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.

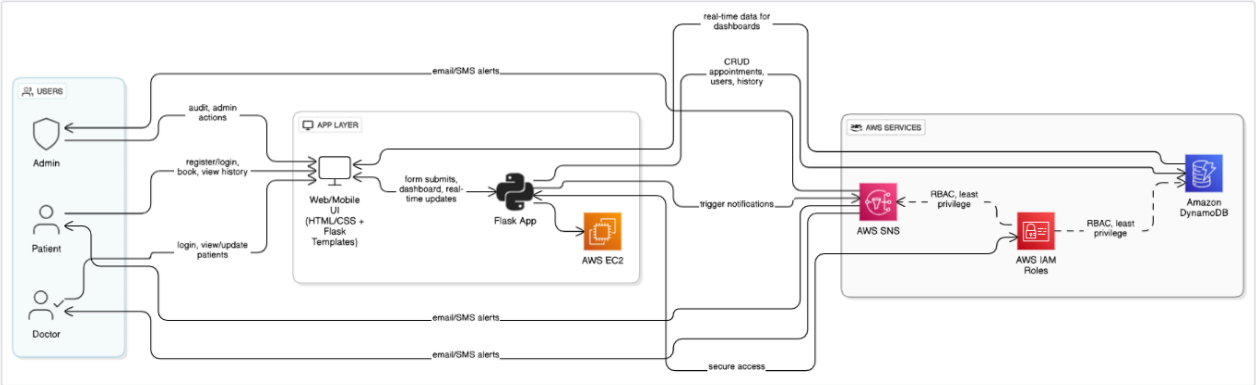
Scenario 2: Secure User Management with IAM

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.

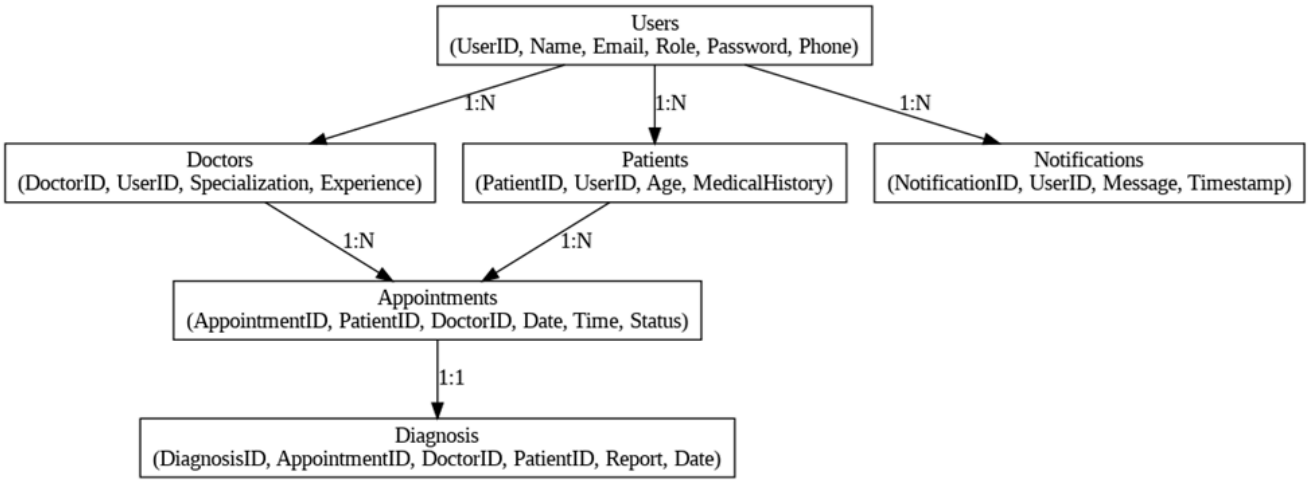
Scenario 3: Easy Access to Medical History and Resources

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:

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

Project WorkFlow:

1. AWS Account Setup and Login

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

Activity 1.2: Log in to the AWS Management Console

2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.

Activity 2.2: Configure Attributes for User Data and 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

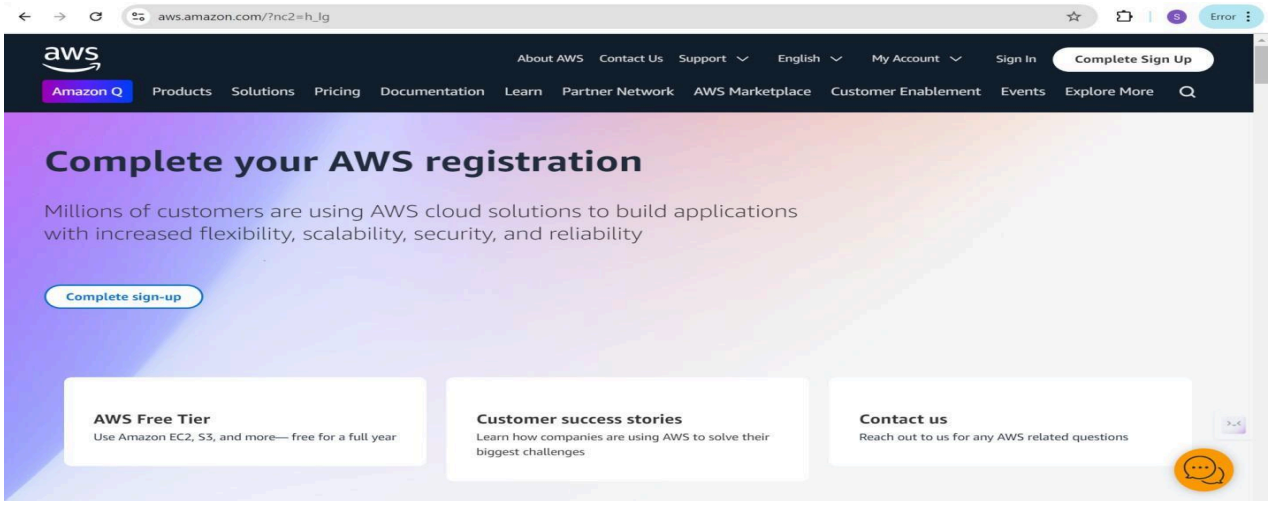
Activity 7.2: Run the Flask App

8. Testing and Deployment

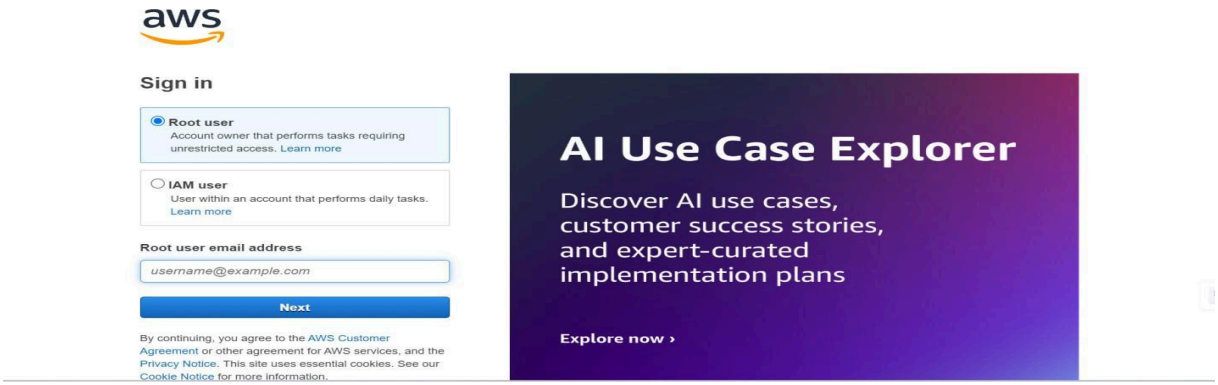
Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.

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.

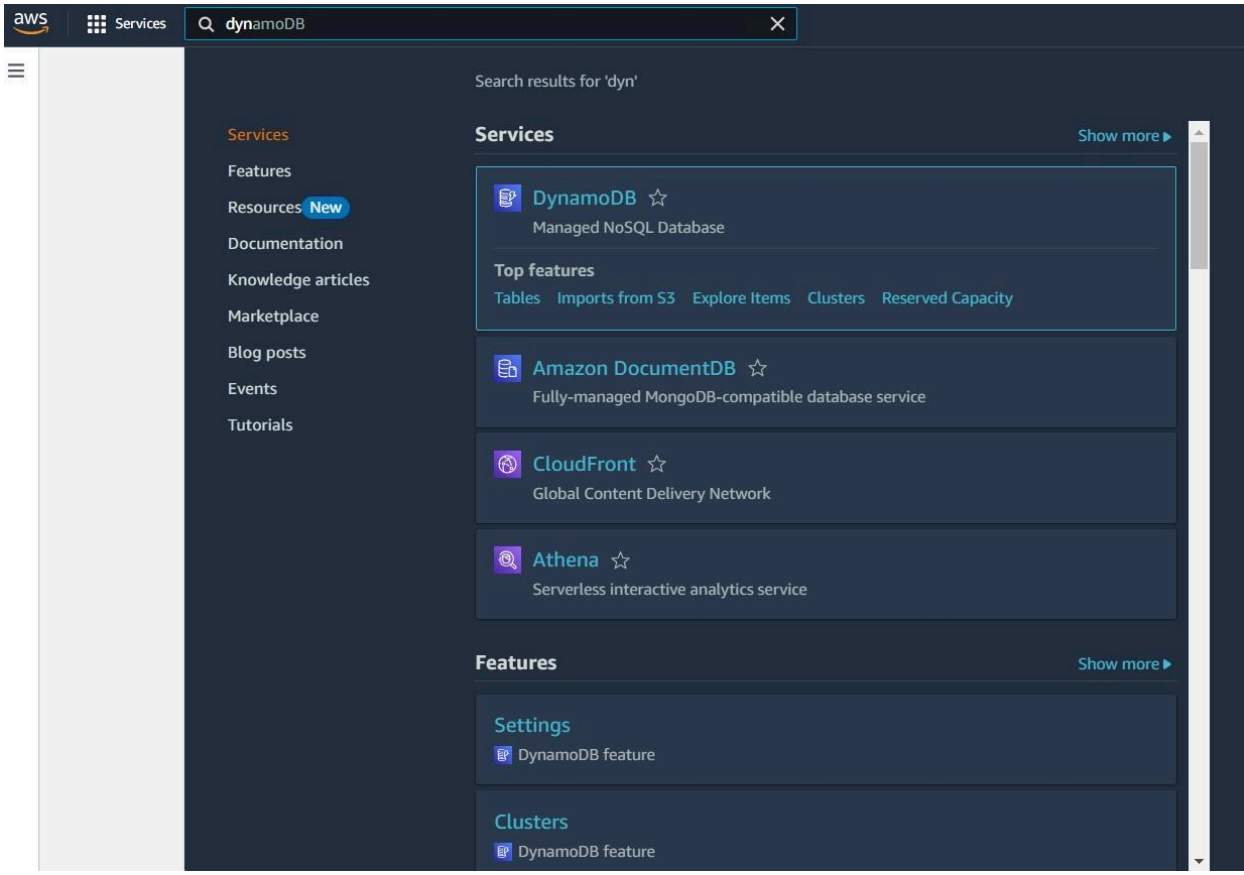


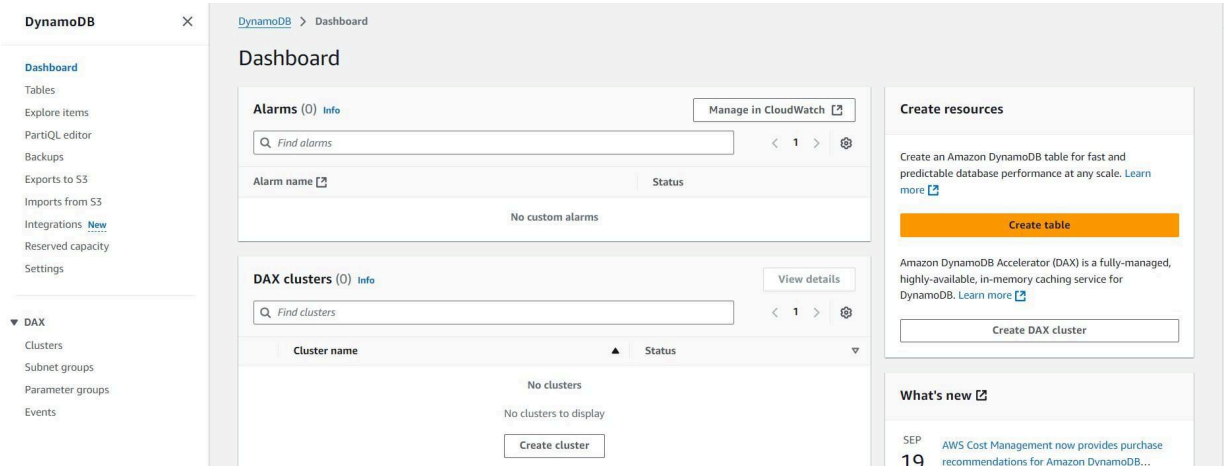
- **Activity 1.2: Log in to the AWS Management Console**
  - After setting up your account, log in to the [AWS Management Console].

