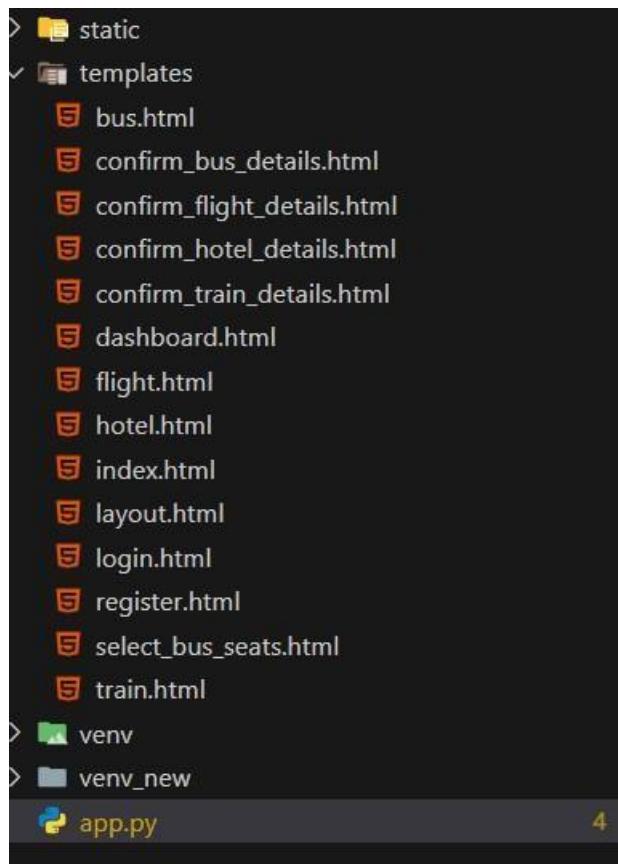


- Successfully done with the SNS mail subscription and setup, now store the ARN link.

Milestone 4:Backend Development and Application Setup

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

- File Explorer Structure



Description: set up the **TravelGO** project with an `app.py` file, a `static/` folder for assets, and a `templates/` directory containing all required HTML pages like home, login, register, booking-specific pages (e.g., `bus.html`, `train.html`)

Description of the code :

- **Flask App Initialization**

```

from flask import Flask, render_template, request, redirect, url_for, session, jsonify, flash
import boto3
from boto3.dynamodb.conditions import Key, Attr
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
from decimal import Decimal
import uuid
import random
  
```

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

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

- **Dynamodb Setup:**

```
3 # AWS Setup using IAM Role
4 REGION = 'us-east-1' # Replace with your actual AWS region
5 dynamodb = boto3.resource('dynamodb', region_name=REGION)
6 sns_client = boto3.client('sns', region_name=REGION)
7
8 users_table = dynamodb.Table('travelgo_users')
9 trains_table = dynamodb.Table('trains') # Note: This table is declared by
0 bookings_table = dynamodb.Table('bookings')
1
```

Description: initialize the DynamoDB resource for the us-east-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.

```
SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:418272775181:TravelGo:b7d6f5bc-7710-49de-97bc-ddecff5fd023'
topic ARN

# Function to send SNS notifications

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: Could not send notification - {e}")
        # Optionally, flash an error message to the user or log it more robustly.
    # Function
```

- **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:**

Description: define /register route to validate registration form fields, hash the user

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

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):**

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if 'username' in session:
        session.pop('username', None)
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        # user = users_collection.find_one({'email': email}) # MongoDB
        user = users_table.get_item(Key={'email': email}) # DynamoDB
        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')
```

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,Bus,Train,Flight,Hotel Routes:**

```

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

    user_email = session['email']
    # user_bookings = list(bookings_collection.find({'user_email': user_email}).sort('booking_date', -1)) # MongoDB
    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

    for booking in bookings:
        if '_id' in booking and isinstance(booking['_id'], ObjectId):
            booking['_id'] = str(booking['_id'])

        # Determine vehicle_type for display based on booking_type
        if booking.get('booking_type') == 'bus':
            booking['vehicle_type'] = booking.get('type', 'Bus')
            # Add seat information for bus bookings
            if booking.get('selected_seats'):
                booking['seats_display'] = ', '.join(booking['selected_seats'])
            else:
                booking['seats_display'] = 'N/A'
        elif booking.get('booking_type') == 'train':
            booking['vehicle_type'] = f"Train {booking.get('train_number', '')}" if booking.get('train_number') else "Train"
            # Add seat information for train bookings
            if booking.get('selected_seats'):
                booking['seats_display'] = ', '.join(booking['selected_seats'])
            else:
                booking['seats_display'] = 'N/A'
        elif booking.get('booking_type') == 'flight':
            booking['vehicle_type'] = f"Flight {booking.get('flight_number', '')} ({booking.get('airline', '')})"
            # Add seat information for flights
            if booking.get('selected_seats'):
                booking['seats_display'] = ', '.join(booking['selected_seats'])
            else:
                booking['seats_display'] = 'N/A'
        else:
            booking['vehicle_type'] = booking.get('booking_type', 'N/A')

    return render_template('dashboard.html', username=user_email, bookings=bookings)

```

Description: define /dashboard-page to render the main homepage, to handle booking selection and redirection.

- **confirmbooking Routes:**

```

@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'))

    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 = float(request.args.get('price'))
    travel_date = request.args.get('date')
    num_persons = int(request.args.get('persons'))
    train_id = request.args.get('trainId')

    total_price = price_per_person * num_persons

    booking_details = {
        'name': name,
        'train_number': train_number,
        'source': source,
        'destination': destination,
        'departure_time': departure_time,
        'arrival_time': arrival_time,
        'price_per_person': Decimal(price_per_person),
        'travel_date': travel_date,
        'num_persons': num_persons,
        'total_price': Decimal(total_price),
        'item_id': train_id,
        'booking_type': 'train',
        'user_email': session['email']
    }
    session['pending_booking'] = booking_details
    return render_template('confirm_train_details.html', booking=booking_details)

```

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:

