MedTrack: AWS Cloud-Enabled Healthcare Management System

Project Description:

In today's fast-evolving healthcare landscape, efficient communication and coordination between doctors and patients are crucial. MedTrack is a cloud-based healthcare management system that streamlines patient doctor interactions by providing a centralized platform for booking appointments, managing medical histories, and enabling diagnosis submissions. To address these challenges, the project utilizes Flask for backend development, AWS EC2 for hosting, and DynamoDB for managing data. MedTrack allows patients to register, log in, book appointments, and submit diagnosis reports online. The system ensures real-time notifications, enhancing communication between doctors and patients regarding appointments and medical submissions. Additionally, AWS Identity and Access Management (IAM) is employed to ensure secure access control to AWS resources, allowing only authorized users to access sensitive data. This cloud-based solution improves accessibility and efficiency in healthcare services for all users.

Scenario 1: Efficient Appointment Booking System for Patients

In the MedTrack system, AWS EC2 provides a reliable infrastructure to manage multiple patients accessing the platform simultaneously. For example, a patient can log in, navigate to the appointment booking page, and easily submit a request for an appointment. Flask handles backend operations, efficiently retrieving and processing user data in real-time. The cloud-based architecture allows the platform to handle a high volume of appointment requests during peak periods, ensuring smooth operation without delays.

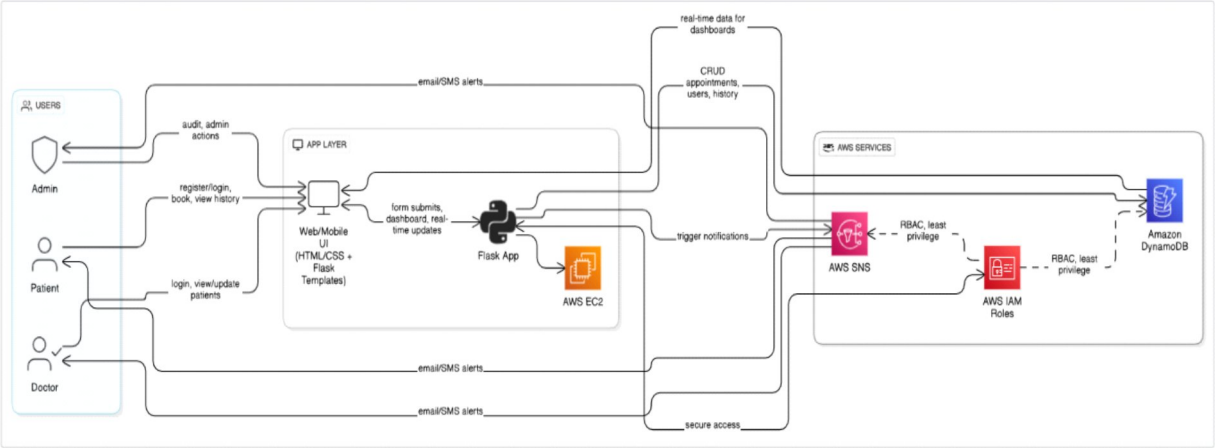
Scenario 2: Secure User Management with IAM

MedTrack utilizes AWS IAM to manage user permissions and ensure secure access to the system. For instance, when a new patient registers, an IAM user is created with specific roles and permissions to access only the features relevant to them. Doctors have their own IAM configurations, allowing them access to patient records and appointment details while maintaining strict security protocols. This setup ensures that sensitive data is accessible only to authorized users

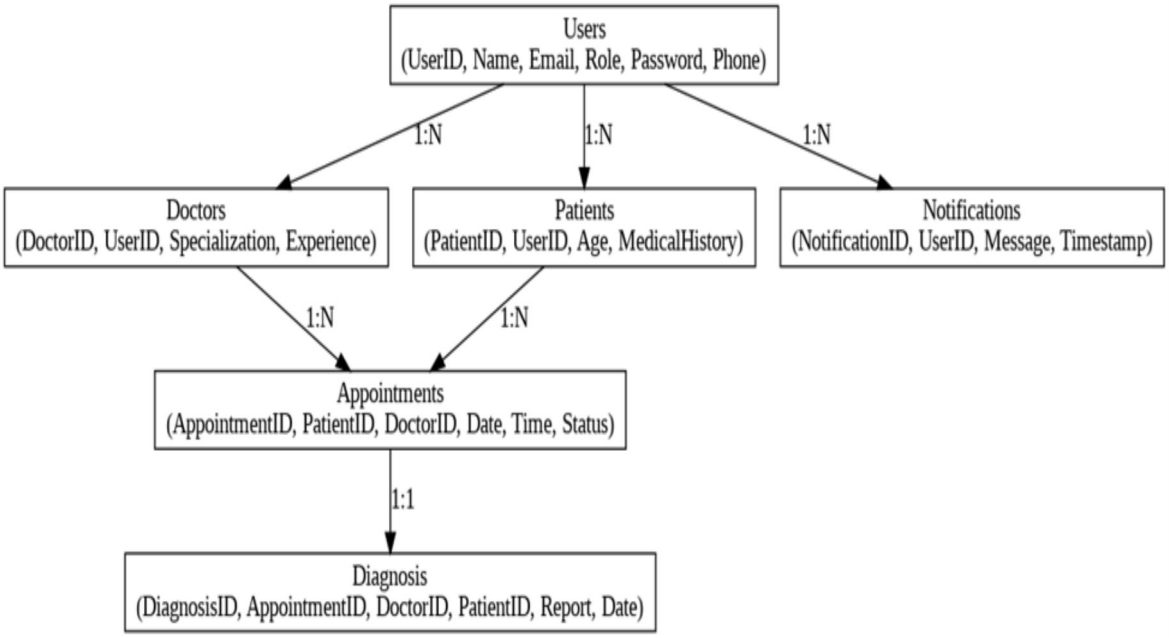
Scenario 3: Easy Access to Medical History and Resources

The MedTrack system provides doctors and patients with easy access to medical histories and relevant resources. For example, a doctor logs in to view a patient's medical history and upcoming appointments. They can quickly access, and update records as needed. Flask manages real-time data fetching from DynamoDB, while EC2 hosting ensures the platform performs seamlessly even when multiple users access it simultaneously, offering a smooth and uninterrupted user experience.

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Pre-requisites:

- .**AWS Account Setup**: AWS HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh"Account HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh" HYPERLINK "https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh"Setup
- **Understanding IAM**: IAM HYPERLINK "https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR" HYPERLINK "https://youtu.be/gsgdAyGhV0o?si=3qg-

bULgkD4LXNvR" HYPERLINK "https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR" HYPERLINK "https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR" HYPERLINK "https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR" HYPERLINK "https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR"Overview

- **Amazon EC2 Basics**: EC2 HYPERLINK "https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz" HYPERLINK "https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz" HYPERLINK "https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz" HYPERLINK "https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz" HYPERLINK "https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz" HYPERLINK "https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz"Tutorial
- **DynamoDB Basics**: DynamoDB HYPERLINK "https://docs.aws.amazon.com/dynamodb" HYPERLINK "https://docs.aws.amazon.com/dynamodb" HYPERLINK "https://docs.aws.amazon.com/dynamodb" HYPERLINK "https://docs.aws.amazon.com/dynamodb" HYPERLINK "https://docs.aws.amazon.com/dynamodb" HYPERLINK "https://docs.aws.amazon.com/dynamodb"Introduction
- **SNS Overview**: SNS HYPERLINK "https://docs.aws.amazon.com/sns" HYPERLINK "https://docs.aws.amazon.com/sns" HYPERLINK "https://docs.aws.amazon.com/sns" HYPERLINK "https://docs.aws.amazon.com/sns" HYPERLINK "https://docs.aws.amazon.com/sns"Documentation
- **Git Version Control**: Git HYPERLINK "https://git-scm.com/doc" HYPERLINK "https://git-scm.com/doc" HYPERLINK "https://git-scm.com/doc" HYPERLINK "https://git-scm.com/doc" HYPERLINK "https://git-scm.com/doc" HYPERLINK "https://git-scm.com/doc"Documentation

Project WorkFlow:

- **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

- **DynamoDB Database Creation and Setup**

Activity 2.1: Create a DynamoDB Table.

Activity 2.2: Configure Attributes for User Data and Book Requests.

- **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.

- **Backend Development and Application Setup**

Activity 4.1:Develop the Backend Using Flask.

Activity 4.2: Integrate AWS Services Using boto3.

- **IAM Role Setup**

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

- **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.

- **Deployme**

nt on EC2

Activity

7.1:Upload

Flask Files

Activity 7.2:

Run the Flask

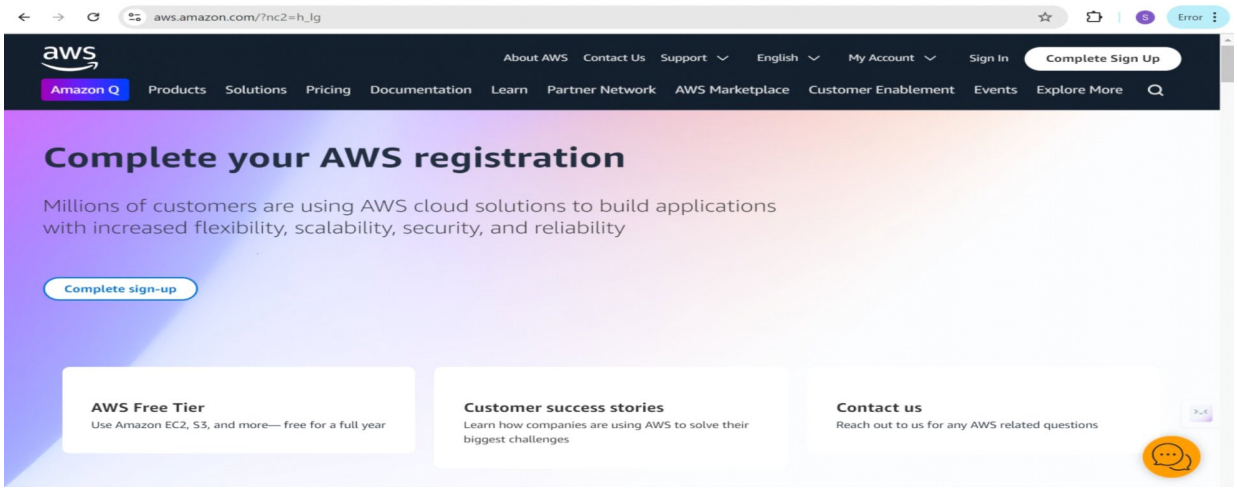
App

- **Testing and Deployment**

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

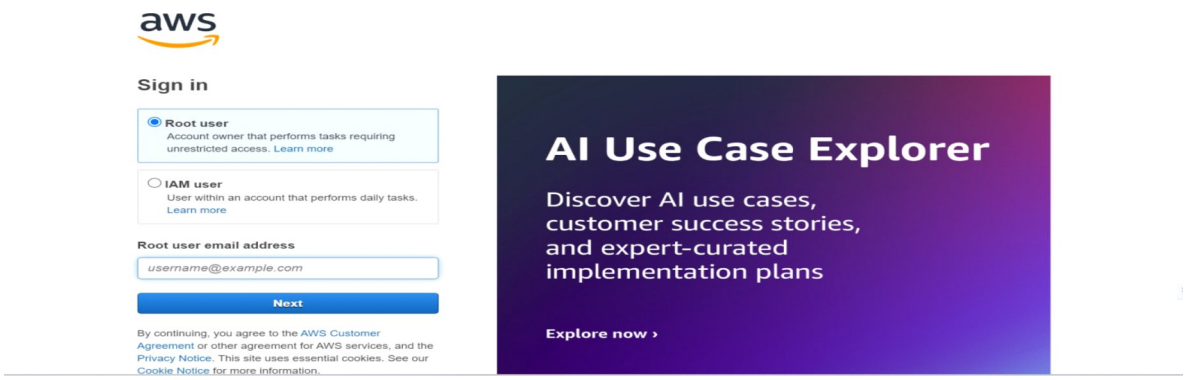
Milestone 1: AWS Account Setup and Login

- **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**

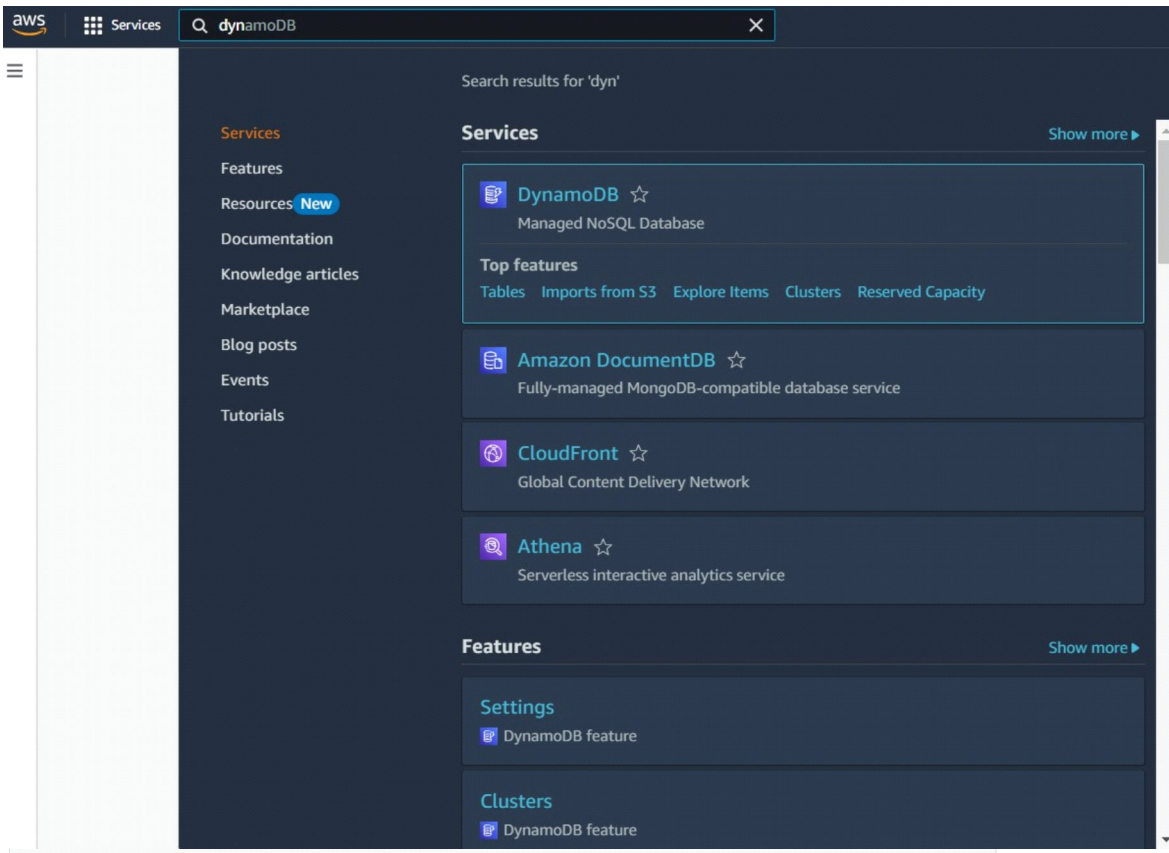
    - After setting up your account, log in to the AWS HYPERLINK "https://aws.amazon.com/console/" HYPERLINK "https://aws.amazon.com/console/" HYPERLINK "https://aws.amazon.com/console/" HYPERLINK "https://aws.amazon.com/console/" HYPERLINK "https://aws.amazon.com/console/" HYPERLINK "https://aws.amazon.com/console/"Management HYPERLINK "https://aws.amazon.com/console/" HYPERLINK "https://aws.amazon.com/console/" HYPERLINK "https://aws.amazon.com/console/" HYPERLINK "https://aws.amazon.com/console/" HYPERLINK "https://aws.amazon.com/console/"Console.

