

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 submitting diagnoses. The system leverages Flask for backend development, AWS EC2 for hosting, and DynamoDB for data management. It supports real-time notifications through AWS SNS and secure access control via AWS IAM, ensuring both accessibility and data integrity. MedTrack is designed to improve healthcare accessibility, efficiency, and real-time communication.

Scenarios

Scenario 1: Efficient Appointment Booking System for Patients

AWS EC2 supports multiple concurrent users, allowing patients to log in and book appointments. Flask handles backend operations and integrates with DynamoDB to store real-time data efficiently.

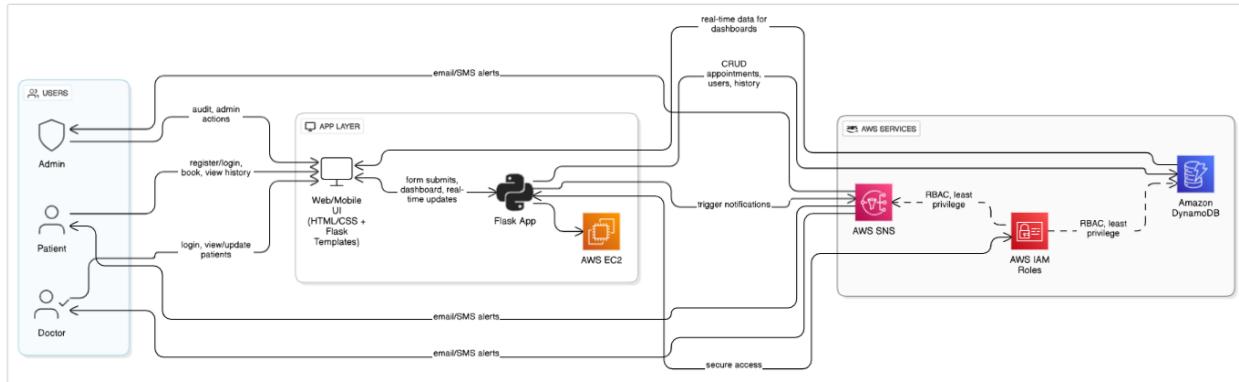
Scenario 2: Secure User Management with IAM

MedTrack uses IAM roles for secure, role-based access control. Patients and doctors receive permissions specific to their roles, ensuring secure access to sensitive data.

Scenario 3: Easy Access to Medical History and Resources

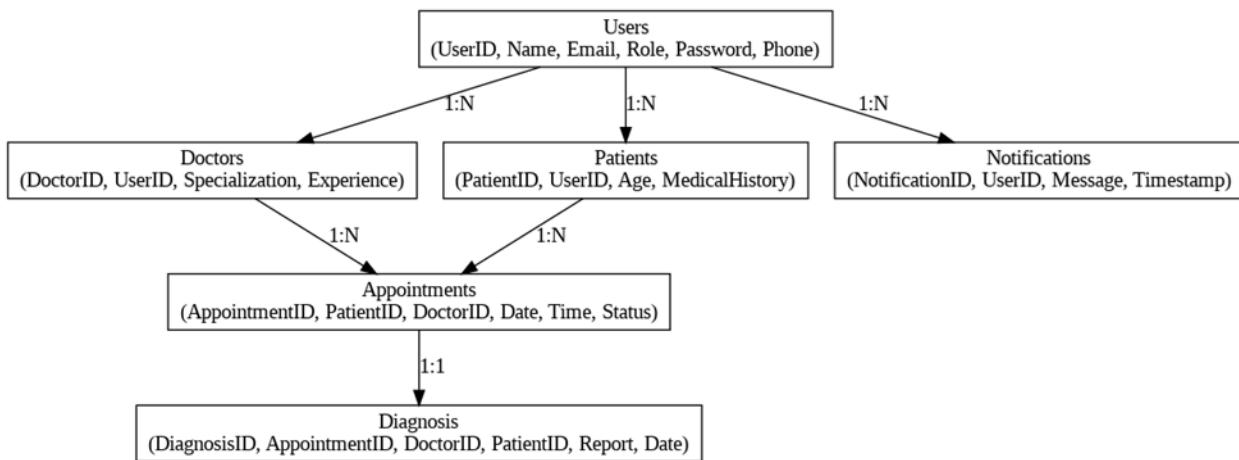
Doctors can retrieve patient records and update diagnoses in real time. Flask and DynamoDB integration enables high-speed access and data updates, ensuring seamless operation during peak usage.

AWS Architecture



- **Flask (Python Framework):** Handles routing and backend logic.
- **Amazon EC2:** Hosts the Flask application.
- **Amazon DynamoDB:** Stores patient data, appointments, and records.
- **AWS SNS:** Sends real-time alerts and appointment confirmations.
- **AWS IAM:** Controls user access and permissions securely.

Entity Relationship (ER) Diagram



The ER diagram illustrates entities such as Users (patients/doctors), Appointments, and their relationships. Key attributes include user ID, name, email, appointment ID, doctor ID, date, and status. This structure supports efficient query processing and normalization.

Pre-requisites

- AWS Account Setup:
<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
- IAM Overview:
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
- EC2 Tutorial:
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
- DynamoDB Introduction:
<https://docs.aws.amazon.com/amazondynamodb/Introduction.html>
- SNS Documentation:
<https://docs.aws.amazon.com/sns/latest/dg/welcome.html>
- Git Documentation:
<https://git-scm.com/doc>
- VS Code:
<https://code.visualstudio.com/download>

Project Workflow

Milestone 1: Web Application Development and Setup

- Set up the Flask app with routing and templates.
- Use local Python lists/dictionaries for initial testing.
- Integrate AWS services (DynamoDB, SNS) using boto3.

Milestone 2: AWS Account Setup

- Access AWS via Troven Labs.
- Avoid personal AWS account usage to prevent billing issues.

Milestone 3: DynamoDB Database Creation and Setup

- Create a Users table (Primary key: Email).
- Create an Appointments table (Primary key: appointment_id).

Milestone 4: SNS Notification Setup

- Create an SNS topic.
- Subscribe to users/admin via email.
- Confirm subscriptions and note Topic ARN.

Milestone 5: IAM Role Setup

- Create IAM Role (e.g., flask dynamodb sns).
- Attach policies: AmazonDynamoDBFullAccess, AmazonSNSFullAccess.

Milestone 6: EC2 Instance Setup

- Launch EC2 instance (Amazon Linux 2/Ubuntu, t2.micro).
- Assign IAM Role and key pair.
- Configure security groups for HTTP/SSH.

Milestone 7: Deployment on EC2

- Install Python3, Flask, Git.
- Clone the GitHub repo.

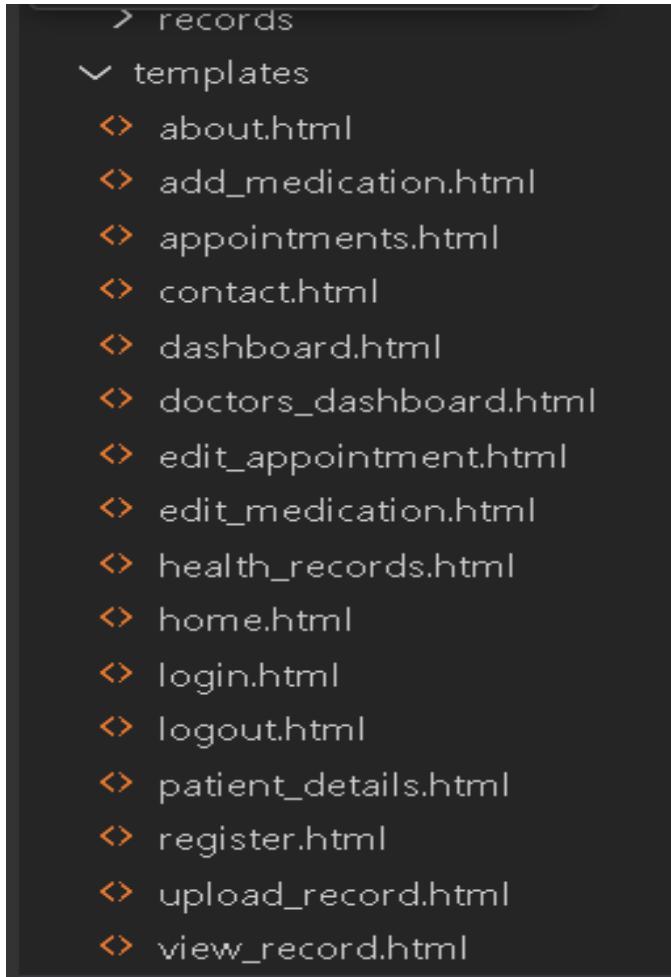
- Run Flask app: sudo flask run --host=0.0.0.0 --port=5000

Milestone 8: Testing and Deployment

- Verify registration, login, appointment booking, and SNS notifications.
-

Milestone 1: Web Application Development and Setup

- Set up Flask app with routing and templates.



```
> records
✓ templates
  ◊ about.html
  ◊ add_medication.html
  ◊ appointments.html
  ◊ contact.html
  ◊ dashboard.html
  ◊ doctors_dashboard.html
  ◊ edit_appointment.html
  ◊ edit_medication.html
  ◊ health_records.html
  ◊ home.html
  ◊ login.html
  ◊ logout.html
  ◊ patient_details.html
  ◊ register.html
  ◊ upload_record.html
  ◊ view_record.html
```