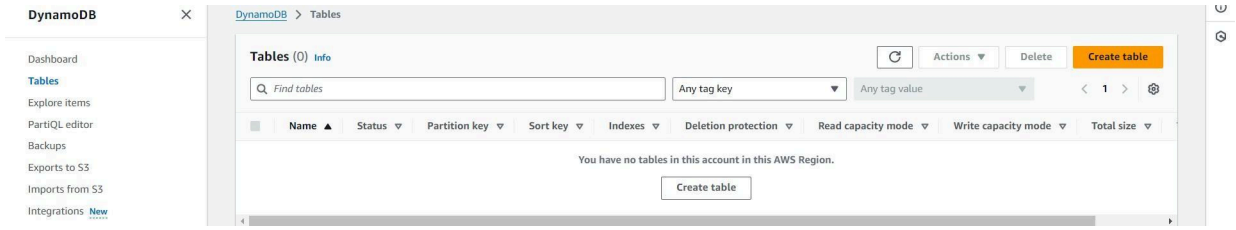


Milestone 2: DynamoDB Database Creation and Setup

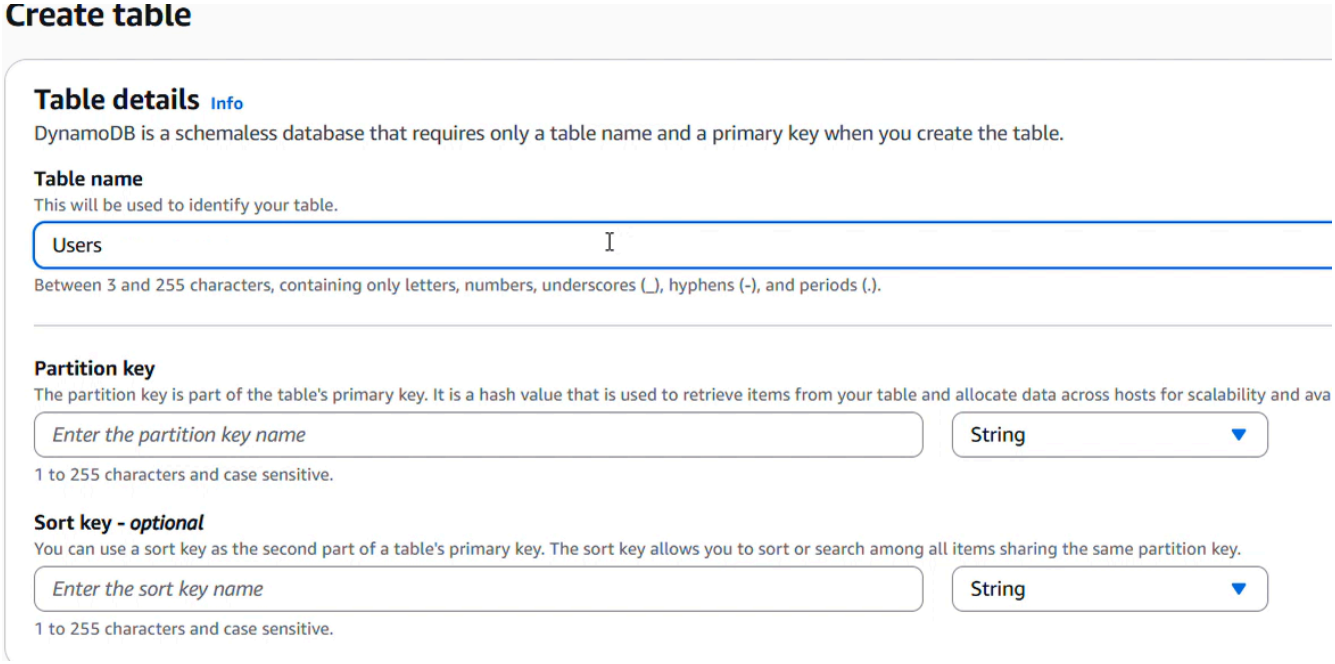
- **Activity 2.1:Navigate to the DynamoDB**
    - In the AWS Console, navigate to DynamoDB and click on create tables.

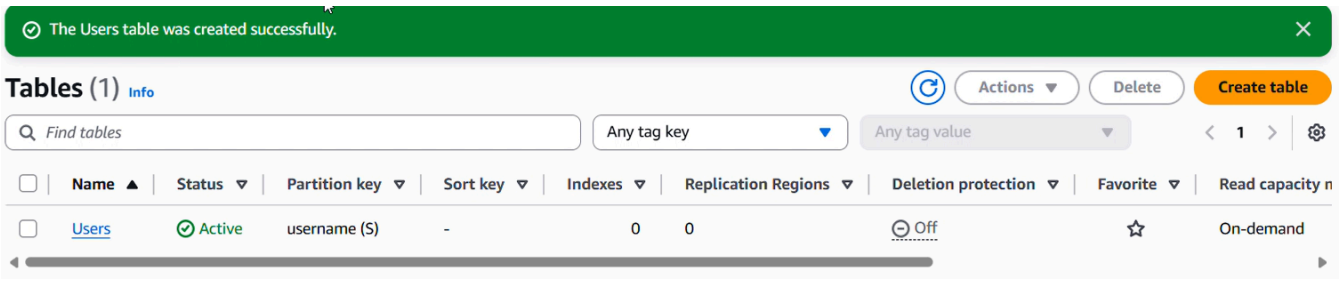




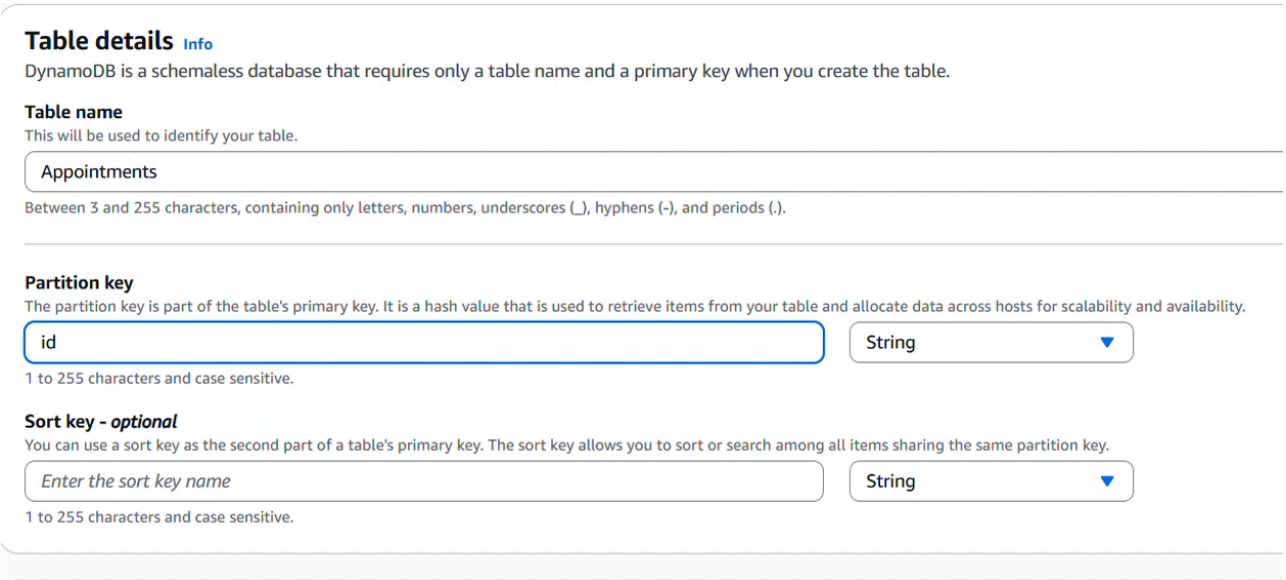


- **Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**
    - Create Users table with partition key “Id” with type String and click on create tables.





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



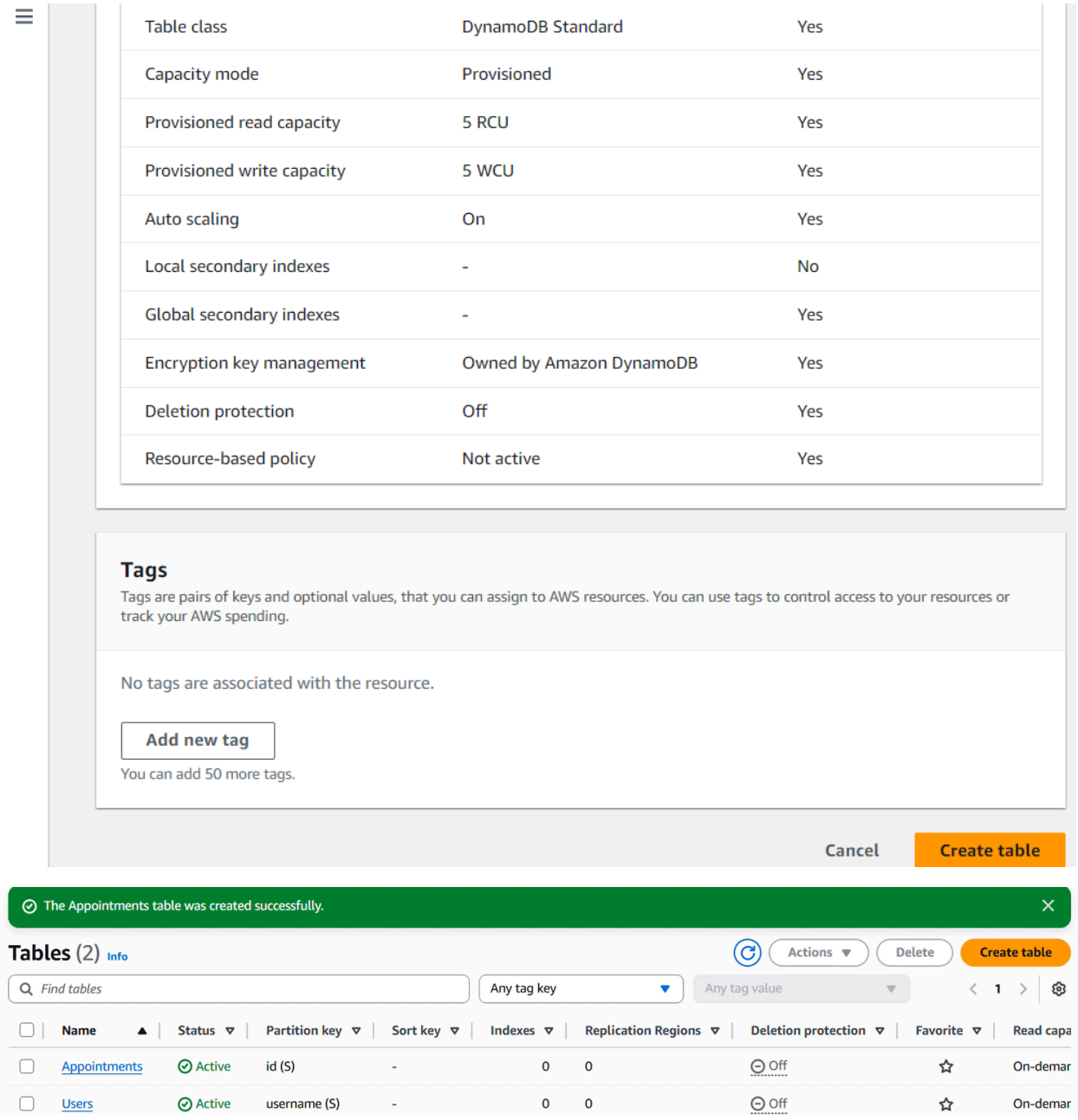
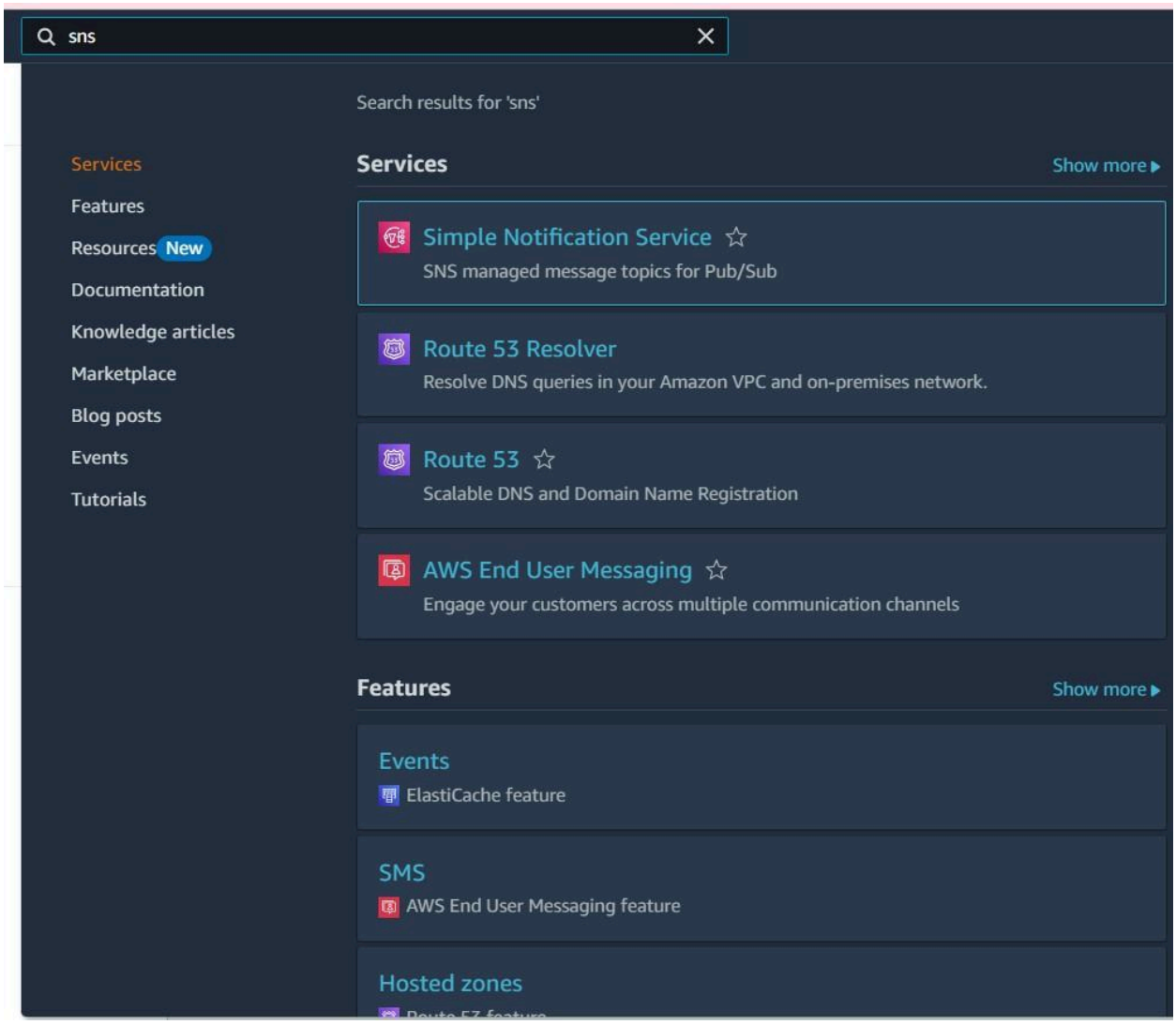


