

# 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

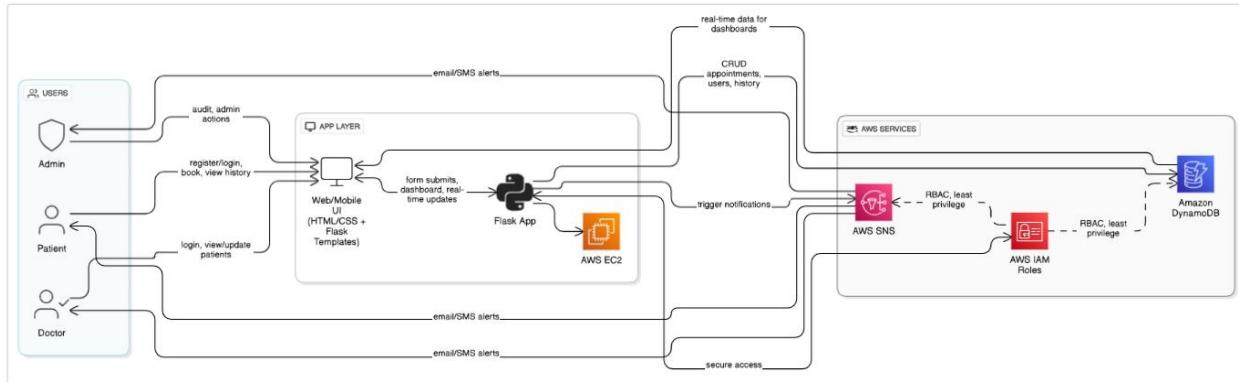
MedTrack uses IAM roles for secure, role-based access control. Patients and doctors receive permissions specific to their roles, ensuring secure access to sensitive data.

### 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.

---

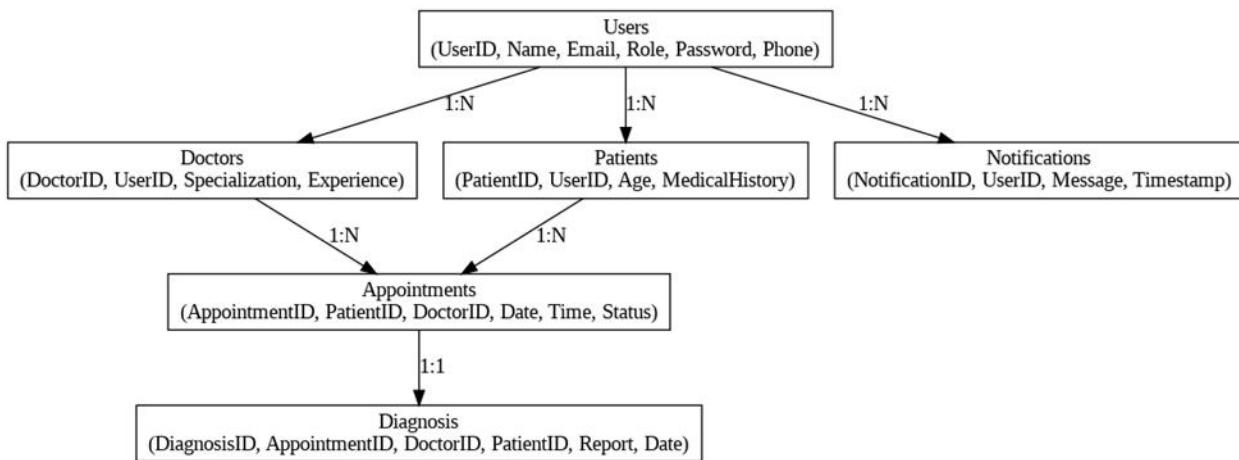
# AWS Architecture



- **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.

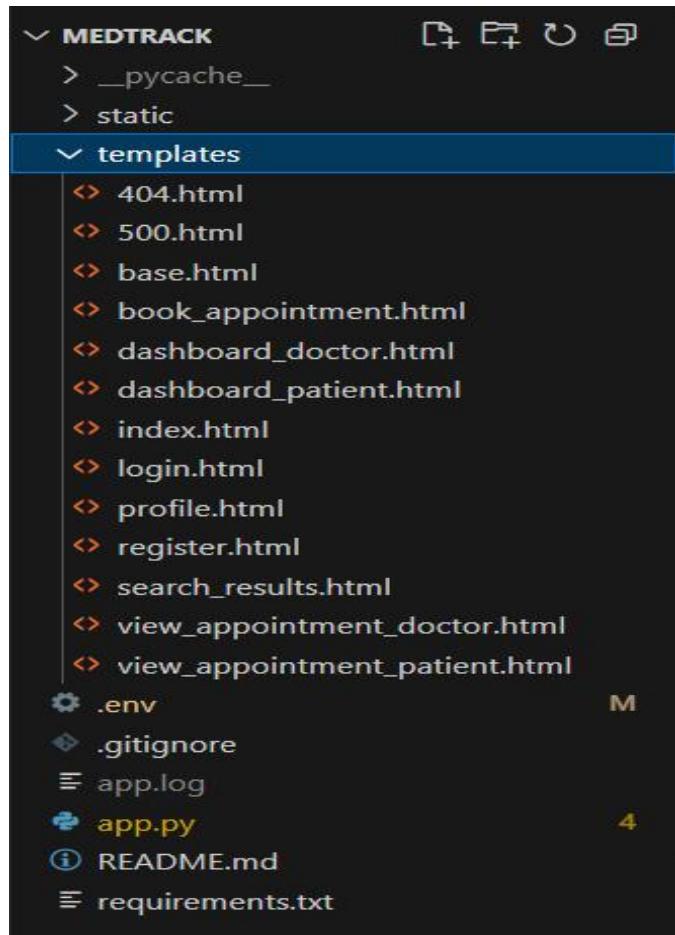
- Run Flask app: sudo flask run --host=0.0.0.0 --port=5000