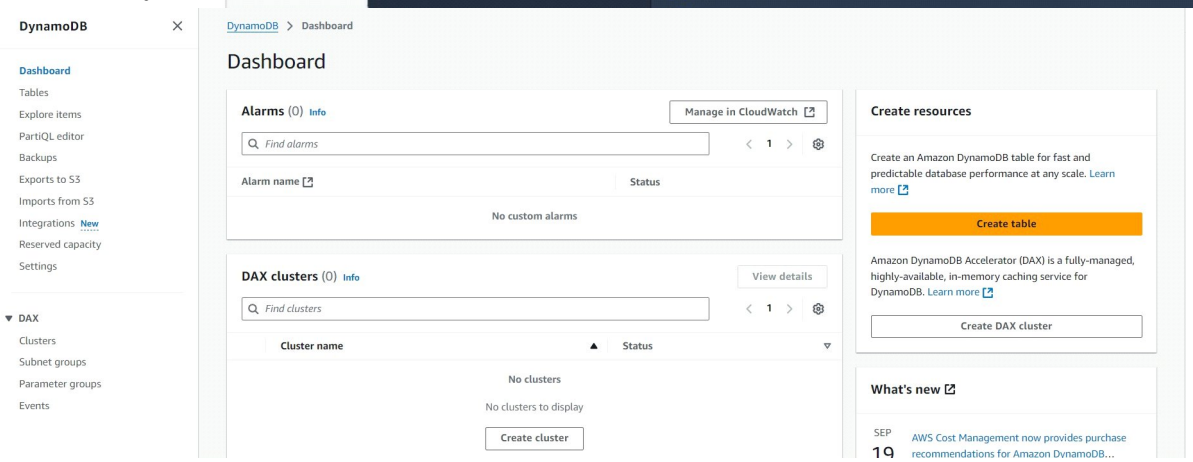


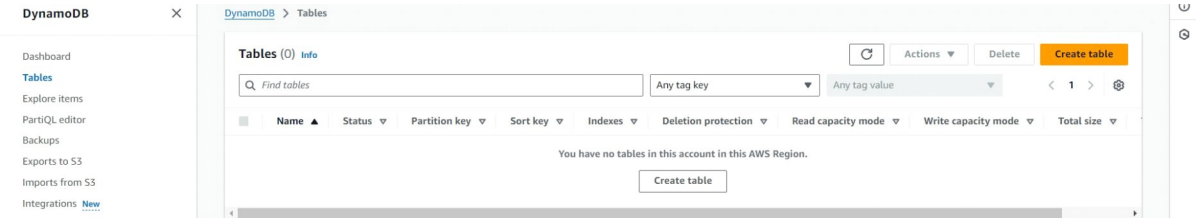
Milestone 2: DynamoDB Database Creation and Setup

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

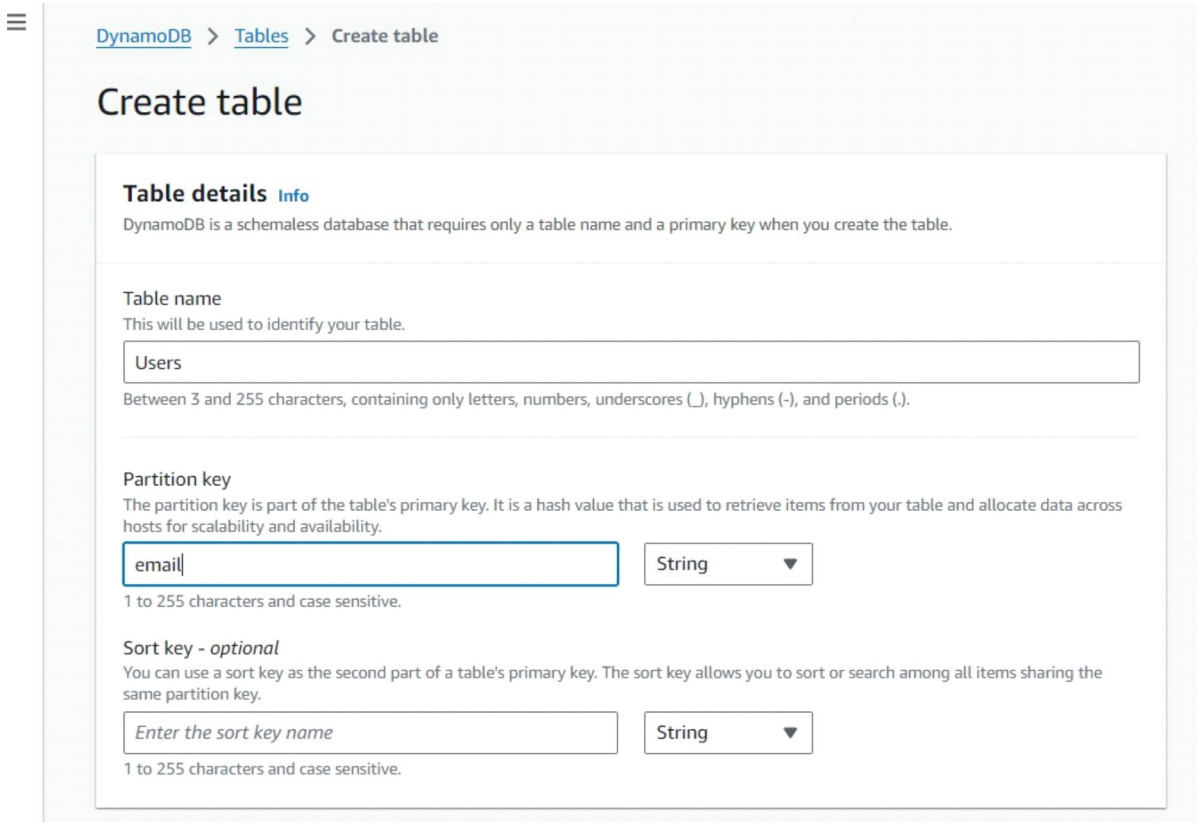
    - In the AWS Console, navigate to DynamoDB and click on create tables.



- 



- **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.


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.
Users
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).
Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
email
String
1 to 255 characters and case sensitive.
Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
Enter the sort key name
String
1 to 255 characters and case sensitive.

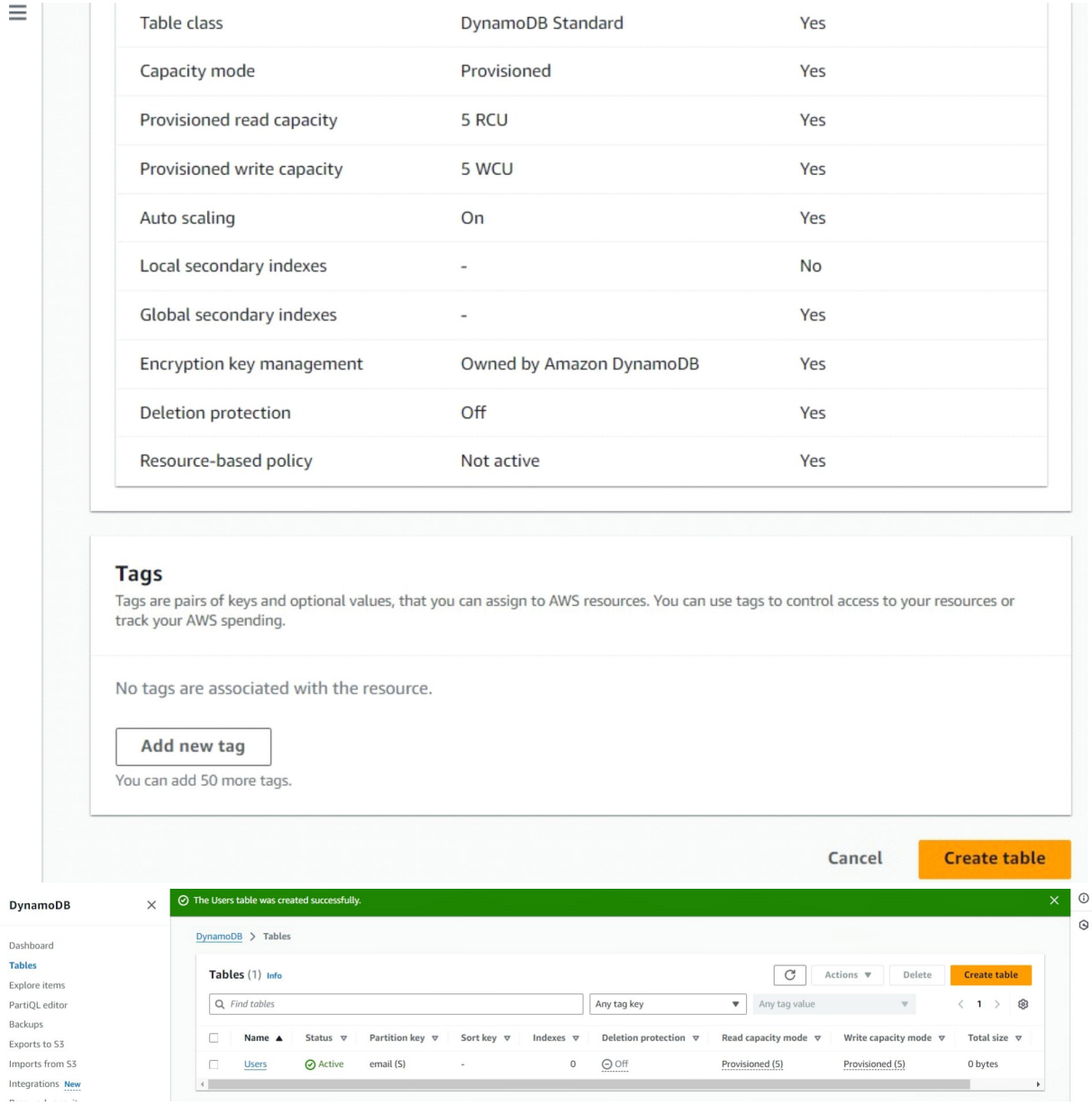


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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

DynamoDB > Tables > Create table

Create table

Table details Info

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

Table name

This will be used to identify your table.

Requests

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

Partition key

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

email | String

1 to 255 characters and case sensitive.

Sort key - *optional*

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

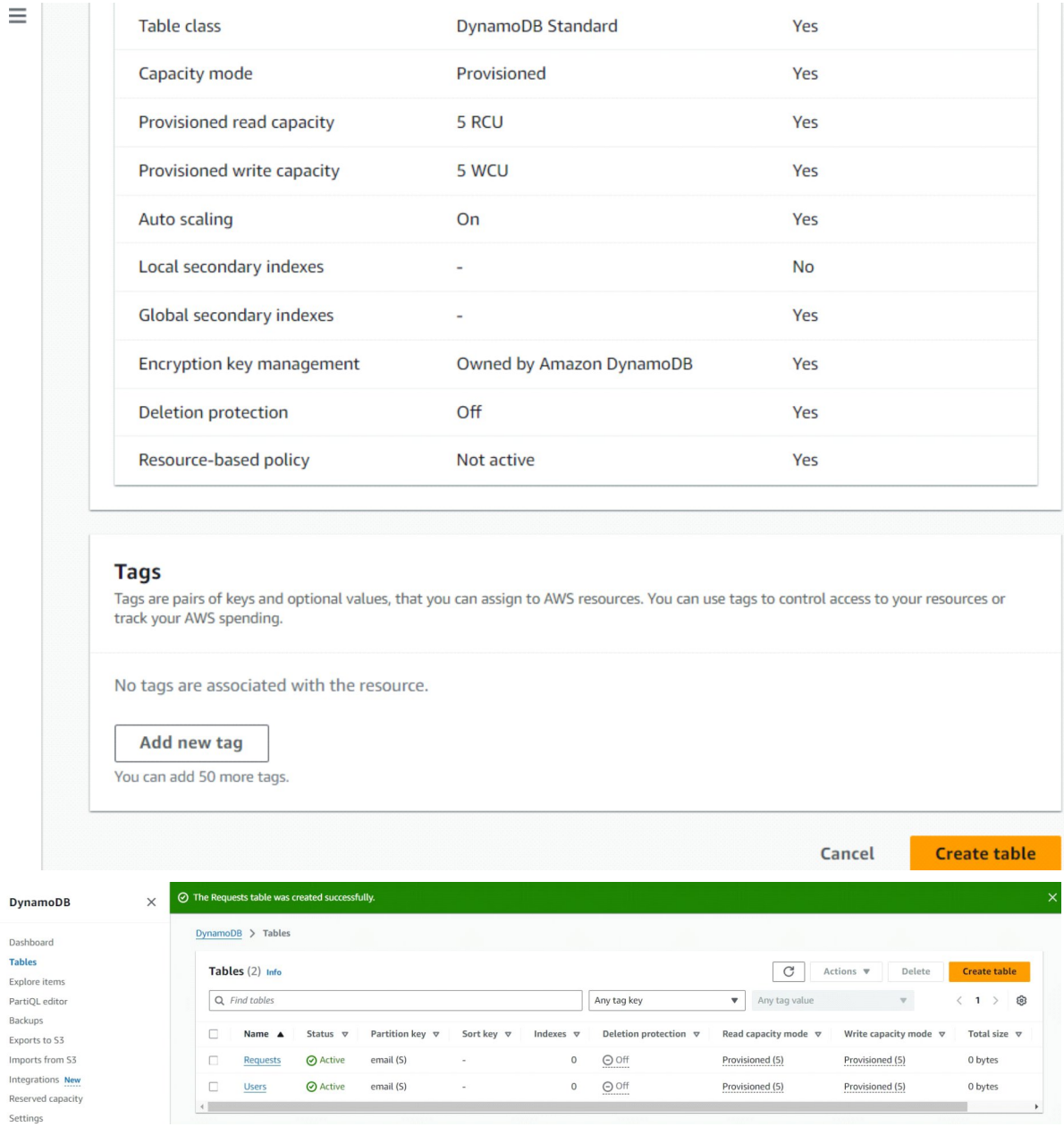
Enter the sort key name | String

1 to 255 characters and case sensitive.