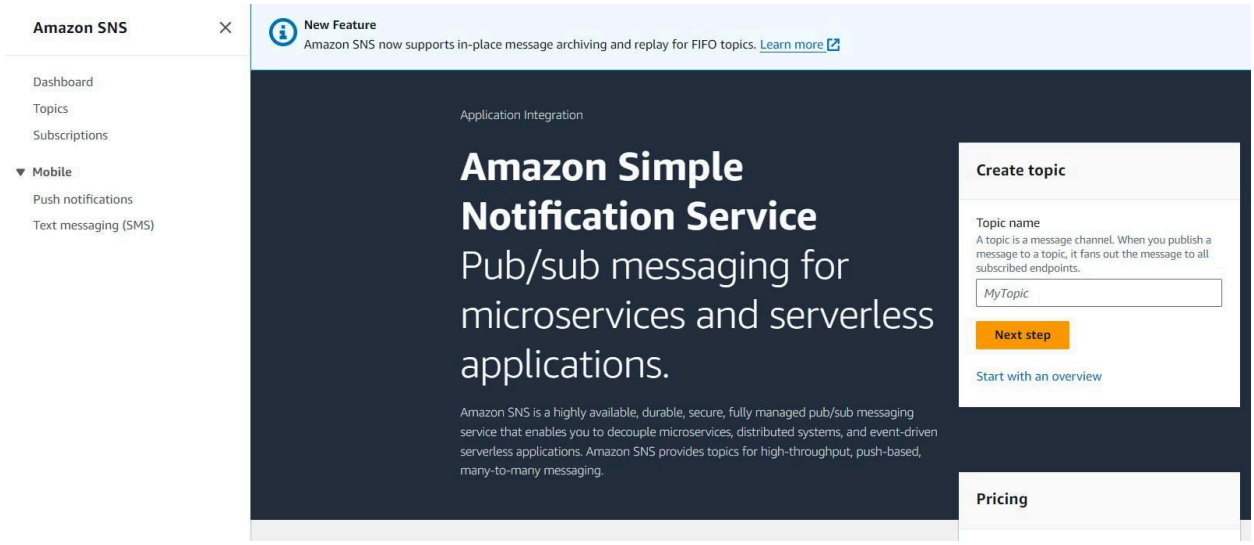
Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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

Milestone 3: SNS Notification Setup

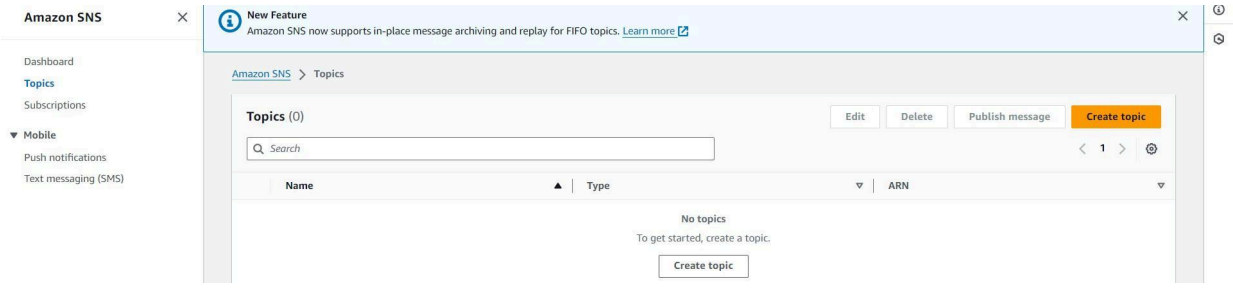
- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**

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

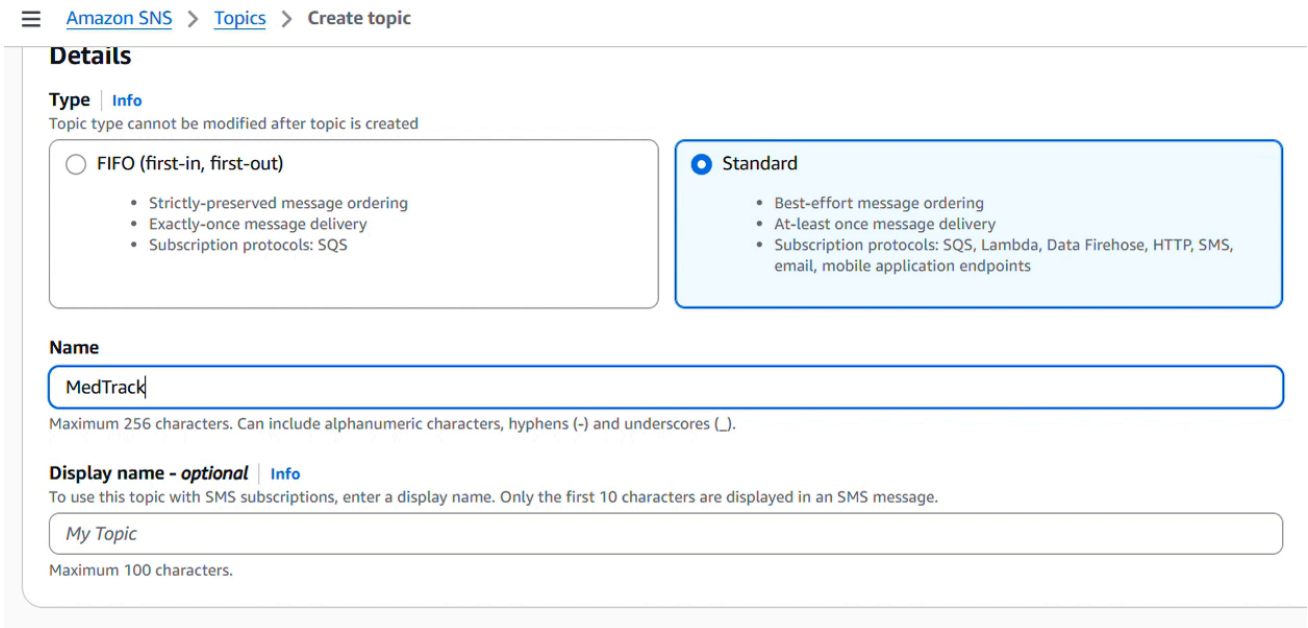


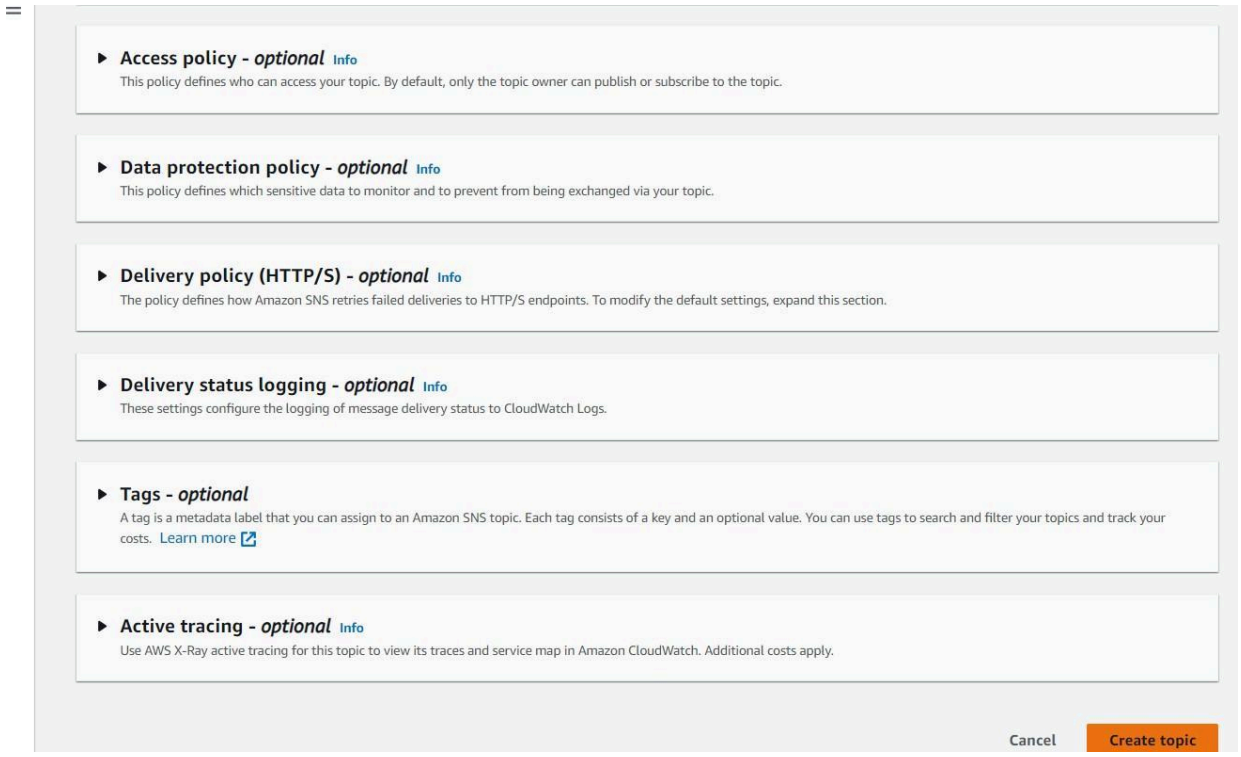


- Click on **Create Topic** and choose a name for the topic.

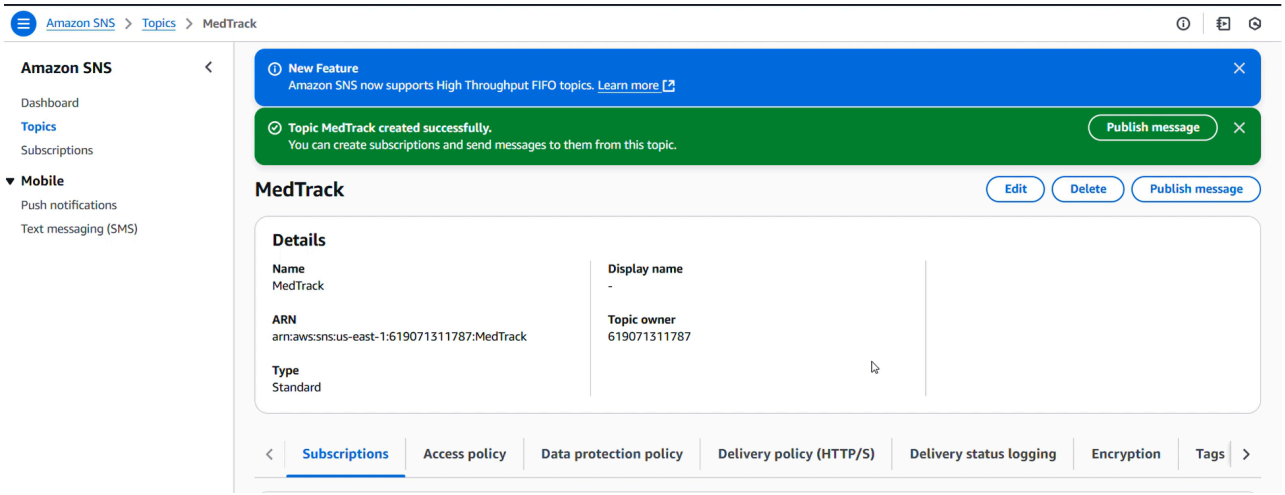


- Choose Standard type for general notification use cases and Click on Create Topic.


Amazon SNS > Topics > 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
Standard
Best-effort message ordering
At-least once message delivery
Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints
Name
MedTrack
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).
Display name - optional | Info
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.
My Topic
Maximum 100 characters.

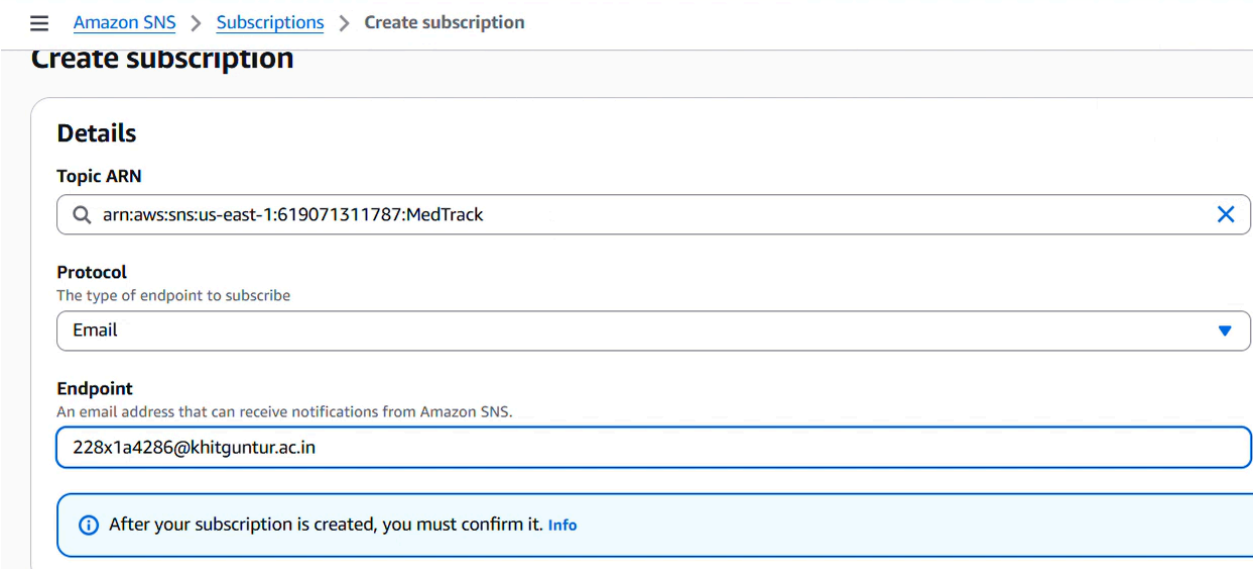


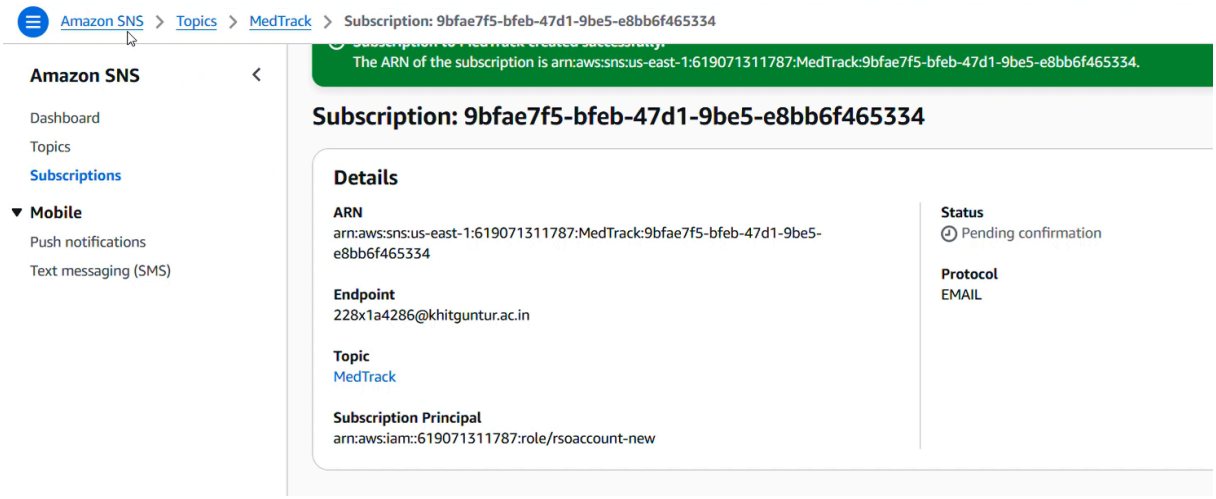
- Configure the SNS topic and note down the **Topic ARN**.



