

# 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

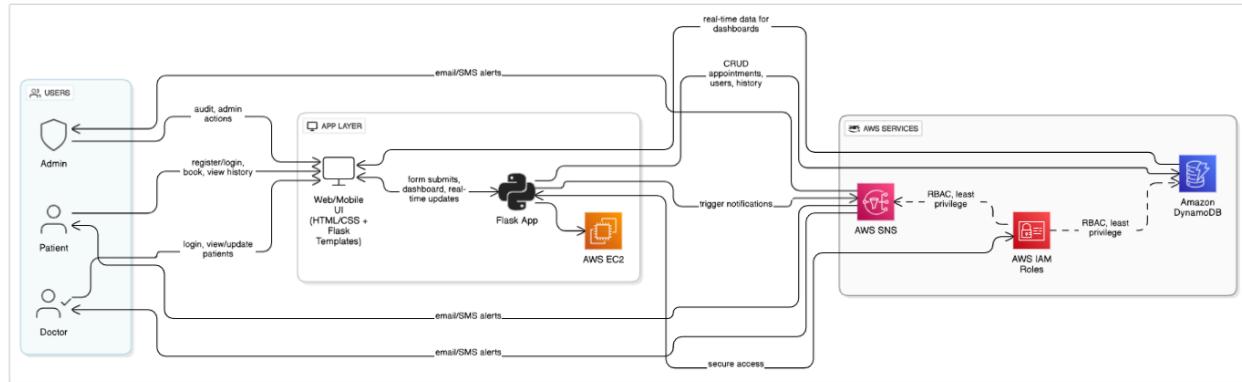
MedTrack uses IAM roles for secure, role-based access control. Patients and doctors receive permissions specific to their roles, ensuring secure access to sensitive data.

### 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.

---

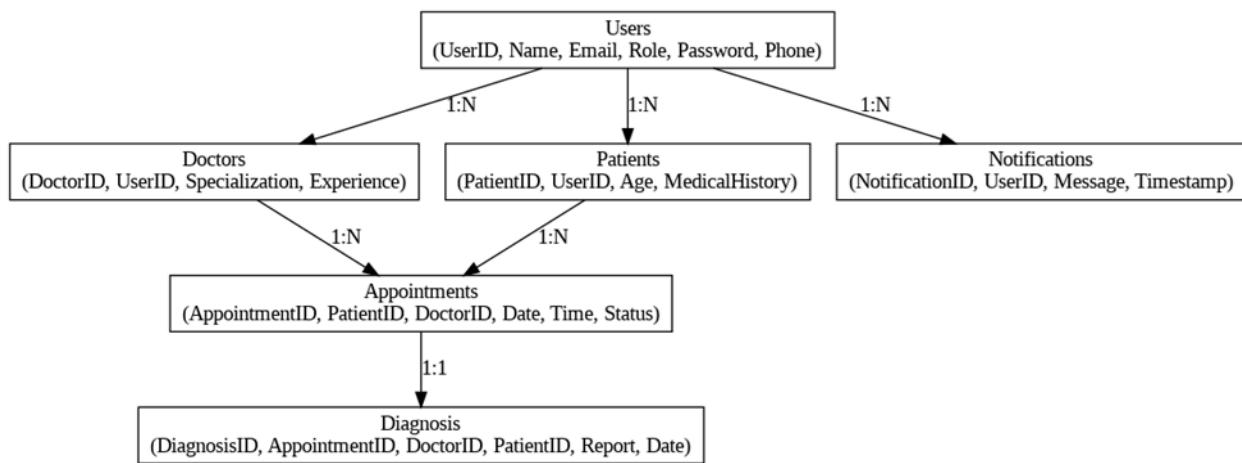
# AWS Architecture



- **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.

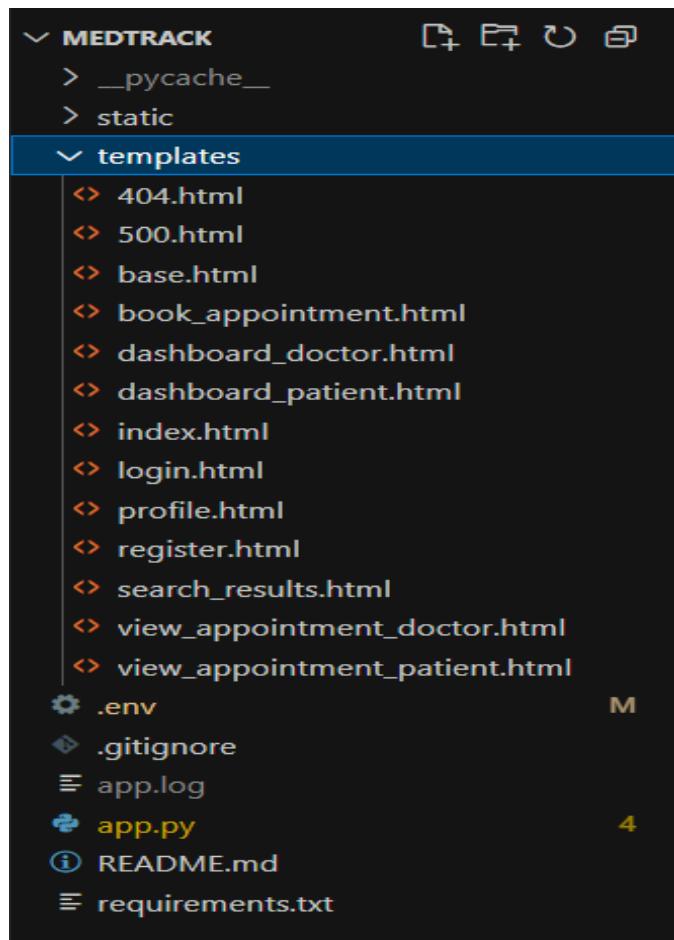
- Run Flask app: sudo flask run --host=0.0.0.0 --port=5000

## 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.



- Use local Python lists/dictionaries for initial testing.

```

# --- Mock Data Stores ---
users = []
appointments = []

# --- Register a User ---
def register_user(email, name, role, password):
    for user in users:
        if user['email'] == email:
            return "User already exists."
    users.append({
        "email": email,
        "name": name,
        "role": role,
        "password": password # In real app, hash it!
    })
    return "User registered successfully."

# --- Login User ---
def login_user(email, password):
    for user in users:
        if user['email'] == email and user['password'] == password:
            return f"Welcome {user['name']} ({user['role']})"
    return "Invalid credentials."


# --- Book Appointment ---
def book_appointment(patient_email, doctor_email, date, time):
    appointment_id = len(appointments) + 1
    appointments.append({
        "appointment_id": appointment_id,
        "patient": patient_email,
        "doctor": doctor_email,
        "date": date,
        "time": time,
        "status": "pending"
    })