Table settings

(•) Default settings

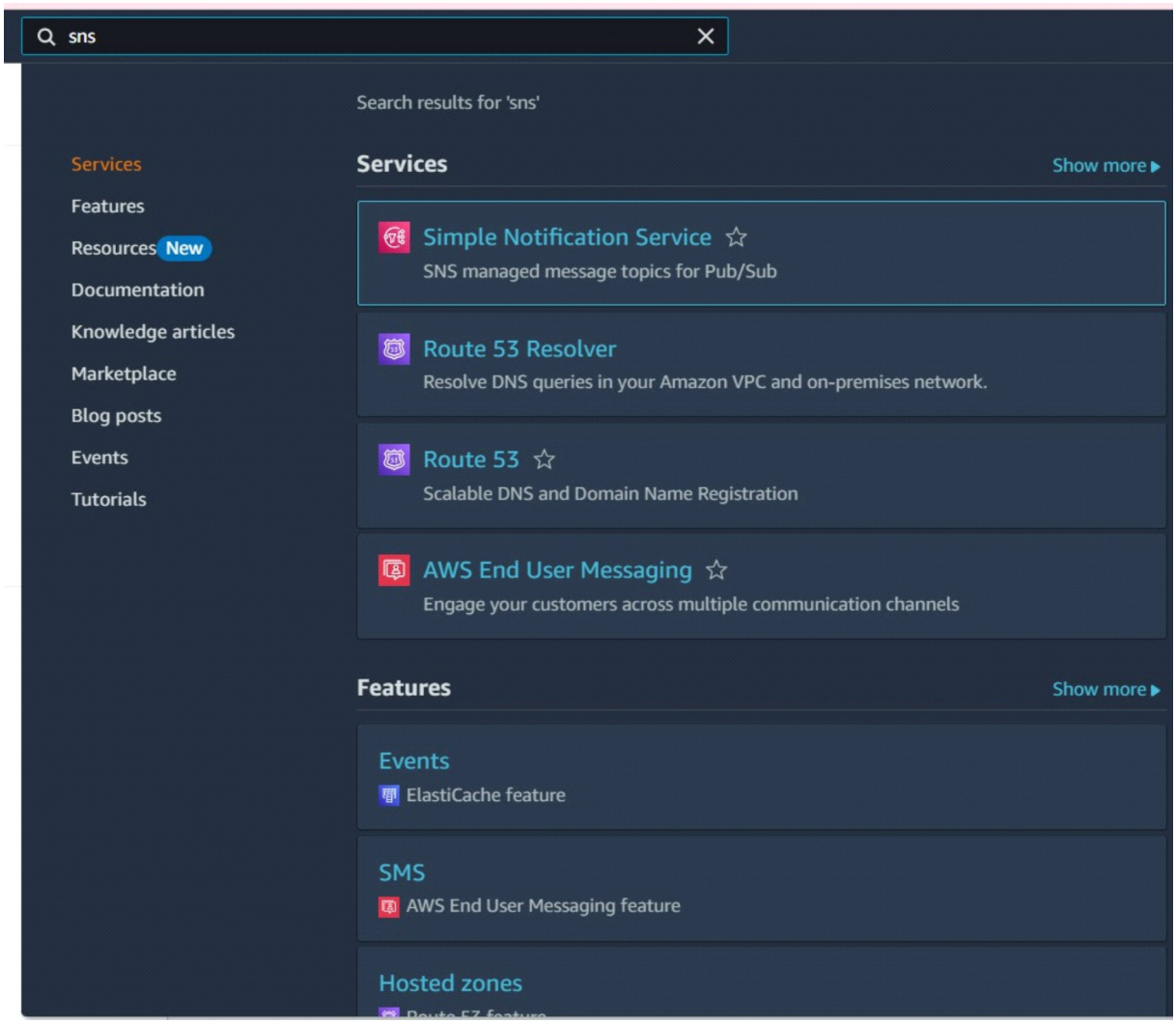
( ) Customize settings

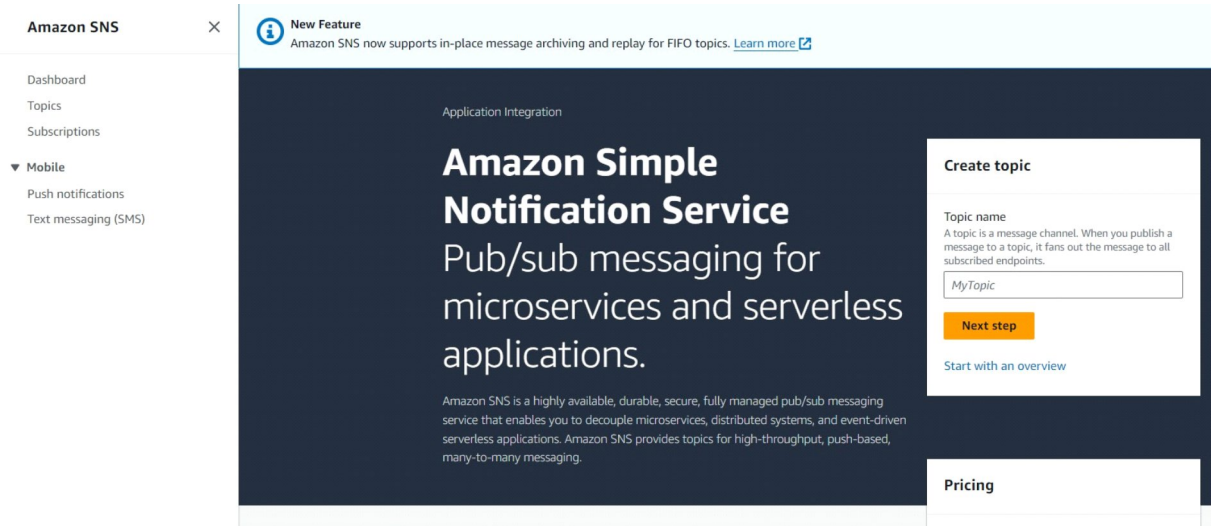


Milestone 3: SNS Notification Setup

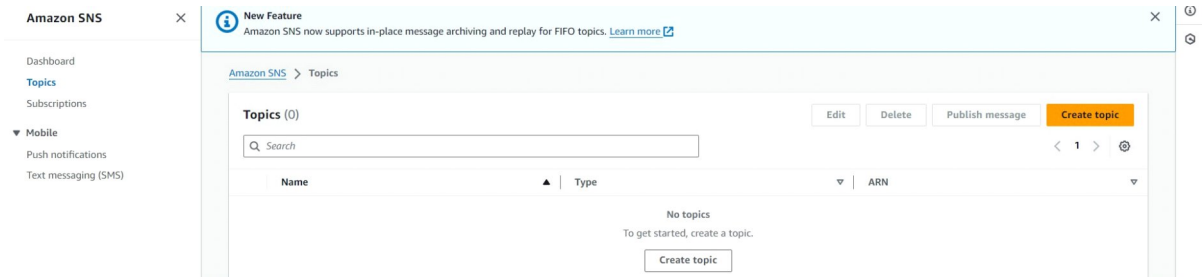
- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**

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

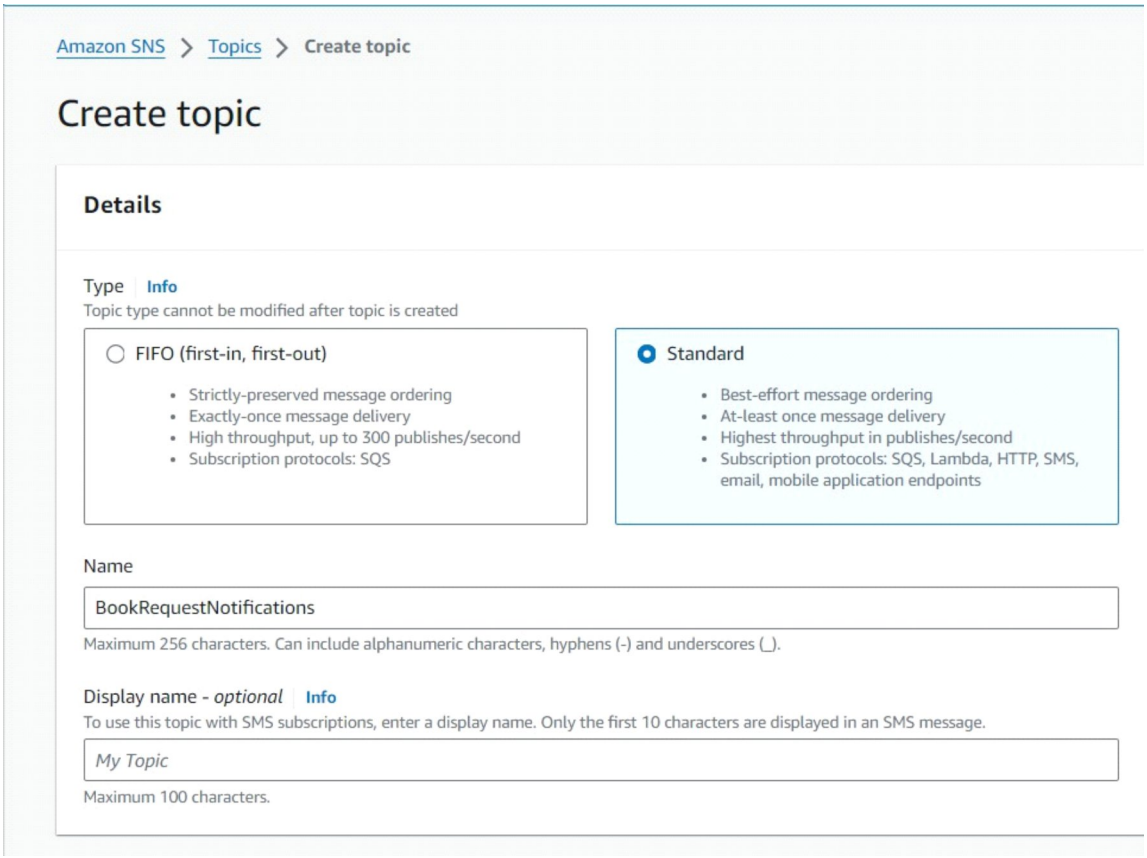


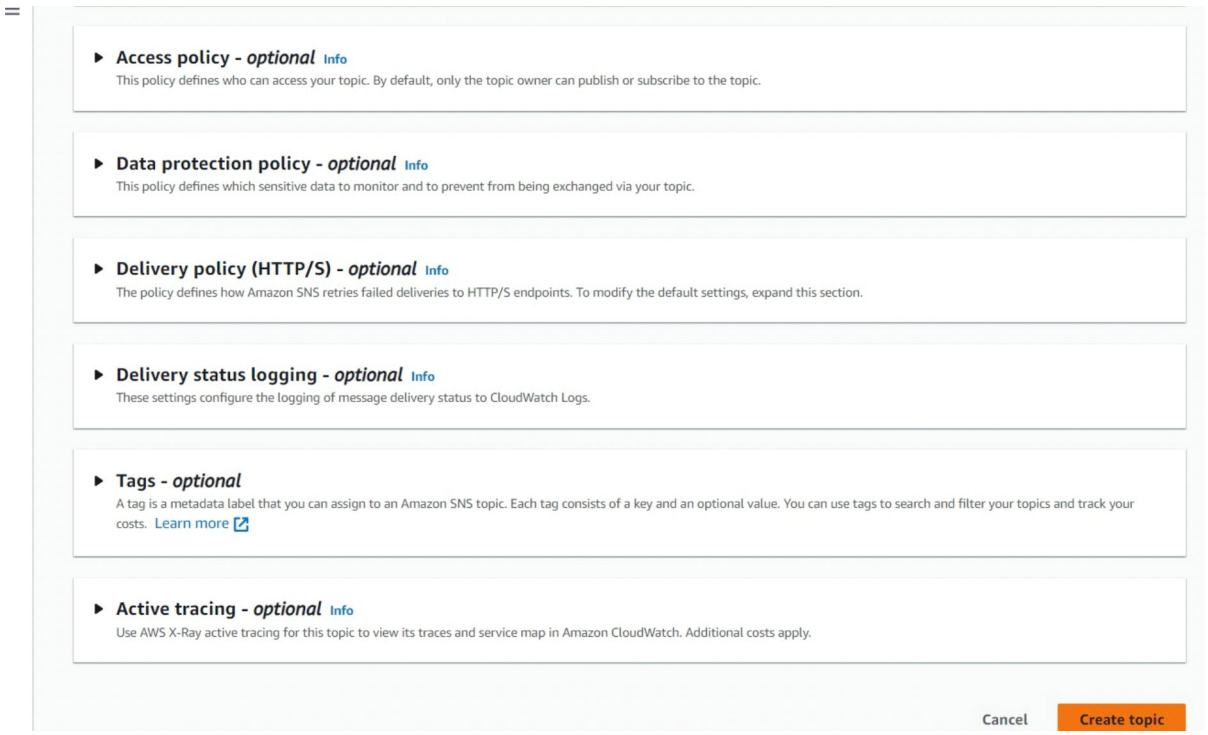


- Click on **Create Topic** and choose a name for the topic.

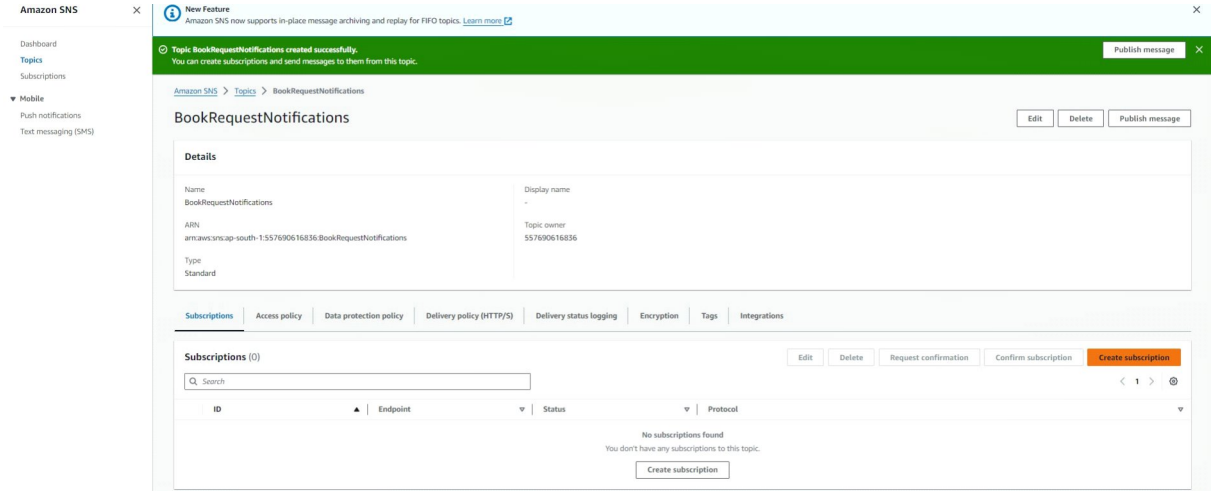


- Choose Standard type for general notification use cases and Click on Create Topic.