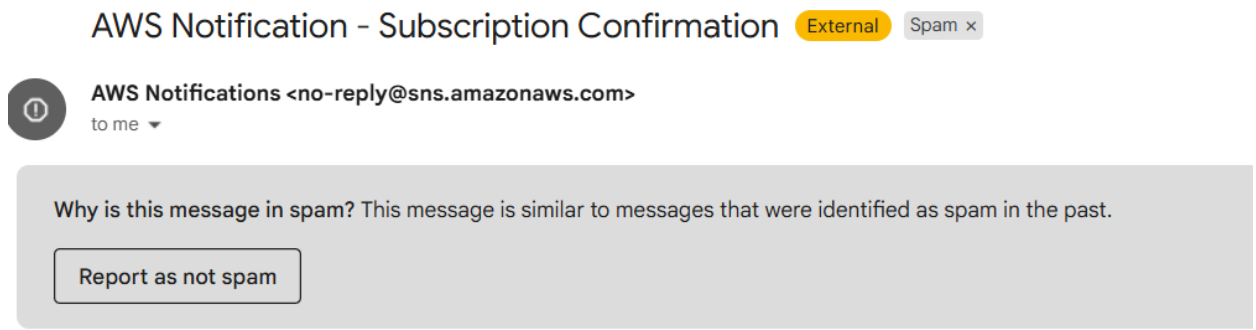
- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

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

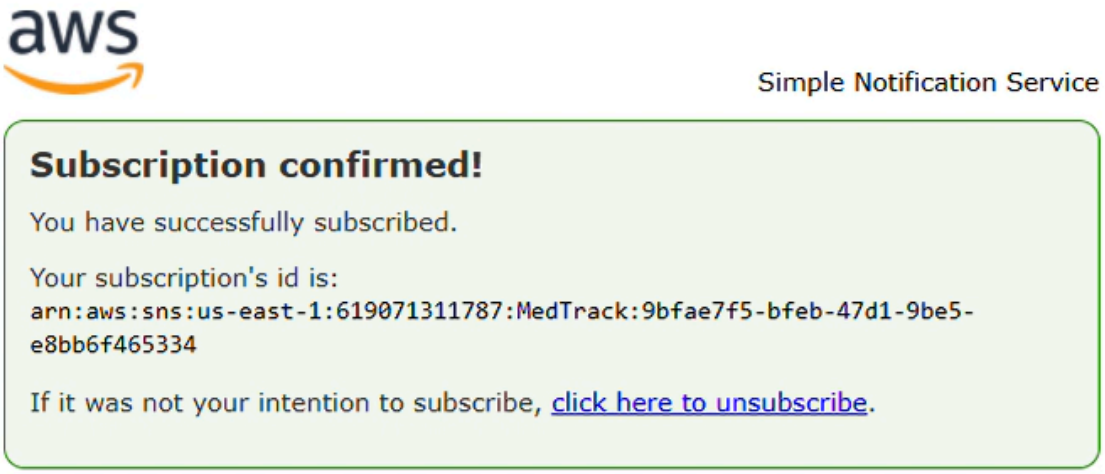




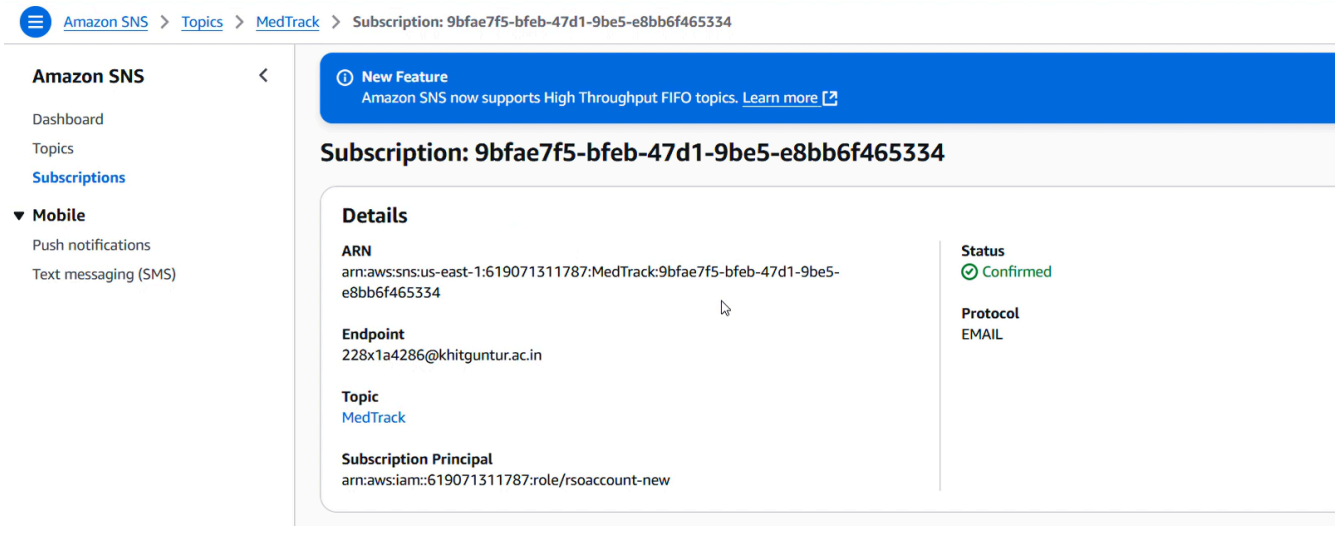
- After subscription request for the mail confirmation



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

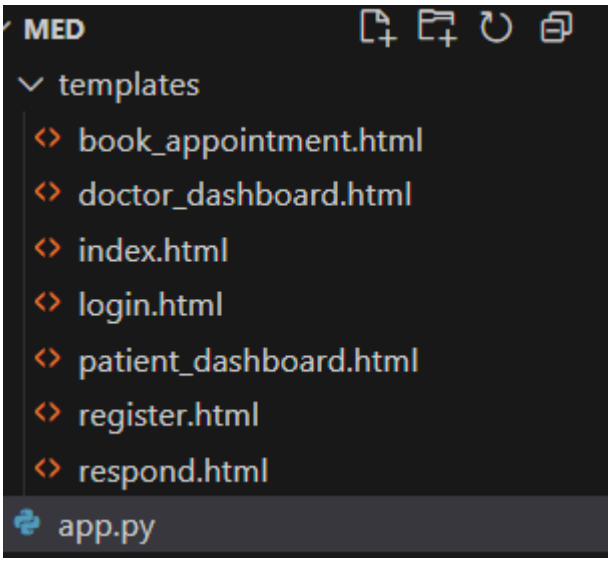


- Successfully done with the SNS mail subscription and setup, now store the ARN link.



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 and a templates/ directory containing all required HTML pages like home, login, register, subject-specific pages (e.g., index..html, login.html), and utility pages (e.g., respond.html).

Description of the code :

- **Flask App Initialization**

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
import boto3
from boto3.dynamodb.conditions import Key
import os
import uuid
from datetime import datetime
import smtplib
from email.mime.text import MIMEText
```

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:**

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 Configuration
# Replace with your SNS topic ARN
SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:123456789012:MedTrackNotifications'

# Email (SMTP) Configuration
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'your_email@gmail.com'
app.config['MAIL_PASSWORD'] = 'your_app_password'

mail = Mail(app)
```

Description: Configure **SNS** to send notifications when a book request is submitted. Paste your stored ARN link in the sns_topic_arn space, along with the region_name where the SNS topic is created. Also, specify the chosen email service in SMTP_SERVER (e.g., Gmail, Yahoo, etc.) and enter the subscribed email in the SENDER_EMAIL section. Create an 'App password' for the email ID and store it in the SENDER_PASSWORD section.

- **Routes for Web Pages**

  - **Home Route:**

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

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

- **Register Route:**

```
# Register
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        data = request.form
        role = data['role']
        item = {
            'username': data['username'],
            'password': data['password'],
            'role': role,
            'specialization': data.get('specialization', ''),
            'experience': data.get('experience', '')
        }

        try:
            users_table.put_item(Item=item, ConditionExpression='attribute_not_exists(username)')
            flash(f'Registration successful as {role.capitalize()}! Please login.', 'success')
            return redirect('/login')
        except:
            flash('Username already exists. Please choose another.', 'danger')
            return redirect('/register')

    return render_template('register.html')
```

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

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

```
# Login
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        data = request.form
        response = users_table.get_item(Key={'username': data['username']})
        user = response.get('Item')

        if user and user['password'] == data['password']:
            session['username'] = user['username']
            session['role'] = user['role']
            return redirect(f"/{user['role']}")
        else:
            flash('Invalid username or password', 'danger')
            return redirect('/login')

    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

- **Patient Dashboard Route:**

```python
# Patient Dashboard
@app.route('/patient')
def patient_dashboard():
    if session.get('role') != 'patient':
        return redirect('/login')

    response = appointments_table.scan(FilterExpression="patient_name = :p", ExpressionAttributeValues={":p": session['username']})
    appointments = response['Items']

    doctors = users_table.scan(FilterExpression="role = :r", ExpressionAttributeValues={":r": 'doctor'})['Items']

    return render_template('patient_dashboard.html', username=session['username'], appointments=appointments, doctors=doctors)
```

Description: Loads the patient dashboard after login. Patients can view their upcoming appointments, cancel them, and browse available doctors to book new appointments.

- **Book Appointment Route:**

```python
# Book Appointment
@app.route('/book/<doctor_name>/<specialization>', methods=['GET', 'POST'])
def book_appointment(doctor_name, specialization):
    if session.get('role') != 'patient':
        return redirect('/login')

    if request.method == 'POST':
        appointment_id = str(uuid.uuid4())
        item = {
            'id': appointment_id,
            'patient_name': session['username'],
            'doctor_name': doctor_name,
            'specialization': specialization,
            'date': request.form['date'],
            'time': request.form['time'],
            'symptoms': request.form['symptoms'],
            'status': 'Pending',
            'response': '',
            'email': request.form['email']
        }

        appointments_table.put_item(Item=item)
```

Description: Displays the appointment booking form for the selected doctor. Patients provide date, time, symptoms, and email to book an appointment and receive notification.

- **Doctor Dashboard Route:**

```
# Doctor Dashboard
@app.route('/doctor')
def doctor_dashboard():
    if session.get('role') != 'doctor':
        return redirect('/login')

    username = session['username']
    doctor_info = users_table.get_item(Key={'username': username})['Item']
    specialization = doctor_info['specialization']

    appointments = appointments_table.scan(FilterExpression="doctor_name = :d", ExpressionAttributeValues={":d": username})['Items']

    total = len(appointments)
    pending = len([a for a in appointments if a['status'] == 'Pending'])
    solved = len([a for a in appointments if a['status'] == 'Solved'])

    return render_template('doctor_dashboard.html', appointments=appointments, total=total, pending=pending, solved=solved, specializatio
```

Description: Loads the doctor dashboard after login. Doctors can view pending and completed appointments based on their specialization, and respond to them with prescriptions.

- **Respond to Appointment Route:**

```
# Respond to Appointment
@app.route('/respond/<string:id>', methods=['GET', 'POST'])
def respond(id):
    if session.get('role') != 'doctor':
        return redirect('/login')

    if request.method == 'POST':
        response_text = request.form['response']
        appointments_table.update_item(
            Key={'id': id},
            UpdateExpression="set response=:r, status='Solved'",
            ExpressionAttributeValues={':r': response_text}
        )
```

Description: Allows doctors to respond to a specific appointment by submitting their diagnosis or treatment notes. This also updates the appointment status to "Solved".

- **Cancel Appointment Route:**

```
# Cancel Appointment
@app.route('/cancel/<string:id>', methods=['POST'])
def cancel_appointment(id):
    if session.get('role') != 'patient':
        return redirect('/login')

    appointments_table.delete_item(Key={'id': id})
    flash('Appointment cancelled successfully.', 'success')
    return redirect('/patient')
```

Description: Allows patients to cancel a specific appointment they previously booked. It removes the appointment record from the system.

- **Logout Route:**