```

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

```

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

Deployment Code:

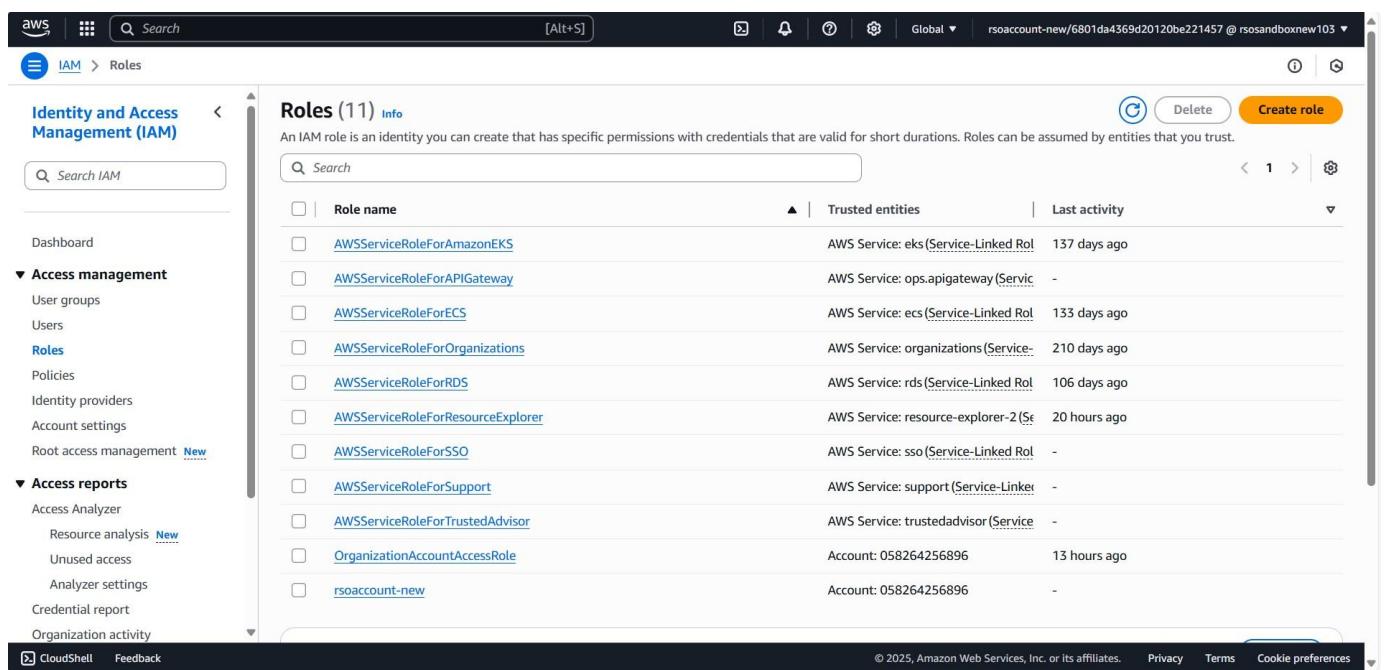
```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Description: start the Flask server to listen on all network interfaces (0.0.0.0) 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.



The screenshot shows the AWS IAM Roles page with 11 roles listed:

| Role name | Trusted entities | Last activity |
|---|---------------------------------------|---------------|
| AWSServiceRoleForAmazonEKS | AWS Service: eks (Service-Linked Rol) | 137 days ago |
| AWSServiceRoleForAPIGateway | AWS Service: ops.apigateway (Servic | - |
| AWSServiceRoleForECS | AWS Service: ecs (Service-Linked Rol) | 133 days ago |
| AWSServiceRoleForOrganizations | AWS Service: organizations (Service- | 210 days ago |
| AWSServiceRoleForRDS | AWS Service: rds (Service-Linked Rol) | 106 days ago |
| AWSServiceRoleForResourceExplorer | AWS Service: resource-explorer-2 (Se | 20 hours ago |
| AWSServiceRoleForSSO | AWS Service: sso (Service-Linked Rol | - |
| AWSServiceRoleForSupport | AWS Service: support (Service-Linker | - |
| AWSServiceRoleForTrustedAdvisor | AWS Service: trustedadvisor (Service | - |
| OrganizationAccountAccessRole | Account: 058264256896 | 13 hours ago |
| rsoaccount-new | Account: 058264256896 | - |

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

Trusted entity type:

- AWS service: Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account: Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity: Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- SAML 2.0 federation: Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- Custom trust policy: Create a custom trust policy to enable others to perform actions in this account.

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 connect and terminate EC2 instances on your behalf.

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

Permissions policies (2/1061) Info

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

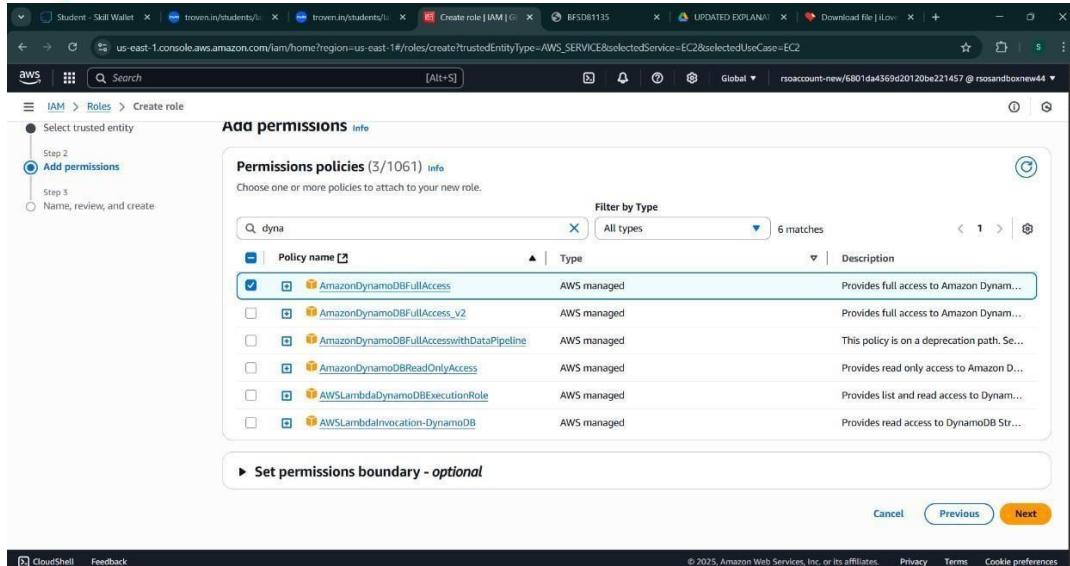
Filter by Type: All types | 33 matches

| Policy name | Type | Description |
|---|-------------|---|
| AmazonEC2ContainerRegistryFullAccess | AWS managed | Provides administrative access to Amazon E... |
| AmazonEC2ContainerRegistryPowerUser | AWS managed | Provides full access to Amazon EC2 Co... |
| AmazonEC2ContainerRegistryPullOnly | AWS managed | Provides access to pull images from A... |
| AmazonEC2ContainerRegistryReadOnly | AWS managed | Provides read-only access to Amazon E... |
| AmazonEC2ContainerServiceAutoscaleRole | AWS managed | Policy to enable Task AutoScaling for A... |
| AmazonEC2ContainerServiceEventsRole | AWS managed | Policy to enable CloudWatch Events fo... |
| AmazonEC2ContainerServiceforEC2Role | AWS managed | Default policy for the Amazon EC2 Rol... |
| AmazonEC2ContainerServiceRole | AWS managed | Default policy for Amazon ECS service ... |
| <input checked="" type="checkbox"/> AmazonEC2FullAccess | AWS managed | Provides full access to Amazon EC2 via... |
| AmazonEC2ReadOnlyAccess | AWS managed | Provides read only access to Amazon E... |
| AmazonEC2RoleforAWSCodeDeploy | AWS managed | Provides EC2 access to S3 bucket to do... |
| AmazonEC2PolicyForAWSCodeDeployInitiated | AWS managed | Provides EC2 limited access to S3 buck... |

● 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.