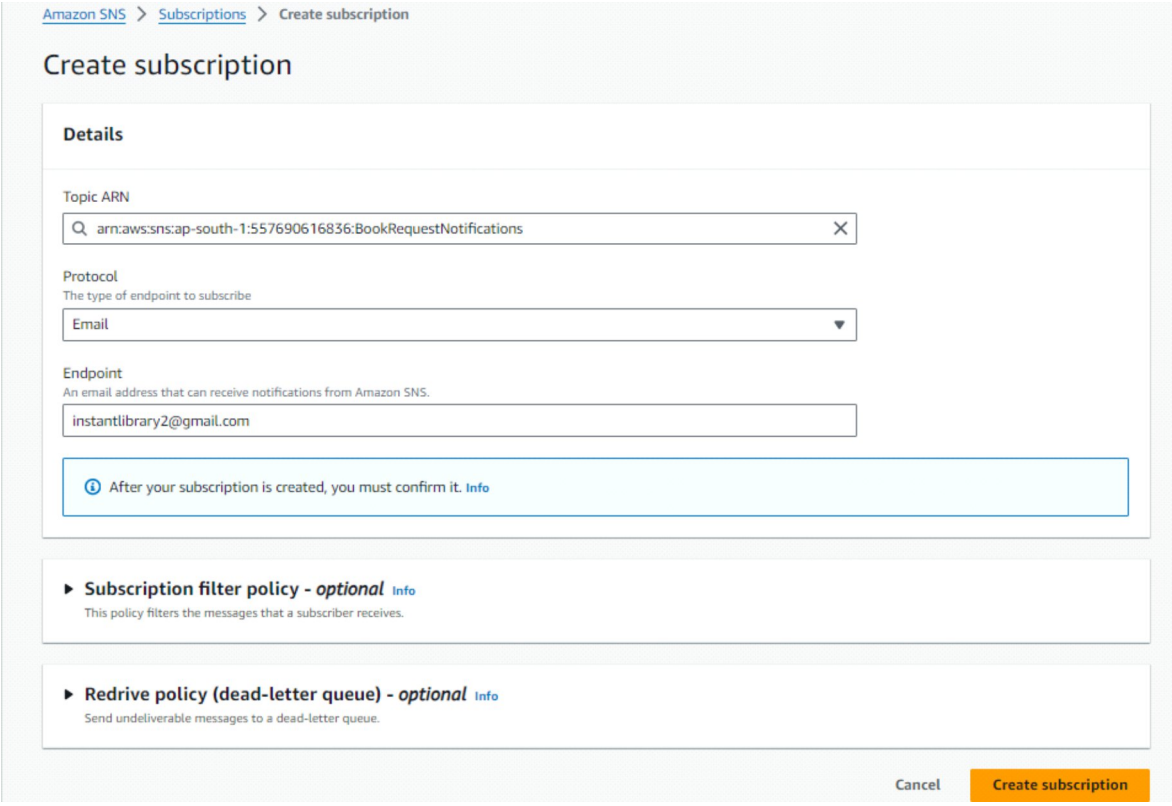


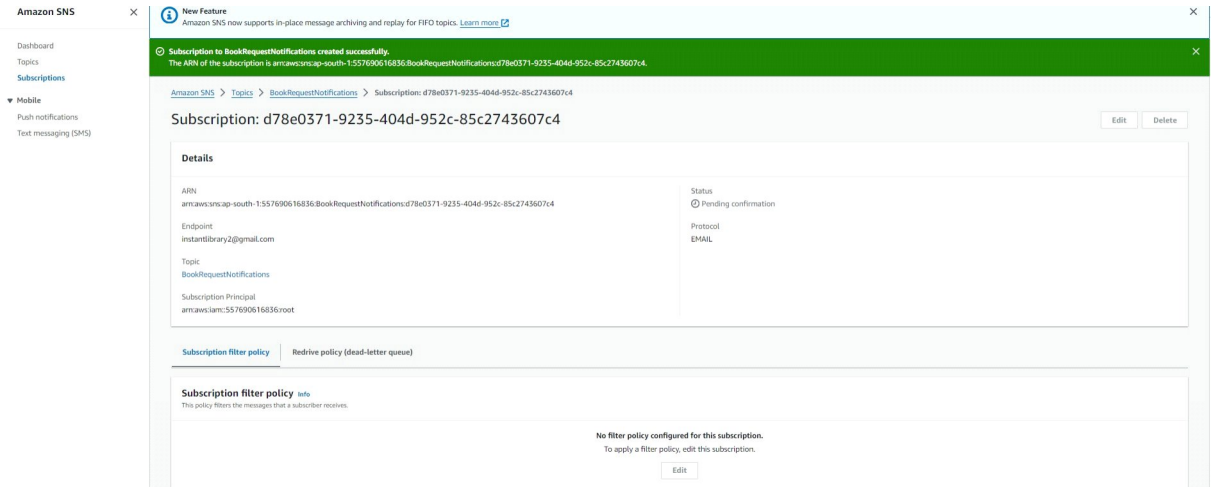
- Configure the SNS topic and note down the **Topic ARN**.



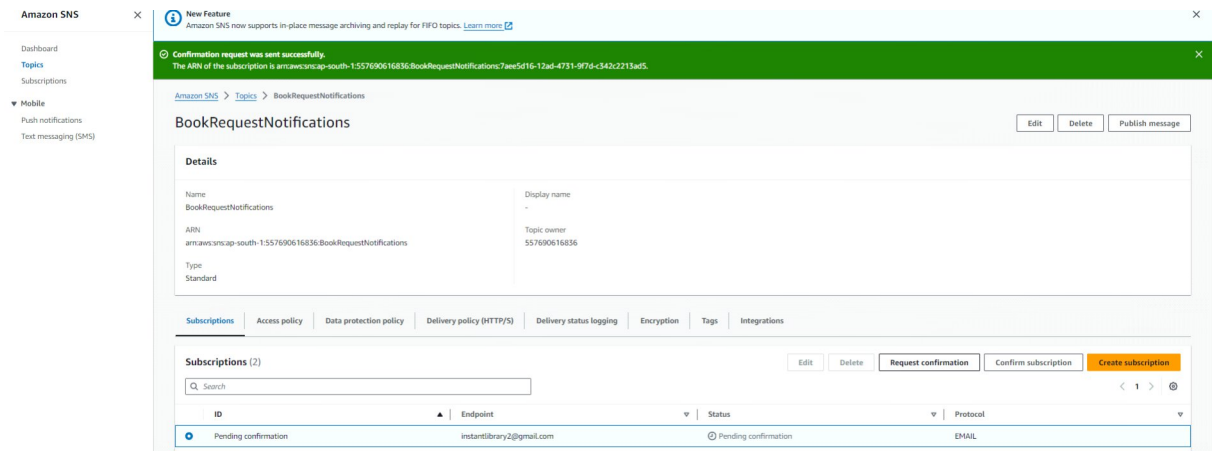
- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

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

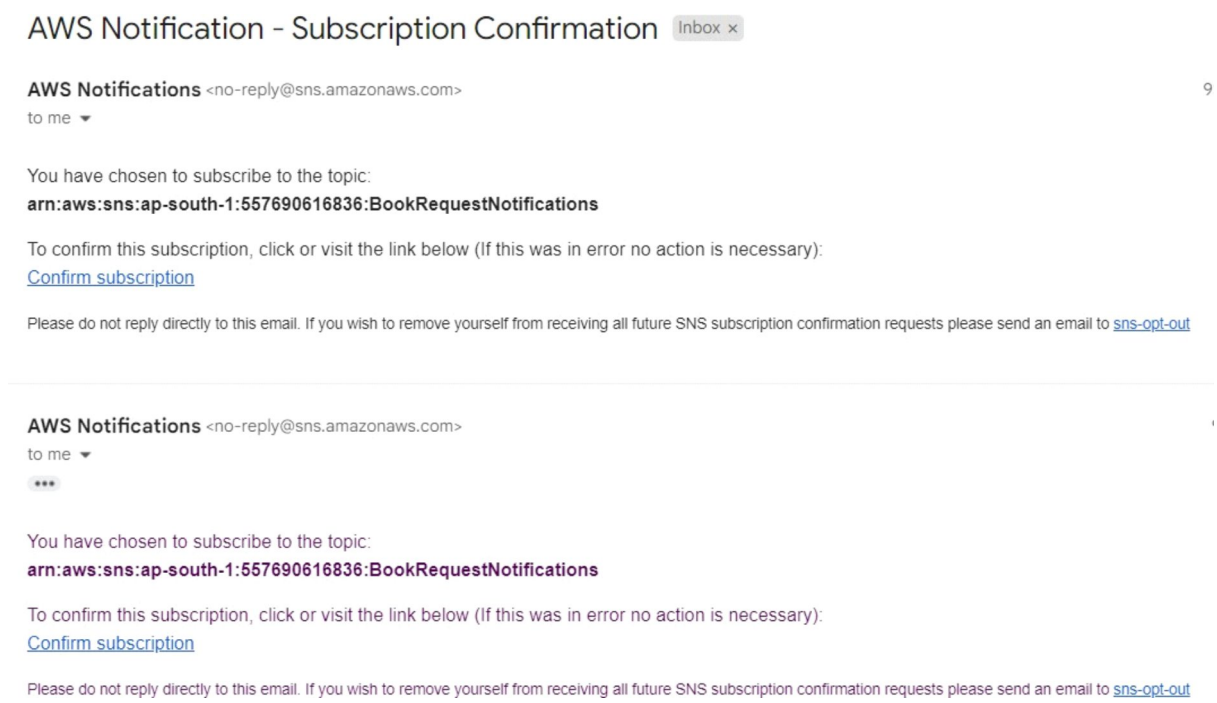


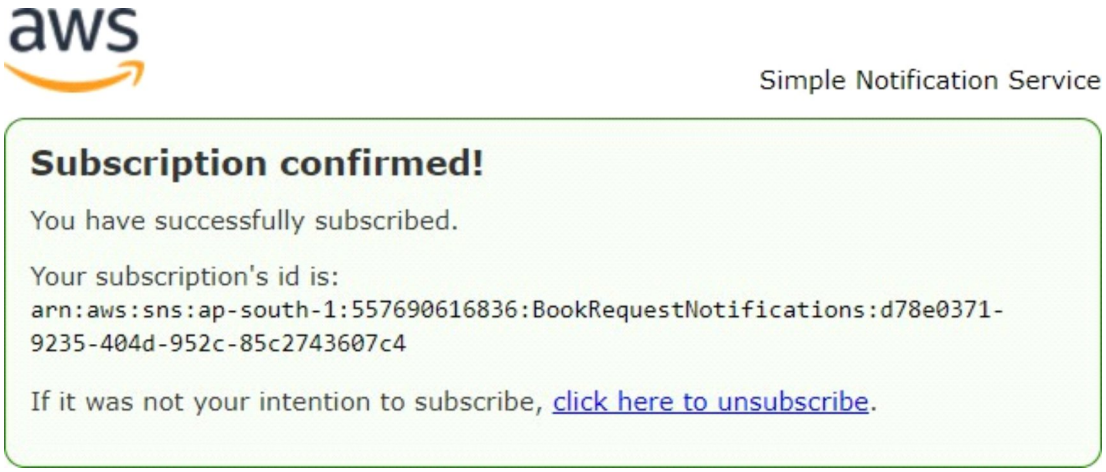


- After subscription request for the mail confirmation

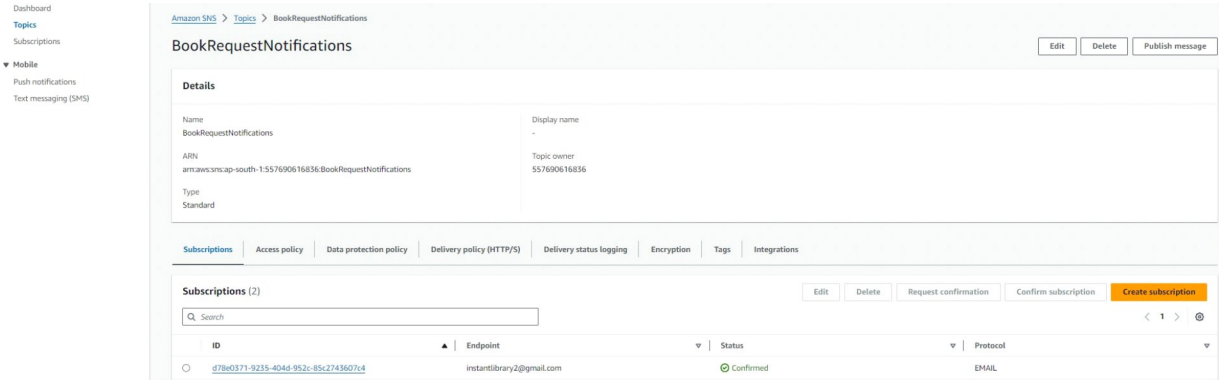


- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.



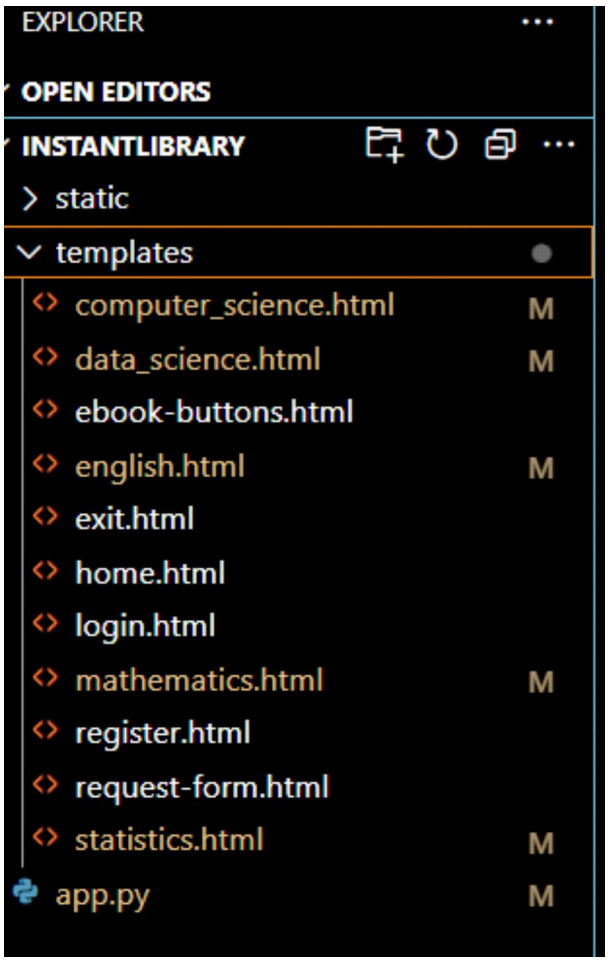


- Successfully done with the SNS mail subscription and setup, now store the ARN link.