- Use local Python lists/dictionaries for initial testing.

```
app.py > ⌂ home
1 from flask import Flask, render_template, request, redirect, url_for, session, flash
2 import boto3
3 from werkzeug.security import generate_password_hash, check_password_hash
4 import uuid
5
6 app = Flask(__name__)
7 app.secret_key = 'your_secret_key'
8
9 # --- AWS Setup ---
10 dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')
11 sns = boto3.client('sns', region_name='ap-south-1')
12 sns_topic_arn = 'arn:aws:sns:ap-south-1:YOUR_ACCOUNT_ID:YourSNSTopic' # Replace with your actual topic ARN
13
14 # --- DynamoDB Tables ---
15 users_table = dynamodb.Table('Users')
16 appointments_table = dynamodb.Table('Appointments')
17 medications_table = dynamodb.Table('Medications')
18 records_table = dynamodb.Table('HealthRecords')
19
20 # --- Routes ---
21
22 @app.route('/')
23 def home():
24     return render_template('home.html')
25
26 @app.route('/register', methods=['GET', 'POST'])
27 def register():
28     if request.method == 'POST':
29         role = request.form.get('role') or request.args.get('role')
30         name = request.form['name']
31         email = request.form['email']
32         gender = request.form['gender']
33         problem = request.form.get('health_problem', '')
34         phone = request.form['phone']
35         password = generate_password_hash(request.form['password'])
36
37         existing = users_table.get_item(Key={'email': email}).get('Item')
```

- Integrate AWS services (DynamoDB, SNS) using boto3.

```
from flask import Flask, request, session, redirect, url_for, render_template, flash
import boto3
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import logging
import os
import uuid
from dotenv import load_dotenv
```

Milestone 2: AWS Account Setup

- Access AWS via Troven Labs.

The screenshot shows the Troven Labs interface. On the left, there's a sidebar with a profile picture, the name 'Kanithi', and the email '22A51A4425@pudhyotekkal.edu.in'. The main area displays a lab titled 'AWS - MedTrack Cloud based Patient Medication Tracker'. The lab details are as follows:

Platform	Lab	Duration	Difficulty	Progress
AWS	1	4 hour(s) 0 minute(s)	Expert	AWS

Below the details, there's an 'Overview' section which describes MedTrack as a cloud-native healthcare app for managing medication schedules. It mentions Flask on Amazon EC2, DynamoDB for storage, SNS for reminders, and IAM roles for security. The 'Skills' section lists EC2, Database on AWS, IAM, and Monitoring and Observability. The 'Lab' section contains a box for 'AWS Final Deployment' with the following information:

- AWS Final Deployment**
- AWS - MedTrack Cloud based Patient Medication Tracker
- AWS
- Status: Expired
- Difficulty: easy
- Task: 5
- Duration: 240 mins

- Avoid personal AWS account usage to prevent billing issues.

Milestone 3: DynamoDB Database Creation and Setup

Activity 3.1: Navigate to the DynamoDB

- In the AWS Console, navigate to DynamoDB and click on Create table

The screenshot shows the AWS Services search results page. The search bar at the top contains the query 'dynamoDB'. Below the search bar, the results are categorized under 'Services' and 'Features'.

- Services:**
 - DynamoDB (Managed NoSQL Database)
 - Amazon DocumentDB (Fully-managed MongoDB-compatible database service)
 - CloudFront (Global Content Delivery Network)
 - Athena (Serverless interactive analytics service)
- Features:**
 - Settings (DynamoDB feature)
 - Clusters (DynamoDB feature)

The screenshot shows the DynamoDB Dashboard. The left sidebar includes links for Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, Settings, DAX (Clusters, Subnet groups, Parameter groups, Events), and Metrics.

The main dashboard area displays sections for Alarms (0) and DAX clusters (0). On the right side, there is a 'Create resources' section with a prominent orange 'Create table' button. A status message indicates that Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB.

Activity 3.2: Create a DynamoDB table for the Create Users table (Primary key: Email).

- Create Users table with partition key “Email” with type String and click on create tables.

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The path 'DynamoDB > Tables > Create table' is visible at the top. The main title is 'Create table'. The 'Table details' section is active, indicated by a blue border. It contains the following fields:

- Table name:** A text input field containing 'Users'. Below it is a note: "This will be used to identify your table." and a character limit note: "Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)".
- Partition key:** A text input field containing 'email'. To its right is a dropdown menu set to 'String'. Below this is a note: "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." and a character limit note: "1 to 255 characters and case sensitive."
- Sort key - optional:** A text input field containing 'Enter the sort key name'. To its right is a dropdown menu set to 'String'. Below this is a note: "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." and a character limit note: "1 to 255 characters and case sensitive."

Activity 3.3:Create Appointments table (Primary key: appointment_id).

The screenshot shows the AWS DynamoDB console with the URL us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables. The left sidebar shows navigation options like Dashboard, Tables, Explore items, PartQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Under Tables, there's a section for DAX with Clusters, Subnet groups, Parameter groups, and Events. The main area displays a table titled 'Tables (2)'. The table has columns: Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read capac. Two rows are listed:

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capac
AppointmentsTable	Active	appointment_id (\$)	-	0	0	Off		On-demand
UsersTable	Active	email (\$)	-	0	0	Off		On-demand

At the top right, there are buttons for Actions, Delete, and Create table. A feedback banner at the top says 'Share your feedback on Amazon DynamoDB'.

Follow the same steps to create an Appointments table with Email as the primary key for booking diagnosis data.

Milestone 4: SNS Notification Setup

- **Activity 4.1: Create SNS topics for sending email notifications to users and doctor prescriptions.**
 - In the AWS Console, search for SNS and navigate to the SNS Dashboard.

SNS

Search results for 'sns'

Services

- Features
- Resources **New**
- Documentation
- Knowledge articles
- Marketplace
- Blog posts
- Events
- Tutorials

Show more ▶

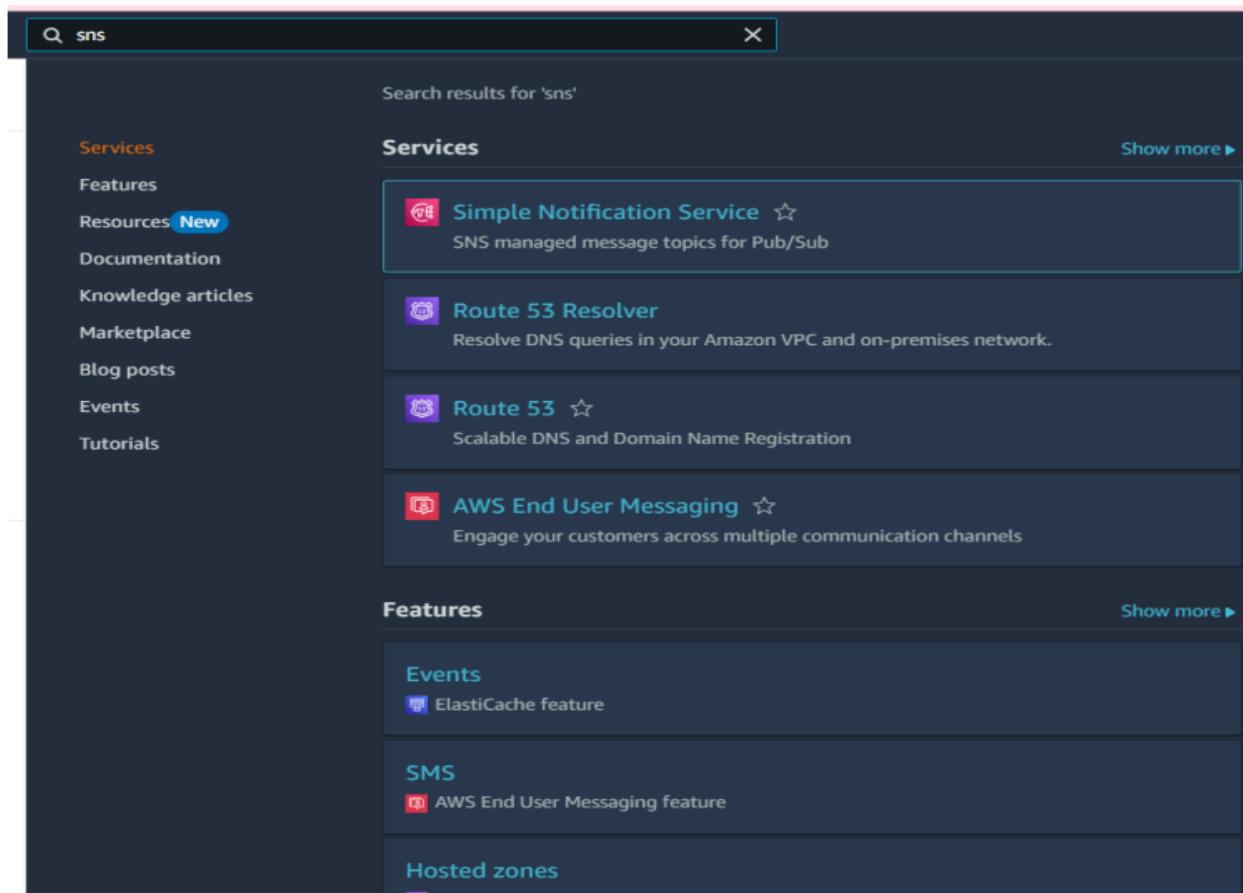
Services

- Simple Notification Service** ☆
SNS managed message topics for Pub/Sub
- Route 53 Resolver**
Resolve DNS queries in your Amazon VPC and on-premises network.
- Route 53** ☆
Scalable DNS and Domain Name Registration
- AWS End User Messaging** ☆
Engage your customers across multiple communication channels

Features

- Events
 - ElastiCache feature
- SMS
 - AWS End User Messaging feature
- Hosted zones
 - Route 53

Show more ▶



MedTrack Directory | Welcome to Skill Wallet | Student - Skill Wallet | trover.in/students/ | trover.in/students/ | Simple Notification | ONLY FOR IAM ROLI | School | ...

aws Search [Alt+S] United States (N. Virginia) rsoaccount-new/68034028d24717cbb9270727 @ rsosandboxnew123

New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more ↗

Application Integration

Amazon Simple Notification Service

Pub/sub messaging for microservices and serverless applications.

Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.

Create topic

Topic name
MyTopic

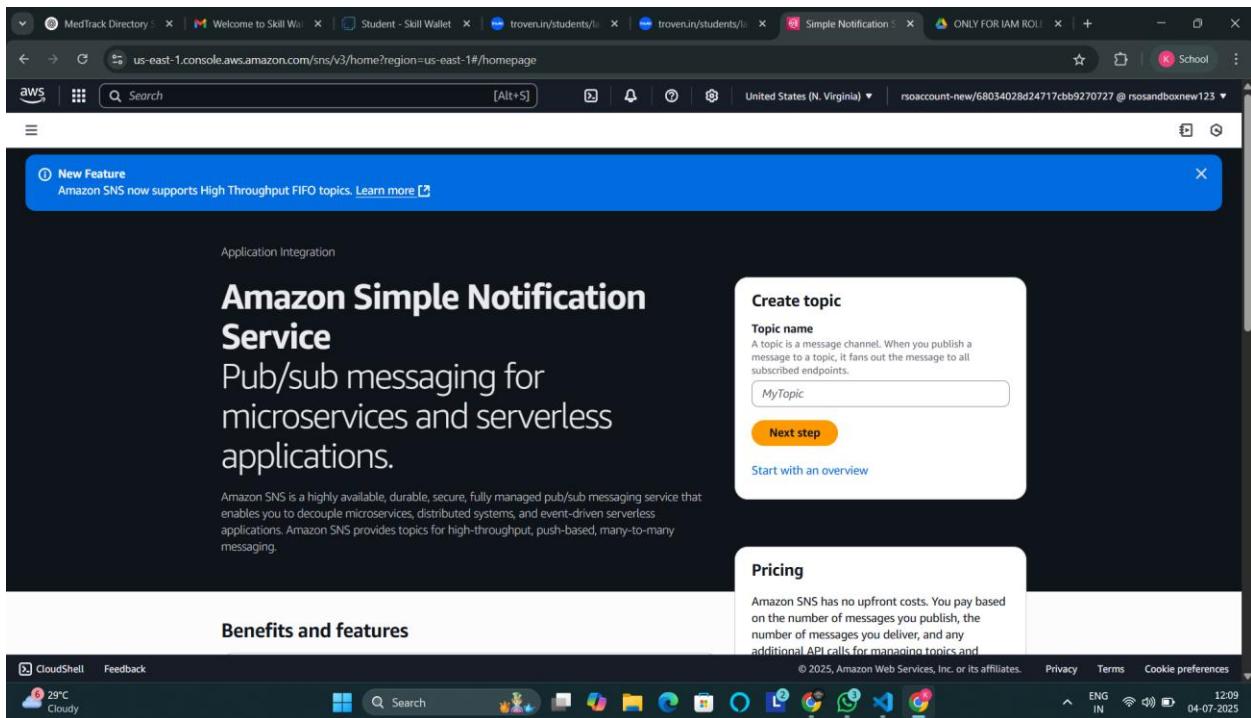
Next step

Start with an overview

Pricing

Amazon SNS has no upfront costs. You pay based on the number of messages you publish, the number of messages you deliver, and any additional API calls for managing topics and

CloudShell Feedback 29°C Cloudy Search Privacy Terms Cookie preferences ENG IN 12:09 04-07-2025



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

The screenshot shows the Amazon SNS Topics page. On the left, there's a navigation sidebar with links like Dashboard, Topics, Subscriptions, Mobile, Push notifications, and Text messaging (SMS). The main area has a header 'Amazon SNS > Topics' with a 'New Feature' banner about FIFO topics. Below is a table titled 'Topics (0)' with columns Name, Type, and ARN. A search bar and buttons for Edit, Delete, Publish message, and Create topic are at the top right. A message says 'No topics' and 'To get started, create a topic.' with a 'Create topic' button.

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

The screenshot shows the 'Create topic' wizard. The path is 'Amazon SNS > Topics > Create topic'. The first step is 'Details'. It shows two options: 'FIFO (first-in, first-out)' and 'Standard'. 'Standard' is selected. The 'FIFO' section contains a note about message ordering and throughput. The 'Standard' section lists benefits like best-effort delivery, at-least once delivery, and support for various protocols.

Type	Info
Topic type cannot be modified after topic is created	

FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

Standard

- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

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

The screenshot shows the AWS SNS console with a successful subscription creation message:

- New Feature:** Amazon SNS now supports High Throughput FIFO topics. Learn more.
- Subscription to Medtrack created successfully.** The ARN of the subscription is arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb.

Subscription: 837e417d-317b-4b43-97c3-054566ac3bbb

Details	
ARN	arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb
Endpoint	22a51a4425@adityatekkali.edu.in
Topic	Medtrack
Subscription Principal	arn:aws:iam::715841346262:role/rsoaccount-new
Status	Pending confirmation
Protocol	EMAIL

Subscription filter policy | Redrive policy (dead-letter queue)

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 29°C Cloudy ENG IN 12:22 04-07-2025

- **Activity 4.2: Subscribe users and Doctors to relevant SNS topics to receive real-time notifications when a book appointment is made.**

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

Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN
 X

Protocol
The type of endpoint to subscribe
 ▼

Endpoint
An email address that can receive notifications from Amazon SNS.

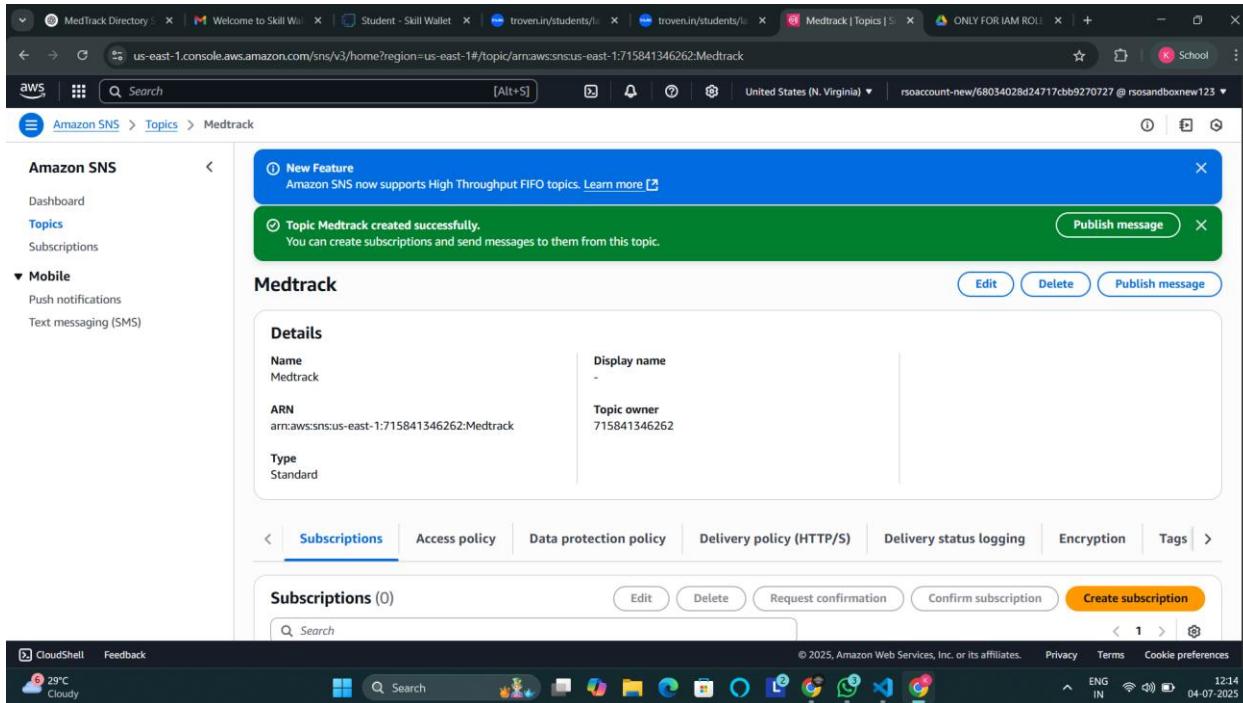
ⓘ After your subscription is created, you must confirm it. [Info](#)

► Subscription filter policy - optional [Info](#)
This policy filters the messages that a subscriber receives.

► Redrive policy (dead-letter queue) - optional [Info](#)
Send undeliverable messages to a dead-letter queue.

Cancel Create subscription

After the subscription request for the mail confirmation.



The screenshot shows the AWS SNS Topics page. On the left, there's a navigation sidebar with links for Dashboard, Topics, Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and CloudShell. The main area shows a topic named "Medtrack". A blue banner at the top right says "Topic Medtrack created successfully. You can create subscriptions and send messages to them from this topic." Below this, there's a "Details" section with fields for Name (Medtrack), Display name (-), ARN (arn:aws:sns:us-east-1:715841346262:Medtrack), Topic owner (715841346262), and Type (Standard). At the bottom, there are tabs for Subscriptions (selected), Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags. The Subscriptions tab shows 0 subscriptions. There are buttons for Edit, Delete, Request confirmation, Confirm subscription, and Create subscription. The status bar at the bottom shows the date (04-07-2025), time (12:14), and network connection information.