Add permissions

Permissions policies (3/1061) Info

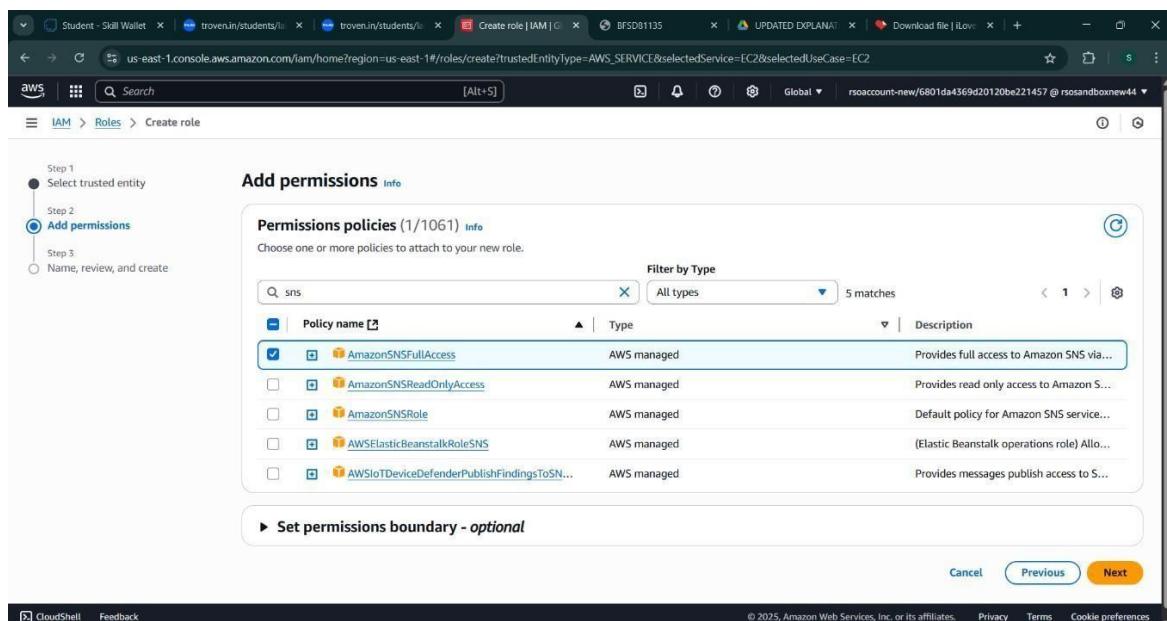
Choose one or more policies to attach to your new role.

Filter by Type All types 6 matches

| Policy name | Type | Description |
|---|-------------|---|
| <input checked="" type="checkbox"/>  AmazonDynamoDBFullAccess | AWS managed | Provides full access to Amazon Dynamo... |
| <input type="checkbox"/>  AmazonDynamoDBFullAccess_v2 | AWS managed | Provides full access to Amazon Dynam... |
| <input type="checkbox"/>  AmazonDynamoDBFullAccessWithDataPipeline | AWS managed | This policy is on a deprecation path. Se... |
| <input type="checkbox"/>  AmazonDynamoDBReadOnlyAccess | AWS managed | Provides read only access to Amazon D... |
| <input type="checkbox"/>  AWSLambdaDynamoDBExecutionRole | AWS managed | Provides list and read access to Dynam... |
| <input type="checkbox"/>  AWSLambdaInvocation-DynamoDB | AWS managed | Provides read access to DynamoDB Str... |

▶ Set permissions boundary - optional

Cancel Previous Next



Add permissions

Permissions policies (1/1061) Info

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

Filter by Type All types 5 matches

| Policy name | Type | Description |
|---|-------------|---|
| <input checked="" type="checkbox"/>  AmazonSNSFullAccess | AWS managed | Provides full access to Amazon SNS via... |
| <input type="checkbox"/>  AmazonSNSReadOnlyAccess | AWS managed | Provides read only access to Amazon S... |
| <input type="checkbox"/>  AmazonSNSRole | AWS managed | Default policy for Amazon SNS service... |
| <input type="checkbox"/>  AWSxElasticBeanstalkRoleSNS | AWS managed | (Elastic Beanstalk operations role) Allo... |
| <input type="checkbox"/>  AWSIoTDeviceDefenderPublishFindingsToSN... | AWS managed | Provides messages publish access to S... |

▶ Set permissions boundary - optional

Cancel Previous Next

Milestone 6: EC2 Instance Setup

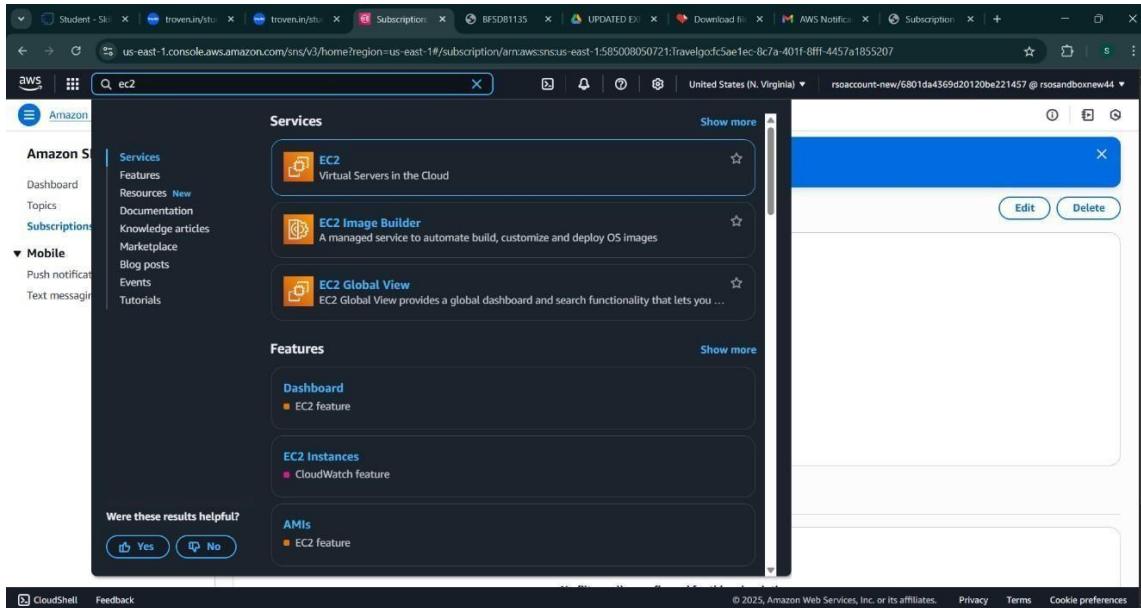
- Note: Load your Flask app and Html files into GitHub repository.

| | |
|---|--------------------------|
|  static | Added full project files |
|  templates | app.py changed |
|  venv | Added full project files |
|  venv_new | Added full project files |
|  README.md | first commit |
|  app.py | Update app.py |