```
# Logout
@app.route('/logout')
def logout():
    session.clear()
    flash('You have been logged out.', 'info')
    return redirect('/login')
```

Description: Clears the current session and logs out the user, redirecting them to the login page with a confirmation flash message.

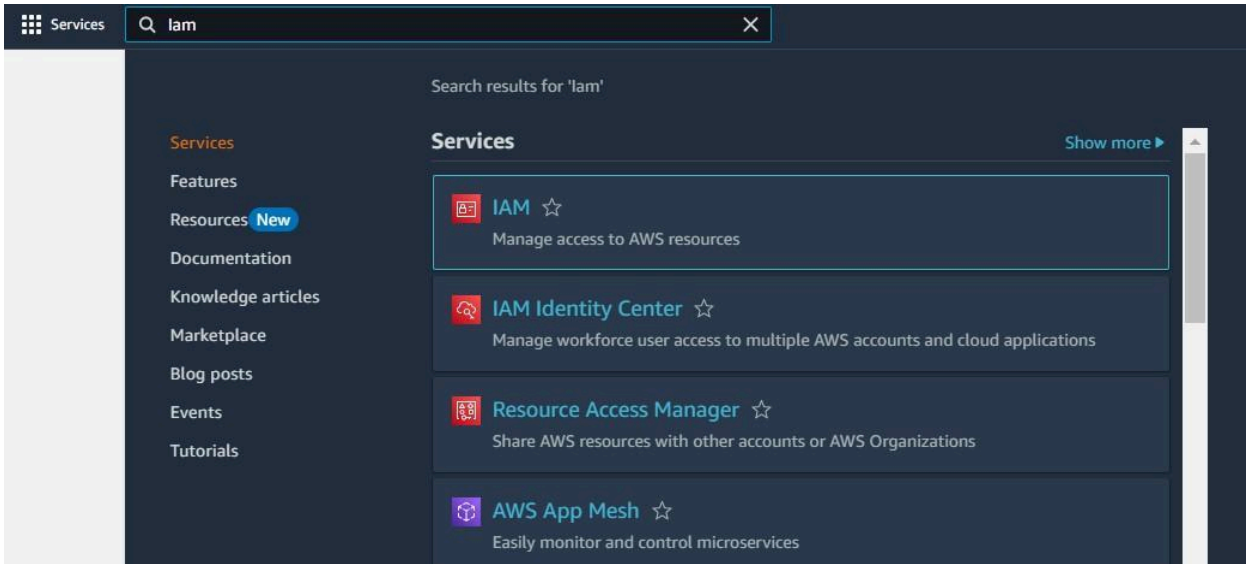
- **Deployment Code:**

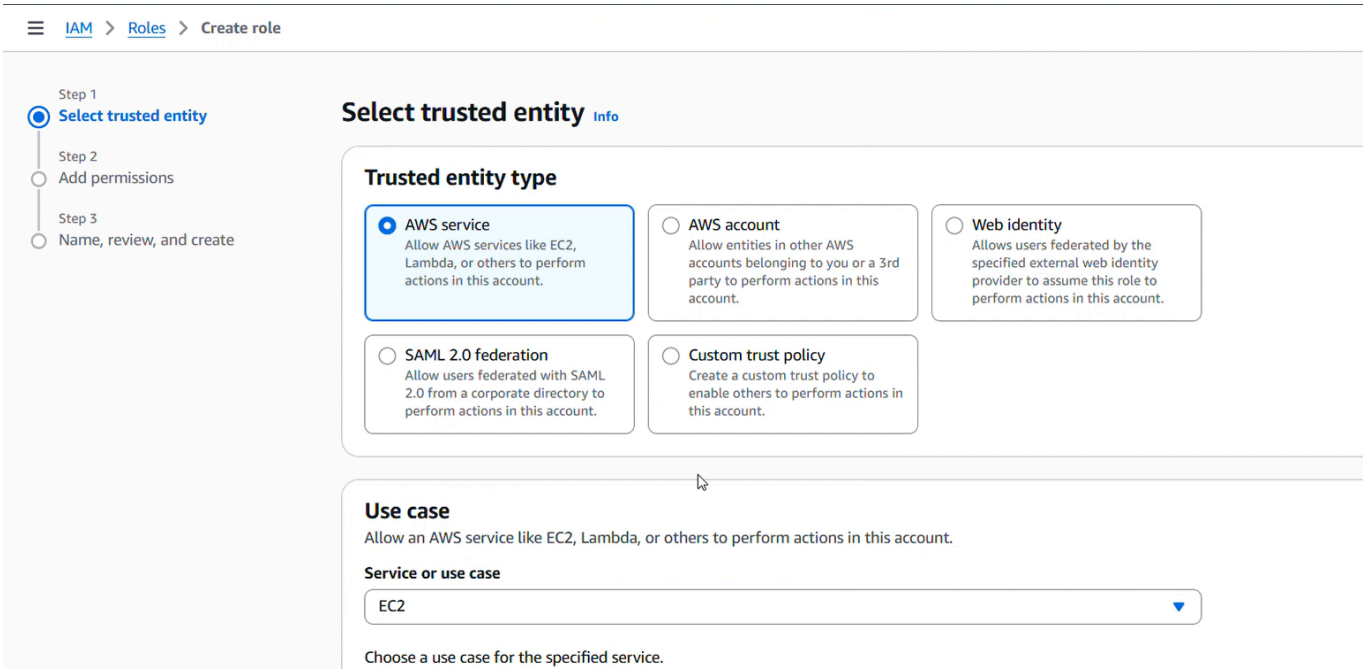
```
# Run the application
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Description: Starts the Flask app in debug mode on port 5000, accessible from any network interface (0.0.0.0). This setup is ideal for development on a remote server or local network.

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.

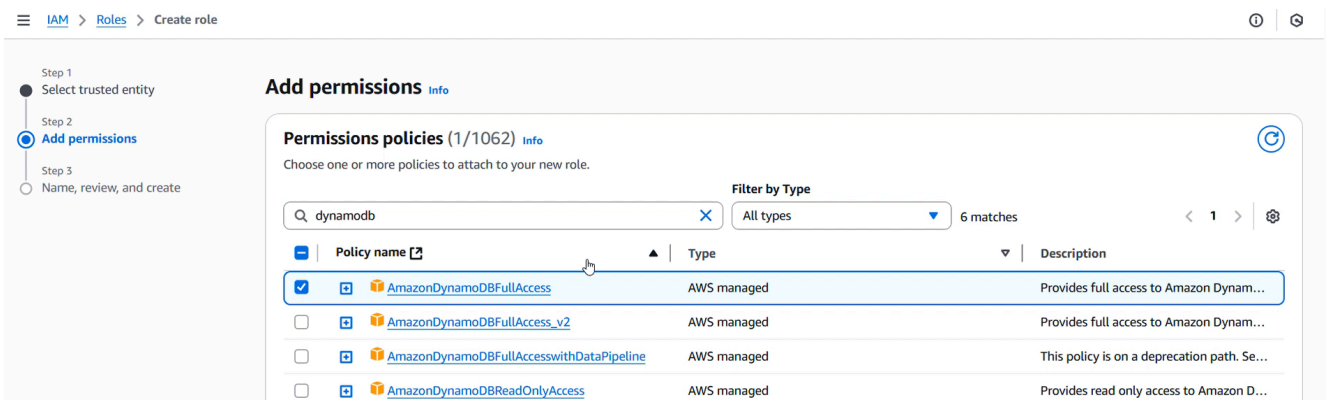


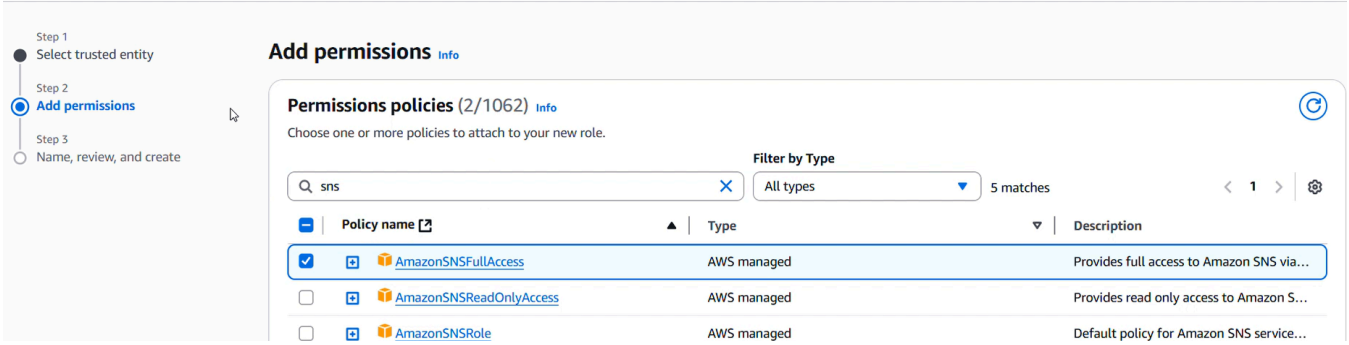


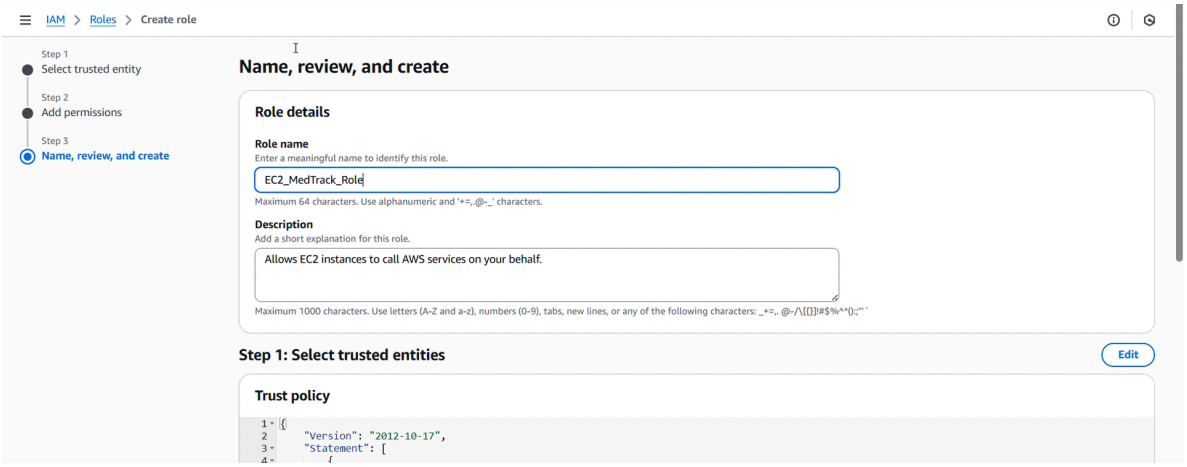
- **Activity 5.2: Attach Policies.**

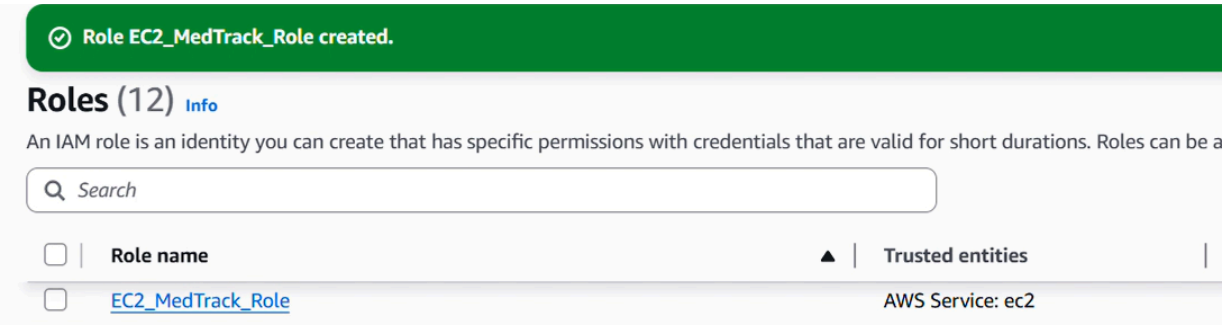
  Attach the following policies to the role:

  - AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
  - AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.



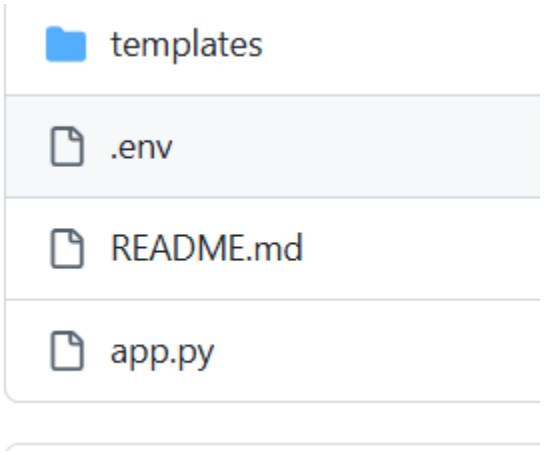

Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (2/1062) Info
Choose one or more policies to attach to your new role.
Filter by Type
sns
All types
5 matches
1
Policy name
Type
Description
AmazonSNSFullAccess
AWS managed
Provides full access to Amazon SNS via...
AmazonSNSReadOnlyAccess
AWS managed
Provides read only access to Amazon S...
AmazonSNSRole
AWS managed
Default policy for Amazon SNS service...

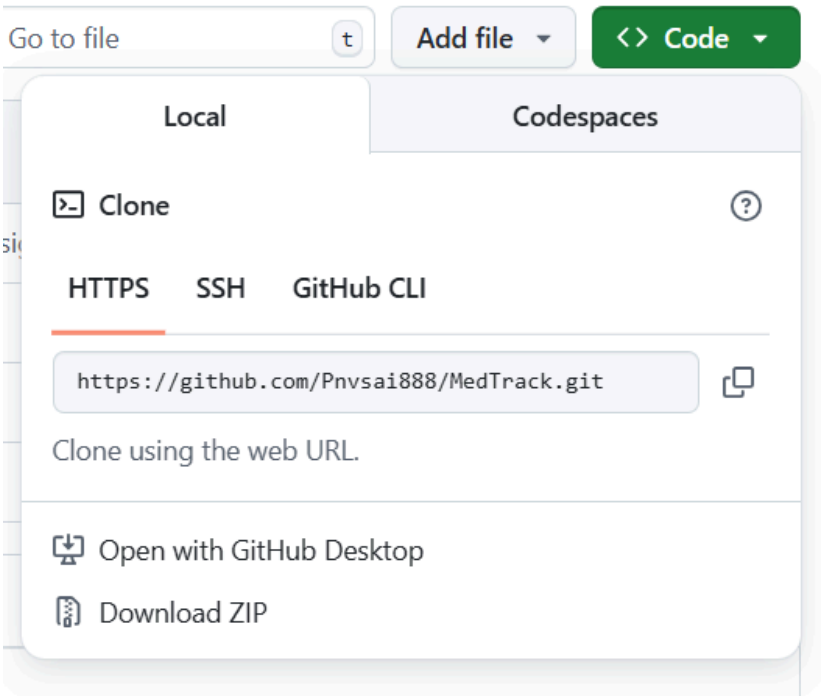

IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Name, review, and create
Role details
Role name
Enter a meaningful name to identify this role.
EC2_MedTrack_Role
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.
Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*();"'`
Step 1: Select trusted entities
Edit
Trust policy
{
"Version": "2012-10-17",
"Statement": [



Milestone 6: EC2 Instance Setup

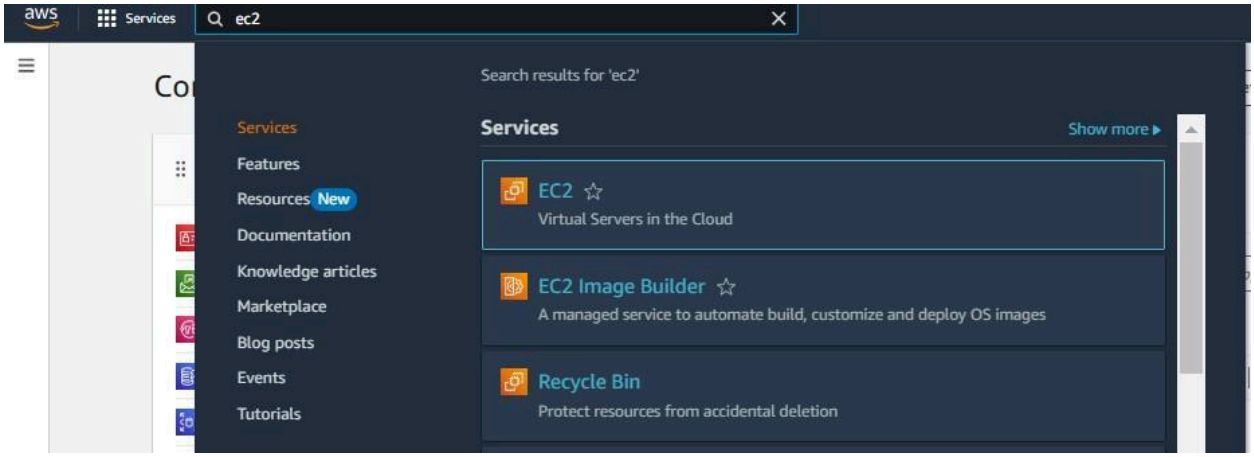
- **Note: Load your Flask app and Html files into GitHub repository.**