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

The screenshot shows an email from AWS Notifications (no-reply@sns.amazonaws.com) titled "AWS Notification - Subscription Confirmation". The email is marked as spam and is dated Fri 4 Jul, 12:22 (5 days ago). It contains a message about being identified as spam in the past and provides a link to "Report as not spam". The body of the email states: "You have chosen to subscribe to the topic: arn:aws:sns:us-east-1:904233124678:MedTrack". It includes a confirmation link: "To confirm this subscription, click or visit the link below (If this was in error no action is necessary): [Confirm subscription](#)". A note at the bottom says: "Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)". Below the email are standard reply, forward, and smiley face buttons.

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

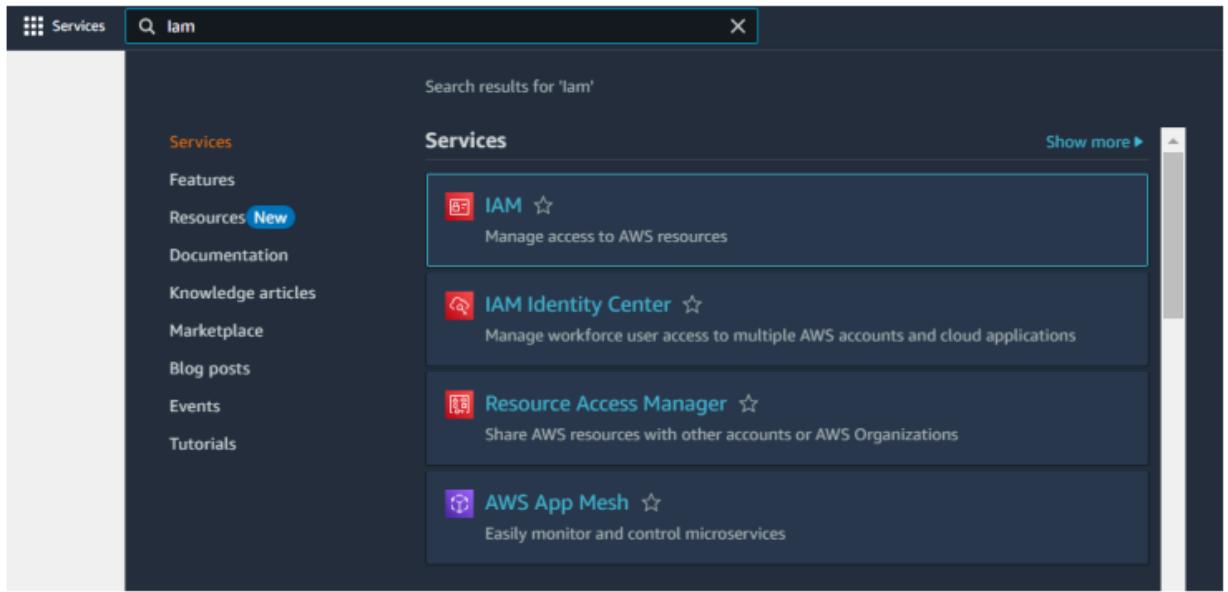
The screenshot shows the AWS SNS console with a list of topics. One topic, "Medtrack", is selected. Under "Subscriptions" for this topic, there is one entry: "Subscription: 837e417d-317b-4b43-97c3-054566ac3bbb". The details for this subscription are shown in a modal window:

- ARN:** arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb
- Endpoint:** 22a51a4425@adityatekkali.edu.in
- Topic:** Medtrack
- Subscription Principal:** arn:aws:siam::715841346262:role/rsoaccount-new

The status is listed as "Status: Confirmed". The modal also includes "Edit" and "Delete" buttons. At the bottom of the page, there are tabs for "Subscription filter policy" and "Redrive policy (dead-letter queue)". The footer of the browser window shows various AWS services and the date 04-07-2025.

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.

A screenshot of the AWS IAM Dashboard. The left sidebar shows navigation options: Identity and Access Management (IAM), Dashboard, Access management (User groups, Users, Roles, Policies, Identity providers, Account settings, Root access management), Access reports (Resource analysis, Unused access, Analyzer settings, Credential report), Organization activity, CloudShell, and Feedback. The main content area is titled 'IAM Dashboard' and includes sections for 'IAM resources' (Resources in this AWS Account), 'What's new' (Updates for features in IAM), and 'AWS Account' (Access denied errors). The 'Access denied' section shows two entries:

- User: arn:aws:sts::715841346262:assumed-role/rsoaccount-new/68034028d24717ccb9270727 @rsoaccountnew123
Action: iam:GetAccountSummary
Context: no identity-based policy allows the action
- User: arn:aws:sts::715841346262:assumed-role/rsoaccount-new/68034028d24717ccb9270727 @rsoaccountnew123
Action: iam>ListAccountAliases
Context: no identity-based policy allows the action

Both entries have a 'Diagnose with Amazon Q' button.

- **Create IAM Roles:**

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
EC2_MedTrack_Role

Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy

```

1+ [
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ]
    }
  ]
]
  
```

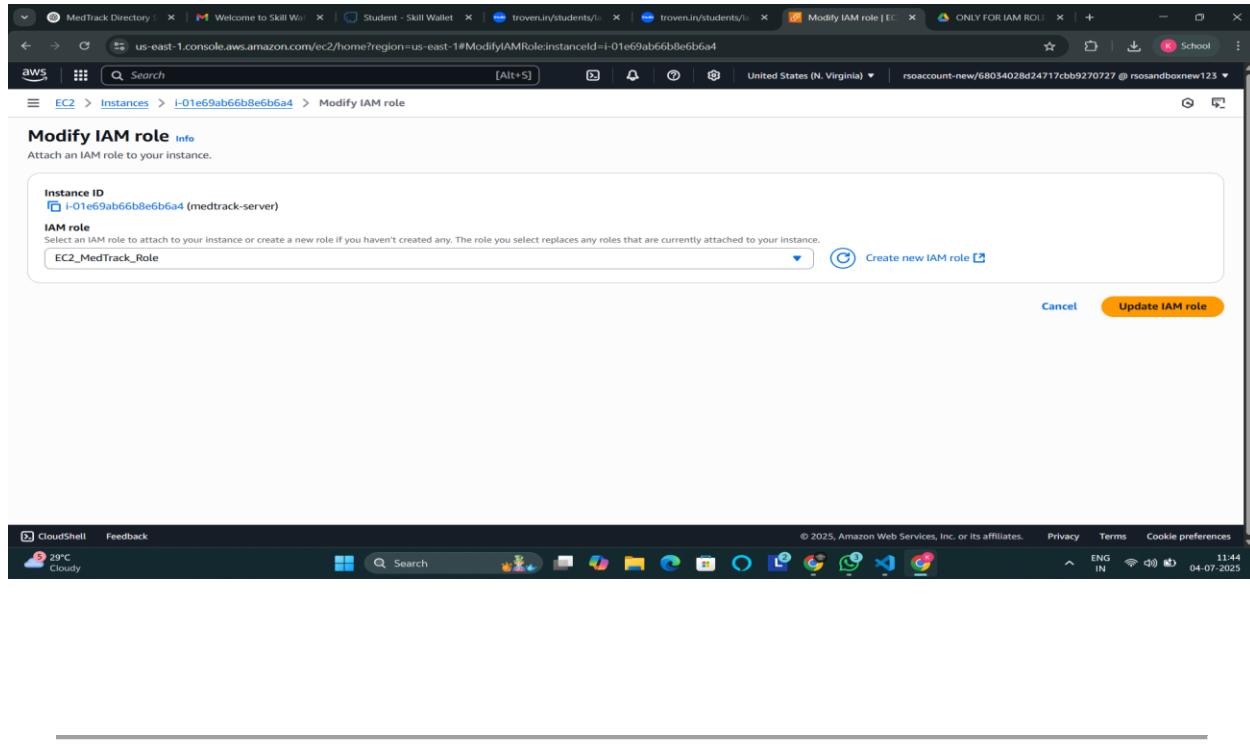
- Attach policies: **AmazonDynamoDBFullAccess**, **AmazonSNSFullAccess**.

Role EC2_MedTrack_Role created.

Roles (11) Info

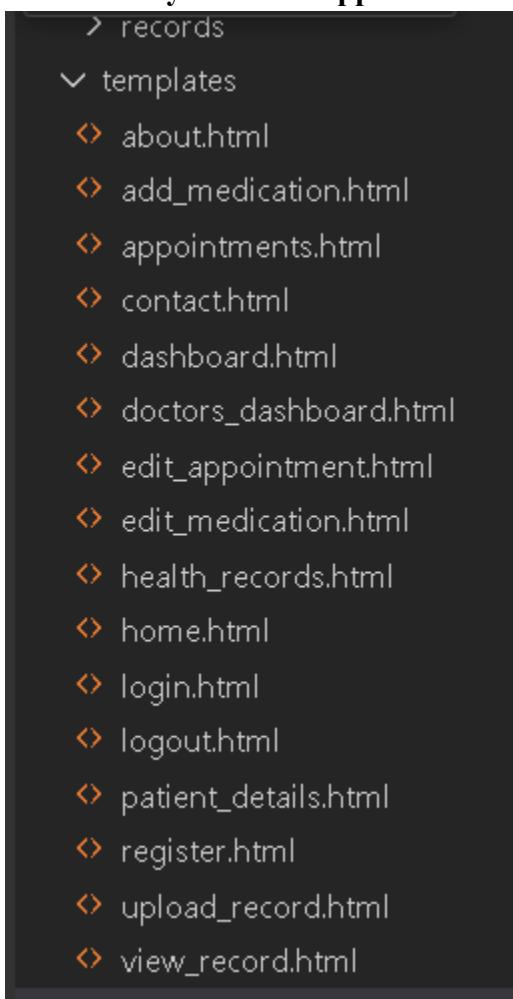
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service-Linked Role)	-
AWSServiceRoleForAutoScaling	AWS Service: autoscaling (Service-Linked Role)	144 days ago
AWSServiceRoleForECS	AWS Service: ecs (Service-Linked Role)	144 days ago
AWSServiceRoleForOrganizations	AWS Service: organizations (Service-Linked Role)	212 days ago
AWSServiceRoleForRDS	AWS Service: rds (Service-Linked Role)	108 days ago
AWSServiceRoleForSSO	AWS Service: sso (Service-Linked Role)	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
EC2_MedTrack_Role	AWS Service: ec2	-
OrganizationAccountAccessRole	Account: 058264256896	58 minutes ago