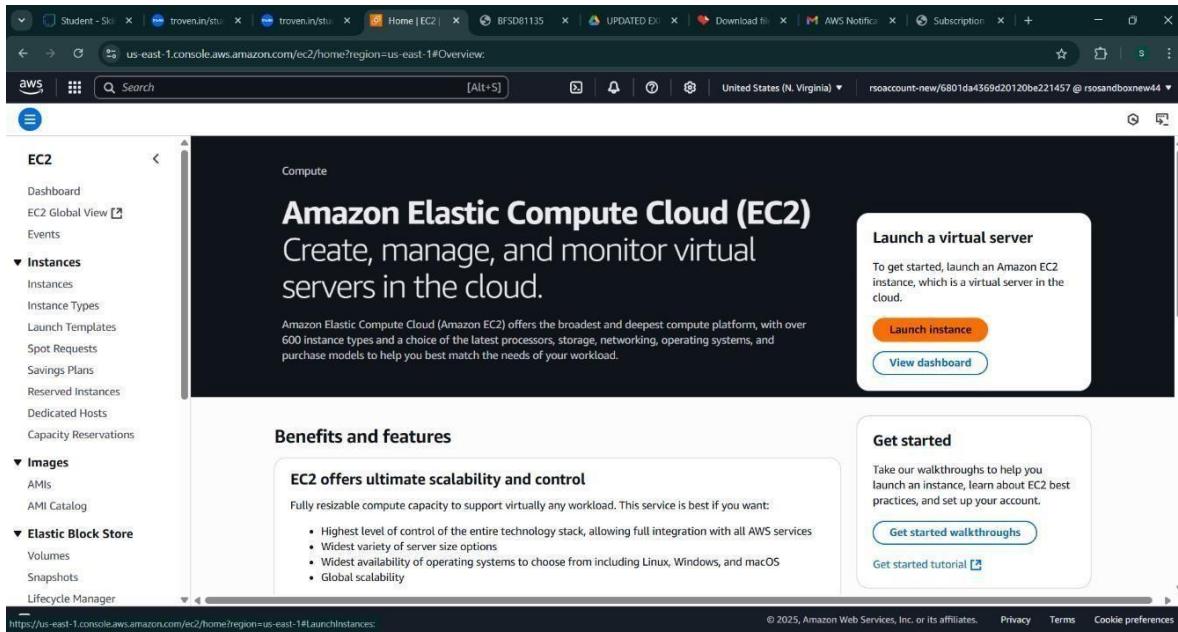
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- **Launch EC2 Instance**

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



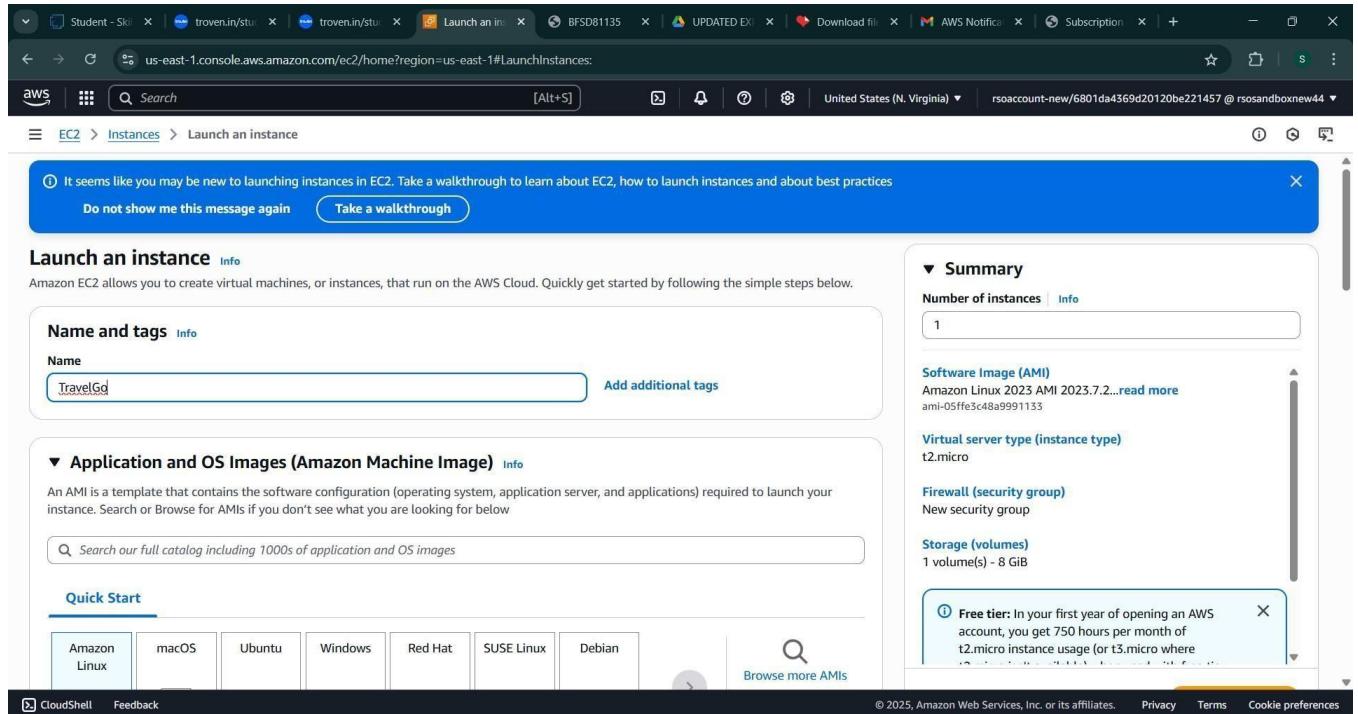
The screenshot shows the AWS search results for 'ec2'. The left sidebar has a 'Mobile' section under 'Amazon Services'. The main search results area has a 'Services' section with 'EC2' highlighted, followed by 'EC2 Image Builder' and 'EC2 Global View'. Below that is a 'Features' section with 'EC2 Instances' highlighted, followed by 'AMIs'. A large preview window on the right shows the EC2 Instances page with a single instance listed.



The screenshot shows the AWS EC2 Home Overview page. The left sidebar has sections for EC2 (Dashboard, Compute, Global View, Events), Instances (Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes, Snapshots, Lifecycle Manager). The main content area features a large 'Compute' section with the heading 'Amazon Elastic Compute Cloud (EC2)'. It says 'Create, manage, and monitor virtual servers in the cloud.' Below this is a paragraph about EC2's offerings and a 'Launch a virtual server' call-to-action button. To the right is a 'Benefits and features' section with a sub-section 'EC2 offers ultimate scalability and control' and a 'Get started' section with walkthroughs and tutorials.

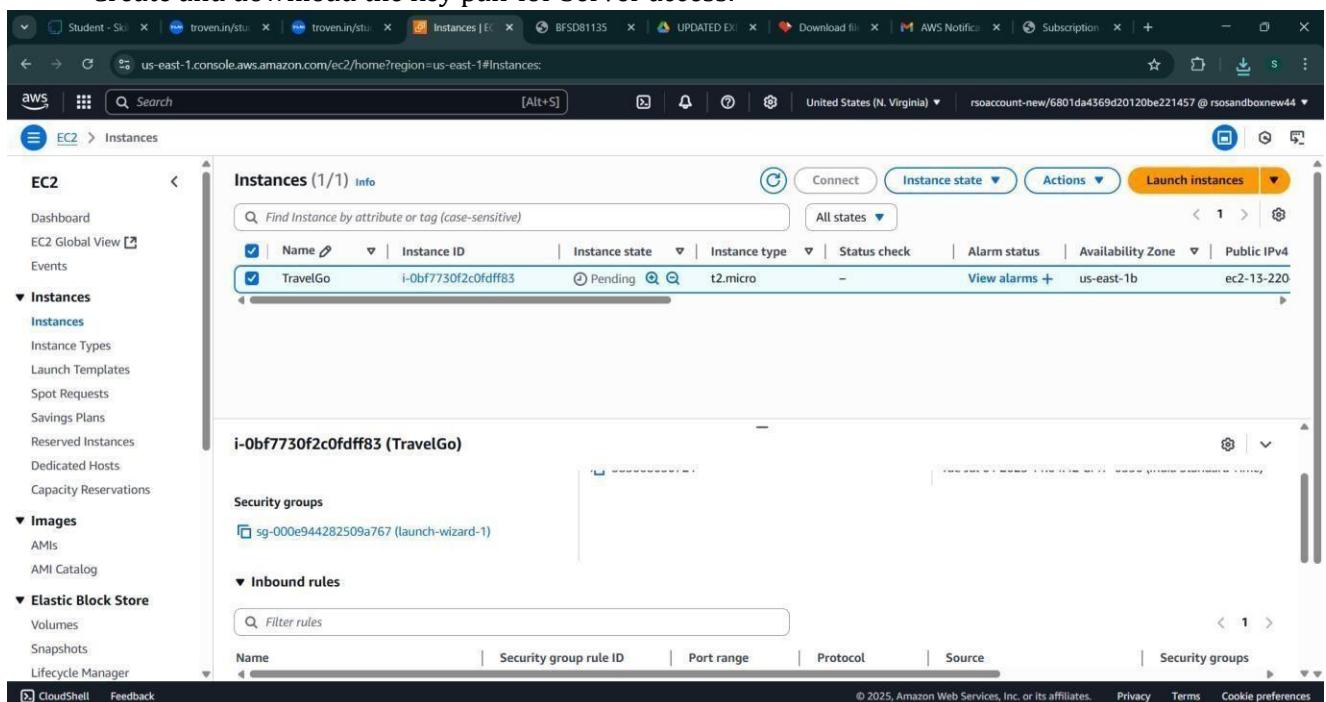
- Click on Launch instance to launch EC2 instance