## 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 is a sidebar with 'troven' logo, 'Labs' button, and 'Log out' link. The main area has a header '← AWS - MedTrack Cloud based Patient Medication Tracker' with a 'Back' link. Below is a table with columns: Platform (AWS), Lab (1), Duration (4 hour(s) 0 minute(s)), Difficulty (Expert), and Progress (AWS). Under 'Overview', it says 'MedTrack is a cloud-native healthcare app that helps users manage and track their medication schedules. Built with Flask on Amazon EC2, it uses DynamoDB for data storage, SNS for email reminders, and IAM roles for secure AWS service integration.' Under 'Skills', there's a bulleted list: EC2, Database on AWS, IAM, Monitoring and Observability. Under 'Lab', there's a box titled 'AWS Final Deployment' for 'AWS - MedTrack Cloud based Patient Medication Tracker'. It shows 'AWS' under 'Status' and '1/5 Tasks Validated Attempted' under 'Progress'.

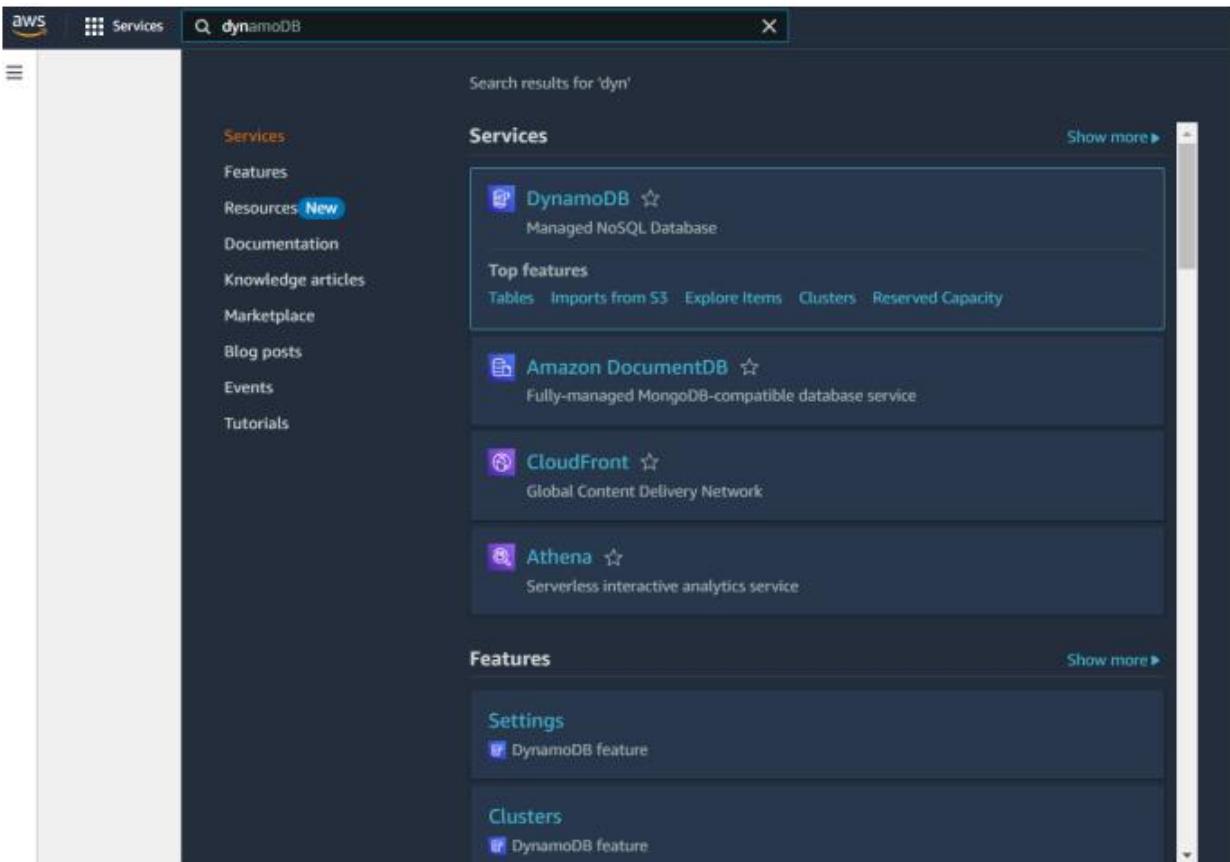
- Avoid personal AWS account usage to prevent billing issues.