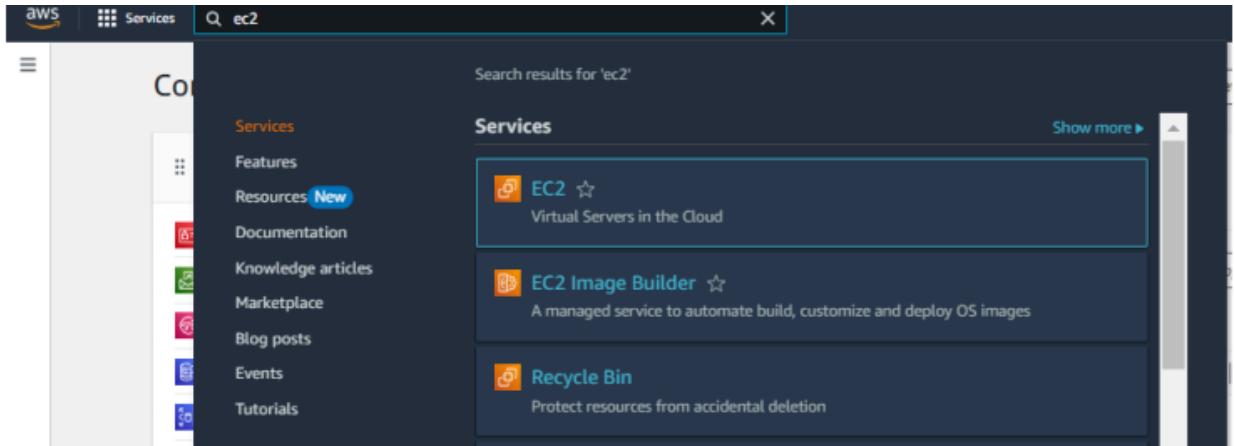
Milestone 6: EC2 Instance Setup

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

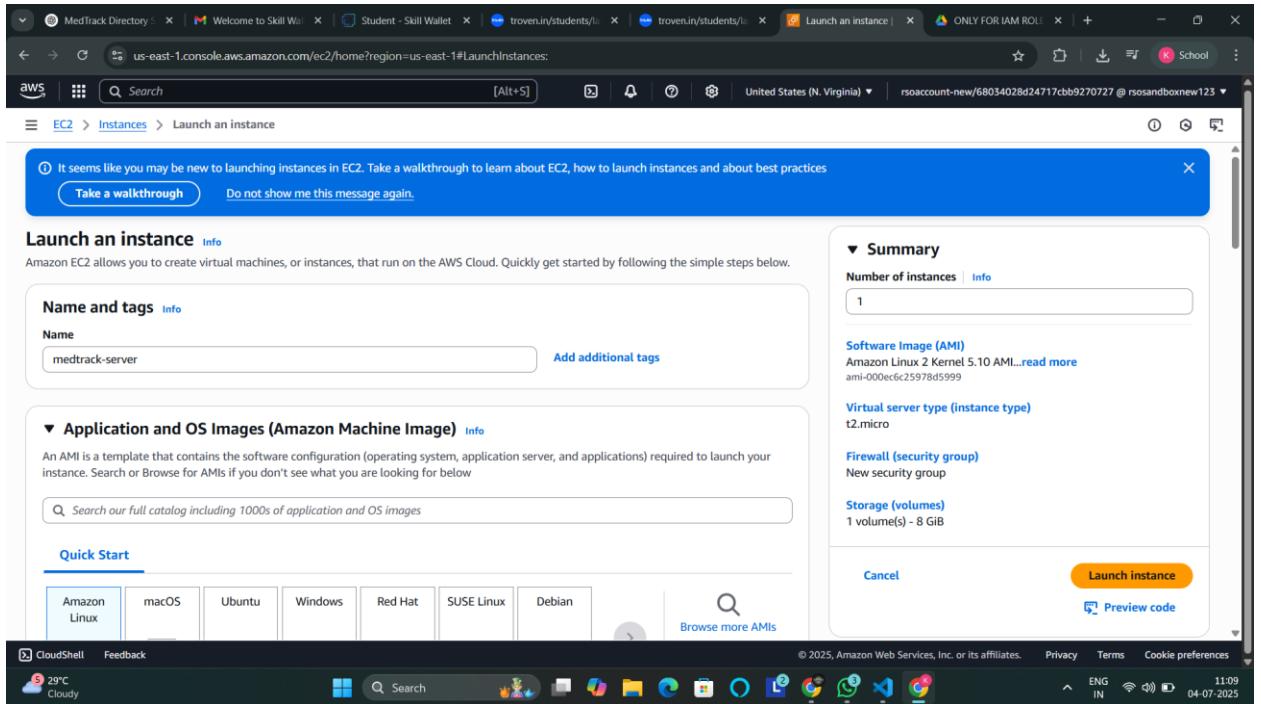


Launch an EC2 instance to host the Flask application.

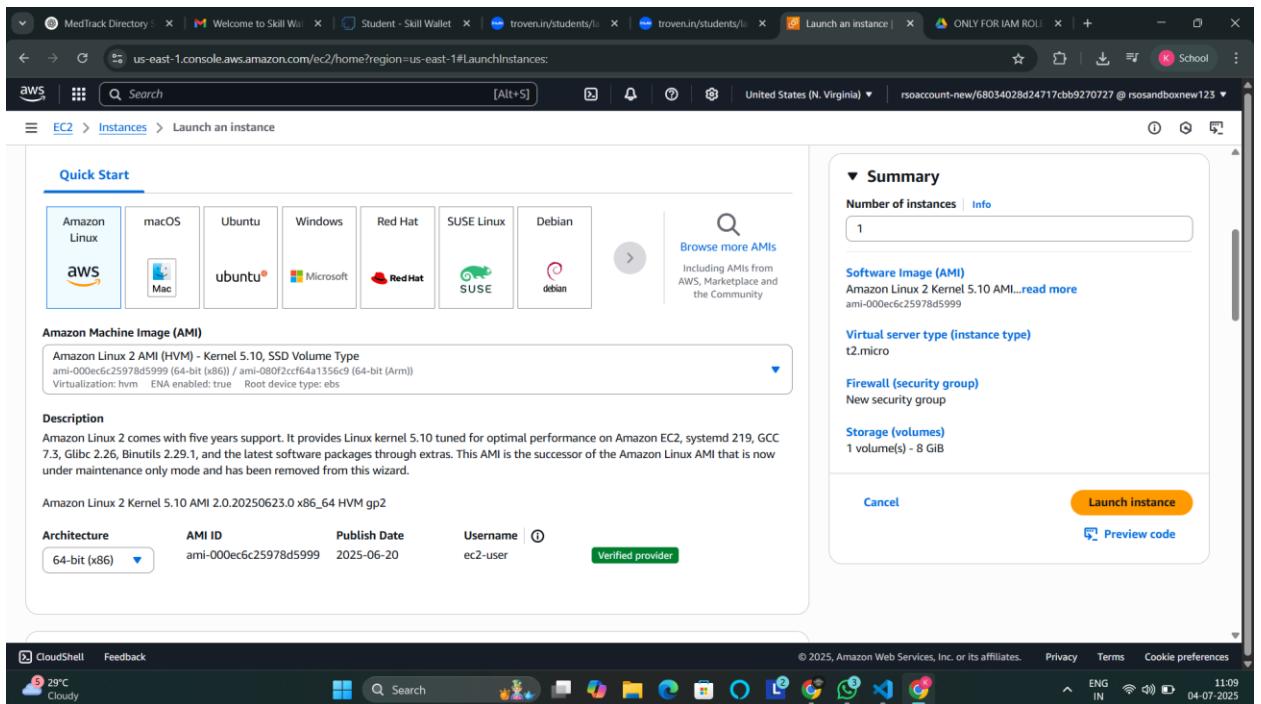
- In the AWS Console, navigate to EC2 and launch a new instance.