Milestone 4:Backend Development and Application Setup

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



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

Description of the code :

- **Flask App Initialization**

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

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

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

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

- **Dynamodb Setup:**

```
# 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:**

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

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

- **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:

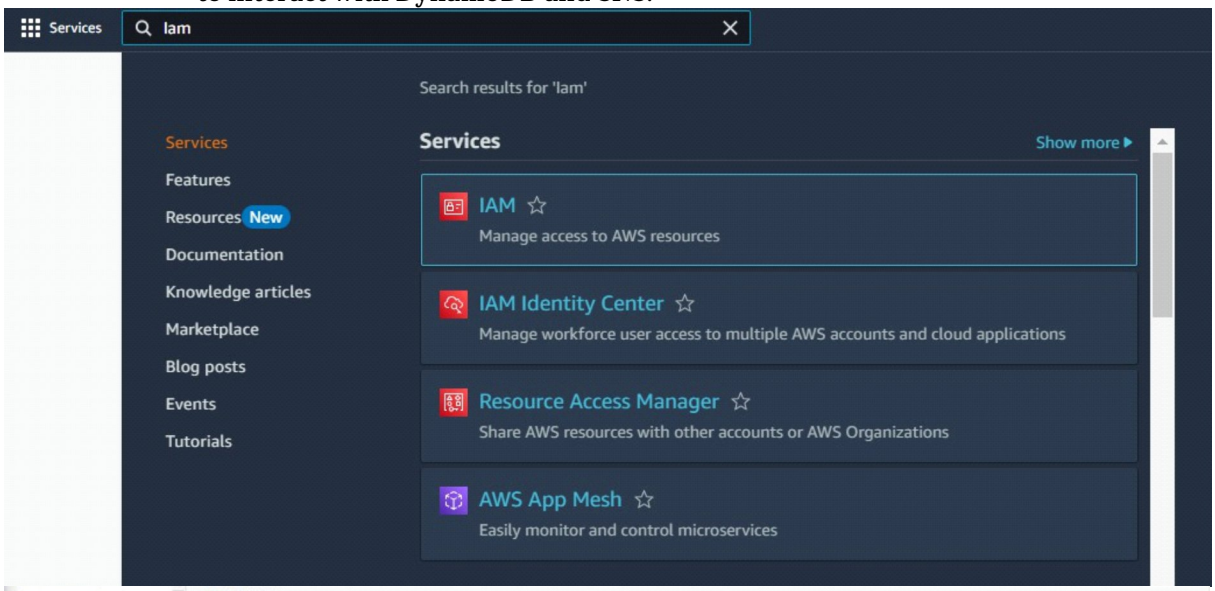
```python
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

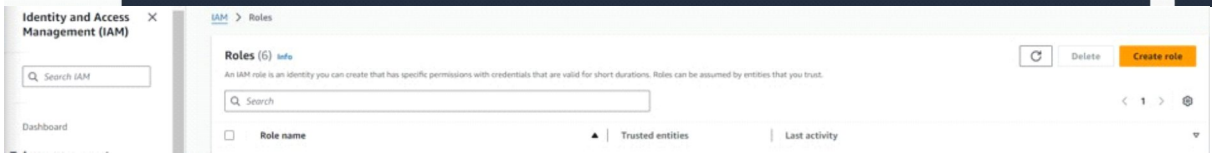
Description: start the Flask server to listen on all network interfaces (`0.0.0.0`) at port `80` with debug mode enabled for development and testing.

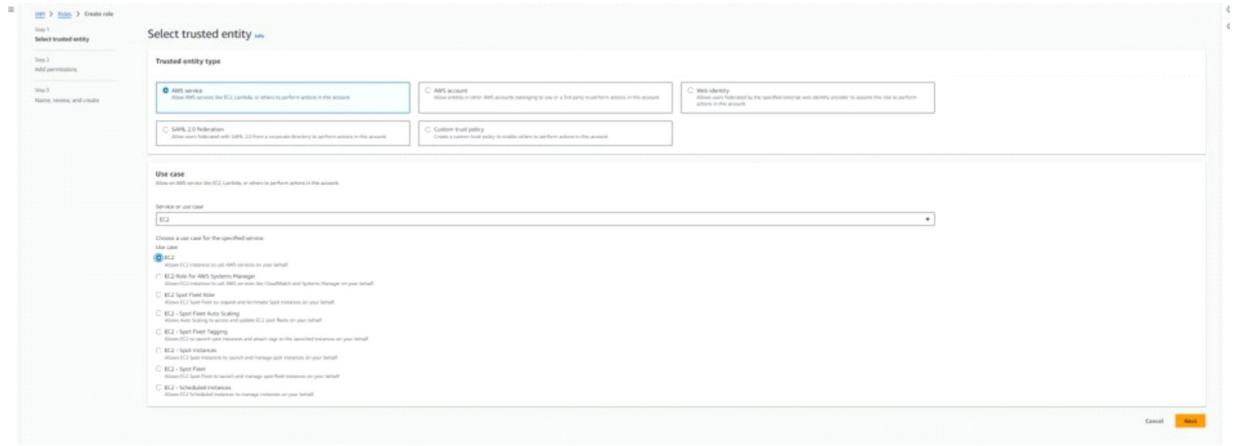
Milestone 5: IAM Role Setup

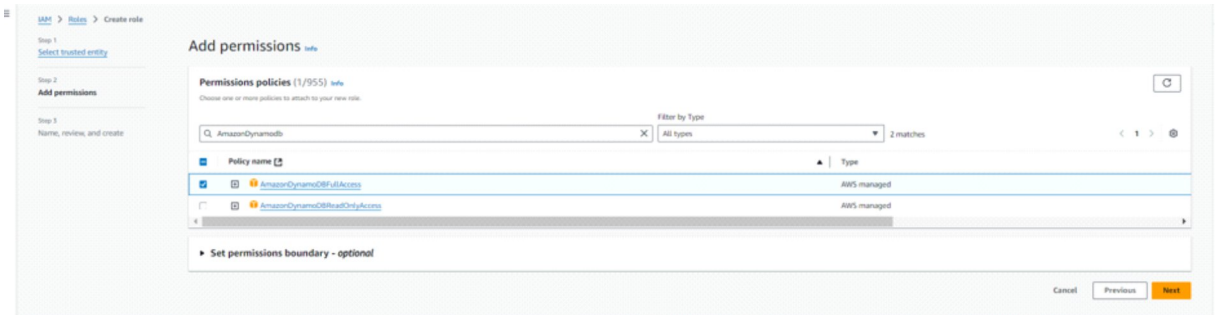
- **Activity 5.1:Create IAM Role.**

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





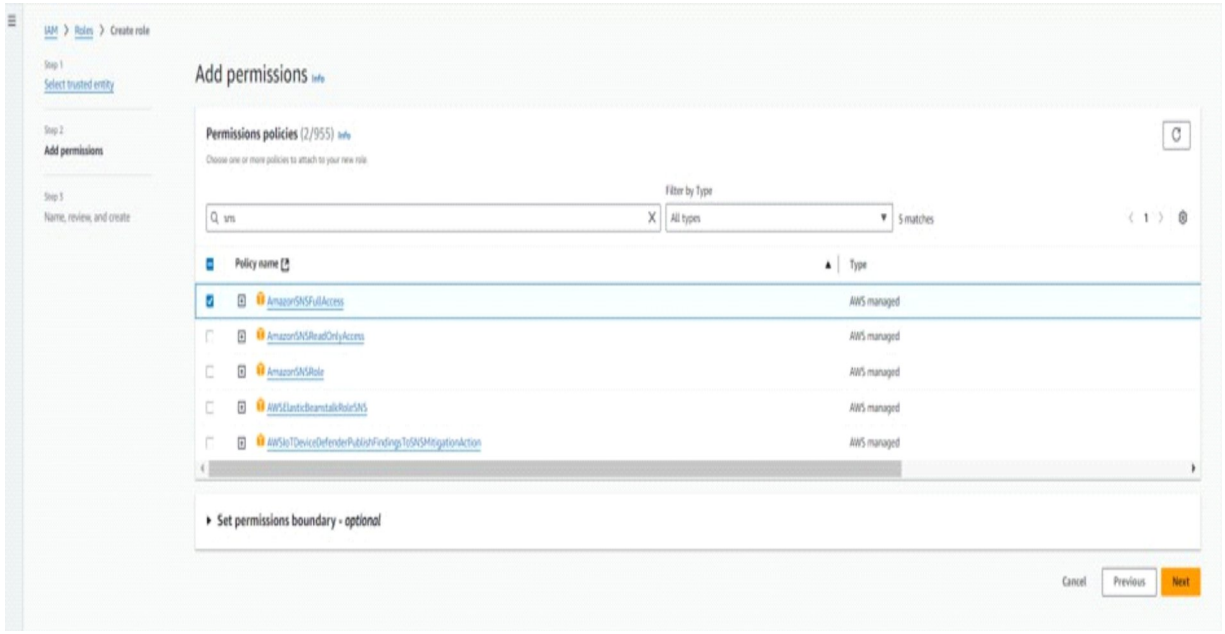


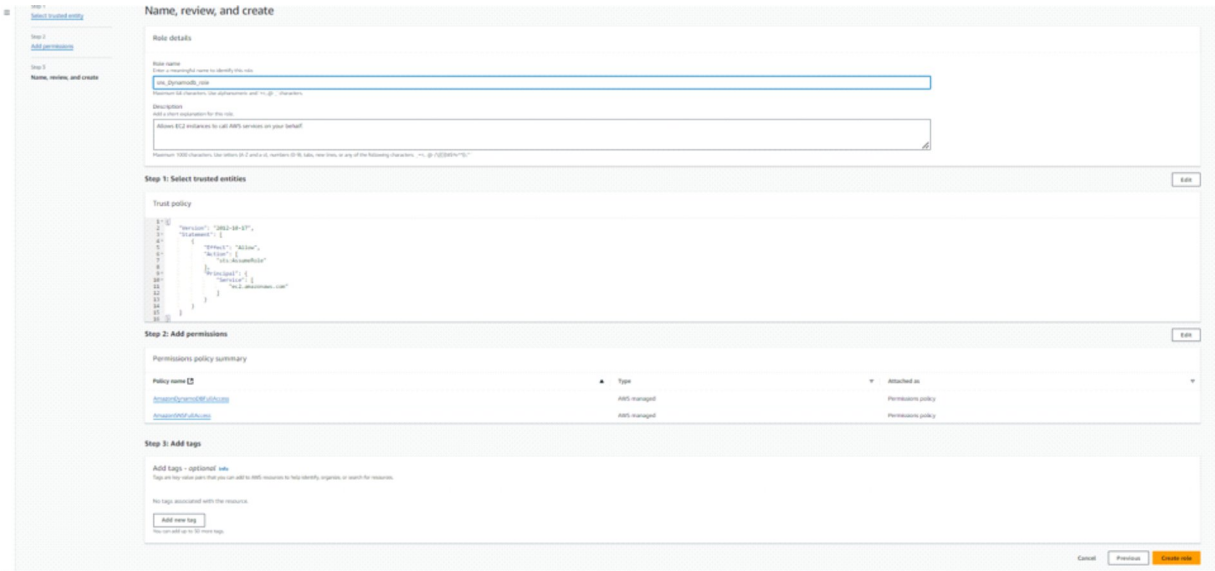


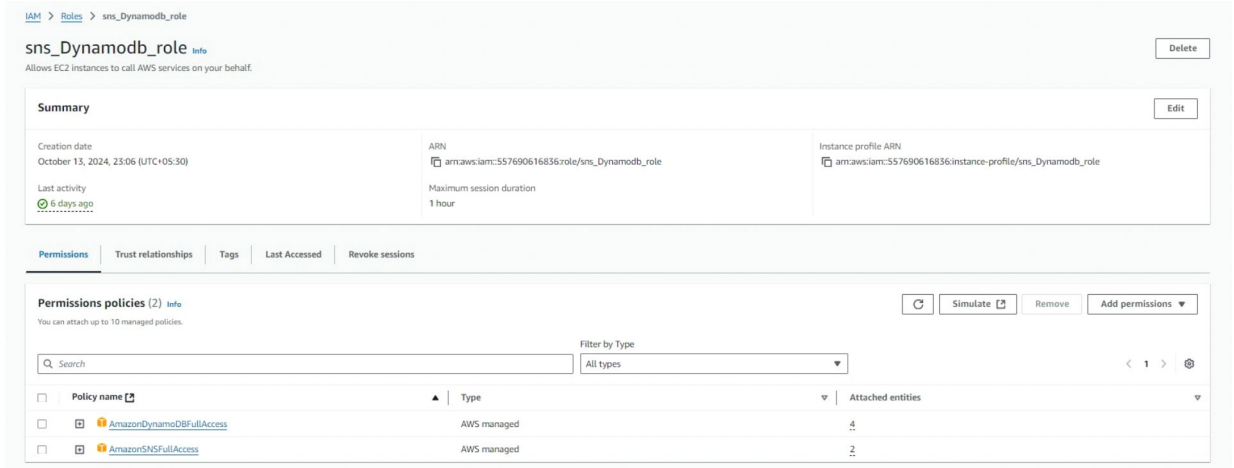
- **Activity 5.2: Attach Policies.**

  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.

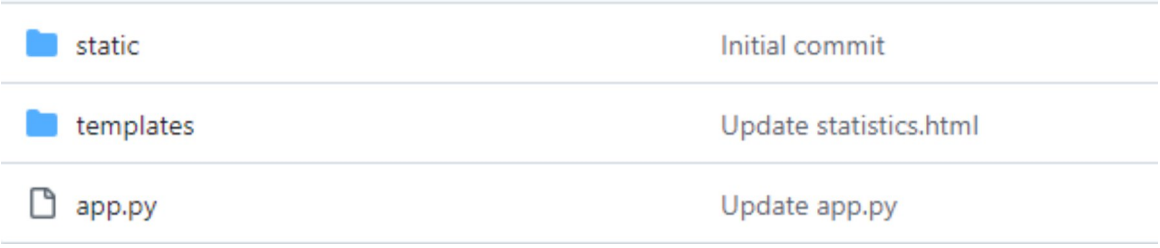


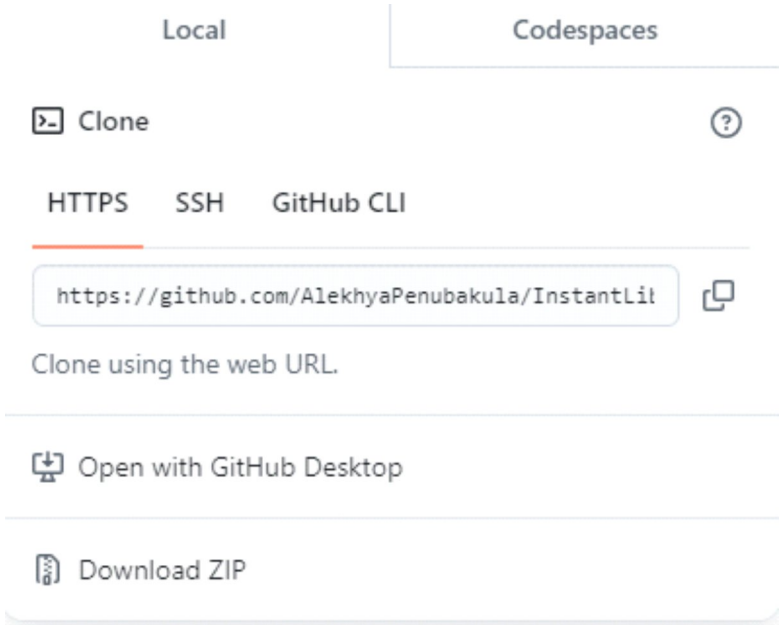




Milestone 6: EC2 Instance Setup

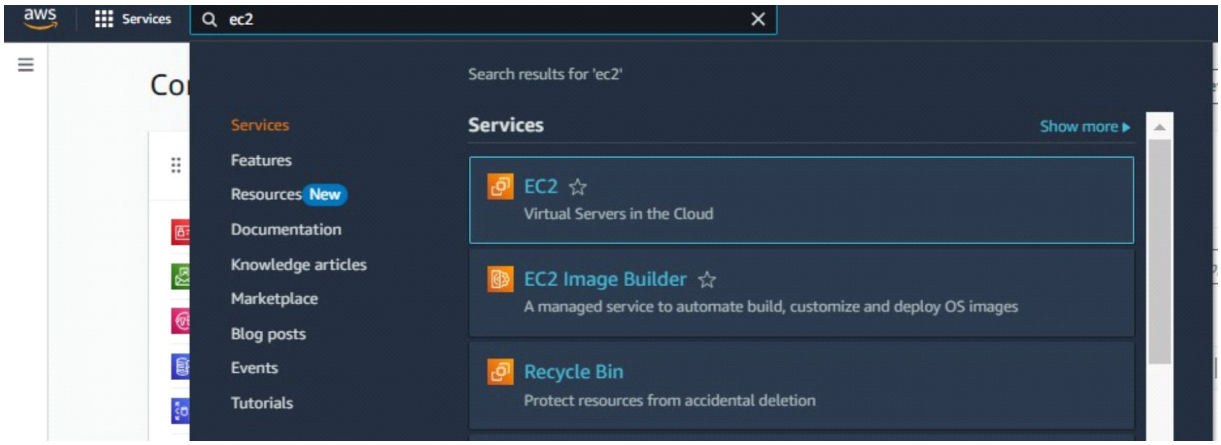
- **Note: Load your Flask app and Html files into GitHub repository.**