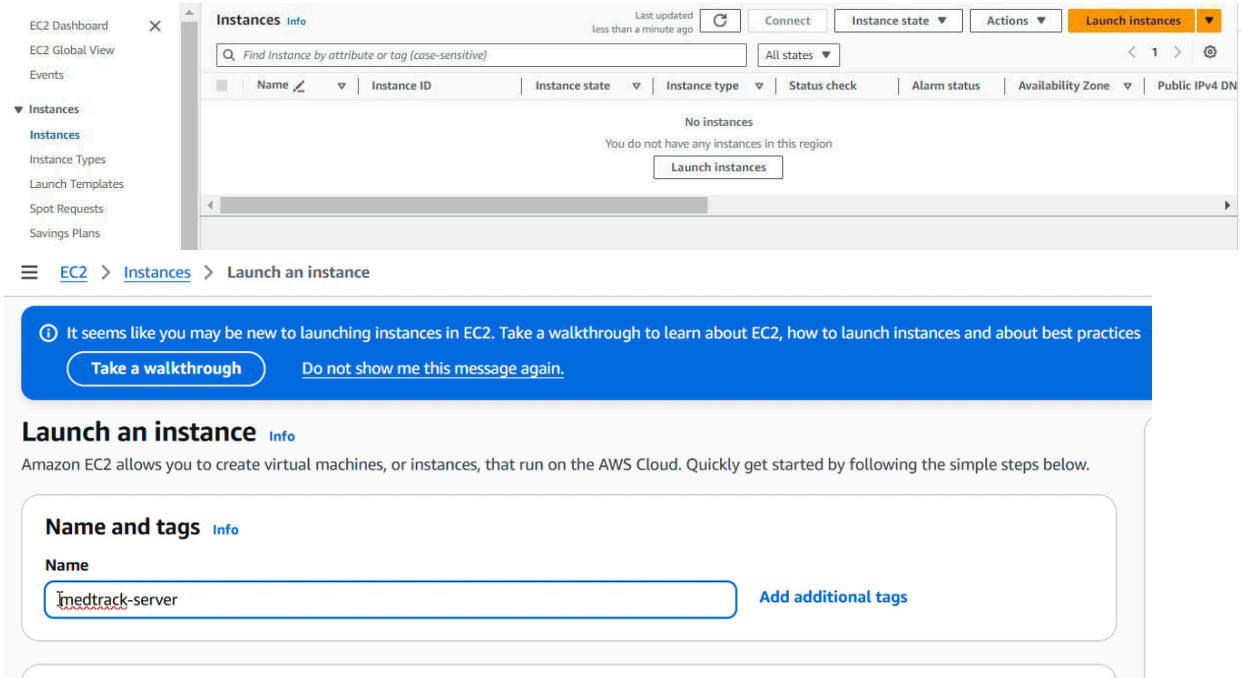


- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

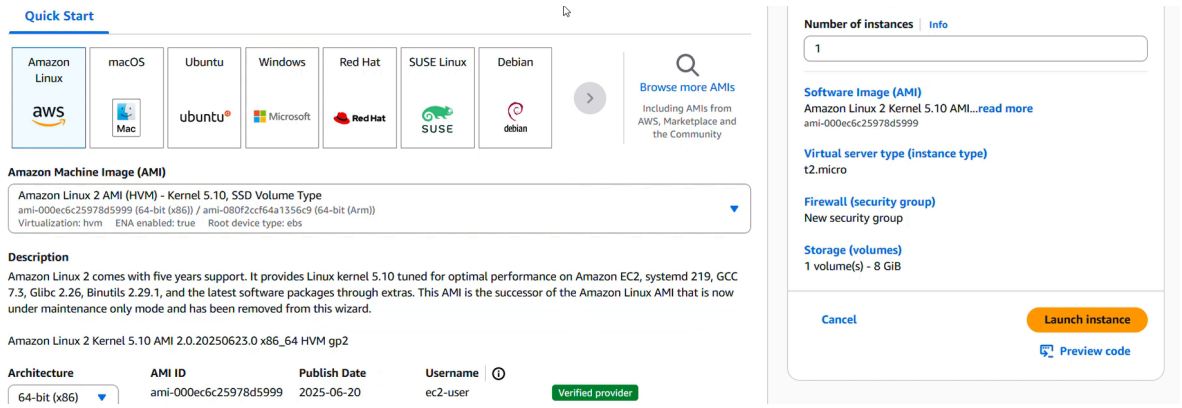
    - **Launch EC2 Instance**
        - In the AWS Console, navigate to EC2 and launch a new instance.



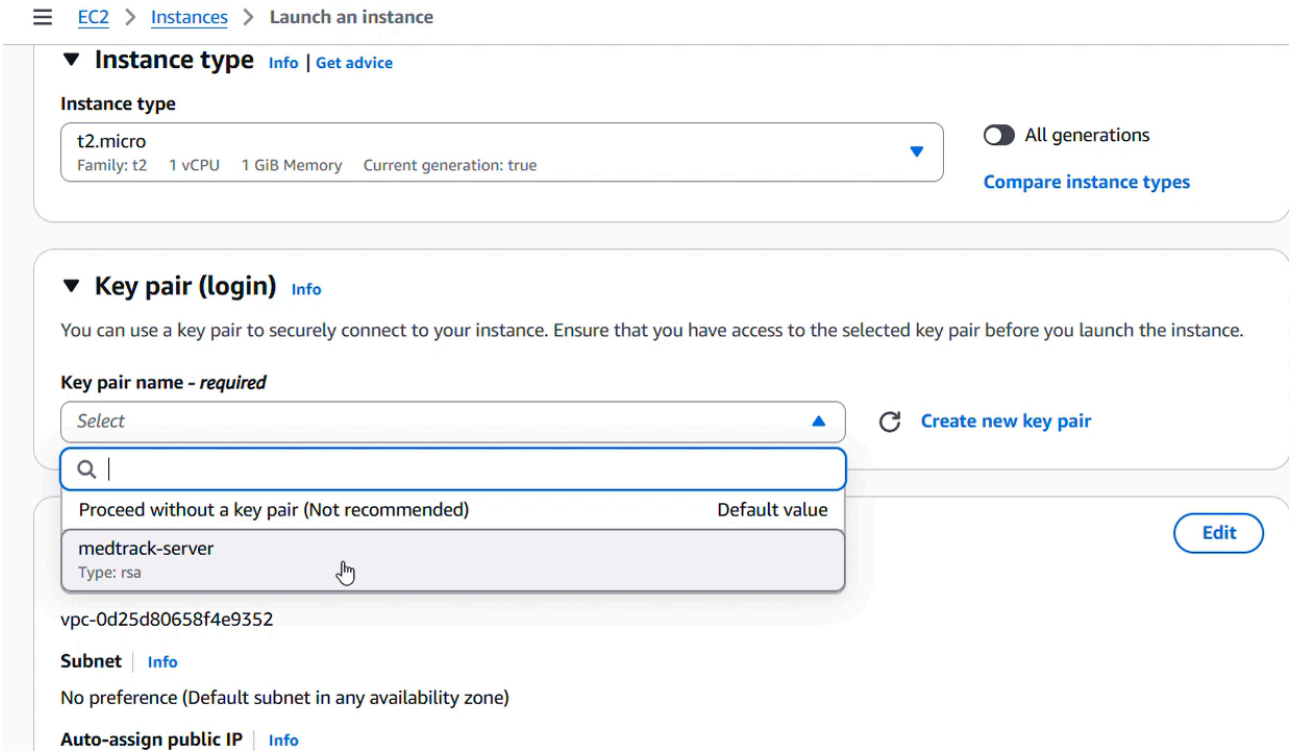
- Click on Launch instance to launch EC2 instance



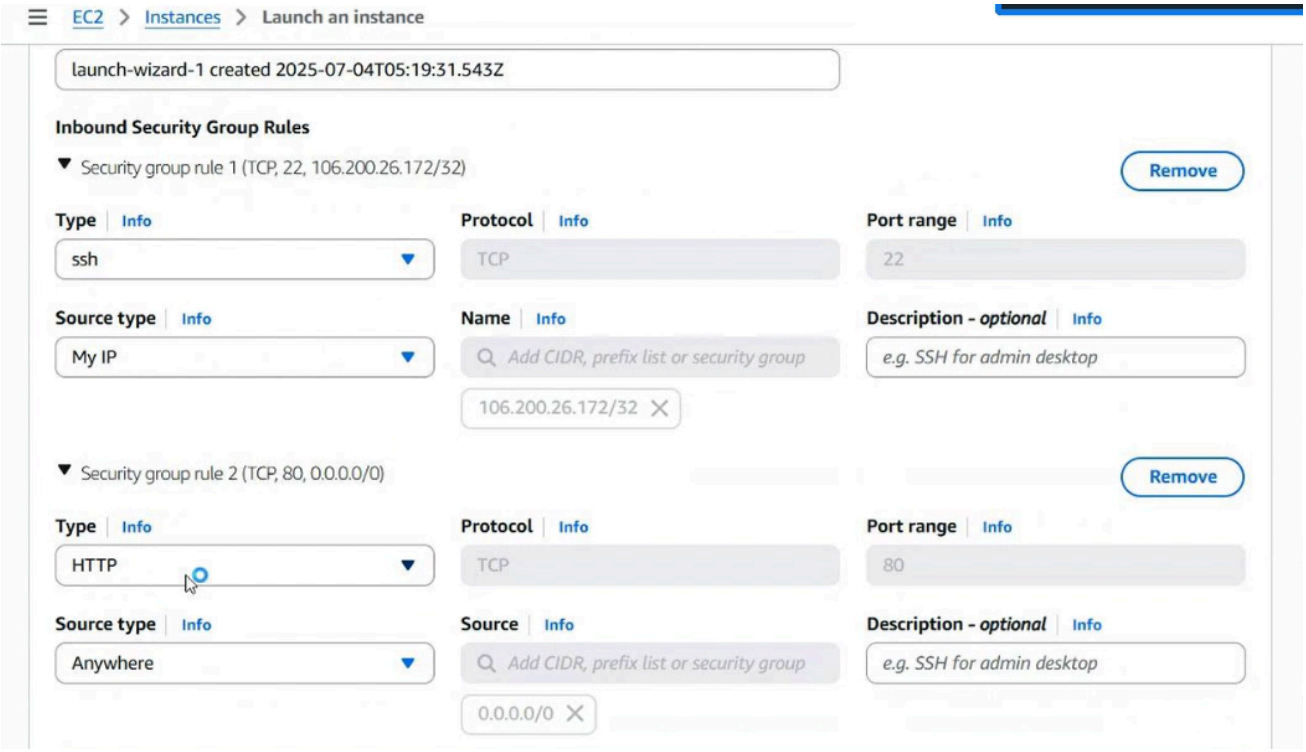
- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

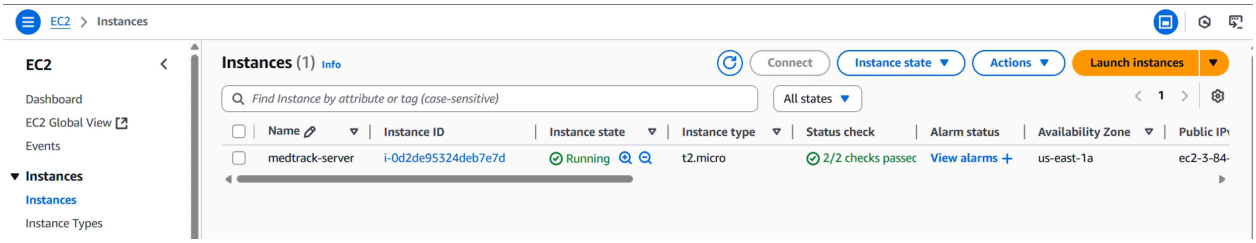


- Create and download the key pair for Server access.

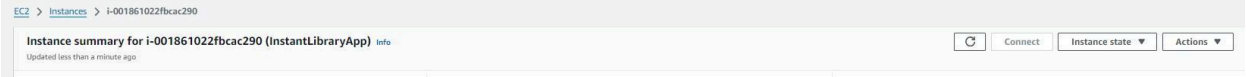


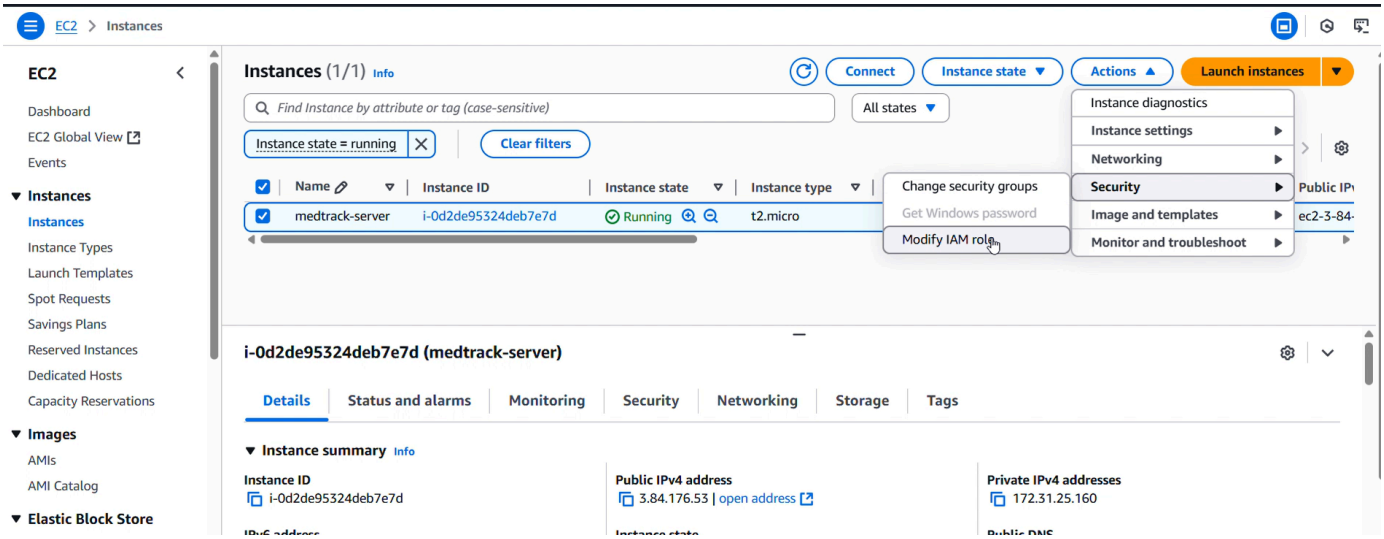
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**