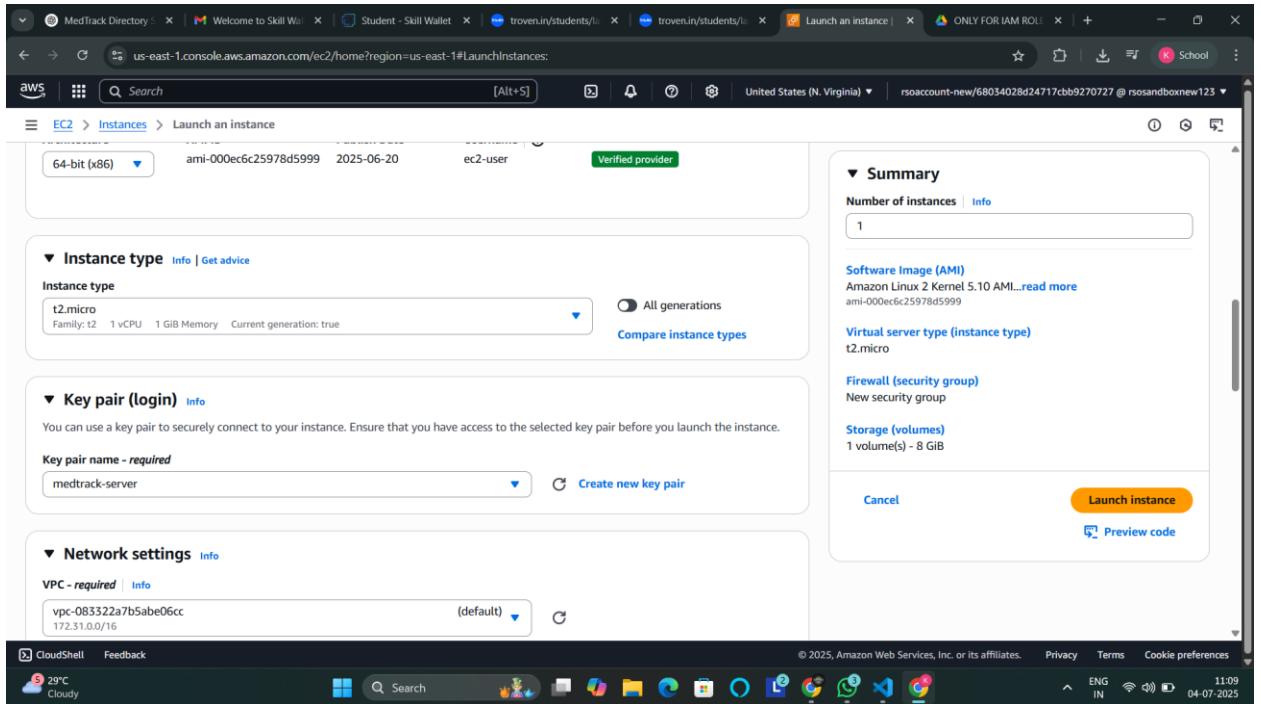
- Click on Launch instance to launch EC2 instance



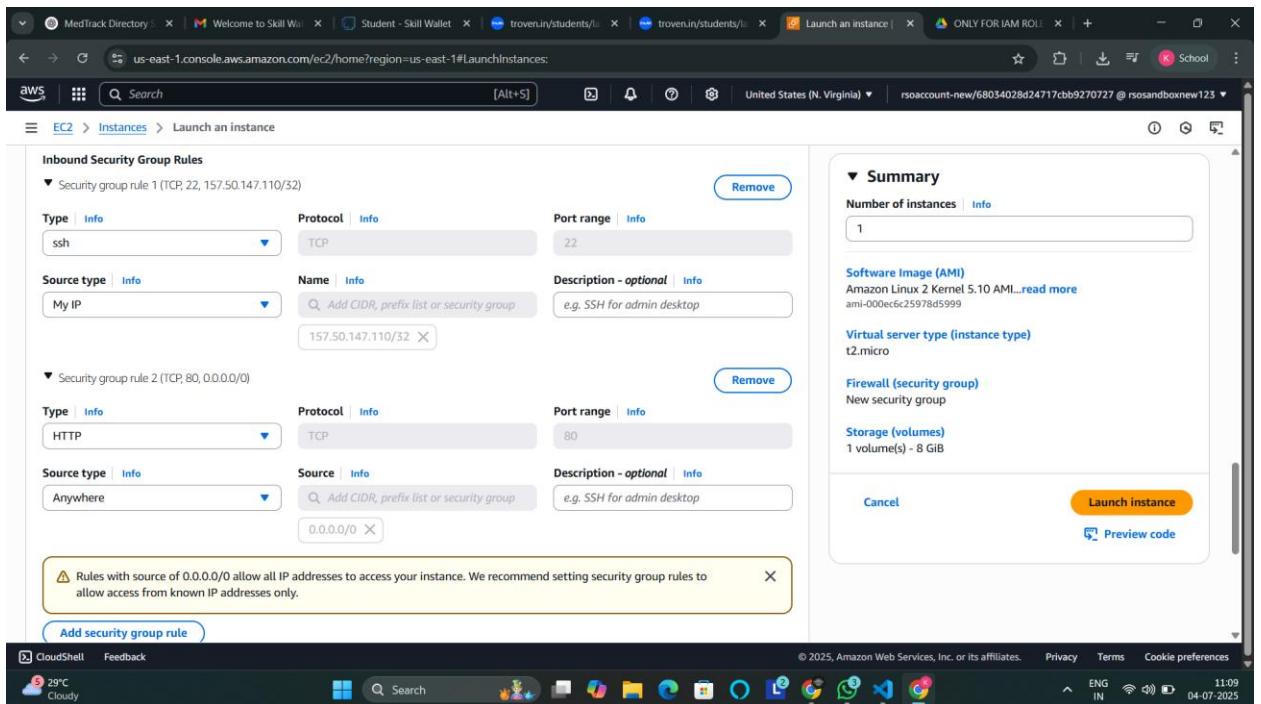
- **Launch EC2 instance (Amazon Linux 2/Ubuntu, t2.micro).**



- **Assign an IAM Role and key pair.**

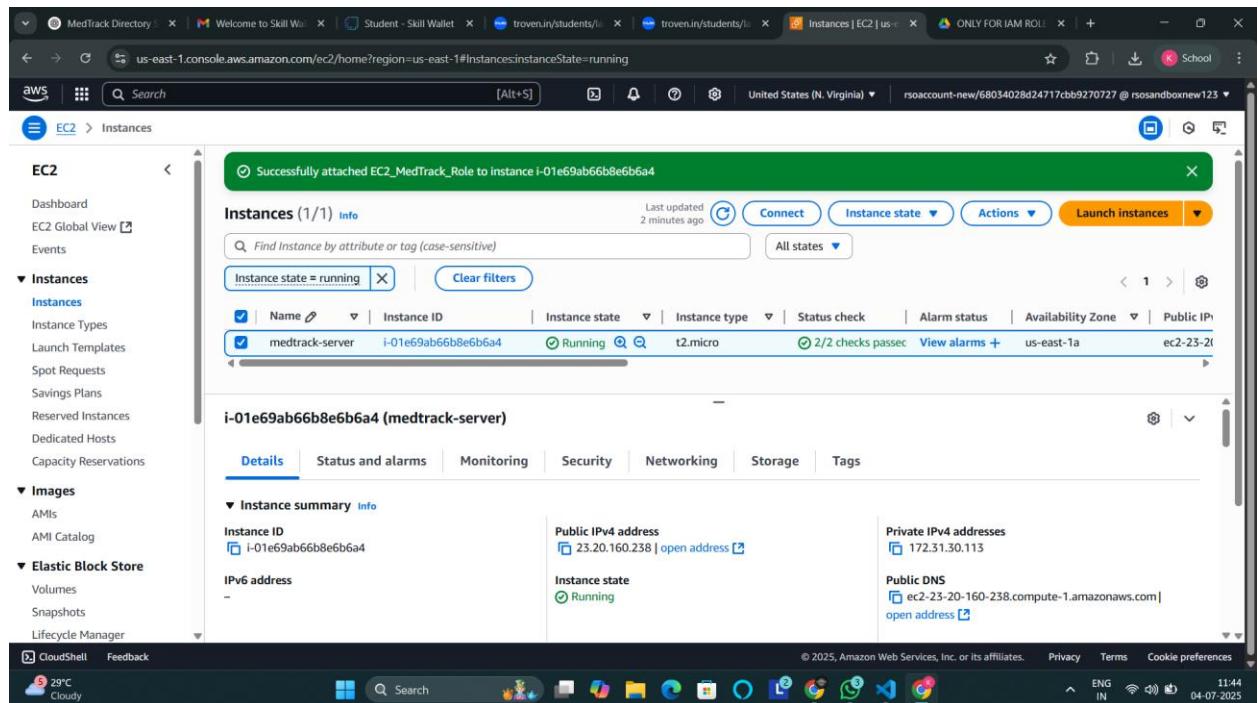


- Configure security groups for HTTP/SSH.



- 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.

```

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 2038-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-23-158 ~]# su
[ec2-user@ip-172-31-23-158 ~]#

```

Milestone 7: Deployment on EC2

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

Clone Your Flask Project from GitHub:

Run: ‘git clone https://github.com/Thanguduvandana/Med_Track

Note: change your-github-username and your-repository-name with your credentials.
here: ‘git clone https://github.com/Thanguduvandana/Med_Track

- This will download your project to the EC2 instance.
- To navigate to the project directory, run the following
- command: cd Medtrack.