---

## 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 DynamoDB Dashboard. The left sidebar includes links for 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations New', 'Reserved capacity', and 'Settings'. A 'DAX' section is also present with links for 'Clusters', 'Subnet groups', 'Parameter groups', and 'Events'. The main dashboard area has two main sections: 'Alarms (0)' and 'DAX clusters (0)'. Both sections include search bars, status indicators, and 'Create' buttons. To the right, there is a 'Create resources' section for creating a new DynamoDB table, a 'What's new' section with a recent update about AWS Cost Management, and a 'Create DAX cluster' button.

**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 navigation bar at the top indicates the user is in the 'Tables' section under 'DynamoDB'. The main title is 'Create table'. Below it, the 'Table details' section is active, with a sub-section titled 'Table name'. A text input field contains the value 'Users'. A note below the input field specifies that the table name must be between 3 and 255 characters, containing letters, numbers, underscores, hyphens, and periods. The 'Partition key' section follows, with a text input field containing 'email' and a dropdown menu set to 'String'. A note below the input field states that the partition key is a hash value used for retrieving items and allocating data across hosts. The 'Sort key - optional' section is present but empty, with a note explaining that it allows sorting or searching among items sharing the same partition key.

DynamoDB > Tables > Create table

## Create table

**Table details** Info

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

**Table name**  
This will be used to identify your table.

Users

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

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

email

String

1 to 255 characters and case sensitive.

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

Enter the sort key name

String

1 to 255 characters and case sensitive.

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

The screenshot shows the AWS DynamoDB Tables page. A green banner at the top right says "The AppointmentsTable table was created successfully." The main area displays a table with two rows:

	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capac
<input type="checkbox"/>	<a href="#">AppointmentsTable</a>	<span>Active</span>	appointment_id (\$)	-	0	0	<span>Off</span>	<span>☆</span>	On-demand
<input type="checkbox"/>	<a href="#">UsersTable</a>	<span>Active</span>	email (\$)	-	0	0	<span>Off</span>	<span>☆</span>	On-demand

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.

The screenshot shows the AWS search interface with the query 'sns' entered in the search bar. The results are categorized under 'Services' and 'Features'.

**Services**

- Simple Notification Service (SNS) - 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

**Show more ▶**

The screenshot shows the AWS EC2 landing page. The left sidebar includes navigation links for EC2, Dashboard, EC2 Global View, Events, Instances, Instances Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, and AMI Catalog. The main content area features a large heading 'Amazon Elastic Compute Cloud (EC2)' and sub-headings 'Create, manage, and monitor virtual servers in the cloud.' and 'Benefits and features'. A callout box on the right says 'Launch a virtual server' with 'Launch instance' and 'View dashboard' buttons. Another callout box at the bottom right says 'Get started' with 'Get started walkthroughs' and 'Get started tutorial' buttons.

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

The screenshot shows the Amazon SNS Topics page. A blue banner at the top left says "New Feature" and "Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more". Below the banner, there's a search bar and a table header with columns for "Name" and "Type". A message below the table says "No topics" and "To get started, create a topic." A prominent orange "Create topic" button is located at the bottom right of the table area.

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

The screenshot shows the "Create topic" page. At the top, the breadcrumb navigation is "Amazon SNS > Topics > Create topic". Below it, the title "Create topic" is displayed. Underneath the title, there's a "Details" section. In this section, there are two tabs: "Type" (selected) and "Info". A note below the tabs says "Topic type cannot be modified after topic is created". There are two options: "FIFO (first-in, first-out)" and "Standard". The "Standard" option is selected and highlighted with a blue border. Both options have associated bullet-point lists describing their features.

Type	Info
<input type="radio"/> FIFO (first-in, first-out) <ul style="list-style-type: none"><li>Strictly-preserved message ordering</li><li>Exactly-once message delivery</li><li>High throughput, up to 300 publishes/second</li><li>Subscription protocols: SQS</li></ul>	<input checked="" type="radio"/> Standard <ul style="list-style-type: none"><li>Best-effort message ordering</li><li>At-least once message delivery</li><li>Highest throughput in publishes/second</li><li>Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints</li></ul>

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

The screenshot shows the AWS EC2 Instances page. A green success message at the top states: "Successfully replaced EC2\_MedTrack\_Role with EC2\_MedTrack\_Role on instance i-01e0509dd662ee181". Below this, the "Instances (1/1) Info" section displays a single instance named "medtrack-server" with the ID "i-01e0509dd662ee181". The instance is listed as "Running" with the type "t2.micro". It has 2/2 checks passed and is located in the "us-east-1c" availability zone, with the public IP "ec2-35-1". The "Details" tab is selected, showing the instance summary with fields like Instance ID (i-01e0509dd662ee181), Public IPv4 address (35.175.196.185), Private IPv4 address (172.31.31.251), and Public DNS (ec2-35-175-196-185.compute-1.amazonaws.com).

- **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.