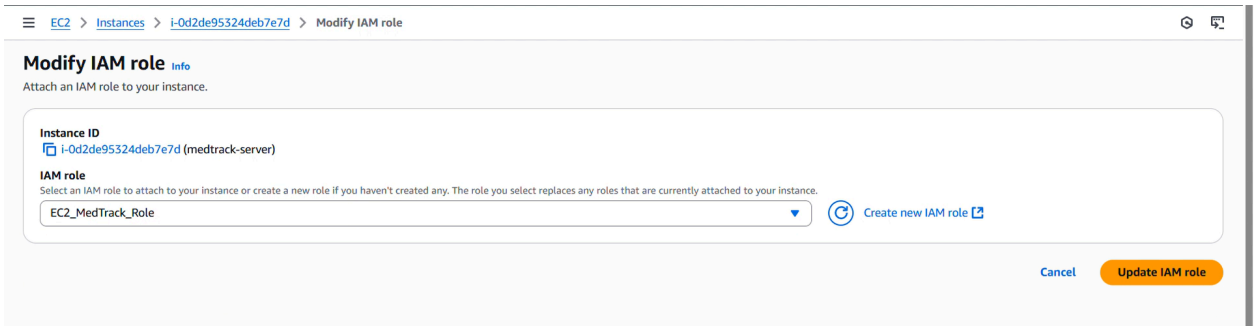




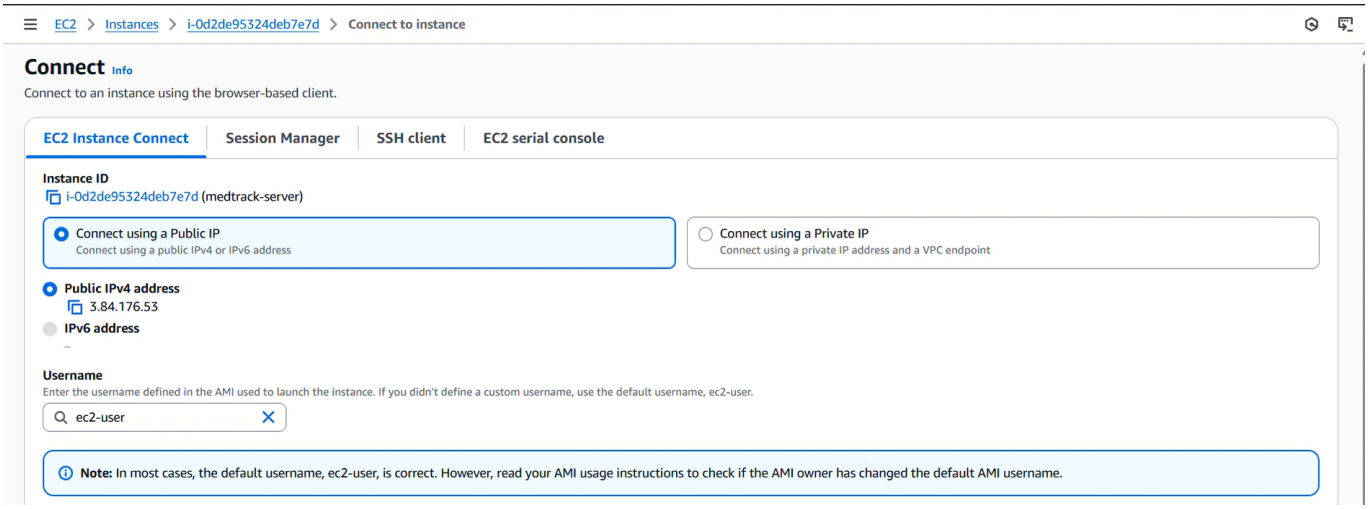
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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

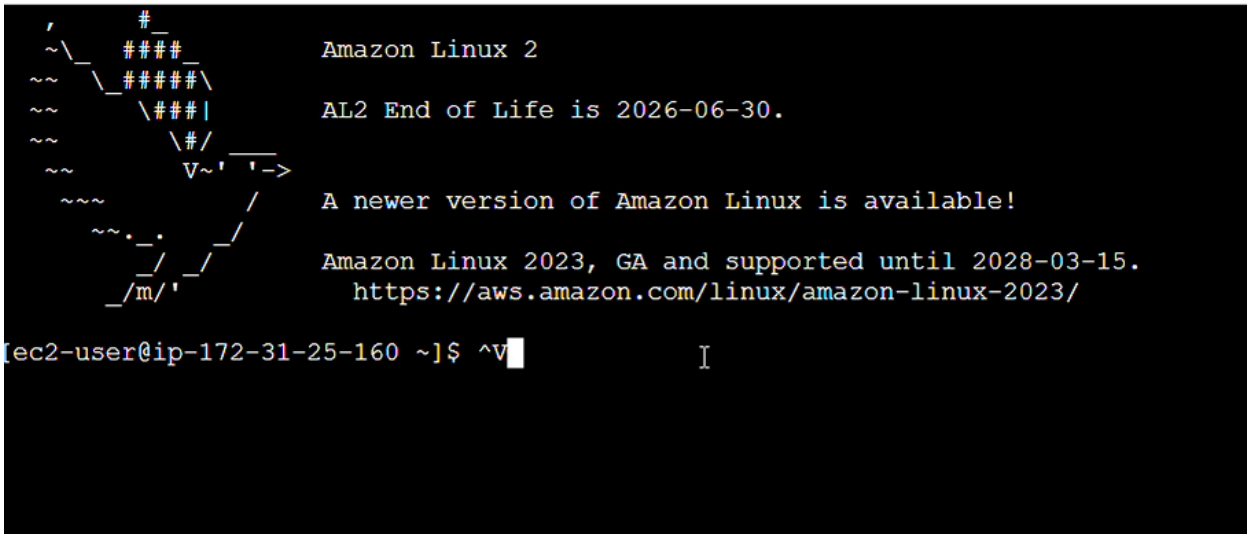






- Now connect the EC2 with the files


EC2 > Instances > i-0d2de95324deb7e7d > Connect to instance
Connect Info
Connect to an instance using the browser-based client.
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Instance ID
i-0d2de95324deb7e7d (medtrack-server)
Connect using a Public IP
Connect using a public IPv4 or IPv6 address
Connect using a Private IP
Connect using a private IP address and a VPC endpoint
Public IPv4 address
3.84.176.53
IPv6 address
Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.
ec2-user
Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.


Amazon Linux 2
AL2 End of Life is 2026-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-25-160 ~]$ ^V

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/Pnvsai888/MedTrack.git'

- This will download your project to the EC2 instance.

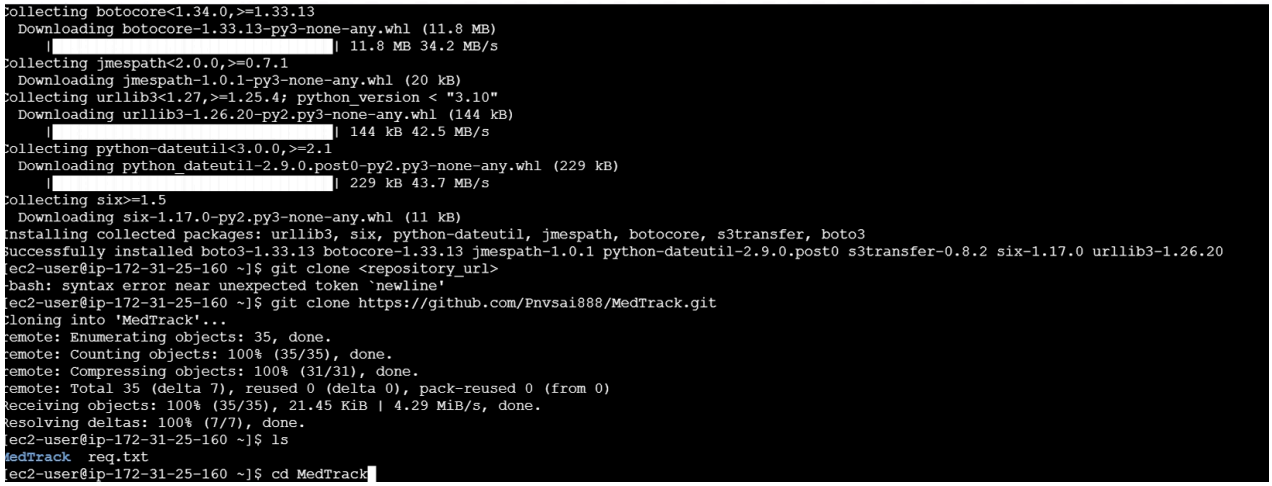
To navigate to the project directory, run the following command:

```
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=5000
```



Verify the Flask app is running:

http://3.84.176.53:5000

- Run the Flask app on the EC2 instance

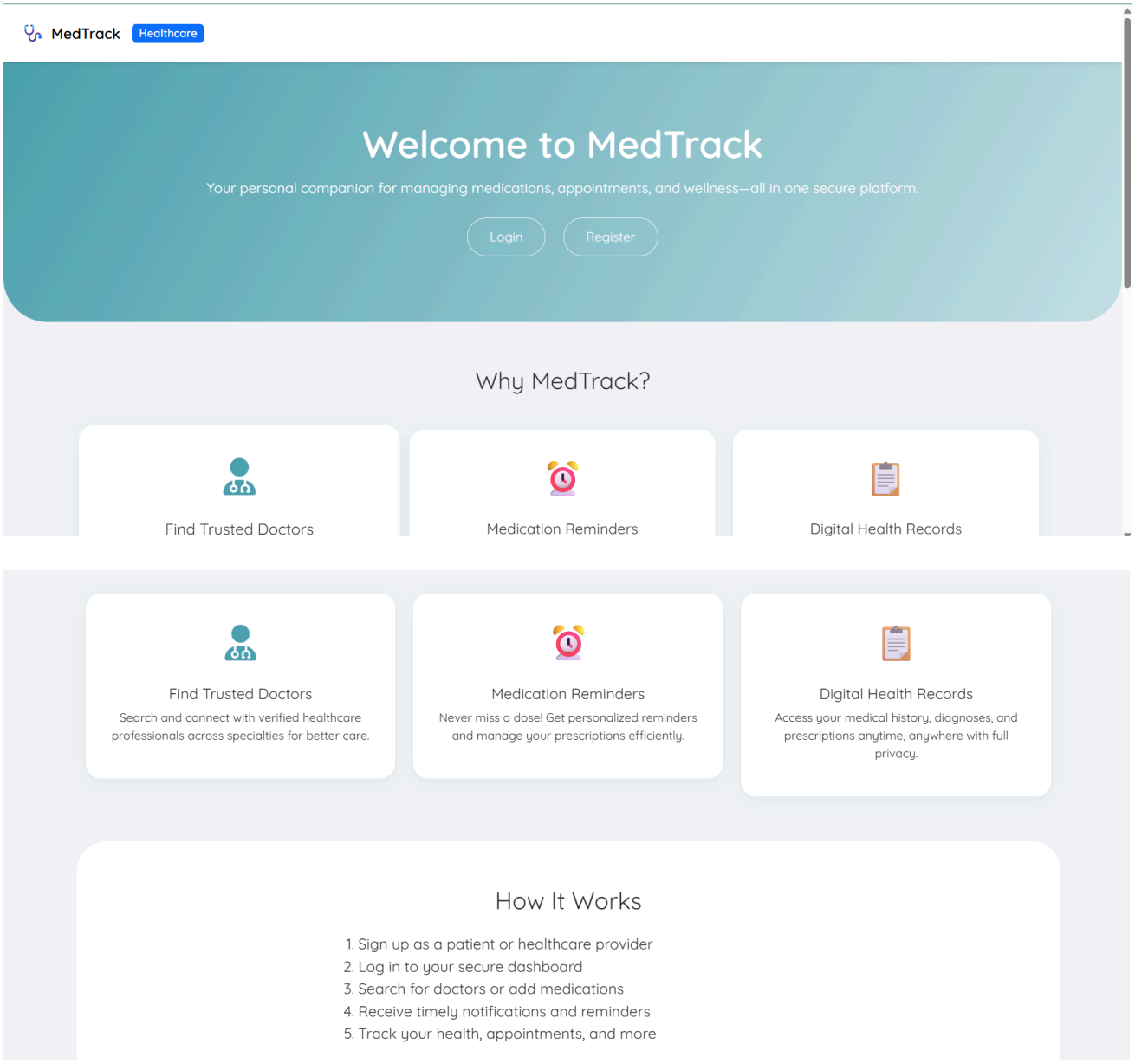
Access the website through:

PublicIPs: http://3.84.176.53:5000

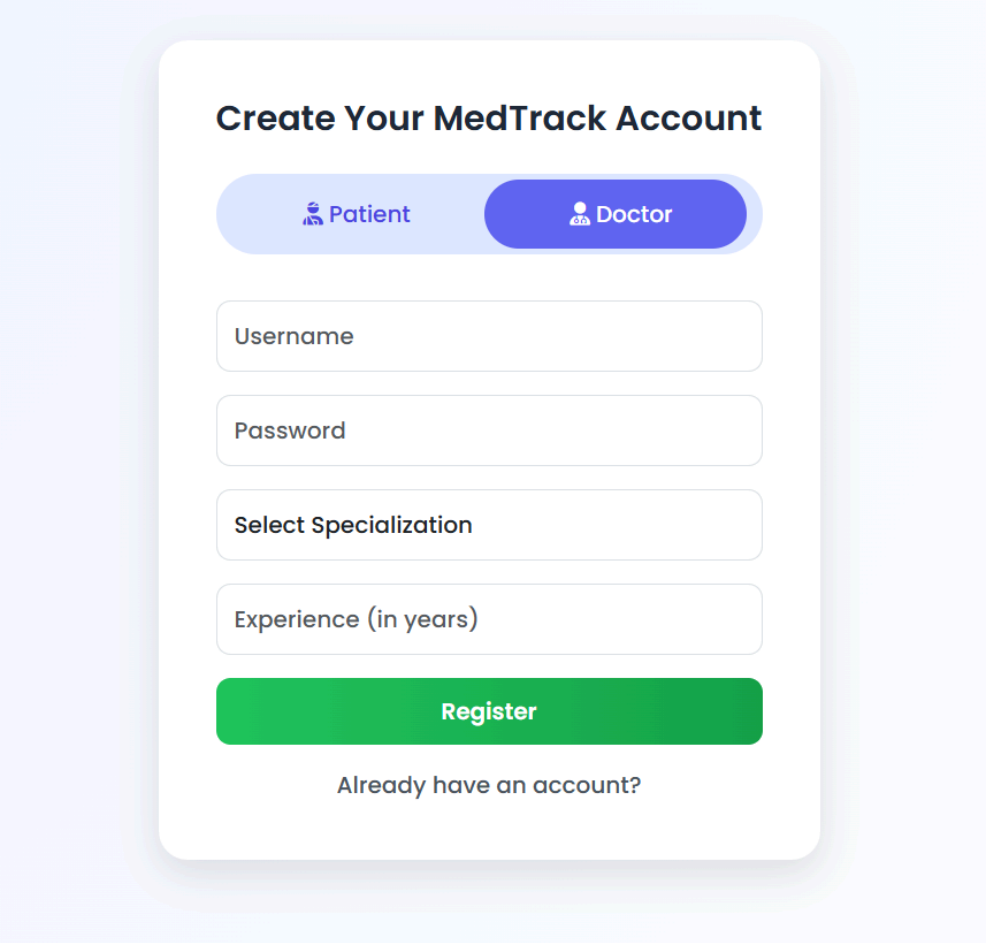
Milestone 8: Testing and Deployment

- **Activity 8.1: Conduct functional testing to verify user sign-up, login, appointment booking, prescriptionand SNS notifications.**