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

- Run Flask app: sudo flask run --host=0.0.0.0 --port=5000

Verify the Flask app is running:

- http://your-ec2-public-ip
- Run the Flask app on the EC2 instance

```

Running on all addresses (0.0.0.0)
Running on http://127.0.0.1:5000
Running on http://192.168.77.51:5000
5-06-22 14:28:29,397 - werkzeug - INFO - [33mPress CTRL+C to quit[0m
5-06-22 14:28:47,806 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:28:47] "GET / HTTP/1.1" 200 -
5-06-22 14:28:47,972 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:28:47] "GET /static/css/custom.css HTTP/1.1" 200 -
5-06-22 14:28:47,991 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:28:47] "GET /static/js/custom.js HTTP/1.1" 200 -
5-06-22 14:29:18,369 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:29:18] "GET / HTTP/1.1" 200 -
5-06-22 14:29:18,407 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:29:18] "GET /static/css/custom.css HTTP/1.1" 200 -
5-06-22 14:29:18,637 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:29:18] "GET /static/js/custom.js HTTP/1.1" 200 -
5-06-22 14:30:04,882 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:30:04] "GET /login HTTP/1.1" 200 -
5-06-22 14:30:05,098 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:30:05] "[36mGET /static/css/custom.css HTTP/1.1[0m" 304 -
5-06-22 14:30:05,104 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:30:05] "[36mGET /static/js/custom.js HTTP/1.1[0m" 304 -
5-06-22 14:30:11,713 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:30:11] "GET /register HTTP/1.1" 200 -
5-06-22 14:30:12,006 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:30:12] "[36mGET /static/css/custom.css HTTP/1.1[0m" 304 -
5-06-22 14:30:12,054 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:30:12] "[36mGET /static/js/custom.js HTTP/1.1[0m" 304 -
5-06-22 14:30:31,104 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:30:31] "GET /login HTTP/1.1" 200 -

```

Access the website through:

- Public IPs: <http://23.20.160.238:5000>
-

Milestone 8: Testing and Deployment

- Verify registration, login, appointment booking, and SNS notifications.
-

Flask Application Structure & Code

App Initialization:

- Setup routes: register, login, dashboard, book appointment, view appointment, search, profile.

```

@app.route('/')
def index():
    if is_logged_in():
        return redirect(url_for('dashboard'))
    return render_template('index.html')

```

- Connect to DynamoDB and SNS using boto3 with the correct region and ARN.

Routes:

- **Register:** Register the user, hash the password, and store it in DynamoDB.

```

26 @app.route('/register', methods=['GET', 'POST'])
27 def register():
28     if request.method == 'POST':
29         role = request.form.get('role') or request.args.get('role')
30         name = request.form['name']
31         email = request.form['email']
32         gender = request.form['gender']
33         problem = request.form.get('health_problem', '')
34         phone = request.form['phone']
35         password = generate_password_hash(request.form['password'])
36
37         existing = users_table.get_item(Key={'email': email}).get('Item')
38         if existing:
39             flash("User already exists!", "danger")
40             return redirect(url_for('register'))
41
42         user_item = {
43             'email': email,
44             'name': name,
45             'password': password,
46             'gender': gender,
47             'phone': phone,
48             'role': role
49         }
50
51         if role.lower() == 'doctor':
52             user_item['specialization'] = problem
53         else:
54             user_item['health_problem'] = problem
55
56         users_table.put_item(Item=user_item)
57
58         try:
59             sns.publish(
60                 TopicArn=sns_topic_arn,
61                 Message=f"New {role} registered: {name} ({email})",

```

- **Login:** Authenticate and update login count.

```

@app.route('/login', methods=['GET', 'POST'])
def login():
    if is_logged_in():
        return redirect(url_for('dashboard'))
    if request.method == 'POST':
        email = request.form.get('email', '').lower()
        password = request.form.get('password', '')
        role = request.form.get('role', '').lower()

        if not email or not password or not role:
            flash('All fields are required', 'danger')
            return render_template('login.html')

        user = user_table.get_item(Key={'email': email}).get('Item')
        if user and user['role'] == role and check_password_hash(user['password'], password):
            session['email'] = email
            session['role'] = role
            session['name'] = user.get('name', '')
            flash('Login successful!', 'success')
            return redirect(url_for('dashboard'))
        flash('Invalid email, password, or role', 'danger')
    return render_template('login.html')

```

- **Logout:** End session.

```
@app.route('/logout')
def logout():
    session.clear()
    flash('Logged out successfully', 'success')
    return redirect(url_for('login'))
```

- **Book Appointment:** Collect and store appointment details, trigger SNS.

```
@app.route('/book_appointment', methods=['GET', 'POST'])
def book_appointment():
    if not is_logged_in() or session.get('role') != 'patient':
        flash('Only patients can book appointments', 'danger')
        return redirect(url_for('login'))

    if request.method == 'POST':
        doctor_email = request.form.get('doctor_email')
        symptoms = request.form.get('symptoms')
        appointment_date = request.form.get('appointment_date')
        patient_email = session.get('email')

        if not doctor_email or not symptoms or not appointment_date:
            flash('Please fill all fields', 'danger')
            return redirect(url_for('book_appointment'))

        # Get doctor and patient info
        doctor = user_table.get_item(Key={'email': doctor_email}).get('Item')
        patient = user_table.get_item(Key={'email': patient_email}).get('Item')

        appointment_id = str(uuid.uuid4())
        appointment_item = {
            'appointment_id': appointment_id,
            'doctor_email': doctor_email,
            'doctor_name': doctor.get('name', 'Doctor'),
            'patient_email': patient_email,
            'patient_name': patient.get('name', 'Patient'),
            'symptoms': symptoms,
            'status': 'pending',
            'appointment_date': appointment_date,
            'created_at': datetime.utcnow().isoformat()
        }
```

- **View Appointments:** Retrieve data from DynamoDB.

```

@app.route('/view_appointment/<appointment_id>', methods=['GET', 'POST'])
def view_appointment(appointment_id):
    if not is_logged_in():
        flash('Please login first', 'danger')
        return redirect(url_for('login'))

    try:
        response = appointment_table.get_item(Key={'appointment_id': appointment_id})
        appointment = response.get('Item')
        if not appointment:
            flash('Appointment not found', 'danger')
            return redirect(url_for('dashboard'))

        # Authorization check
        if session.get('role') == 'doctor' and appointment['doctor_email'] != session.get('email'):
            flash('Unauthorized access', 'danger')
            return redirect(url_for('dashboard'))
        if session.get('role') == 'patient' and appointment['patient_email'] != session.get('email'):
            flash('Unauthorized access', 'danger')
            return redirect(url_for('dashboard'))

        if request.method == 'POST' and session.get('role') == 'doctor':
            diagnosis = request.form.get('diagnosis')
            treatment_plan = request.form.get('treatment_plan')
            prescription = request.form.get('prescription')

            appointment_table.update_item(
                Key={'appointment_id': appointment_id},
                UpdateExpression="SET diagnosis = :d, treatment_plan = :t, prescription = :p, #s = :s, updated_at = :u",
                ExpressionAttributeValues={
                    ':d': diagnosis,
                    ':t': treatment_plan,
                    ':p': prescription,
                    ':s': 'completed',
                    ':u': datetime.utcnow().isoformat()
                },
            )
    
```

Profile: View/edit personal data:

```
@app.route('/profile', methods=['GET', 'POST'])
def profile():
    Chat (CTRL + I) / Share (CTRL + L)
    if not is_logged_in():
        flash('Please login first', 'danger')
        return redirect(url_for('login'))
    email = session.get('email')
    user = user_table.get_item(Key={'email': email}).get('Item')
    if not user:
        flash('User not found', 'danger')
        return redirect(url_for('logout'))

    if request.method == 'POST':
        name = request.form.get('name', user.get('name'))
        age = request.form.get('age', user.get('age'))
        gender = request.form.get('gender', user.get('gender'))
        update_expression = "SET #name = :name, age = :age, gender = :gender"
        expr_values = {':name': name, ':age': age, ':gender': gender}
        expr_names = {'#name': 'name'}

        # If doctor, allow specialization update
        if user['role'] == 'doctor' and 'specialization' in request.form:
            update_expression += ", specialization = :spec"
            expr_values[':spec'] = request.form['specialization']

        user_table.update_item(
            Key={'email': email},
            UpdateExpression=update_expression,
            ExpressionAttributeValues=expr_values,
            ExpressionAttributeNames=expr_names
        )
        session['name'] = name
        flash('Profile updated', 'success')
        return redirect(url_for('profile'))

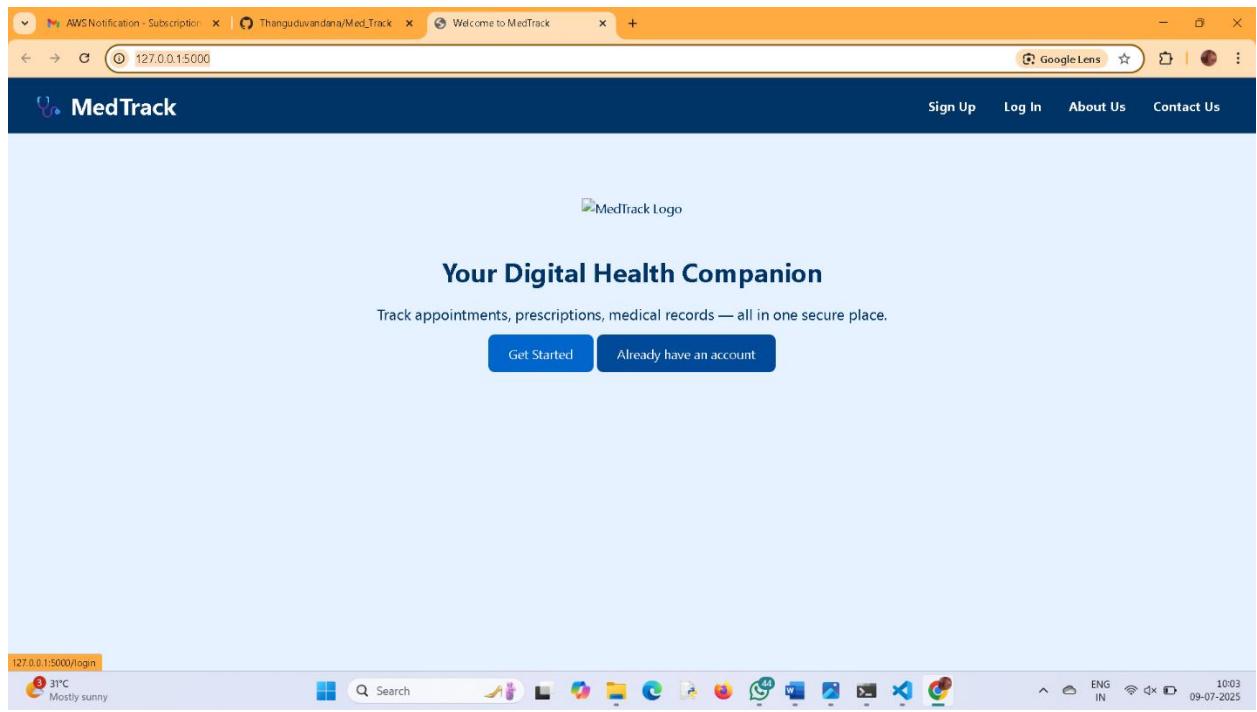
    return render_template('profile.html', user=user)
```