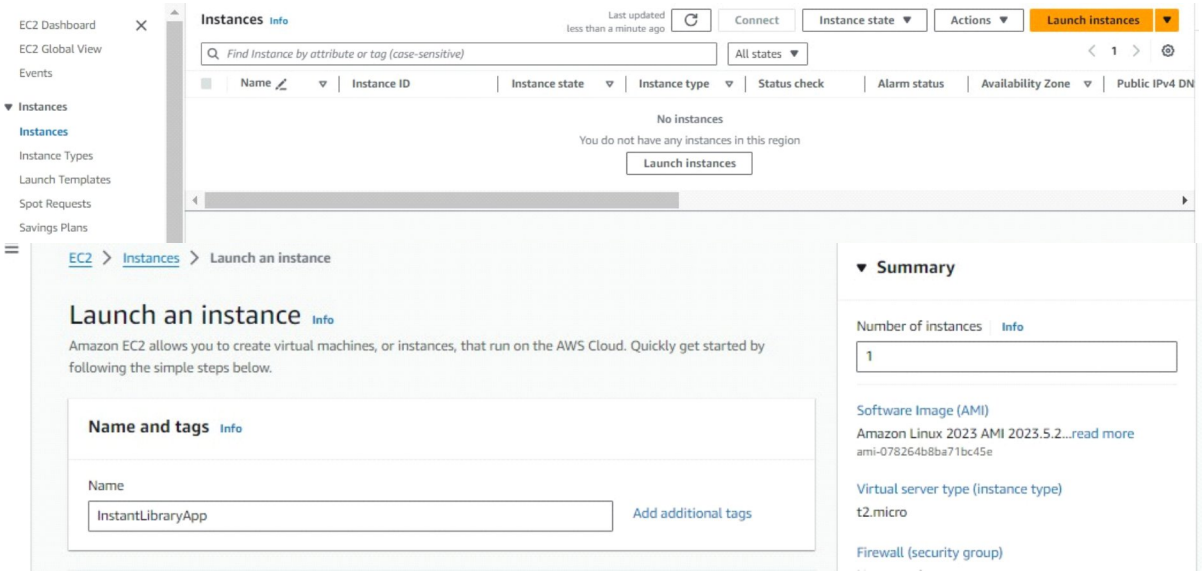


- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

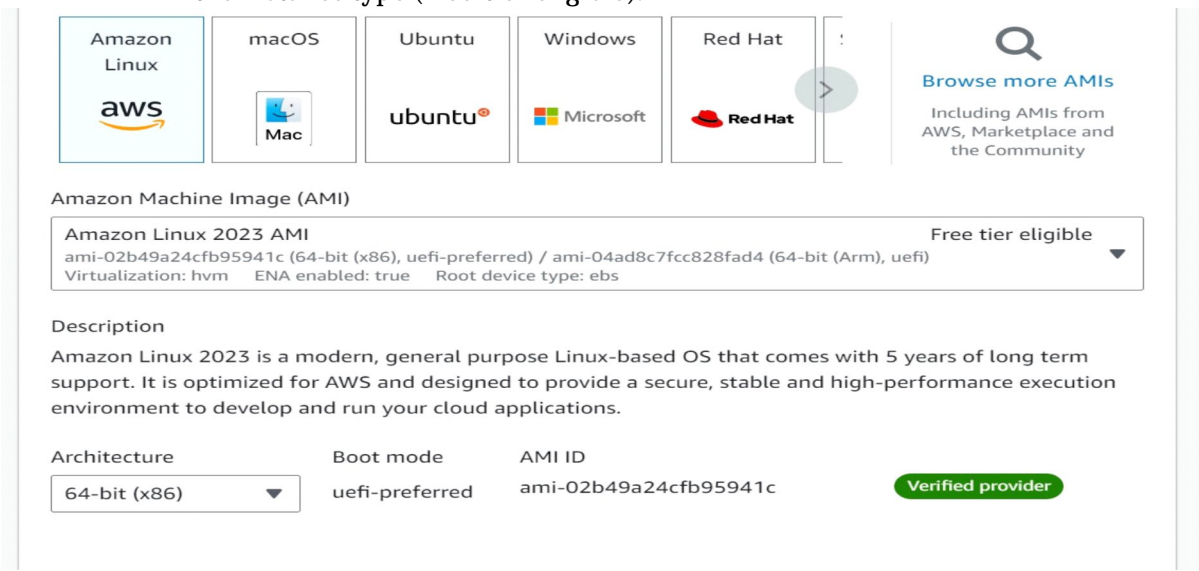
    - **Launch EC2 Instance**
        - In the AWS Console, navigate to EC2 and launch a new instance.



- 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.

Instance type Info | Get advice

Instance type

t2.micro — Free tier eligible

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

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

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

Key pair name - *required*

Select

Create new key pair

Create key pair

Key pair name

Key pairs allow you to connect to your instance securely.

InstantLibrary

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

Key pair type

- RSA — RSA encrypted private and public key pair
- ED25519 — ED25519 encrypted private and public key pair

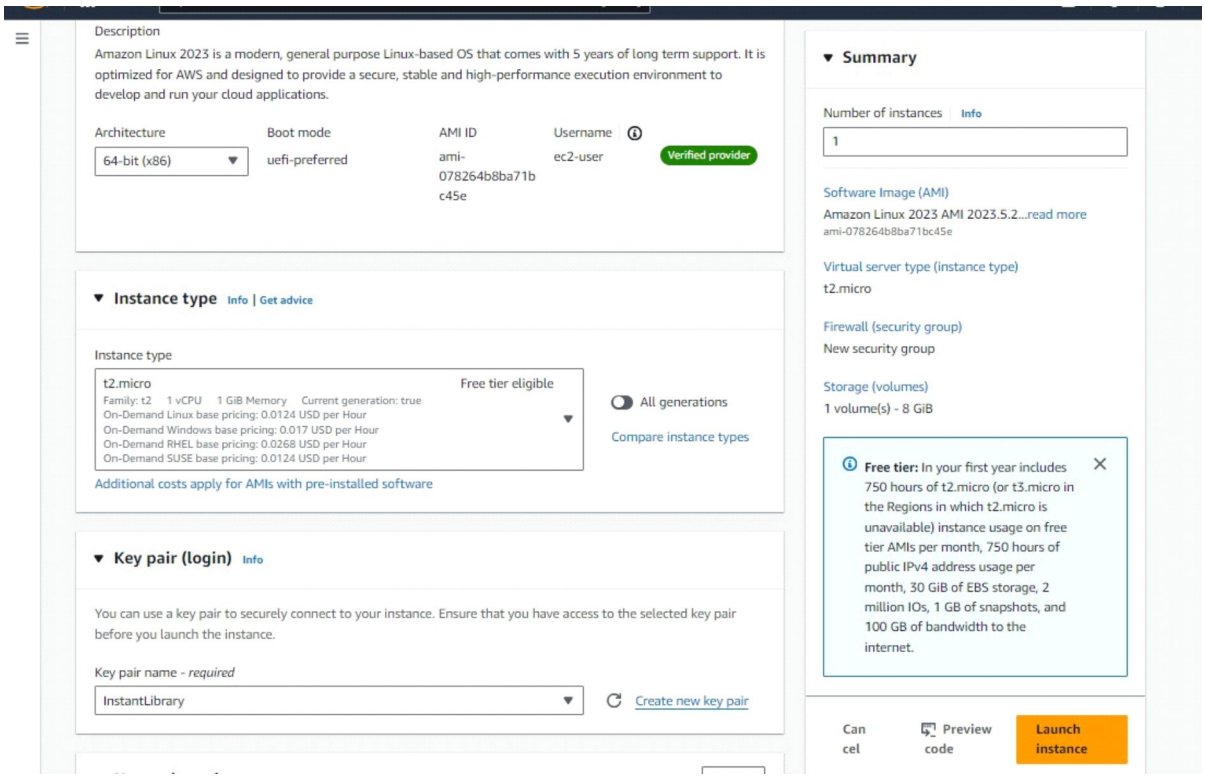
Private key file format

- .pem — For use with OpenSSH
- .ppk — For use with PuTTY

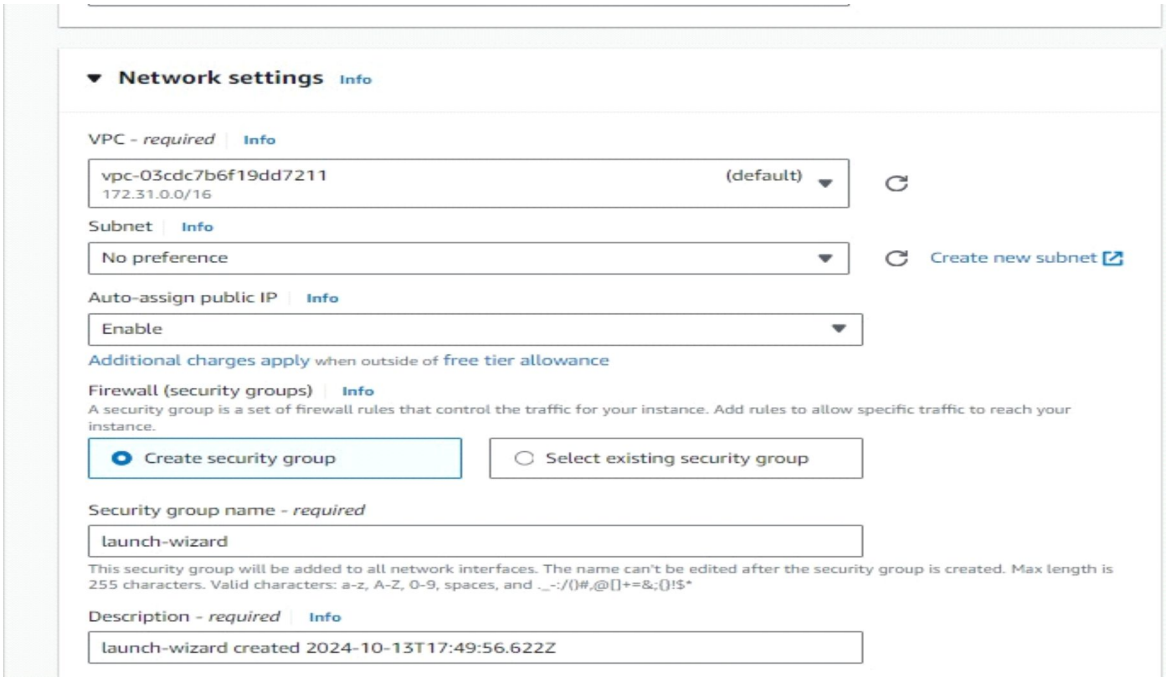
When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** Learn more