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



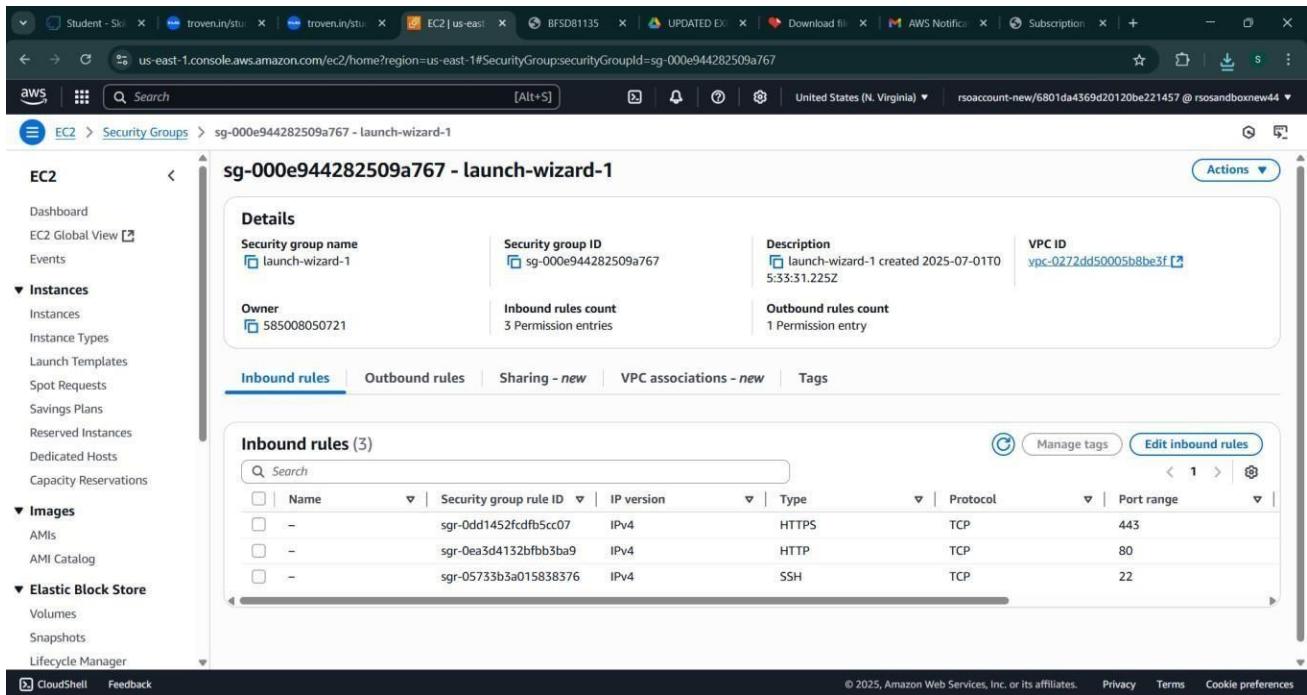
The screenshot shows the 'Launch an instance' wizard in the AWS EC2 console. The 'Name and tags' section has 'TravelGo' entered. The 'Application and OS Images (Amazon Machine Image)' section shows 'Amazon Linux' selected. The 'Virtual server type (instance type)' is set to 't2.micro'. A summary panel on the right shows 1 instance being launched. A tooltip for the free tier is visible.

- Create and download the key pair for Server access.



The screenshot shows the AWS EC2 Instances page. The left sidebar shows 'Instances' selected. The main table lists one instance: 'TravelGo' (i-0bf7730f2c0fdf83), which is currently 'Pending'. The instance details page for 'TravelGo' is shown on the right, including its security group (sg-000e944282509a767) and inbound rules.

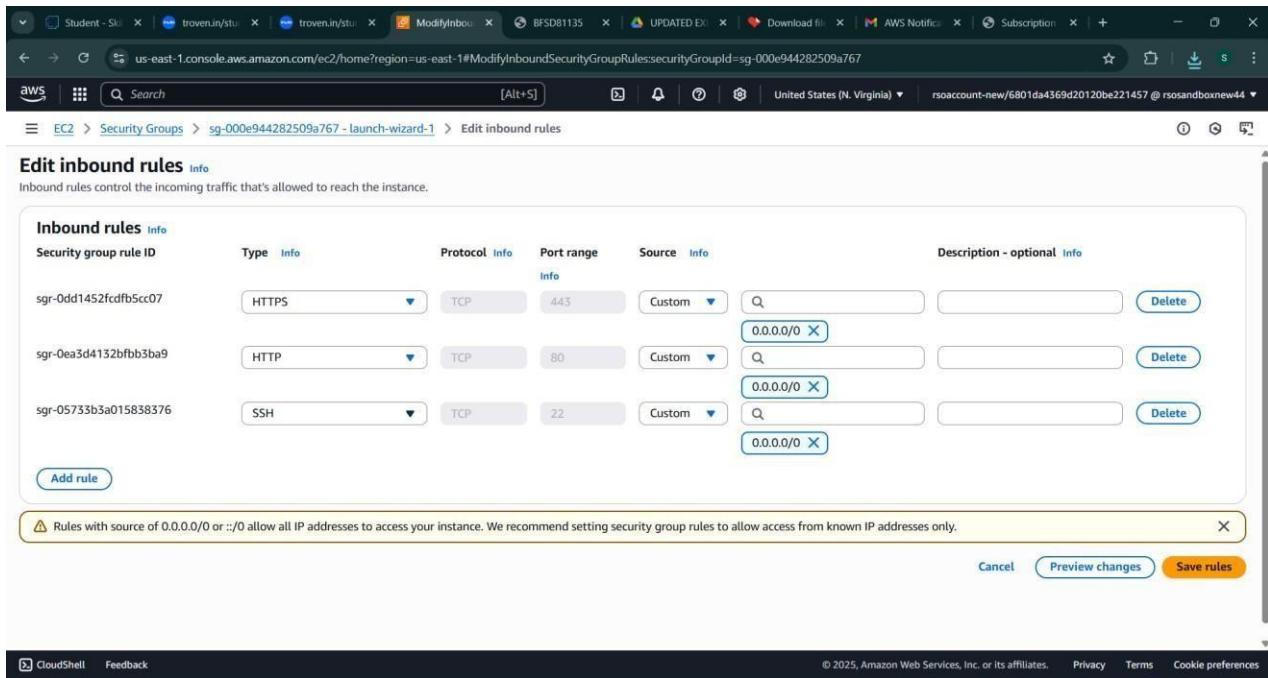
- Activity 6.2: Configure security groups for HTTP, and SSH access.