```

- 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@odityatekkali.edu.in'. The main content area has a header 'AWS - MedTrack Cloud based Patient Medication Tracker' with a back button. Below the header is a table with columns: Platform (AWS), Lab (1), Duration (4 hour(s) 0 minute(s)), Difficulty (Expert), and Progress (AWS). Underneath the table, there's an 'Overview' section with a brief description of the MedTrack app. The 'Skills' section lists EC2, Database on AWS, IAM, and Monitoring and Observability. The 'Lab' section contains a card for 'AWS Final Deployment' with the following details: Platform (AWS), Status (Expired), Difficulty (easy), Task (5), and 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 tables

The screenshot shows the AWS Services search results for 'dynamoDB'. The search bar at the top contains 'dynamoDB'. Below it, the 'Services' section lists several services: Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The 'DynamoDB' service is highlighted with a blue border and is described as a 'Managed NoSQL Database'. Other services listed include Amazon DocumentDB, CloudFront, and Athena. Below the services, there's a 'Features' section with 'Settings' and 'Clusters' categories, both of which mention 'DynamoDB feature'. A 'Show more ▶' link is visible in both the services and features sections.

The screenshot shows the DynamoDB Dashboard. The left sidebar has a 'Dashboard' section with links to Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations (New), Reserved capacity, and Settings. It also has a 'DAX' section with links to Clusters, Subnet groups, Parameter groups, and Events. The main dashboard area shows 'Alarms (0) Info' and 'DAX clusters (0) Info'. On the right, there's a 'Create resources' section with a 'Create table' button and a brief description of Amazon DynamoDB Accelerator (DAX). Below that is a 'What's new' section with a note about AWS Cost Management providing purchase recommendations for Amazon 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 in the top navigation bar is 'DynamoDB > Tables > Create table'. The main title is 'Create table'. Under 'Table details', there is an 'Info' link. A note states: 'DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.' The 'Table name' field contains 'Users'. Below it, a note says: 'This will be used to identify your table.' The 'Partition key' section shows 'email' in the input field and 'String' as the type. A note below says: '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.' The 'Sort key - optional' section shows an empty input field with placeholder 'Enter the sort key name' and 'String' as the type. A note below says: '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.' At the bottom, notes say: '1 to 255 characters and case sensitive.'

### Activity 3.3: Create Appointments table (Primary key: appointment\_id).

The screenshot shows the Amazon DynamoDB console interface. On the left, there's a navigation sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a section for DAX with Clusters, Subnet groups, Parameter groups, and Events. The main content area is titled "Tables (2)" and lists two tables:

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 of the main area, there's a blue banner with the text "Share your feedback on Amazon DynamoDB" and "Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing." There are also "Notifications" and "Create table" buttons. The bottom of the screen shows the AWS navigation bar with CloudShell, Feedback, Search, and various service icons.

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

**Services**

Show more ▶

- 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**

Show more ▶

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

us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/homepage

Search [Alt+S]

**Amazon Simple Notification Service**

Pub/sub messaging for microservices and serverless applications.

Amazon SNS now supports High Throughput FIFO topics. Learn more [↗]

Application Integration

**Create topic**

**Topic name**  
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.

MyTopic

**Next step**

**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 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 sidebar with links like Dashboard, Topics (which is selected), Subscriptions, Mobile (Push notifications and Text messaging (SMS)), and a 'New Feature' banner about FIFO topics. The main area has a header 'Topics (0)' with buttons for Edit, Delete, Publish message, and Create topic. Below is a search bar and a table with columns Name, Type, and ARN. A message says 'No topics 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 title is 'Create topic'. Under 'Details', it says 'Type' and 'Info' (disabled). It notes 'Topic type cannot be modified after topic is created'. There are two options: 'FIFO (first-in, first-out)' and 'Standard'. 'Standard' is selected. Its description includes: 'Best-effort message ordering', 'At-least once message delivery', 'Highest throughput in publishes/second', and '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. The message states: "Subscription to Medtrack created successfully. The ARN of the subscription is arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb". Below this, the subscription details are listed:

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

Below the details, there are tabs for "Subscription filter policy" and "Redrive policy (dead-letter queue)". The status of the subscription is "Pending confirmation".

- **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**

**Protocol**  
The type of endpoint to subscribe

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

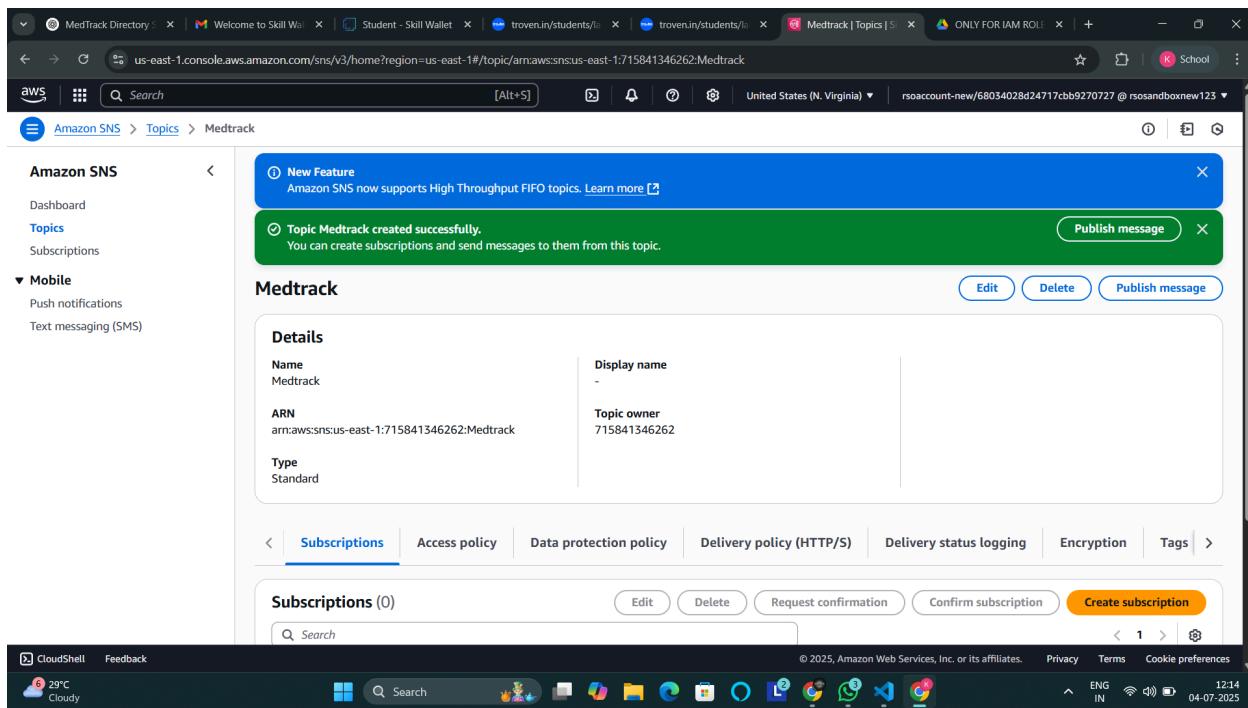
i After your subscription is created, you must confirm it. [Info](#)