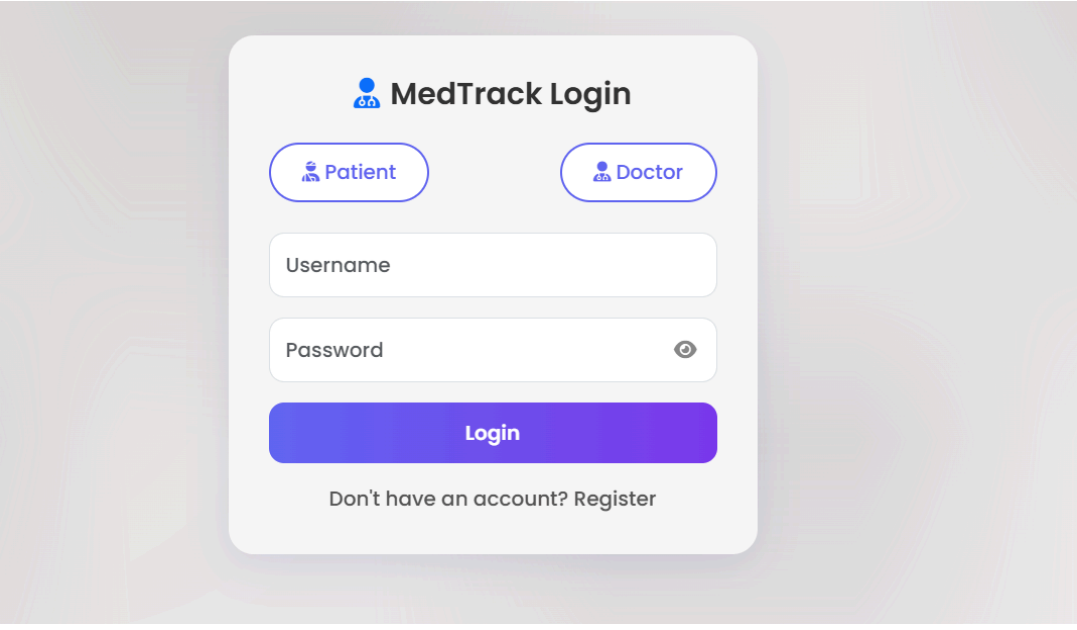
Index Page:



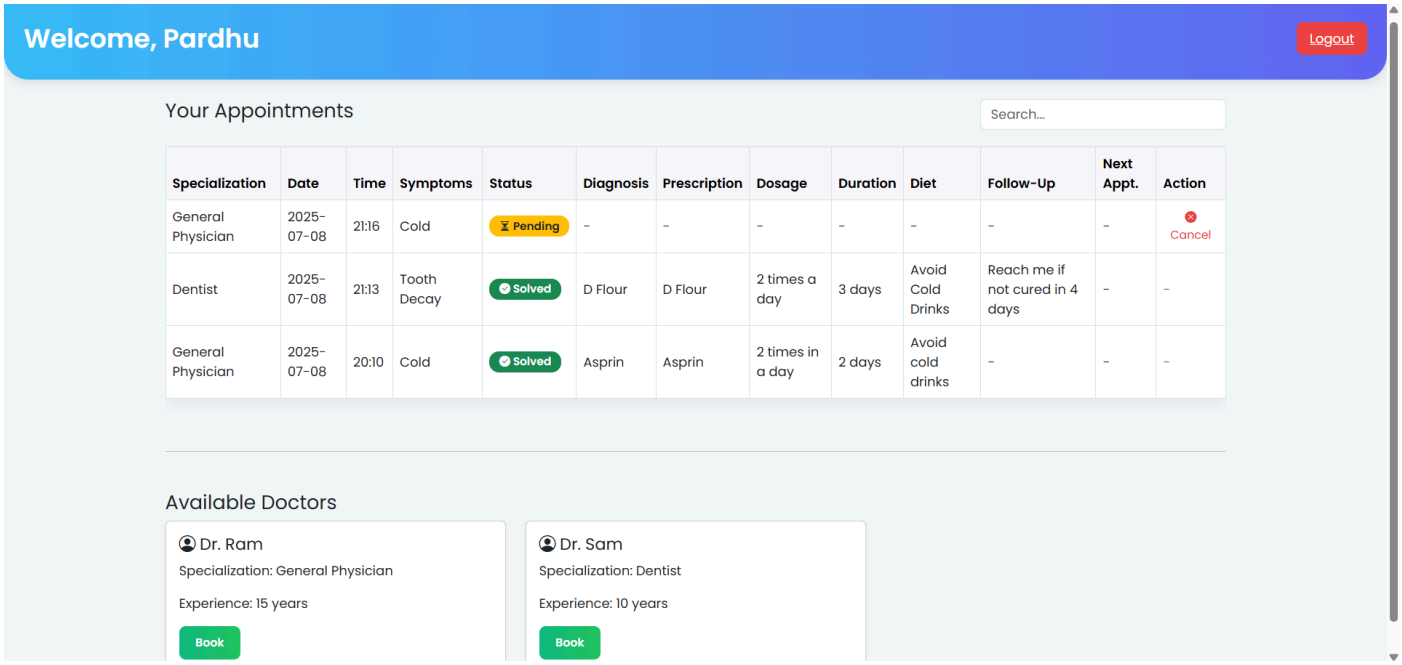
Register Page:



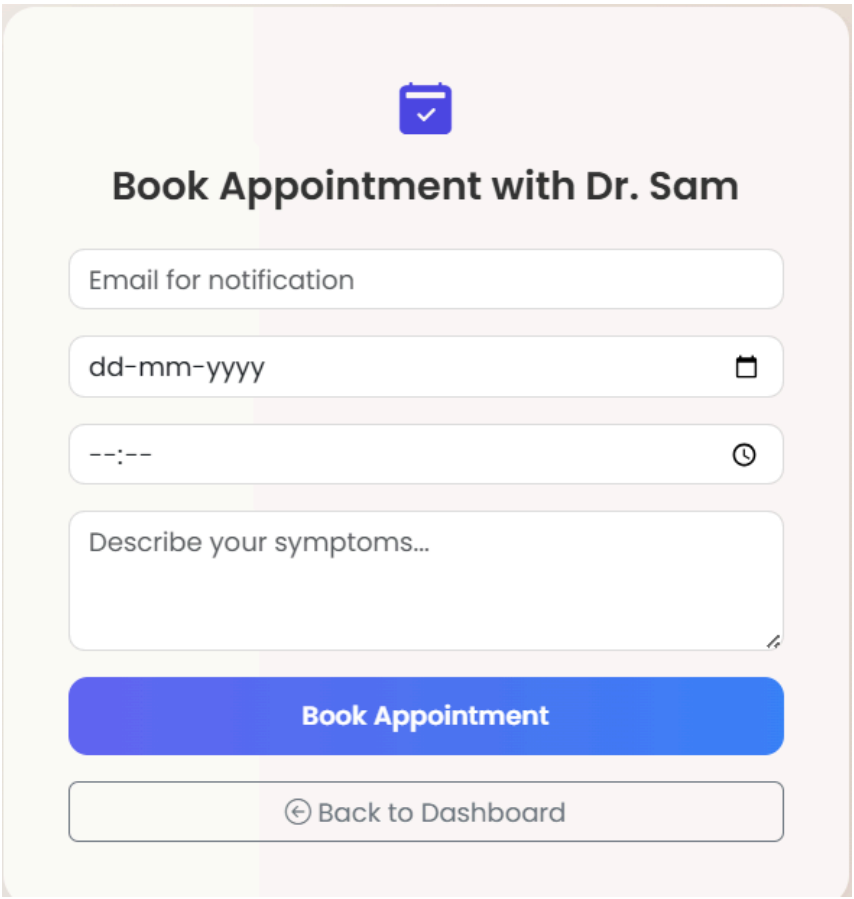
Login Page:



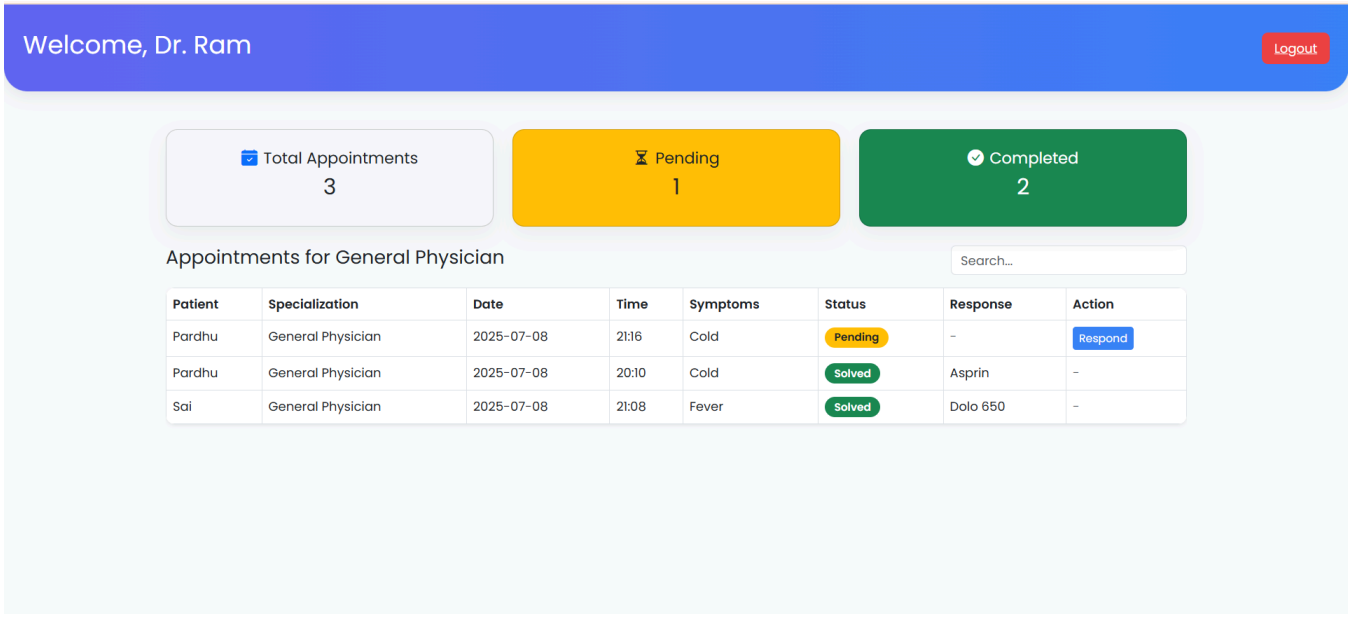
Patient DashBoard:



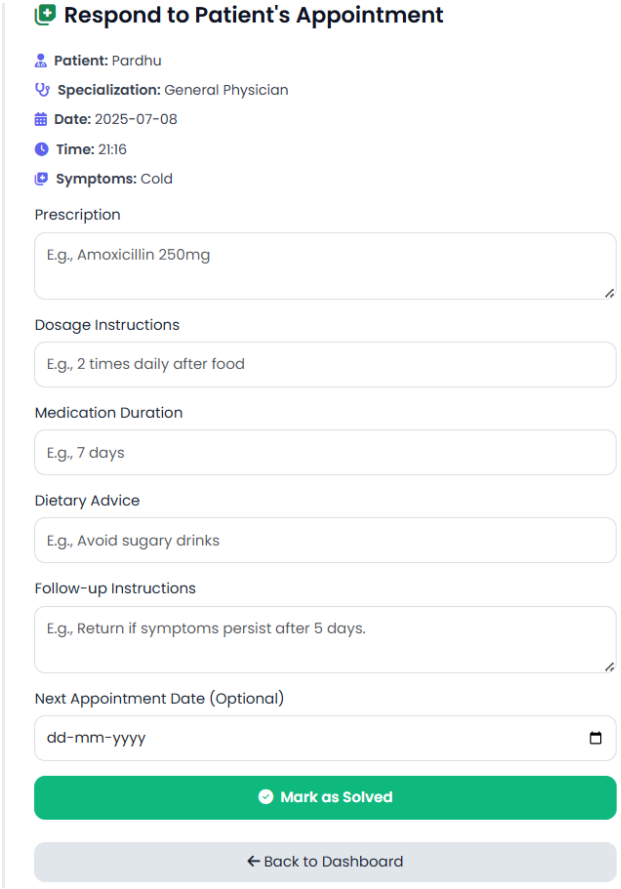
Book Appointment:



Doctor DashBoard:



Response:



Conclusion:

MedTrack represents a significant step forward in modernizing healthcare management through a cloud-powered infrastructure. By integrating AWS services such as EC2, DynamoDB, SNS, and IAM with a Flask-based backend, the platform delivers a secure, scalable, and responsive environment for both patients and doctors.

The system successfully overcomes key healthcare challenges by enabling seamless appointment scheduling, prescription tracking, and timely medication reminders — all within a unified interface. Doctors benefit from centralized tools to manage appointments, issue prescriptions, and oversee patient progress, ultimately enhancing the quality of care.

Extensive testing has validated the platform's core features, ensuring robust performance in user authentication, data handling, and real-time notifications.

In essence, MedTrack stands as a powerful example of how cloud technologies can transform healthcare delivery. It not only streamlines doctor-patient interactions but also empowers users to take control of their health in a secure and efficient manner.