The screenshot shows the AWS EC2 Security Groups console. The left sidebar is collapsed, and the main area displays the details of the security group **sg-000e944282509a767 - launch-wizard-1**. The **Inbound rules** tab is selected, showing three rules:

| Name | Security group rule ID | IP version | Type | Protocol | Port range |
|------|------------------------|------------|-------|----------|------------|
| - | sgr-0dd1452fcdfb5cc07 | IPv4 | HTTPS | TCP | 443 |
| - | sgr-0ea3d4132bfb3ba9 | IPv4 | HTTP | TCP | 80 |
| - | sgr-05733b3a015838376 | IPv4 | SSH | TCP | 22 |

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

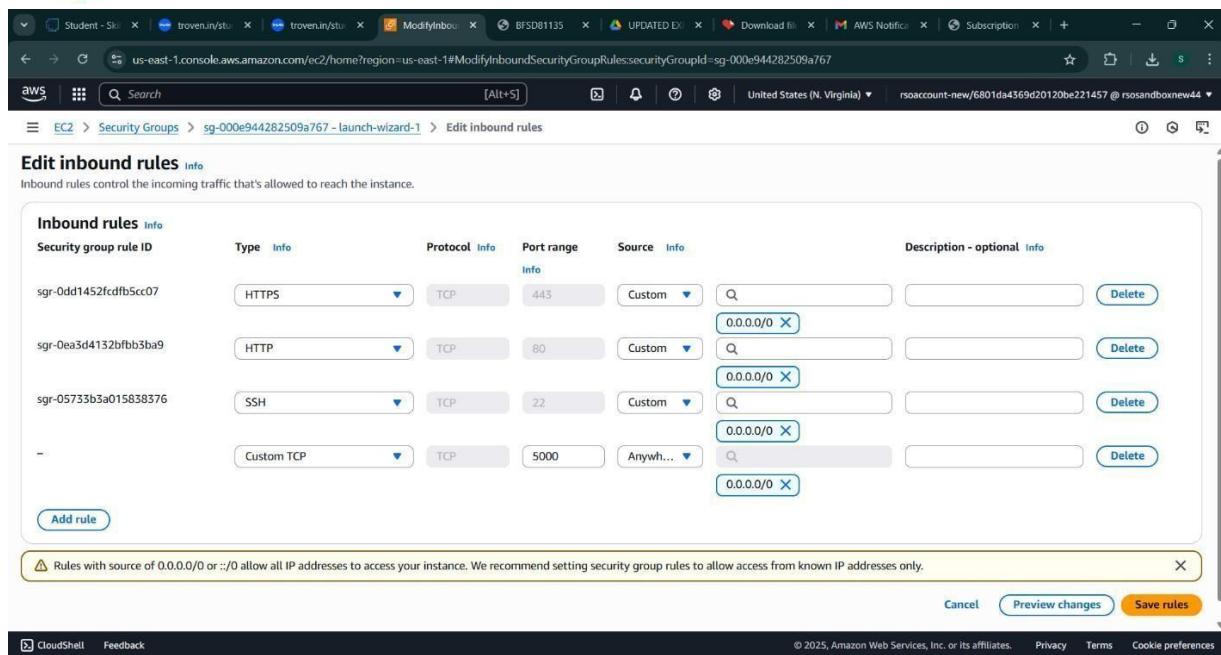


The screenshot shows the **Edit inbound rules** page for the security group **sg-000e944282509a767 - launch-wizard-1**. The table lists the existing inbound rules:

| Security group rule ID | Type | Protocol | Port range | Source | Description - optional |
|------------------------|-------|----------|------------|--------|------------------------|
| sgr-0dd1452fcdfb5cc07 | HTTPS | TCP | 443 | Custom | |
| sgr-0ea3d4132bfb3ba9 | HTTP | TCP | 80 | Custom | |
| sgr-05733b3a015838376 | SSH | TCP | 22 | Custom | |

A warning message at the bottom states: **⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.**

Buttons at the bottom right include **Cancel**, **Preview changes**, and **Save rules**.



The screenshot shows the AWS Management Console with the URL <https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ModifyInboundSecurityGroupRules/securityGroupId=sg-000e944282509a767>. The page is titled "Edit inbound rules". It displays a table of inbound rules:

| Security group rule ID | Type | Protocol | Port range | Source | Description - optional |
|------------------------|------------|----------|------------|----------|------------------------|
| sgr-0dd1452fcdfb5cc07 | HTTPS | TCP | 443 | Custom | 0.0.0.0/0 |
| sgr-0ea3d4132bfbb3ba9 | HTTP | TCP | 80 | Custom | 0.0.0.0/0 |
| sgr-05733b3a015838376 | SSH | TCP | 22 | Custom | 0.0.0.0/0 |
| - | Custom TCP | TCP | 5000 | Anywhere | 0.0.0.0/0 |

A button "Add rule" is visible at the bottom left. A warning message at the bottom states: "⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only." Buttons "Cancel", "Preview changes", and "Save rules" are at the bottom right.

- 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.

Screenshot of the AWS EC2 Instances page showing a single instance named "TravelGo" (i-0bf7730f2c0fdff83) in the "Running" state. The instance type is t2.micro. The Actions menu is open, showing options like "Change security groups", "Get Windows password", and "Modify IAM role". Below the instance list, the "Inbound rules" section is visible.

Modify IAM role Info

Attach an IAM role to your instance.

Instance ID
 i-0bf7730f2c0fdff83 (TravelGo)

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.