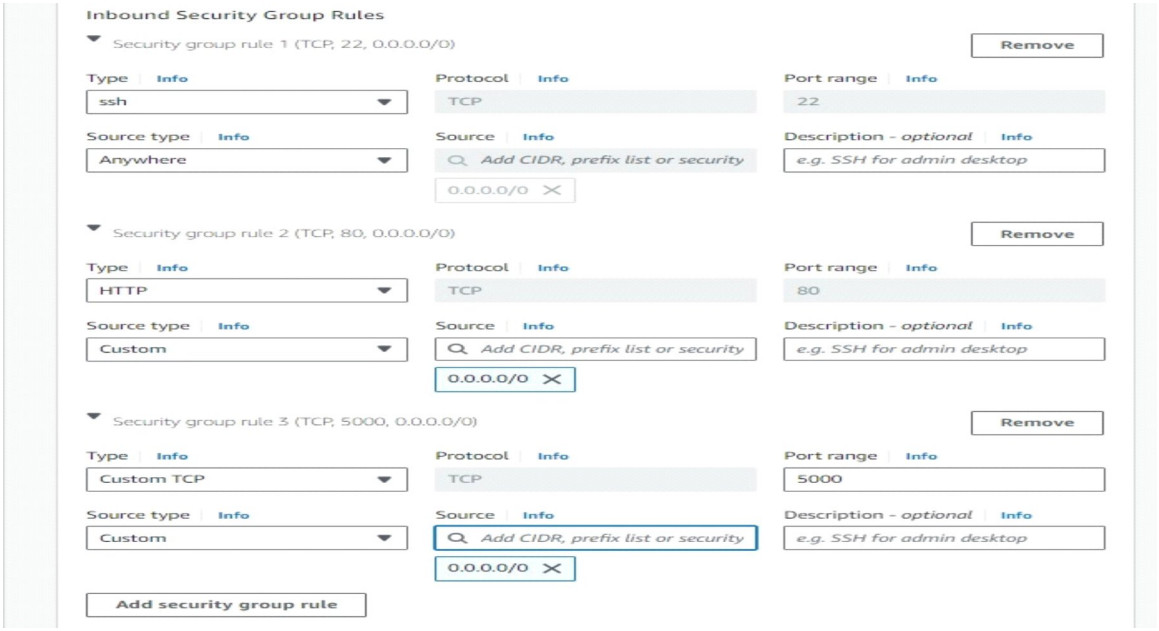
Cancel | Create key pair

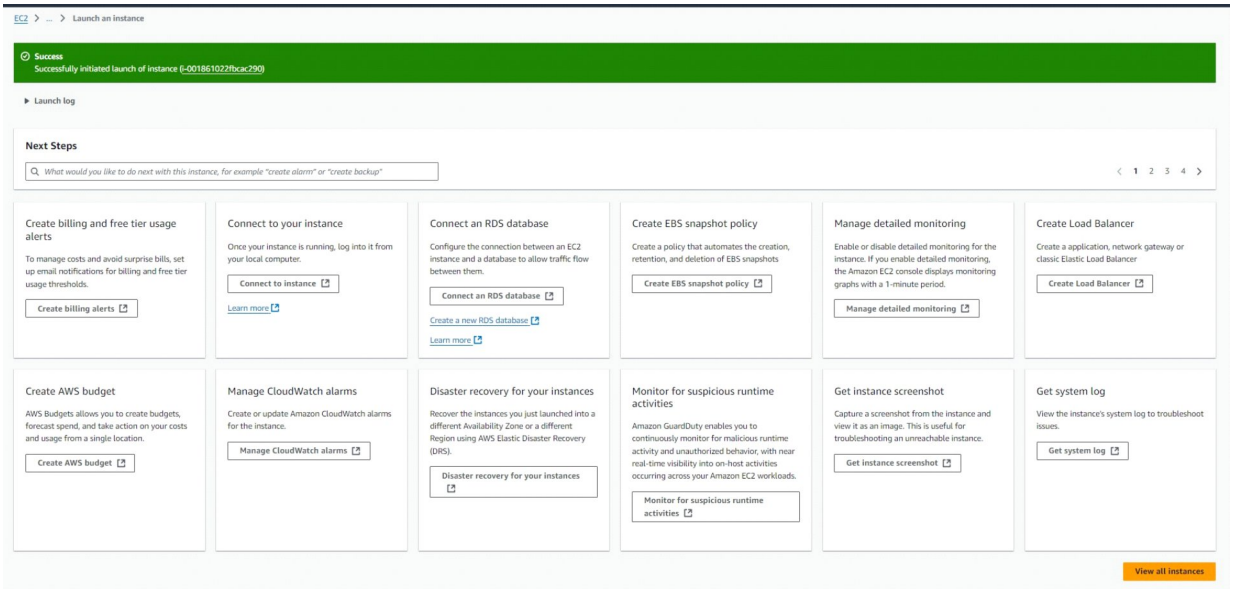
InstantLibrary.pem



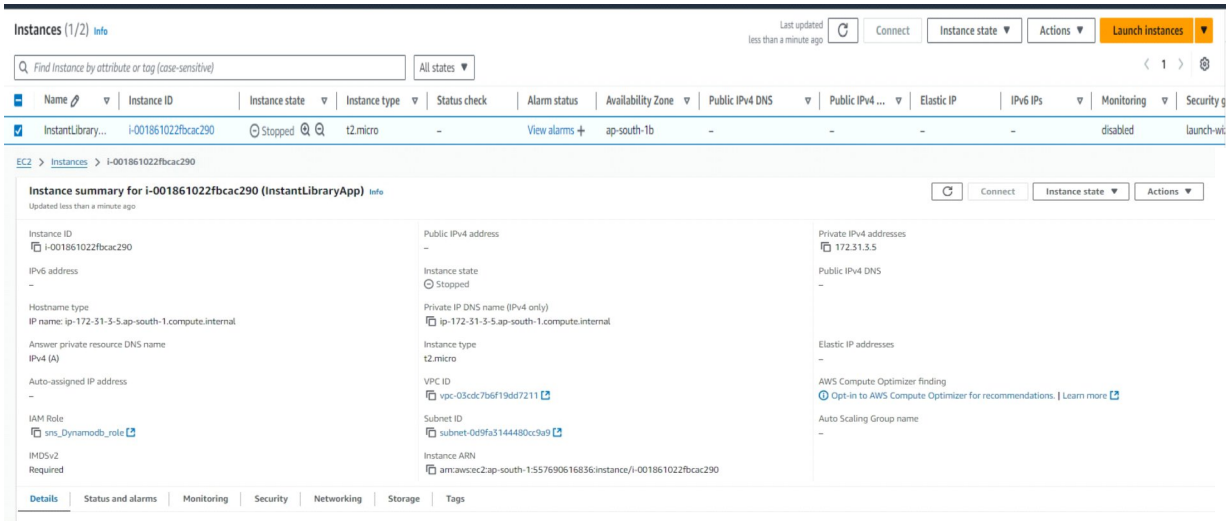
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**

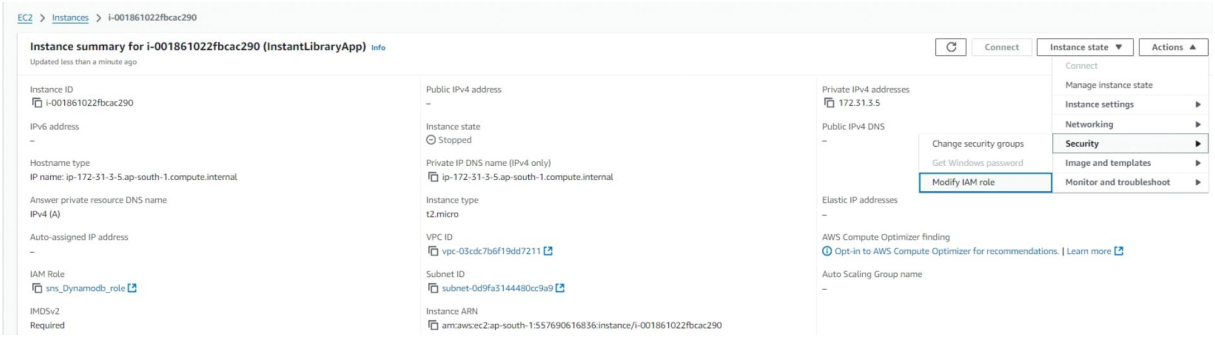


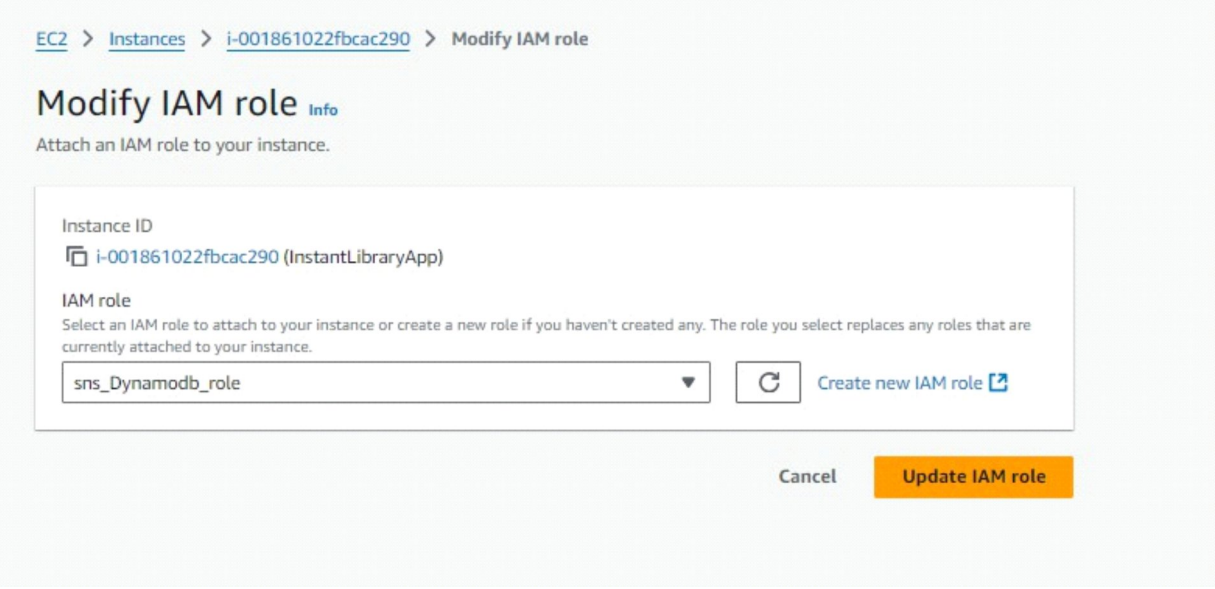




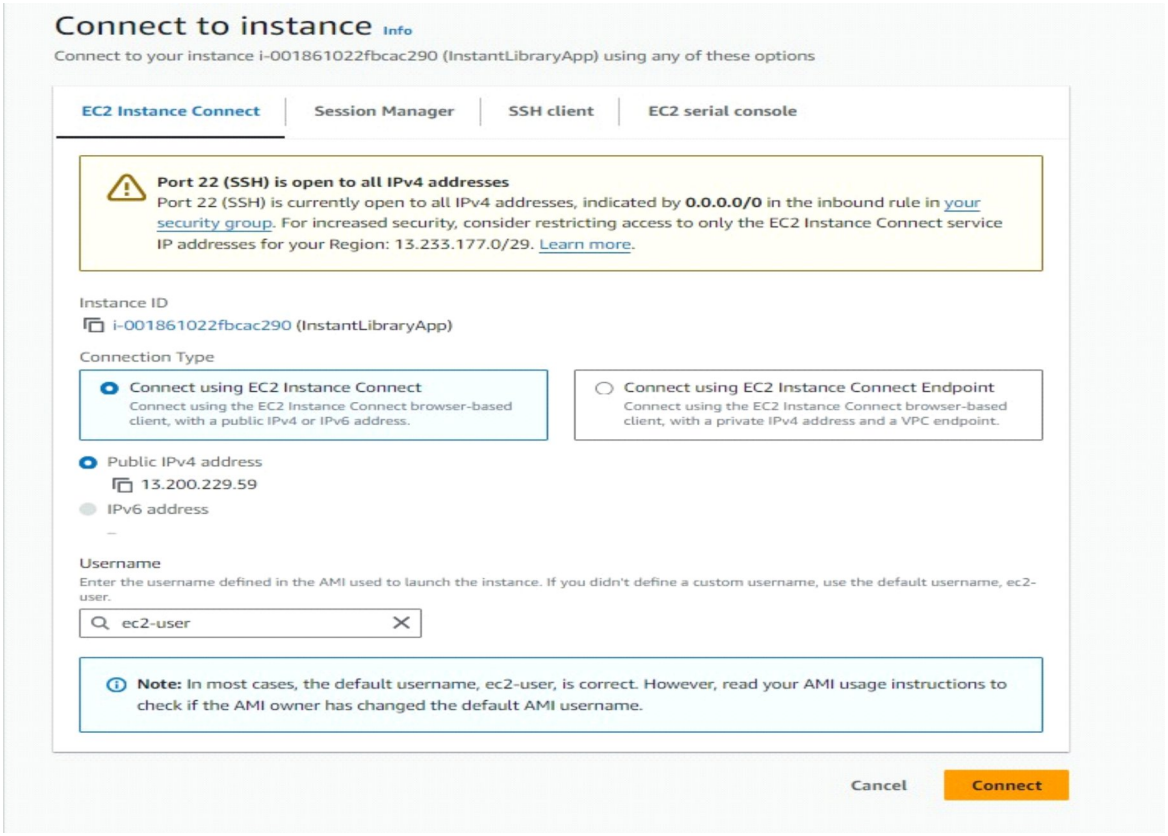
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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

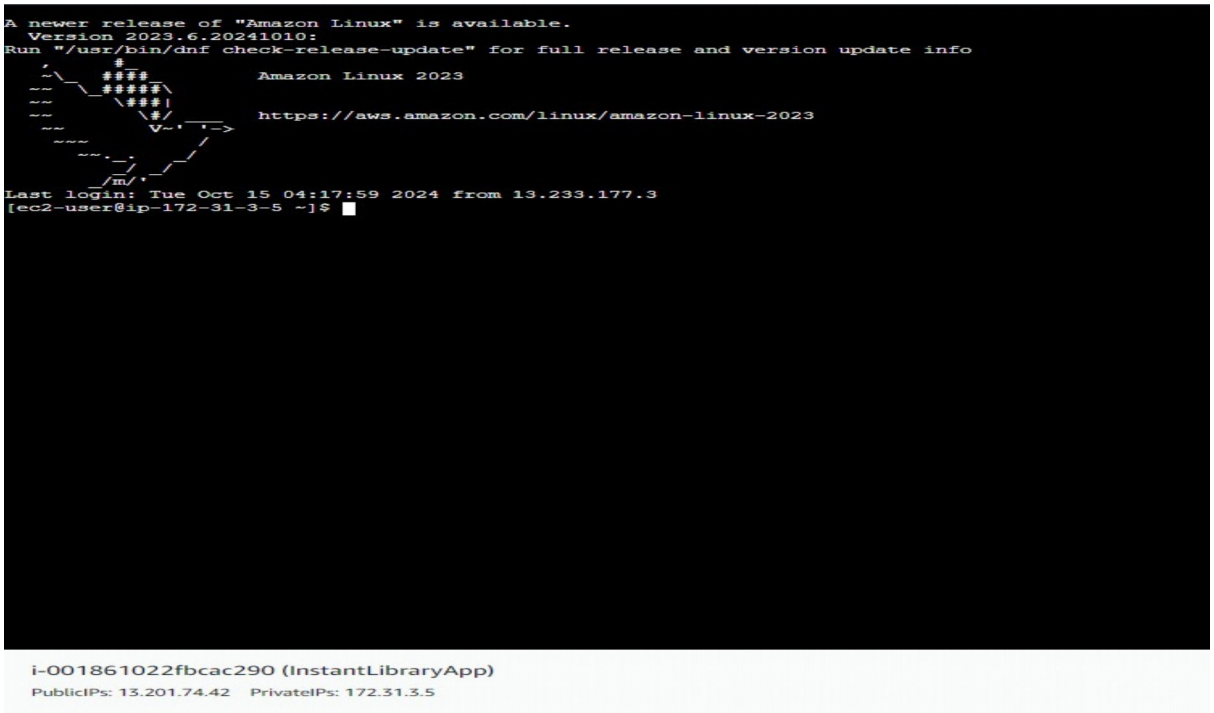






- Now connect the EC2 with the files





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:

f

l

a

s

k

-

-

v

e

r

s

i

o

n

g

i

t

-

-

v

e

r

s

i

o

n

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- HYPERLINK "https://github.com/your-github-username/your-repository-name.git" HYPERLINK "https://github.com/your-github-username/your-repository-name.git" HYPERLINK "https://github.com/your-github-username/your-repository-name.git"name.git'