**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 options like Dashboard, Topics, Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and CloudShell. The main area displays the 'Medtrack' topic details. At the top, there are two notifications: one about High Throughput FIFO topics and another stating 'Topic Medtrack created successfully.' Below this, the 'Medtrack' section shows the topic's details: Name (Medtrack), ARN (arn:aws:sns:us-east-1:715841346262:Medtrack), and Type (Standard). To the right of the details are buttons for Edit, Delete, and Publish message. Below the details, there are tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags. The Subscriptions tab is selected, showing a table with 0 rows. At the bottom of the table are buttons for Edit, Delete, Request confirmation, Confirm subscription, and Create subscription. The status bar at the bottom indicates it's 12:14, ENG IN, and the date is 04-07-2025.

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

AWS Notification - Subscription Confirmation External Spam

AWS Notifications <[no-reply@sns.amazonaws.com](mailto:no-reply@sns.amazonaws.com)>  
to me ▾ 12:22 PM (7 hours ago)

Why is this message in spam? This message is similar to messages that were identified as spam in the past.  
[Report not spam](#)

You have chosen to subscribe to the topic:  
**arn:aws:sns:us-east-1:715841346262:Medtrack**

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):  
[Confirm subscription](#)

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](mailto:sns-opt-out).

[Reply](#) [Forward](#)



## Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

**arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb**

If it was not your intention to subscribe, [click here to unsubscribe](#).

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

The screenshot shows the AWS SNS 'Subscription' details page for a specific subscription. The subscription ARN is `arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb`. The status is confirmed. The endpoint is `2251a4425@adityatekkali.edu.in`. The topic is `Medtrack`. The subscription principal is `arn:aws:iam::715841346262:role/rsoaccount-new`. There are tabs for 'Subscription filter policy' and 'Redrive policy (dead-letter queue)'. A blue banner at the top indicates that Amazon SNS now supports High Throughput FIFO topics.

## Milestone 5: IAM Role Setup

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

The screenshot shows the AWS search results for 'iam'. The search bar at the top contains 'Q iam'. The left sidebar has a 'Services' section with links to 'Features', 'Resources New', 'Documentation', 'Knowledge articles', 'Marketplace', 'Blog posts', 'Events', and 'Tutorials'. The main content area displays search results for 'iam' under the 'Services' heading. The first result is 'IAM' with the subtext 'Manage access to AWS resources'. Below it are 'IAM Identity Center' (Manage workforce user access to multiple AWS accounts and cloud applications), 'Resource Access Manager' (Share AWS resources with other accounts or AWS Organizations), and 'AWS App Mesh' (Easily monitor and control microservices). A 'Show more ▶' link is visible at the top right of the service list.

The screenshot shows the AWS IAM Dashboard. On the left, there's a sidebar with 'Access management' and 'Access reports' sections. The main area has a blue banner at the top stating 'New access analyzers available' and 'Access Analyzer now analyzes internal access patterns to your critical resources within a single account or across your entire organization.' Below this is the 'IAM Dashboard' section. It contains two red-bordered boxes: one for 'Access denied' (iam:GetAccountSummary) and another for 'Access denied' (iam>ListAccountAliases). Both boxes show the user, action, context, and a 'Diagnose with Amazon Q' button. To the right is the 'AWS Account' section, which also contains a red-bordered 'Access denied' box for iam>ListAccountAliases. At the bottom, there's a 'What's new' section and a 'Tools' section with a 'Policy simulator' link.

## ● Create IAM Roles:

The screenshot shows the 'Create role' wizard. The left sidebar shows 'Step 1: Select trusted entity' (selected), 'Step 2: Add permissions', and 'Step 3: Name, review, and create'. The main area is titled 'Name, review, and create' and contains a 'Role details' section. It includes fields for 'Role name' (set to 'EC2\_MedTrack\_Role'), 'Description' (set to 'Allows EC2 instances to call AWS services on your behalf.'), and a note about character limits. Below this is 'Step 1: Select trusted entities' with a 'Trust policy' section containing a JSON policy document. The policy document is as follows:

```
1+ [{
2+     "Version": "2012-10-17",
3+     "Statement": [
4+         {
5+             "Effect": "Allow",
6+             "Action": [
7+                 "sts:AssumeRole"
8+             ]
9+         }
10+    ]
11+ }
```

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

The screenshot shows the AWS IAM Roles page. A green banner at the top says "Role EC2\_MedTrack\_Role created." Below it, the heading "Roles (11) Info" is displayed. A table lists the roles, including the newly created one:

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)	-
<a href="#">EC2_MedTrack_Role</a>	AWS Service: ec2	-
OrganizationAccountAccessRole	Account: 058264256896	58 minutes ago

The screenshot shows the "Modify IAM role" page for the instance i-01e69ab66b8e6b6a4. The heading is "Modify IAM role Info". It says "Attach an IAM role to your instance." Below is a form:

**Instance ID**  
i-01e69ab66b8e6b6a4 (medtrack-server)

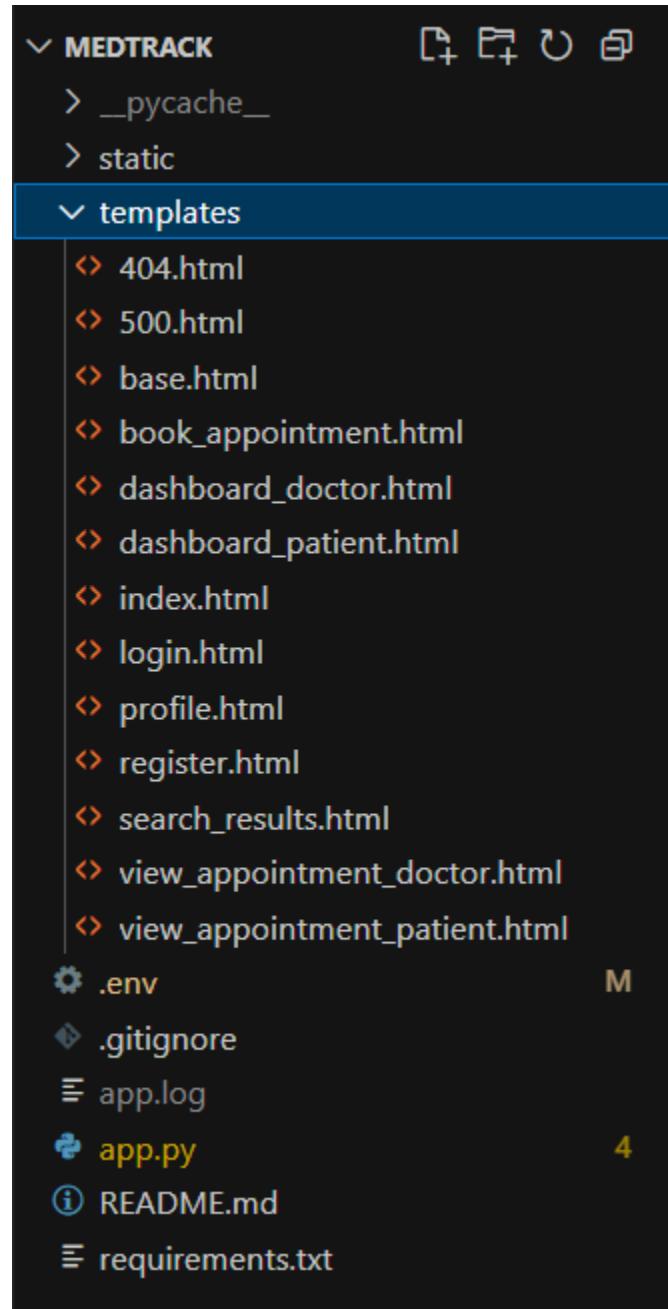
**IAM role**  
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

A dropdown menu shows "EC2\_MedTrack\_Role" selected. There is also a "Create new IAM role" button.

At the bottom are "Cancel" and "Update IAM role" buttons.

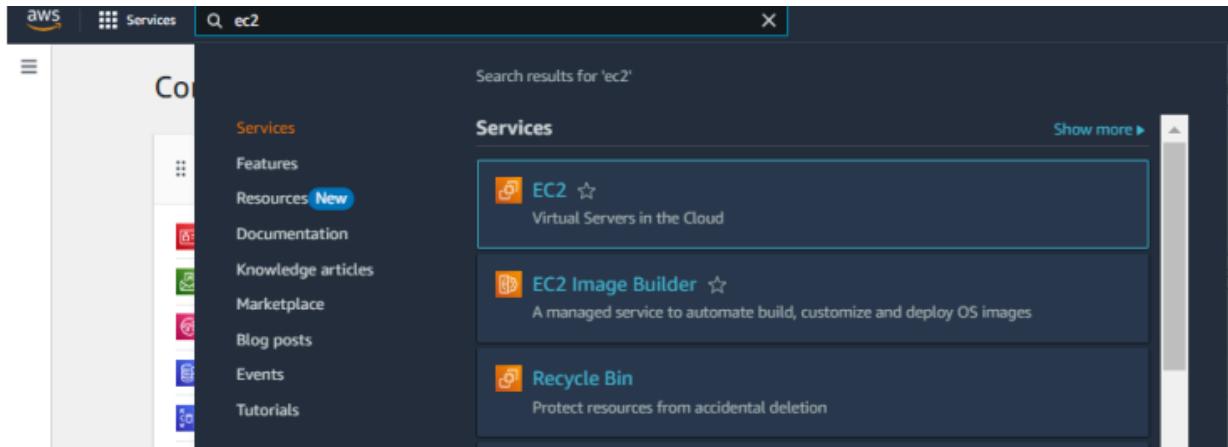
## Milestone 6: EC2 Instance Setup

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



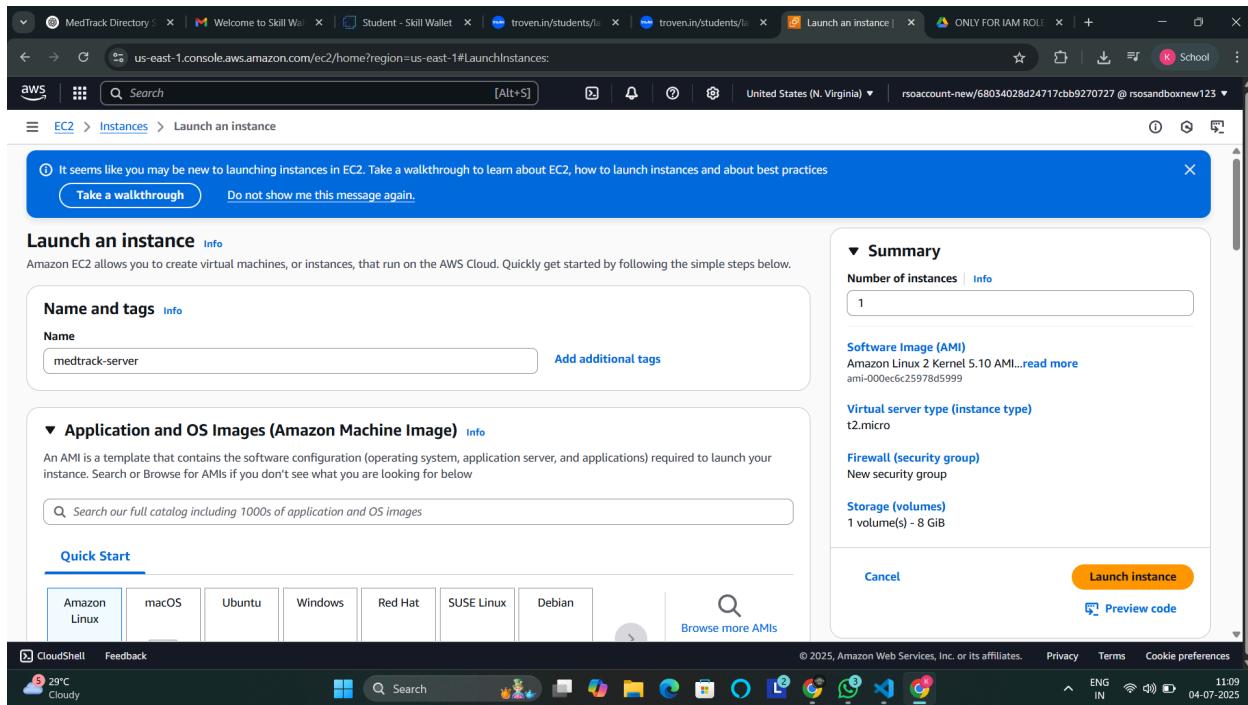
Launch an EC2 instance to host the Flask application.