The screenshot shows the "Create subscription" page in the AWS SNS console. The "Details" section is filled out with the following information:

- Topic ARN:** arn:aws:sns:us-east-1:692768080016:aws-new-feature-updates
- Protocol:** Email
- Endpoint:** jbarr@amazon.com

A note at the bottom of this section states: "After your subscription is created, you must confirm it." Below this, there are two optional sections: "Subscription filter policy - optional" (with a note about filtering messages) and "Redrive policy (dead-letter queue) - optional" (with a note about sending undeliverable messages to a dead-letter queue). At the bottom right are "Cancel" and "Create subscription" buttons.

After the subscription request for the mail confirmation.

The screenshot shows the AWS SNS console with a newly created topic named 'Medtrack'. A green banner at the top right indicates that the topic was created successfully. The 'Subscriptions' tab is selected, showing 0 subscriptions. The topic details include Name: Medtrack, ARN: arn:aws:sns:us-east-1:715841346262:Medtrack, and Type: Standard. The topic owner is listed as 715841346262. The interface includes standard AWS navigation elements like CloudShell and Feedback, and a Windows taskbar at the bottom.

**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 body contains the following text:

You have chosen to subscribe to the topic:  
arn:aws:sns:us-east-1:149536455348: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](#)

At the bottom, there are 'Reply', 'Forward', and 'Compose' buttons.



## 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](#).

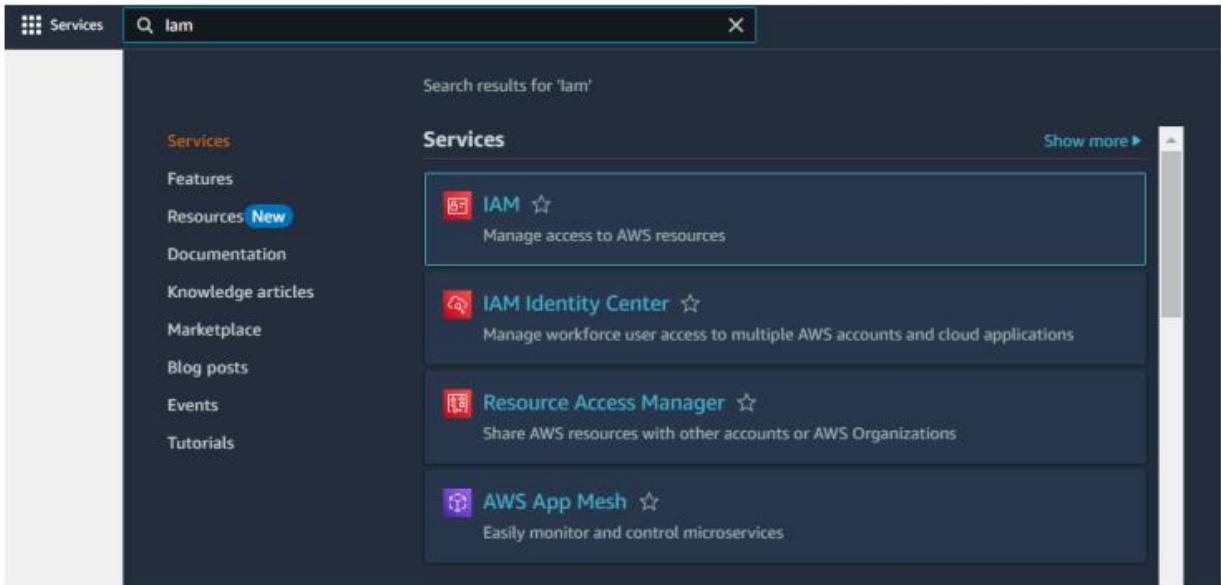
- Successfully done with the SNS mail subscription and setup, now store the ARN link.

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

- Amazon SNS > Topics > Medtrack > Subscription: 837e417d-317b-4b43-97c3-054566ac3bbb**
- New Feature:** Amazon SNS now supports High Throughput FIFO topics. [Learn more](#)
- Subscription: 837e417d-317b-4b43-97c3-054566ac3bbb**
- Details:**
  - ARN:** arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb
  - Status:** Confirmed
  - Protocol:** EMAIL
  - Endpoint:** 22a51a4425@adityatekkali.edu.in
  - Topic:** Medtrack
  - Subscription Principal:** arn:aws:iam::715841346262:role/rsoaccount-new
- Subscription filter policy:** Info. This policy filters the messages that a subscriber receives.
- Subscription filter policy (dead-letter queue):** Redrive policy (dead-letter queue)

## Milestone 5: IAM Role Setup

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



The screenshot shows the AWS IAM Dashboard. The left sidebar includes links for Identity and Access Management (IAM), Dashboard, Access management (User groups, Users, Roles, Policies, Identity providers, Account settings, Root access management), and Access reports (Access Analyzer, Resource analysis, Unused access, Analyzer settings, Credential report, Organization activity). The main content area features a blue banner for 'New access analyzers available'. Below it is the 'IAM Dashboard' section with tabs for IAM resources, AWS Account, and Tools. The IAM resources tab shows an 'Access denied' alert for a user attempting to get account summary. The AWS Account tab shows an 'Access denied' alert for listing account aliases. The Tools tab includes a 'Policy simulator' link.