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

your credentials here: **'git clone**

https://github.com/Mahalakshmi13-creater/Medtrack.git

This is 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

```
A newer release of "Amazon Linux" is available.
  Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$ git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git
fatal: destination path 'InstantLibrary' already exists and is not an empty directory.
[ec2-user@ip-172-31-3-5 ~]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ flask run --host=0.0.0.0 --port=80
 * Debug mode: off
Permission denied
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
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://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
^C[ec2-user@ip-172-31-3-5 InstantLibrary]$
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
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://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```

i-001861022fbcac290 (InstantLibraryApp)

PublicIPs: 13.201.74.42 PrivateIPs: 172.31.3.5

Verify the Flask app is running: http://your-ec2-public-ip

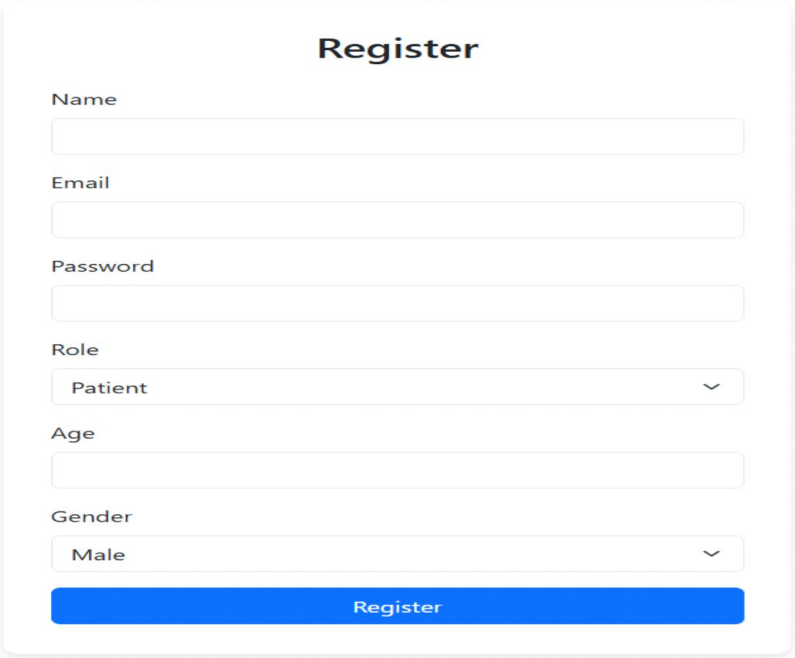
- Run the Flask app on the EC2 instance

```
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
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://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```

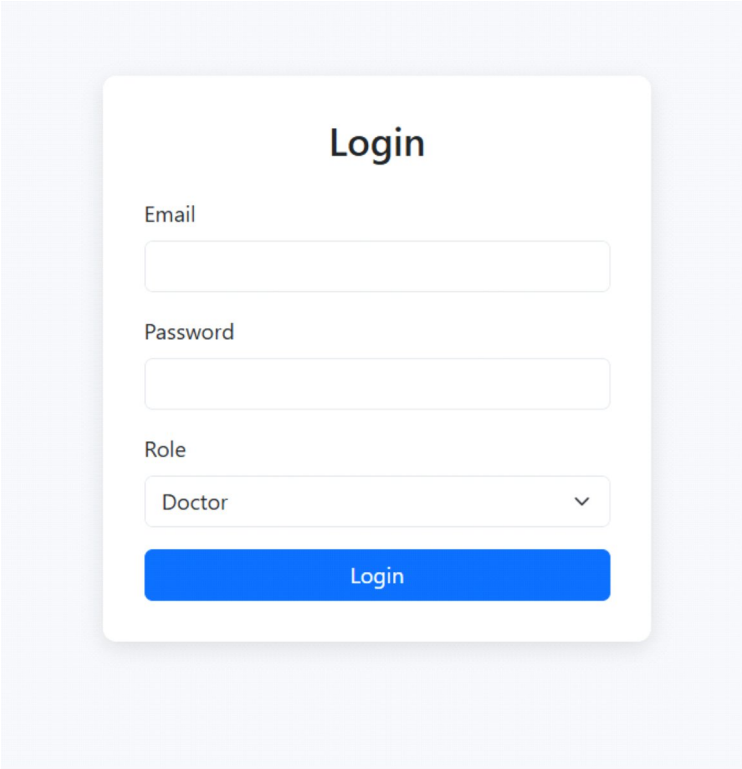
Milestone 8: Testing and Deployment

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

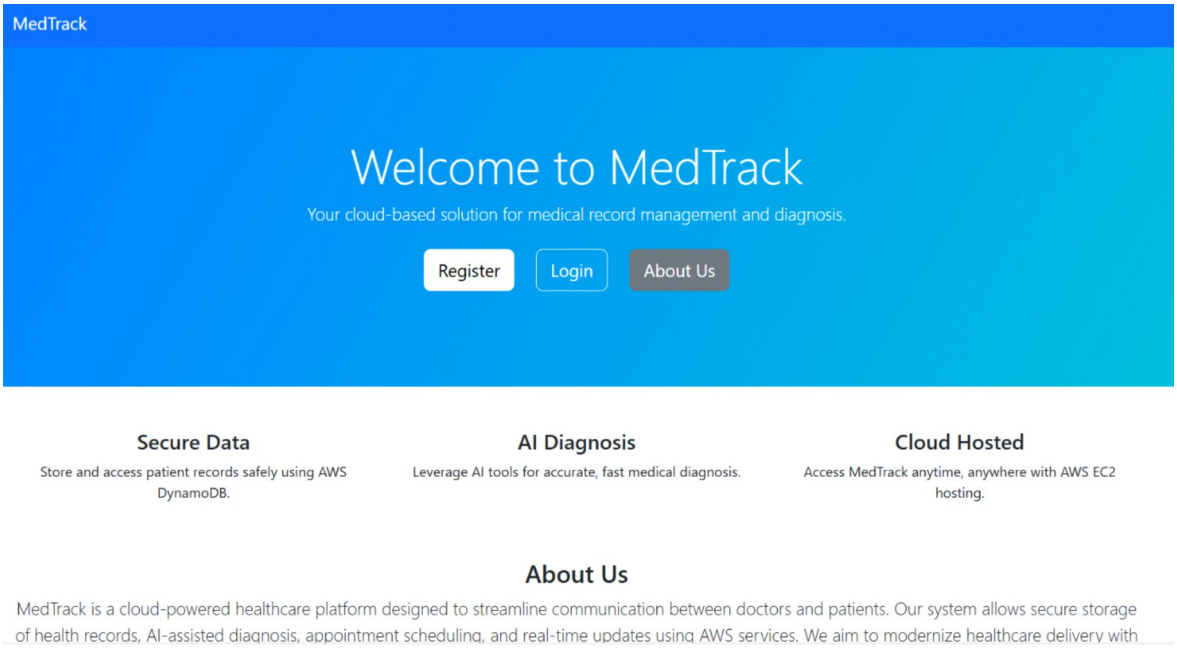
  Register Page:



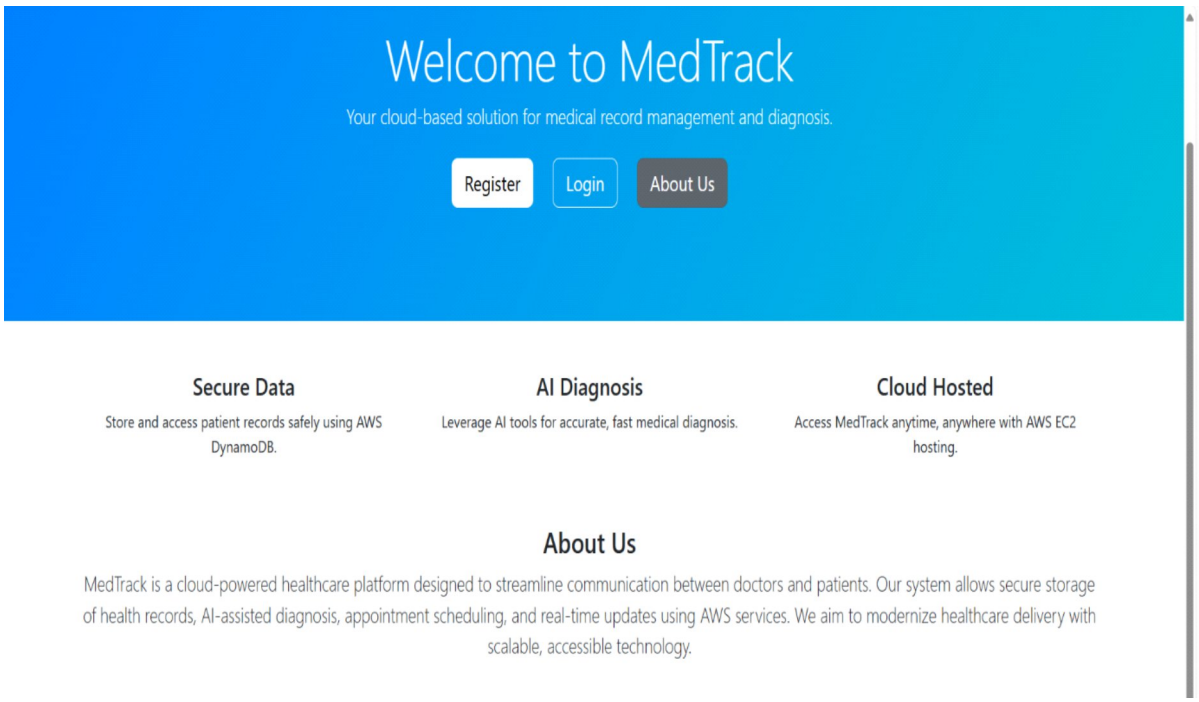
  Login Page:



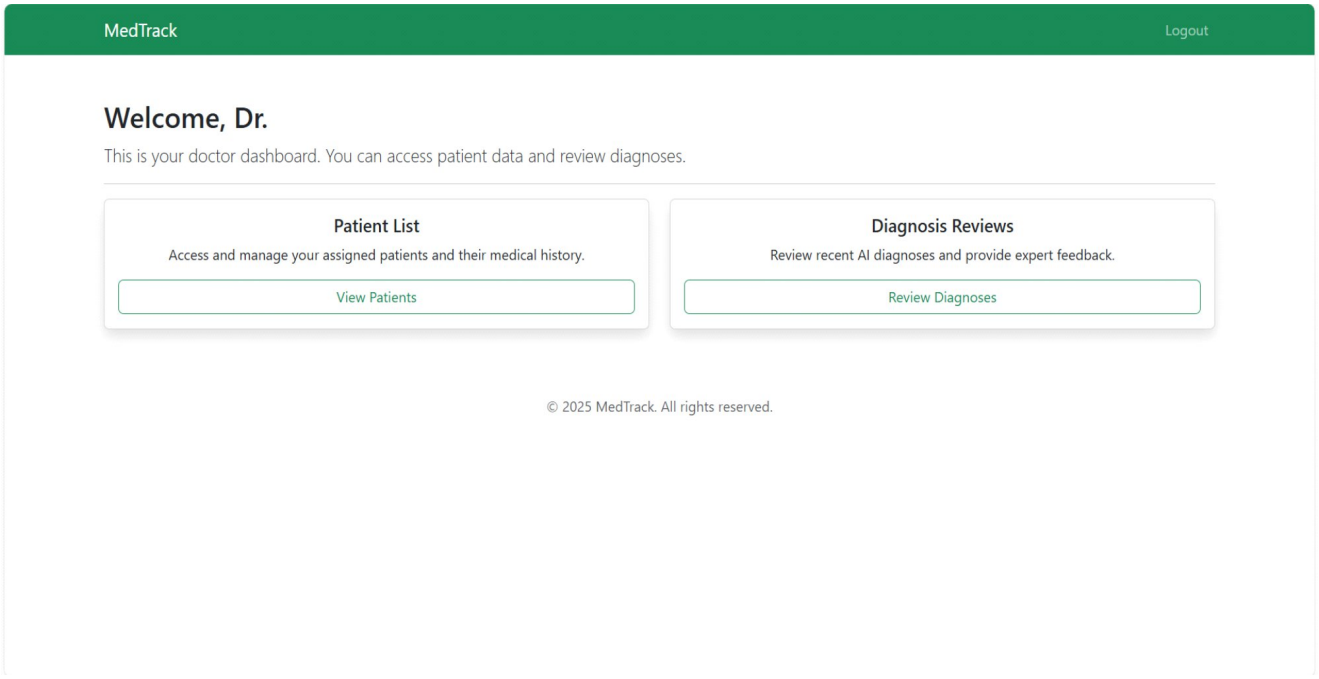
Home page:



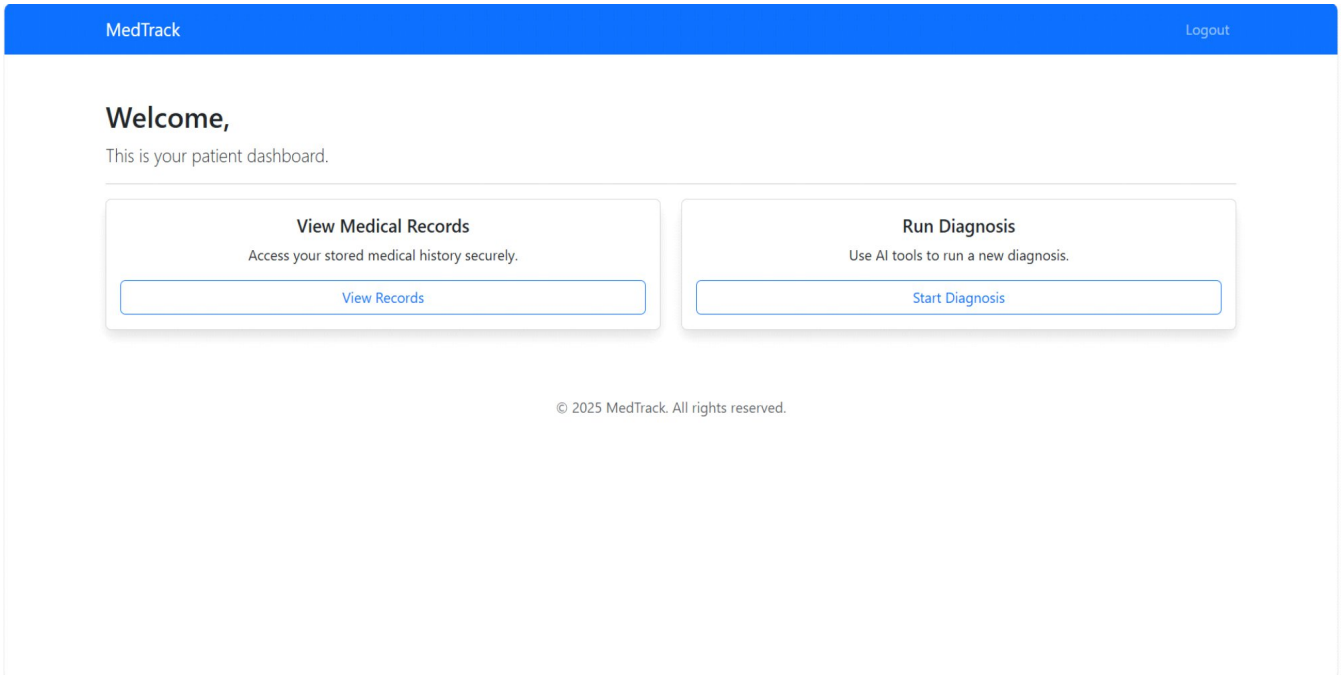
About Us page:



Doctor DashBoard:



Patient DashBoard:



Exit:

Session Ended

Please close this tab.

Conclusion:

MedTrack is a web-based healthcare management system built using Flask that facilitates interaction between doctors and patients. It allows users to register and log in as either a doctor or a patient. Patients can view available doctors, book appointments, and view their scheduled or past visits. Doctors can access their assigned appointments, update them with a diagnosis, treatment plan, and prescription, and mark them as completed. The system uses in-memory storage for users and appointments, making it suitable for demonstration or development purposes without requiring a database. Although email and AWS SNS notification features are present in the code, they are disabled by default. MedTrack is designed with a clear role-based workflow and is easily extendable for real-world deployment with additional features like persistent databases, email alerts, password encryption, and a modern UI.