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

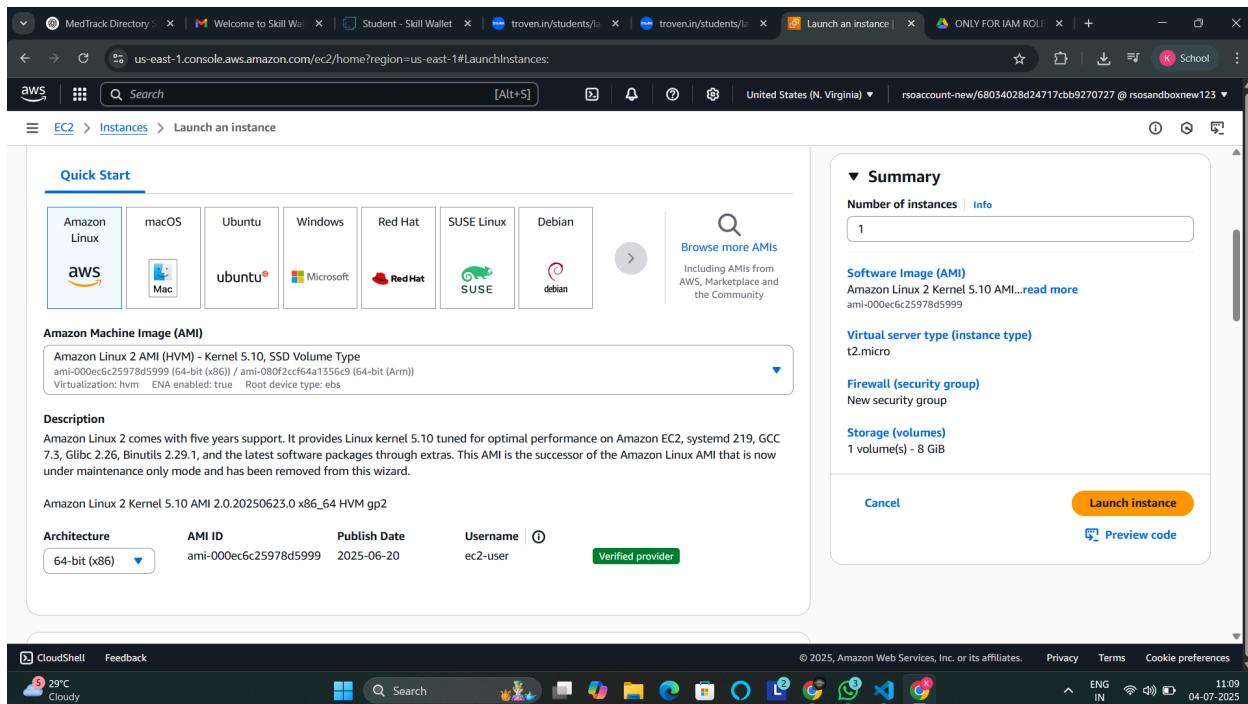


- Click on Launch instance to launch EC2 instance

A screenshot of the AWS EC2 Instances page. The URL in the browser is 'us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances'. The left sidebar shows navigation options: Dashboard, EC2 Global View, Events, Instances (selected), Images, and Elastic Block Store. The main content area is titled 'Instances Info' and displays a search bar with placeholder text 'Find Instance by attribute or tag (case-sensitive)'. It includes filters for 'Name' and 'Instance ID', and dropdowns for 'Instance state', 'Instance type', 'Status check', 'Alarm status', 'Availability Zone', and 'Public IP'. A message 'No instances' is shown, followed by the sub-message 'You do not have any instances in this region'. A prominent blue 'Launch instances' button is located below the search bar. At the bottom of the page, there are links for CloudShell, Feedback, and various system status indicators like temperature and battery level. The footer contains copyright information for Amazon Web Services, Inc., and links for Privacy, Terms, and Cookie preferences.



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



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

The screenshot shows the AWS EC2 'Launch an instance' wizard. The configuration includes:

- Instance type:** t2.micro (64-bit x86) - ami-000ec6c25978d5999 (2025-06-20)
- Key pair (login):** medtrack-server
- Network settings:** VPC: vpc-083322a7b5abe06cc (default)
- Summary:** Number of instances: 1
- Software Image (AMI):** Amazon Linux 2 Kernel 5.10 AMI... (ami-000ec6c25978d5999)
- Virtual server type (instance type):** t2.micro
- Firewall (security group):** New security group
- Storage (volumes):** 1 volume(s) - 8 GiB

Buttons at the bottom include 'Cancel', 'Launch instance', and 'Preview code'.