Choose IAM role

Create new IAM role

No IAM Role

Choose this option to detach an IAM role

Studentuser

arn:aws:iam::585008050721:instance-profile/Studentuser

Studentuser

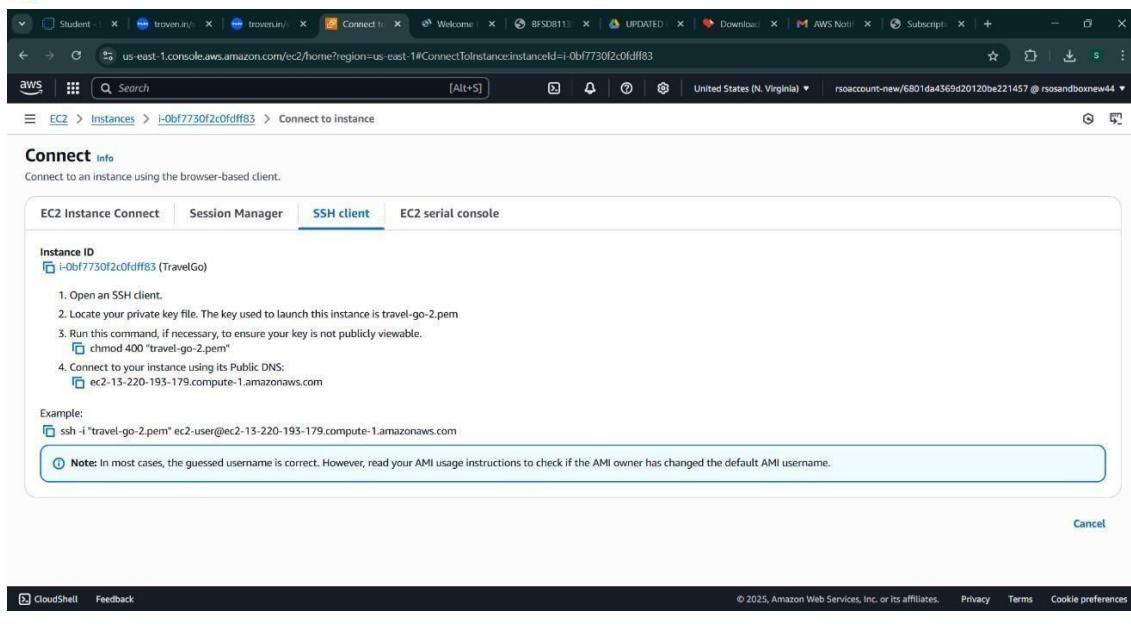
Cancel

Update IAM role

CloudShell Feedback

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

- Now connect the EC2 with the files



The screenshot shows the AWS EC2 Connect interface for instance i-0bf7730f2c0fdf83 (TravelGo). It provides instructions for connecting via SSH:

- Open an SSH client.
- Locate your private key file. The key used to launch this instance is travel-go-2.pem.
- Run this command, if necessary, to ensure your key is not publicly viewable:


```
chmod 400 "travel-go-2.pem"
```
- Connect to your instance using its Public DNS:

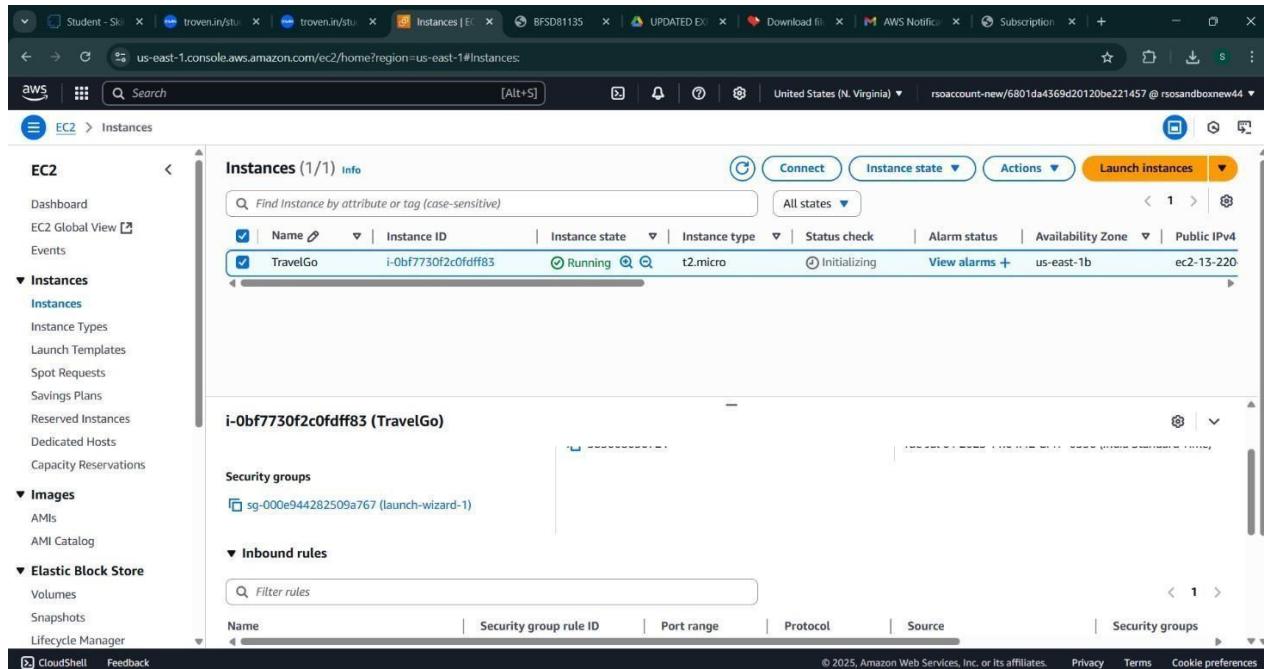

```
ec2-13-220-193-179.compute-1.amazonaws.com
```

Example command:

```
ssh -i "travel-go-2.pem" ec2-user@ec2-13-220-193-179.compute-1.amazonaws.com
```

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

Cancel



The screenshot shows the AWS EC2 Instances page. The left sidebar shows the navigation menu under the EC2 section. The main area displays the instance details for i-0bf7730f2c0fdf83 (TravelGo), which is currently running. The instance is associated with a t2.micro instance type, an sg-000e944282509a767 security group, and is located in us-east-1b Availability Zone, with a public IPv4 address ec2-13-220-193-179.

| Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Availability Zone | Public IPv4 |
|----------|--------------------|----------------|---------------|--------------|---------------|-------------------|--------------------|
| TravelGo | i-0bf7730f2c0fdf83 | Running | t2.micro | Initializing | View alarms + | us-east-1b | ec2-13-220-193-179 |

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

```

ec2-user@ip-172-31-26-113:~/ > + <
=3.6.0->flask) (3.23.0)
[ec2-user@ip-172-31-26-113 travelgoAWS]$ pip install boto3
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: boto3 in /home/ec2-user/.local/lib/python3.9/site-packages (1.39.1)
Requirement already satisfied: botocore<1.40.0,>=1.39.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from boto3) (1.39.1)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Requirement already satisfied: s3transfer<0.14.0,>=0.13.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from bot
o3) (0.13.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1
.39.1->boto3) (2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.1-
>boto3) (1.25.10)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<
1.40.0,>=1.39.1->boto3) (1.15.0)
[ec2-user@ip-172-31-26-113 travelgoAWS]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.26.113:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 568-143-514
157.50.146.49 - - [02/Jul/2025 08:17:08] "GET / HTTP/1.1" 200 -
157.50.146.49 - - [02/Jul/2025 08:17:08] "GET /static/images/travel-bg.jpg HTTP/1.1" 404 -
157.50.146.49 - - [02/Jul/2025 08:17:08] "GET /favicon.ico HTTP/1.1" 404 -
|
```

Milestone 8: Testing and Deployment

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

Welcome Page: Register Page:

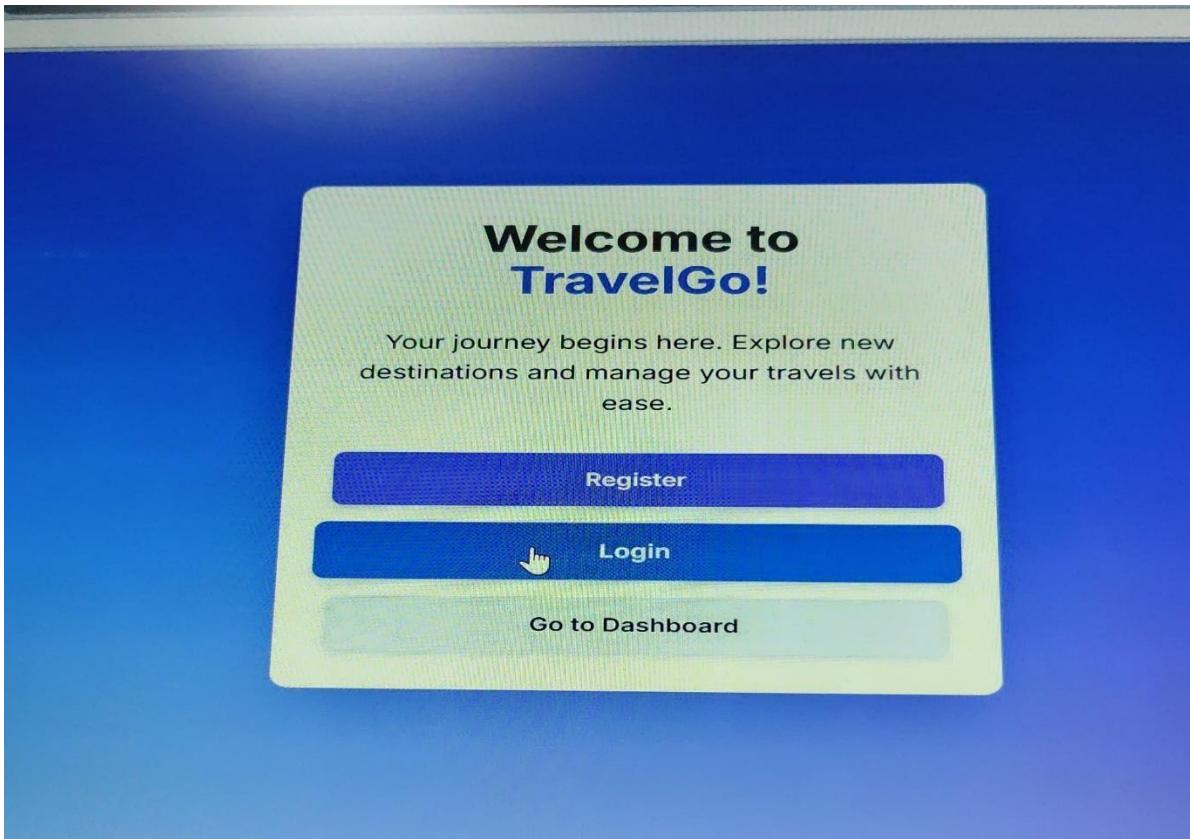
Login Page:

Bus Booking Page:

Train Booking page:

Flight Booking Page:

Hotel Booking Page:



A screenshot of a web browser window showing a login form for "TravelGo". The form is contained within a light blue rectangular box. At the top center, it says "Login to TravelGo". Below that is a text input field labeled "Email Address" containing "tar@gmail.com". Below that is a password input field labeled "Password" showing five dots. At the bottom of the form is a large blue "Login" button. Below the form, there is a link "Don't have an account? Register here!" and a link "Back to Home".

Login to TravelGo

Email Address

tar@gmail.com

Password

.....

Login

Don't have an account? [Register here!](#)

[Back to Home](#)

Welcome,tar@gmail.com



Bus



Train



Flight



Hotel

YOUR BOOKINGS

Bus

- Orange Travels Hyderabad→Vijayawada 08:00 AM (1 persons) (2025-06-29)

[Cancel](#)

Train

- Karnataka Express 12627 Hyderabad→Bengaluru 20:00 - 07:00 (1 persons) (2025-06-29)

[Cancel](#)

Flight

- Indigo 6E 987 Delhi→Hyderabad 11:00 - 13:00 (1 persons) (2025-06-29)

[Cancel](#)

Hotel

- Taj Falaknuma Palace in Hyderabad (1 rooms, 1 guests, 2025-06-29 to 2025-06-30) (2025-06-29)

[Cancel](#)



Booking Summary

Type: Bus (AC Sleeper)

Route: Hyderabad to Vijayawada

Travel Date: 2025-07-09

Departure/Time: 08:00 AM

Price per Person: ₹800.00

Number of Passengers: 1

Please select 1 seat(s).

Available Selected Booked

| Driver | | | | | | Door |
|--------|---|---|---|---|---------------------------------------|------|
| A | 1 | 2 | 3 | 4 | <input checked="" type="checkbox"/> 5 | 6 |
| B | 1 | 2 | 3 | 4 | <input type="checkbox"/> 5 | 6 |
| C | 1 | 2 | 3 | 4 | <input type="checkbox"/> 5 | 6 |
| D | 1 | 2 | 3 | 4 | <input type="checkbox"/> 5 | 6 |
| E | 1 | 2 | 3 | 4 | <input type="checkbox"/> 5 | 6 |

[Proceed to Confirm Booking](#)

Confirm Your Bus Booking

Bus Name: Orange Travels
Route: Hyderabad – Vijayawada
Departure Time: 08:00 AM
Bus Type: AC Sleeper
Travel Date: 2025-07-09
Number of Persons: 1
Price per Person: ₹800.0

Total Price: ₹800.0

[Confirm Booking](#)[Go Back](#) Search & Book Trains

Hyderabad

Vijayawada

09-07-2025

1

[Search](#) Express Superfast AC Sleeper

Sort by: None

Charminar Express (12759)

Express Sleeper • 18:30 - 00:30 • ₹300.00

[Book](#)

Find & Book Hotel Rooms

Hyderabad 09-07-2025 10-07-2025 1 1

Search

5-Star 4-Star 3-Star WiFi Pool Parking

Sort by: **None**

Taj Falaknuma Palace
Hyderabad • 5-Star • ₹25000.00/night
Amenities: WiFi, Pool, Parking, Restaurant

Total for 1 nights, 1 rooms: ₹25000.00

Novotel Hyderabad Convention Centre
Hyderabad • 4-Star • ₹7000.00/night
Amenities: WiFi, Pool, Parking, Gym

Total for 1 nights, 1 rooms: ₹7000.00

Book

A 1 2 3 4 5 6

TravelGo

Home Dashboard

Search & Book Flights

Hyderabad (HYD) Mumbai (BOM) 09-07-2025 1 Search

Indigo Vistara Air India Direct 1 Stop

Sort by: None

Indigo (6E 234)

Direct Economy • 08:00 - 09:30 • Direct • ₹3500.00

Book



Conclusion:

The **TravelGo** Website has been successfully developed and deployed using a scalable and cloud-native architecture. Leveraging AWS services such as EC2 for hosting, DynamoDB for real-time data management, and SNS for instant booking and cancellation notifications, the platform provides a seamless travel booking experience for users. TravelGo enables registered users to search and book buses, trains, flights, and hotels in a centralized, intuitive interface, eliminating the complexities of navigating multiple travel services.

The cloud infrastructure ensures high availability and smooth performance even during peak usage, while the Flask backend ensures efficient handling of user authentication, dynamic booking flows, and data transactions. Real-time notification integration via AWS SNS allows users to receive booking confirmations and cancellations immediately via email, improving communication and user engagement.

In summary, the **TravelGo** Website offers a modern, reliable, and user-friendly solution for managing travel and accommodation needs. It highlights the potential of cloud-based platforms in building unified travel systems, simplifying operations, and enhancing the overall user experience.



THANK YOU