## ● Create IAM Roles:

Step 1 **Select trusted entity**

- Step 2
- Add permissions
- Step 3  
Name, review, and create

**Select trusted entity** Info

**Trusted entity type**

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

**Use case**  
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

**Service or use case**

EC2

Choose a use case for the specified service.

**Use case**

**EC2**  
Allows EC2 instances to call AWS services on your behalf.

CloudShell Feedback 29°C Search © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG 11:29

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

aws us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ModifyIAMRole:instancetype=i-01e69ab66b8e6b6a4

**Modify IAM role** Info

Attach an IAM role to your instance.

**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.

EC2\_MedTrack\_Role

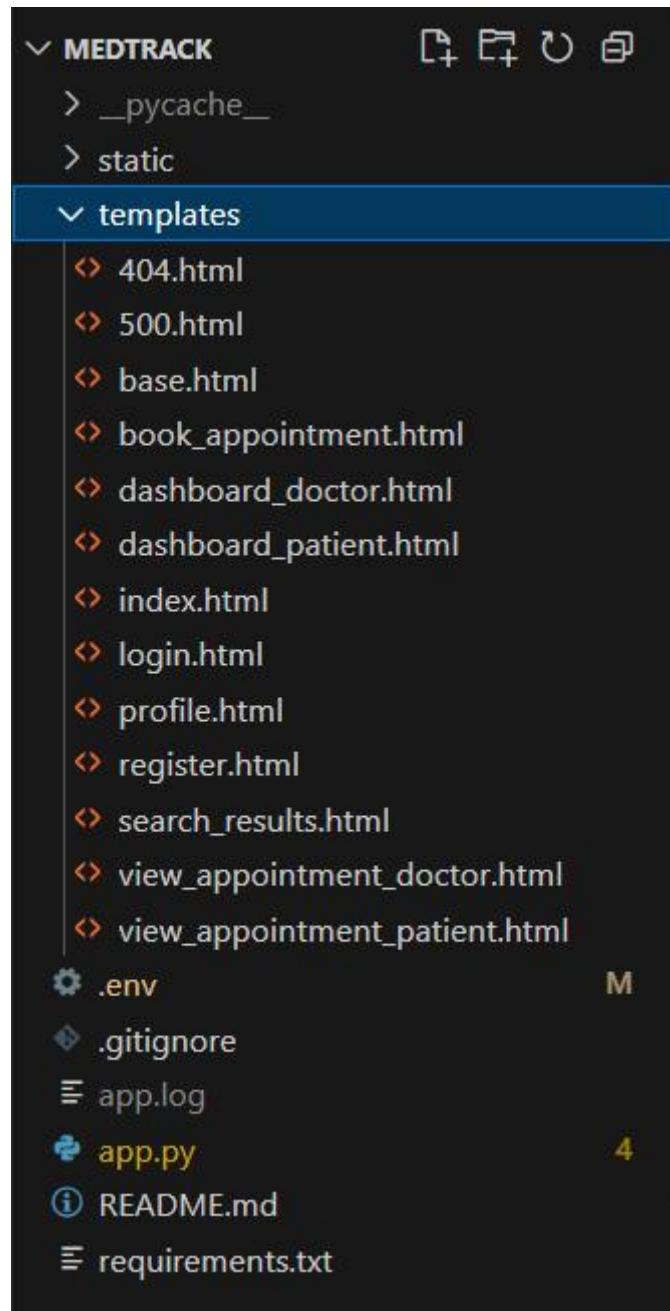
CloudShell Feedback 29°C Search © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 11:44 04-07-2025