## ● Configure security groups for HTTP/SSH.

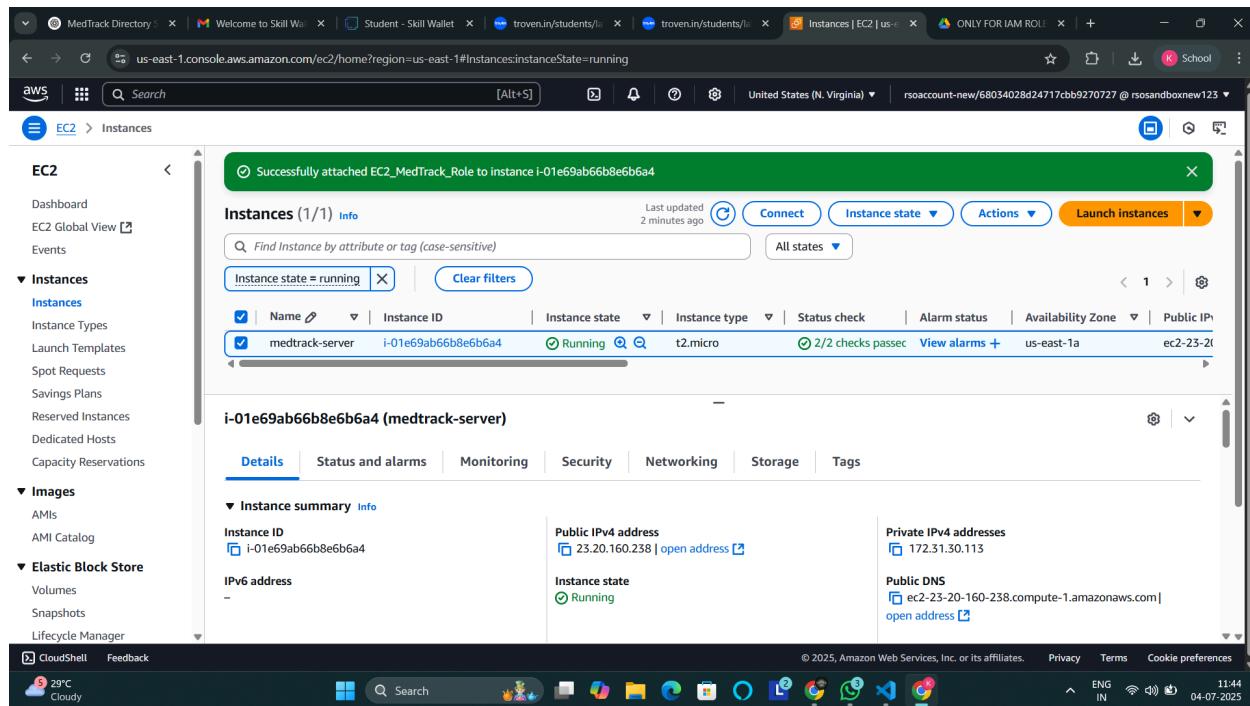
The screenshot shows the AWS EC2 'Launch an instance' wizard with the 'Inbound Security Group Rules' section expanded. It contains two rules:

- Security group rule 1 (TCP, 22, 157.50.147.110/32):**
  - Type: ssh
  - Protocol: TCP
  - Port range: 22
  - Name: Add CIDR, prefix list or security group (e.g. SSH for admin desktop)
  - Source type: My IP
  - Source value: 157.50.147.110/32
- Security group rule 2 (TCP, 80, 0.0.0.0/0):**
  - Type: HTTP
  - Protocol: TCP
  - Port range: 80
  - Name: Add CIDR, prefix list or security group (e.g. SSH for admin desktop)
  - Source type: Anywhere
  - Source value: 0.0.0.0/0

A warning message in a callout box states: "⚠️ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only."

Buttons at the bottom include 'Add security group rule', 'Cancel', 'Launch instance', and 'Preview code'.

- 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 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-30-113 ~]$ sudo su
[root@ip-172-31-30-113 ec2-user]# sudo su
[root@ip-172-31-30-113 ec2-user]# yum install python3
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
Package python3-3.7.16-1.amzn2.0.17.x86_64 already installed and latest version
Nothing to do
[root@ip-172-31-30-113 ec2-user]# yum install git
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
--> Package git.x86_64 0:2.47.1-1.1.amzn2.0.3 will be installed
--> Processing Dependency: git-core = 2.47.1-1.amzn2.0.3 for package: git-2.47.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl-Git = 2.47.1-1.amzn2.0.3 for package: git-2.47.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl(Git) for package: git-2.47.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl(Term::ReadKey) for package: git-2.47.1-1.amzn2.0.3.x86_64
--> Running transaction check

i-01e69ab66b8e6b6a4 (medtrack-server)
Public IPs: 23.20.160.238 Private IPs: 172.31.30.113

```

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 31°C Mostly cloudy ENG IN 13:00 04-07-2025

## 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/Saikamalsuro/MedTrack-.git’

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

- 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.

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if is_logged_in():
        return redirect(url_for('dashboard'))
    if request.method == 'POST':
        required_fields = ['name', 'email', 'password', 'confirm_password', 'age', 'gender', 'role']
        for field in required_fields:
            if not request.form.get(field):
                flash(f'Please enter {field}', 'danger')
                return render_template('register.html')
        if request.form['password'] != request.form['confirm_password']:
            flash('Passwords do not match', 'danger')
            return render_template('register.html')

        email = request.form['email'].lower()
        existing = user_table.get_item(key={'email': email}).get('Item')
        if existing:
            flash('Email already registered', 'danger')
            return render_template('register.html')

        user_data = {
            'email': email,
            'name': request.form['name'],
            'password': generate_password_hash(request.form['password']),
            'age': request.form['age'],
            'gender': request.form['gender'],
            'role': request.form['role'].lower(),
            'created_at': datetime.utcnow().isoformat()
        }
        user_table.put_item(Item=user_data)
```

- **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()
                },
            )
    
```

- **Search:** Filter appointments.

```

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

    if request.method == 'POST':
        search_term = request.form.get('search_term', '').strip()

    try:
        if session.get('role') == 'doctor':
            # Search patient's name for doctor
            filter_expr = boto3.dynamodb.conditions.Attr('doctor_email').eq(session['email']) & boto3.dynamodb.conditions.Attr('pa
            response = appointment_table.scan(FilterExpression=filter_expr)
        else:
            # Patient search doctor name or status
            filter_expr = (
                boto3.dynamodb.conditions.Attr('patient_email').eq(session['email']) &
                (
                    boto3.dynamodb.conditions.Attr('doctor_name').contains(search_term) |
                    boto3.dynamodb.conditions.Attr('status').contains(search_term)
                )
            )
        response = appointment_table.scan(FilterExpression=filter_expr)

        appointments = response.get('Items', [])
        return render_template('search_results.html', appointments=appointments, search_term=search_term)

    except Exception as e:
        logger.error(f"Search failed: {e}")
        flash('Search failed. Please try again.', 'danger')

    return redirect(url_for('dashboard'))

    return redirect(url_for('dashboard'))

```

## Profile: View/edit personal data:

```

@app.route('/profile', methods=['GET', 'POST'])
def profile():
    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.