Deployment Code:

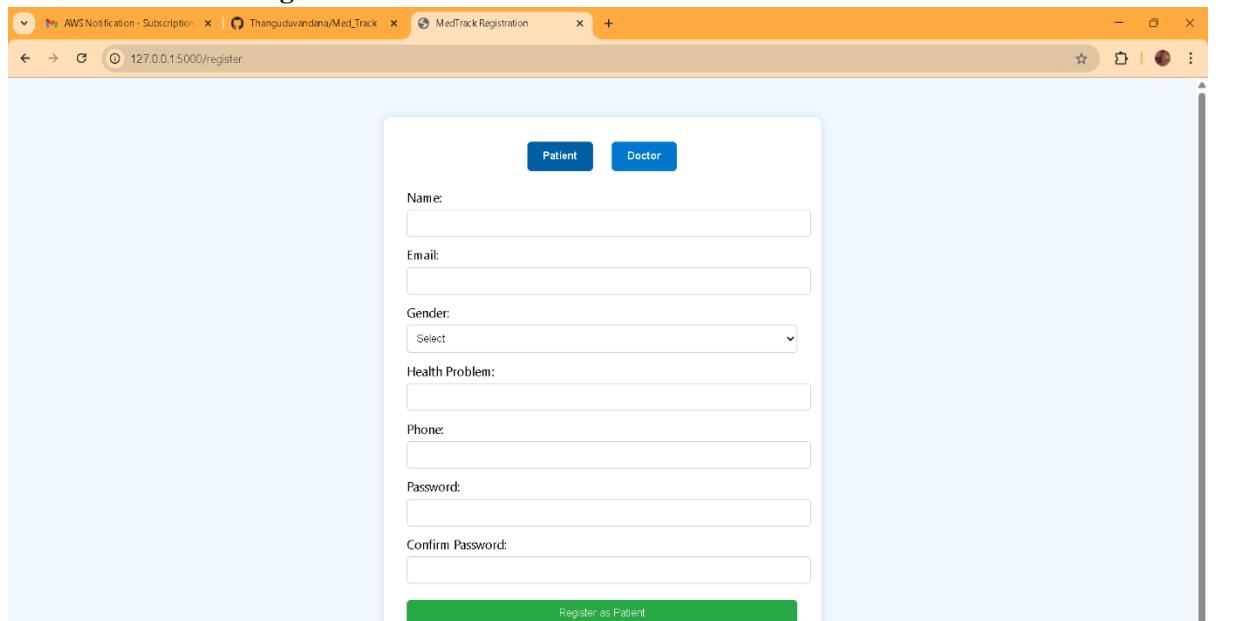
```
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5000))
    debug_mode = os.environ.get('FLASK_ENV', '') == 'development'
    app.run(host='0.0.0.0', port=port, debug=debug_mode)
```

Functional Testing Summary

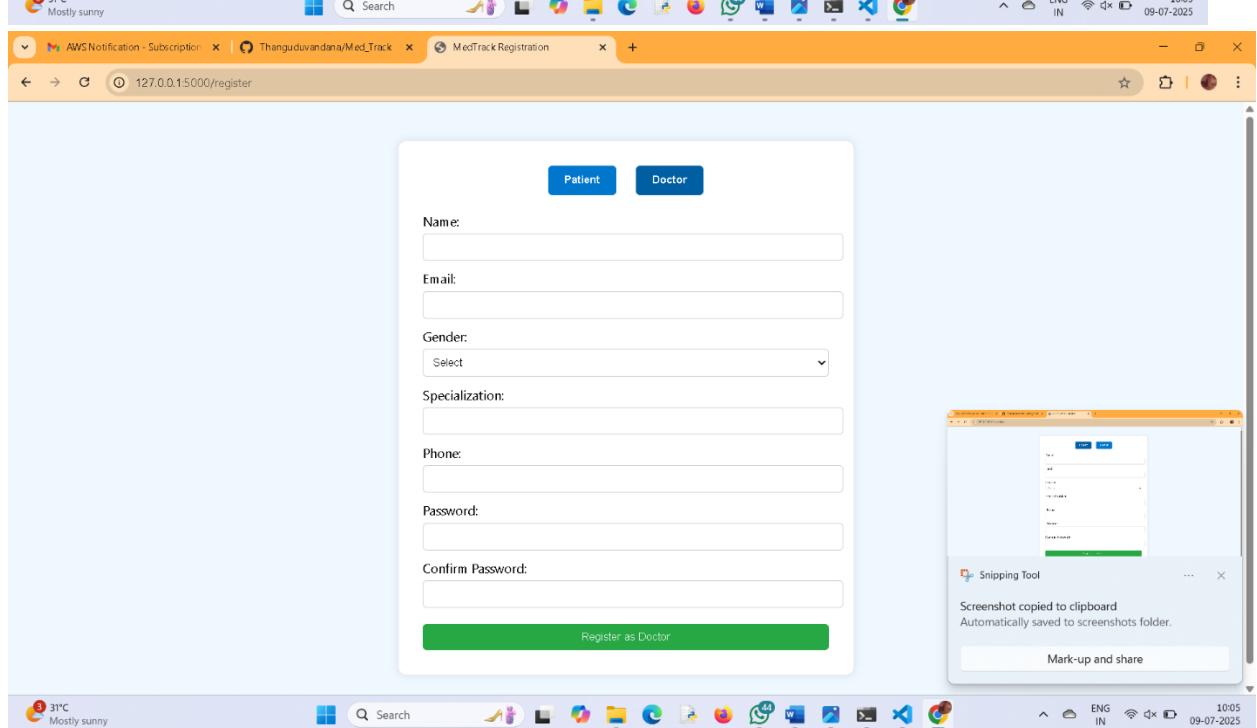
- **Home Page:** Entry point with navigation and responsive design.



- **Doctor & Patient Registration:** Collects and validates credentials.



The screenshot shows the Patient registration form. It features two tabs at the top: "Patient" (selected) and "Doctor". The form includes fields for Name, Email, Gender (with a dropdown menu showing "Select"), Health Problem, Phone, Password, and Confirm Password. A green "Register as Patient" button is at the bottom.



The screenshot shows the Doctor registration form. It has the same structure as the Patient form, with tabs for "Patient" and "Doctor" (disabled). The fields for Name, Email, Gender, Specialization (dropdown menu showing "Select"), Phone, Password, and Confirm Password are present. A green "Register as Doctor" button is at the bottom. A Snipping Tool window is overlaid on the screen, indicating a screenshot was taken at 10:05 on 09-07-2025.

- **Login Pages:** Secures access and redirects to dashboards.
- **Dashboards:** Role-based UIs for managing appointments.

→ Doctor Dashboard

The screenshot shows a web browser window titled "Doctor Dashboard" with the URL "127.0.0.1:5000/dashboard". The dashboard has two main sections: "Patient Appointments" and "Registered Patients".

Patient Appointments

Patient Email	Date	Time	Reason
thanguduvandana@gmail.com	2025-06-26	10:00	consultation
thanguduvandana@gmail.com	2025-07-10	10:00	Health problem
syamgopal@gmail.com	2025-07-10	10:00	skin disease
syamgopal@gmail.com	2025-07-10	10:00	skin disease
syamgopal@gmail.com	2025-07-10	10:00	skin disease

Registered Patients

Name	Email	Gender	Health Problem	Phone	Upload
vandana	thanguduvandana@gmail.com	Female	Sick	9876543211	<button>Upload Record</button>
Thangudu vandana	22a51a4460@adityatekkali.edu.in	Female	fever	9876543212	<button>Upload Record</button>
syam	syamgopal@gmail.com	Male	skin disease	9876543211	<button>Upload Record</button>

At the bottom, there is a weather widget showing "31°C Mostly sunny", a taskbar with various icons, and a system tray with date and time information.

→ Patient Dashboard

The screenshot shows a web browser window titled "MedTrack Dashboard" with the URL "127.0.0.1:5000/dashboard". The dashboard is personalized for "Syamgopal" and includes a welcome message and navigation links for adding medications, viewing appointments, and doctor's dashboard.

Your Medications:

Name	Dosage	Frequency	Notes	Action1	Action2
dolo	650	4	for fever	Delete	Edit

Your Appointments:

Date	Time	Reason	Actions
2025-07-10	10:00	Check-up	View Details

The taskbar at the bottom shows the date as 09-07-2025, time as 10:10, and various system icons.

Conclusion

MedTrack successfully demonstrates how cloud-native technologies can modernize healthcare delivery. Integrating Flask with AWS services such as EC2, DynamoDB, SNS, and IAM ensures secure, scalable, and responsive operations. MedTrack enhances patient care, optimizes appointment workflows, and supports reliable doctor-patient communication. This project stands as a powerful model of how technology can bridge operational gaps in real-world healthcare systems.