Analyzer settings  
Credential report  
CrossRegion activity

OrganizationAccountAccessRole Account: 058264256896 58 minutes ago

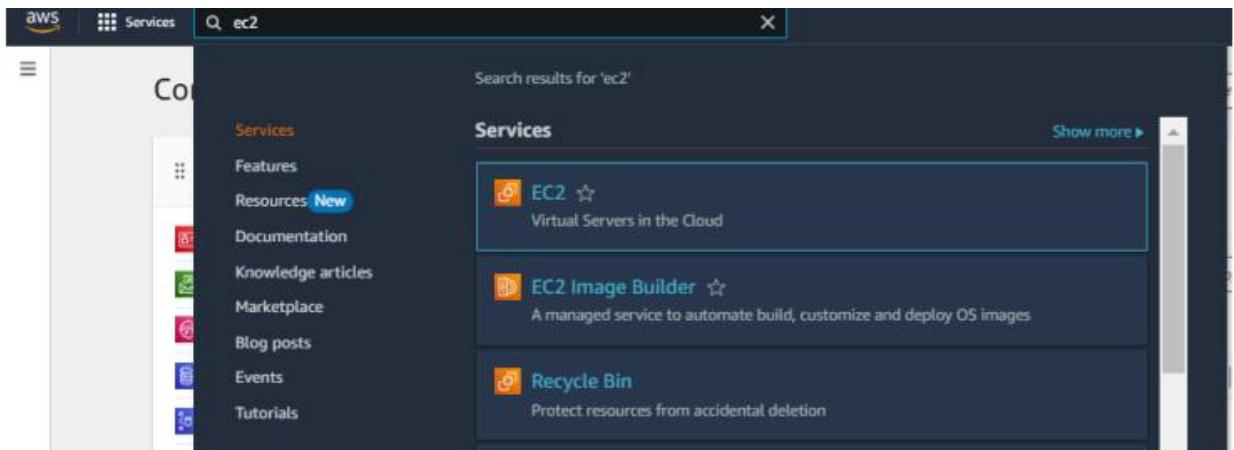
## Milestone 6: EC2 Instance Setup

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

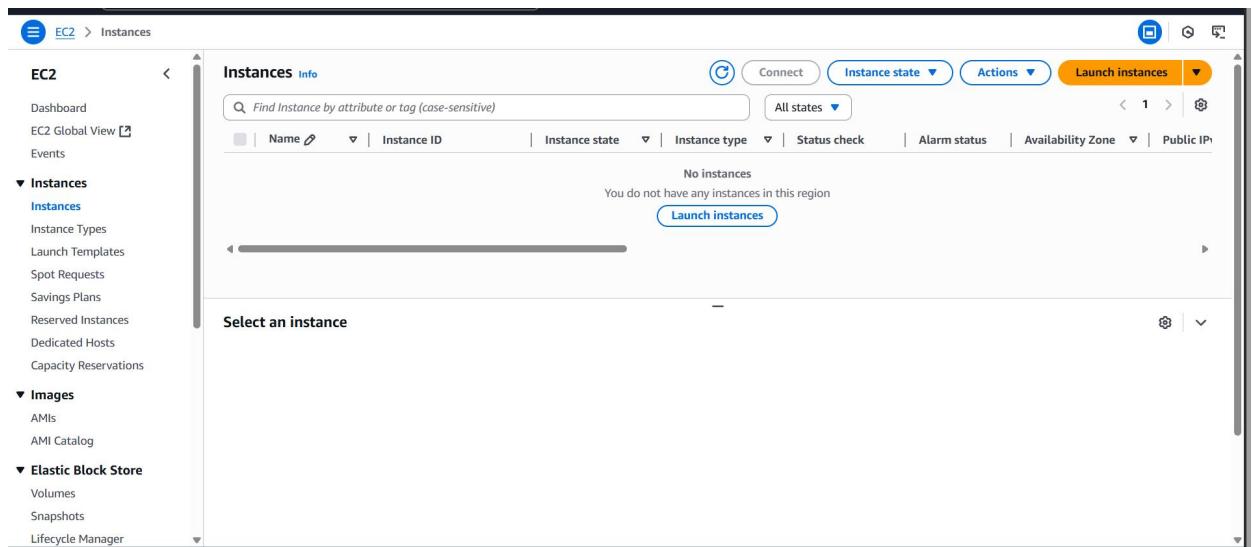


Launch an EC2 instance to host the Flask application.

- 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).

It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices

[Take a walkthrough](#) [Do not show me this message again.](#)

### Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

#### Name and tags [Info](#)

Name  Add additional tags

#### Summary

Number of instances [Info](#)

1

#### Application and OS Images (Amazon Machine Image) [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

**Quick Start**

[Amazon Linux](#) [macOS](#) [Ubuntu](#) [Windows](#) [Red Hat](#) [SUSE Linux](#) [Debian](#) [Browse more AMIs](#)

[Cancel](#)

[Launch instance](#)

[Preview code](#)

EC2 > Instances > Launch an instance

#### Application and OS Images (Amazon Machine Image) [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

**Quick Start**

[Amazon Linux](#) [macOS](#) [Ubuntu](#) [Windows](#) [Red Hat](#) [SUSE Linux](#) [Debian](#) [Browse more AMIs](#)

**Amazon Machine Image (AMI)**

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type  
ami-000ec6c25978d5999 (64-bit (x86)) / ami-080f2ccf64a1356c9 (64-bit (Arm))  
Virtualization: hvm ENA enabled: true Root device type: ebs

**Description**

Amazon Linux 2 comes with five years support. It provides Linux kernel 5.10 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is now under maintenance only mode and has been removed from this wizard.

Amazon Linux 2 Kernel 5.10 AMI 2.0.20250623.0 x86\_64 HVM gp2

#### Summary

Number of instances [Info](#)

1

[Cancel](#)

[Launch instance](#)

[Preview code](#)