# Top Pediatric Hospital in Vizag

Consult experienced pediatricians with 24+ years of care.

[Book an Appointment](#)

## Welcome to HealthCare App

Your digital bridge to qualified doctors, instant booking, and secured health records.

[Login](#)

[Register](#)



### Find Doctors

Browse our trusted network of certified medical professionals in various specialties.



### Easy Appointments

Schedule and manage your appointments with a few taps on your device.



### Digital Records

Safely access your medical reports and history anytime, from any device.

### How It Works

1. Register as a patient or a doctor
2. Login to your personalized dashboard
3. Patients can search for doctors and schedule appointments
4. Doctors manage schedules and provide feedback
5. Automatic email reminders and updates

## Welcome Back

Login to manage your healthcare dashboard

Patient

Doctor

 Email address

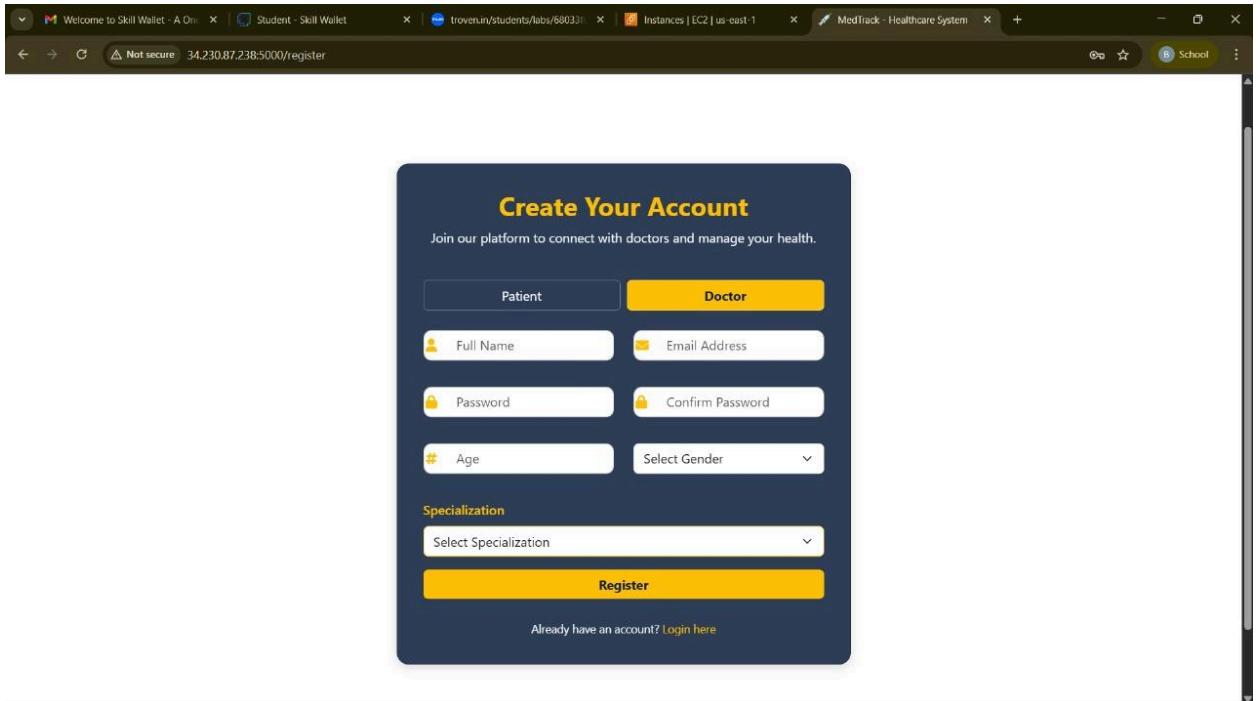
 Password

[Forgot password?](#)

Login

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

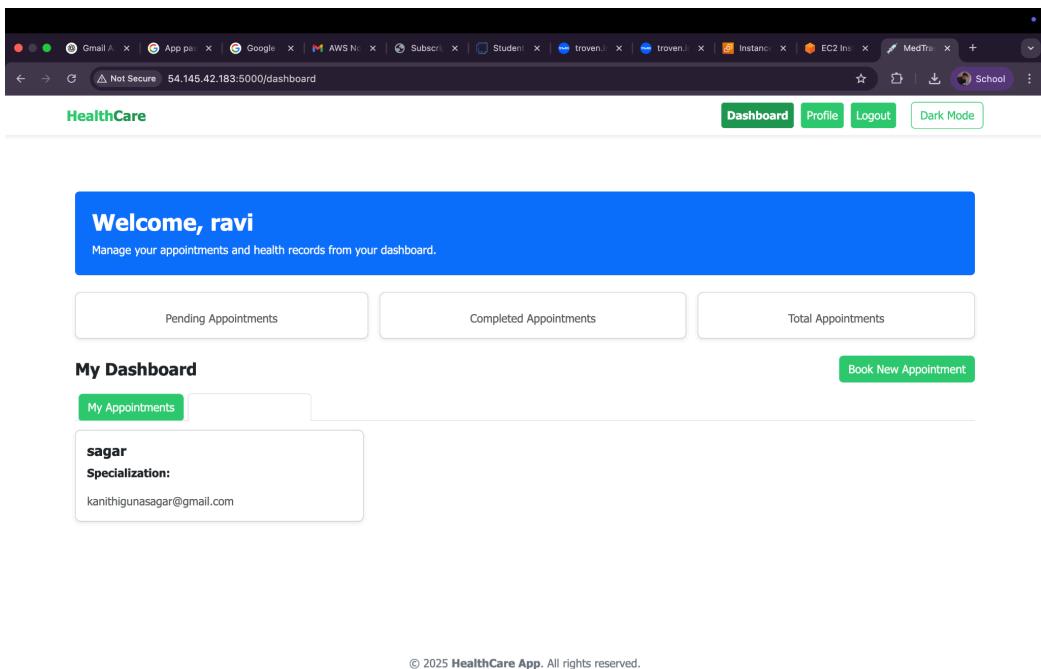
- **Doctor & Patient Registration:** Collects and validates credentials.



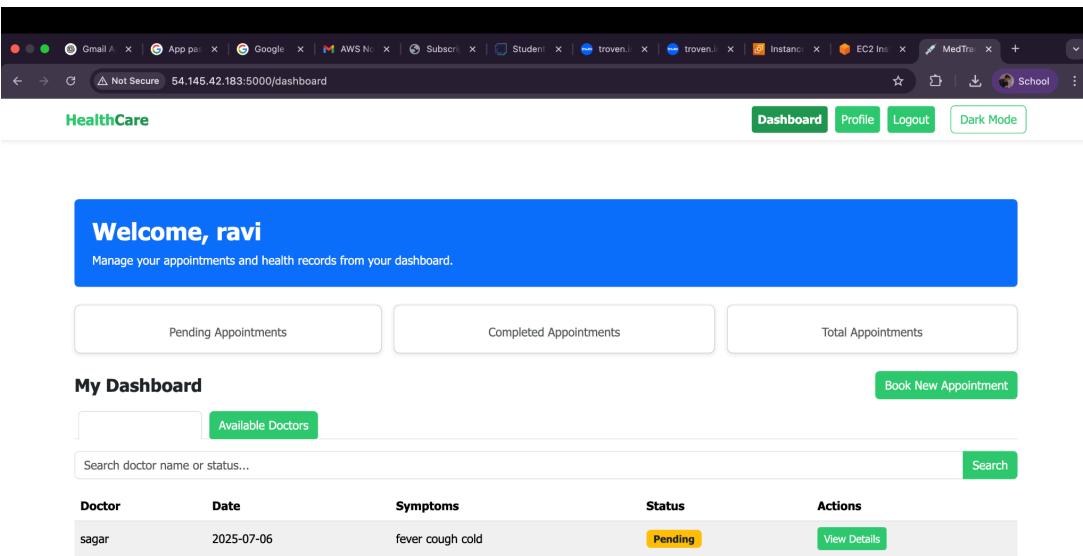
- **Login Pages:** Secures access and redirects to dashboards.
- **Dashboards:** Role-based UIs for managing appointments.

### → Doctor Dashboard

A screenshot of a web browser showing the "Doctor Dashboard" of the "HealthCare" application. The dashboard has a dark blue header with the "HealthCare" logo on the left and "Dashboard", "Profile", and "Logout" links on the right. A search bar at the top right contains the name "sagar" and a "Search" button. A prominent banner at the top says "Welcome, Dr. Kanithi GunaSagar" and "Manage your appointments and patient consultations from your dashboard." Below the banner, a section titled "Doctor Dashboard" displays three summary cards: "Pending Appointments" (0), "Completed Appointments" (2), and "Total Appointments" (2). Below these cards is a table with three tabs: "Pending Appointments", "Completed Appointments", and "All Appointments". The "Completed Appointments" tab is currently active, showing two rows of appointment details. Each row includes columns for "Patient Name" (gayatri, sagar), "Date" (2025-07-09, 2025-07-24), "Symptoms" (fever,cold, kidney prblem), "Status" (Completed, Completed), and an "Actions" column with a "View Details" button. At the bottom of the page, a footer bar contains the copyright notice "© 2025 HealthCare App. All rights reserved.".



## → Patient Dashboard



© 2025 HealthCare App. All rights reserved.

- **Search Feature:** Enables real-time filtering by status/date.

## Database Updates

### Users Table:

- Add a new user (doctor/patient)
- Update Profile
- Track active/inactive status.

The screenshot shows the AWS DynamoDB console interface. The URL is `us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#item-explorer?maximize=true&operation=SCAN&table=UsersTable`. The left sidebar shows tables: AppointmentsTable (unchecked) and UsersTable (checked). The main area has tabs for Scan (selected) and Query. Under Scan, it says "Select a table or index: Table - UsersTable" and "Select attribute projection: All attributes". Below that is a "Filters - optional" section with a Run button. A green message bar at the bottom says "Completed - Items returned: 4 · Items scanned: 4 · Efficiency: 100% · RCU consumed: 2". The table below is titled "Table: UsersTable - Items returned (4)" and shows the following data:

	email (String)	age	created_at	gender	name	password	role
<input type="checkbox"/>	22a51a4446@adityat...	43	2025-07-05...	Male	kamal	pbkdf2:sha...	doctor
<input type="checkbox"/>	ravikira901@gmail.com	33	2025-07-05...	Male	ravi	pbkdf2:sha...	patient
<input type="checkbox"/>	sagarkanithi12345@g...	54	2025-07-05...	Male	gunasagar	pbkdf2:sha...	patient
<input type="checkbox"/>	kanithigunasagar@g...	25	2025-07-05...	Male	sagar	pbkdf2:sha...	doctor

### Appointments Table:

- Create new appointment
- Update Status

- Maintain history

The screenshot shows the AWS DynamoDB console with the following details:

- Left Sidebar:** Shows 'Any tag key' dropdown, 'Any tag value' dropdown, and a 'Find tables' search bar. Below it are two tables: 'AppointmentsTable' (selected) and 'UsersTable'.
- Main Panel:**
  - Scan or query items:** A section with 'Scan' (selected) and 'Query' buttons, 'Select a table or index' dropdown set to 'Table - AppointmentsTable', and 'Select attribute projection' dropdown set to 'All attributes'.
  - Filters - optional:** A section with 'Run' and 'Reset' buttons.
  - Success Message:** A green box at the bottom left says 'Completed - Items returned: 2 · Items scanned: 2 · Efficiency: 100% · RCUs consumed: 2'.
- Results Table:**

Table: AppointmentsTable - Items returned (2)					
Scan started on July 05, 2025, 12:01:57					
	appointment_id (String)	appointment_date	created_at	doctor_email	doctor_name
<input type="checkbox"/>	dfef948-53d4-4fd1-8c1f-ec...	2025-07-09	2025-07-05...	kanithigunasag...	sagar
<input type="checkbox"/>	dfd9ab20-aed2-46b1-881e-...	2025-07-06	2025-07-05...	kanithigunasag...	ravikira901@gm...

## Notifications

- Email confirmation to patients upon booking.



Welcome to HealthCare App Inbox x

22a51a4433@adityatekkali.edu.in

to me ▾

Hello Ravi Kiran, your account was created successfully.

Reply

Forward

- Email alerts to doctors/admins upon new appointments.



## Appointment Confirmation Inbox ×

**22a51a4433@adityatekkali.edu.in**

to me ▾

Dear ravi,

Your appointment with Dr. sagar has been successfully booked on 2025-07-06.

← Reply

→ Forward



---

## 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.