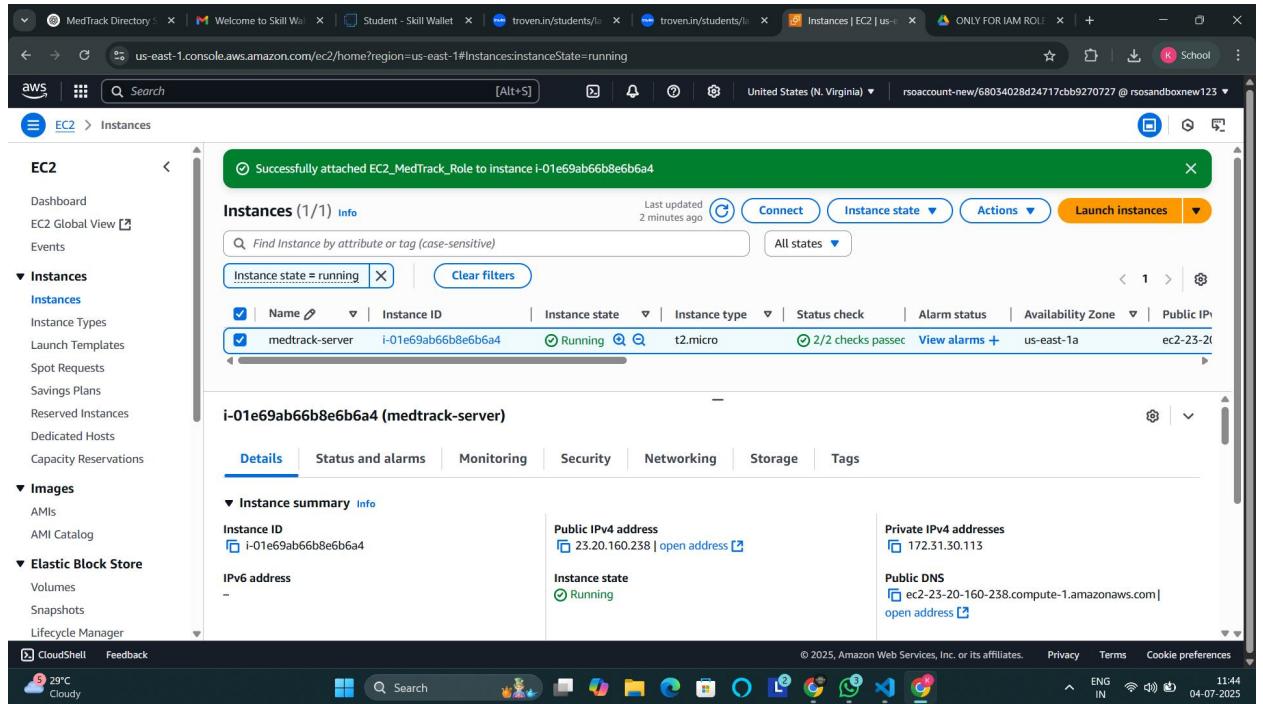
- Assign an IAM Role and key pair.

The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The 'Instance type' section is set to 't2.micro'. The 'Key pair (login)' section has 'medtrack-server' selected. In the 'Network settings' section, the VPC is 'vpc-08d64be79c1a632e3', the subnet is 'No preference', and the availability zone is 'us-west-2a'. On the right, the 'Summary' panel shows 1 instance, AMI 'Amazon Linux 2 Kernel 5.10 AMI...', and a 'Launch instance' button.

- Configure security groups for HTTP/SSH.

The screenshot shows the 'Inbound Security Group Rules' section of the 'Launch an instance' wizard. It contains two rules: one for SSH (TCP port 22) from 'My IP' and another for HTTP (TCP port 80) from 'Anywhere'. A note at the bottom 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.' On the right, the 'Summary' panel shows 1 instance, AMI 'Amazon Linux 2 Kernel 5.10 AMI...', and a 'Launch instance' button.

- 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.

```

  ^__\ #####
  ~~ \####|
  ~~ \|##| AL2 End of Life is 2026-06-30.
  ~~ \|/| |
  ~~ \|/| V-->
  ~~ \|/| /| 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.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: git-core-doc = 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

```

## 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/pavanichebolu/MEDTRACK.git>

Note: change your-github-username and your-repository-name with your credentials.  
here: ‘git clone https://github.com/pavanichebolu/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://192.168.191.91:5000>

## 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()
                },
            )
    except Exception as e:
        flash(str(e), 'danger')
        return redirect(url_for('dashboard'))

```

- **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('patient_name').contains(search_term)
            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():
    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:

```

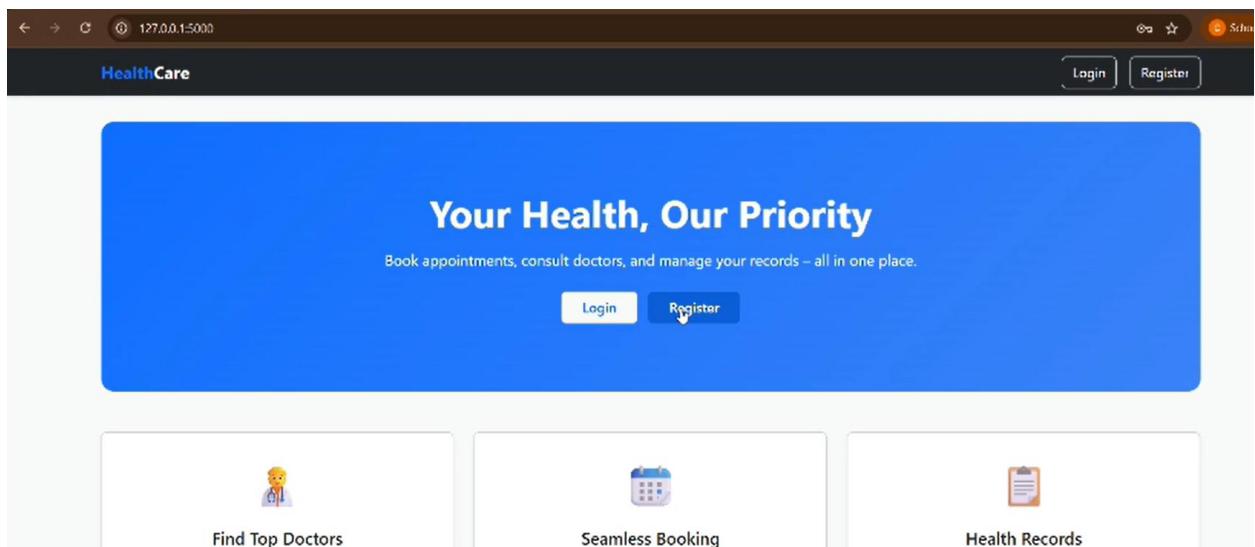
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5000))
    debug_mode = os.environ.get('FLASK_ENV', '') == 'development'
    app.run(host='0.0.0.0', port=port, debug=debug_mode)

```

---

### Functional Testing Summary

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



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

The screenshot shows the login page of the "HealthCare" application. The header "HealthCare" is visible. The main title is "Login to Your Account". A yellow notification bar at the top says "Registration successful! Your User ID is: DOC21104" with a close button. Below it are two buttons: "Patient" (blue background with a person icon) and "Doctor" (light blue background with a heart rate monitor icon). The login form has fields for "Email Address" containing "doc@gamil.com" and "Password" (a masked field). There is a "Show Password" checkbox and a "Login" button with a cursor icon. At the bottom, there is a link "Don't have an account? Register here".

- **Login Pages:** Secures access and redirects to dashboards.

**HealthCare**

Appointment booked successfully.

Welcome, Chebolu Pavani  
Manage your appointments and health records from your dashboard.

My Dashboard

Book New Appointment

1 Pending Appointments	0 Completed Appointments	1 Total Appointments
------------------------	--------------------------	----------------------

My Appointments Available Doctors

Search doctor name or status... Search

Doctor	Date	Time	Status	Actions
Dr. Chebolu Pavani	2025-07-05	Pause (Ctrl+P)	Pending	<a href="#">View Details</a>

- **Dashboards:** Role-based UIs for managing appointments.  
→ Doctor Dashboard

**HealthCare**

Appointment booked successfully.

Welcome, Chebolu Pavani  
Manage your appointments and health records from your dashboard.

My Dashboard

Book New Appointment

1 Pending Appointments	0 Completed Appointments	1 Total Appointments
------------------------	--------------------------	----------------------

My Appointments Available Doctors

Search doctor name or status... Search

Doctor	Date	Time	Status	Actions
Dr. Chebolu Pavani	2025-07-05	Pause (Ctrl+P)	Pending	<a href="#">View Details</a>

- Patient Dashboard

**Appointment Summary**

**Doctor & Appointment Details**

**Doctor:** Dr. Chebolu Pavani  
**Status:** Completed  
**Date:** 2025-07-05  
**Time:** 10:37  
**Booked On:** 2025-07-05 10:36:31.715307

**Your Reported Symptoms**  
Heart Pain

**Diagnosis**  
normal due to stress

**Treatment Plan**  
meditation

00:03:26 ——————> 00:00:07

- **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 with the 'Explore items' feature for the 'UsersTable'. The table has 4 items returned. The attributes for each item are:

	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

DynamoDB > Explore items > AppointmentsTable

**Scan or query items**

Scan      Query

Select a table or index: AppointmentsTable      Select attribute projection: All attributes

Filters - optional

Run      Reset

Completed · Items returned: 2 · Items scanned: 2 · Efficiency: 100% · RCU consumed: 2

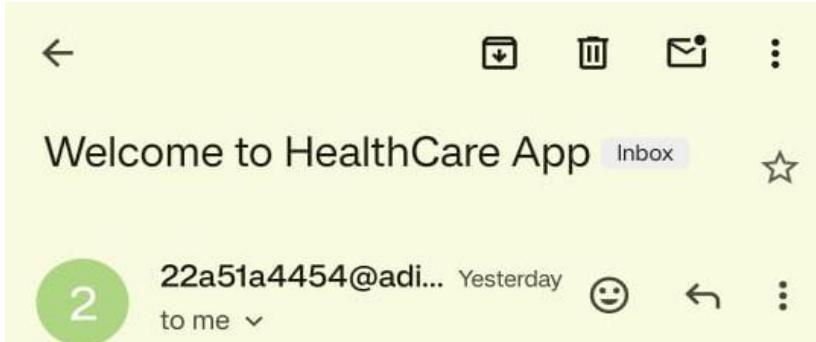
**Table: AppointmentsTable - Items returned (2)**

Scan started on July 05, 2025, 12:01:57

appointment_id	appointment_date	created_at	doctor_email	doctor_name	patient_email	patient_name
dfefd948-53d4-4fd1-8c1f-ec...	2025-07-09	2025-07-05...	kanithigunasag...	sagar	sagarkanithi123...	g
dfd9ab20-aed2-46b1-881e...	2025-07-06	2025-07-05...	kanithigunasag...	sagar	ravikira901@gm...	r

## Notifications

- Email confirmation to patients upon booking.



- Email alerts to doctors/admins upon new appointment.

## **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.