

# 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

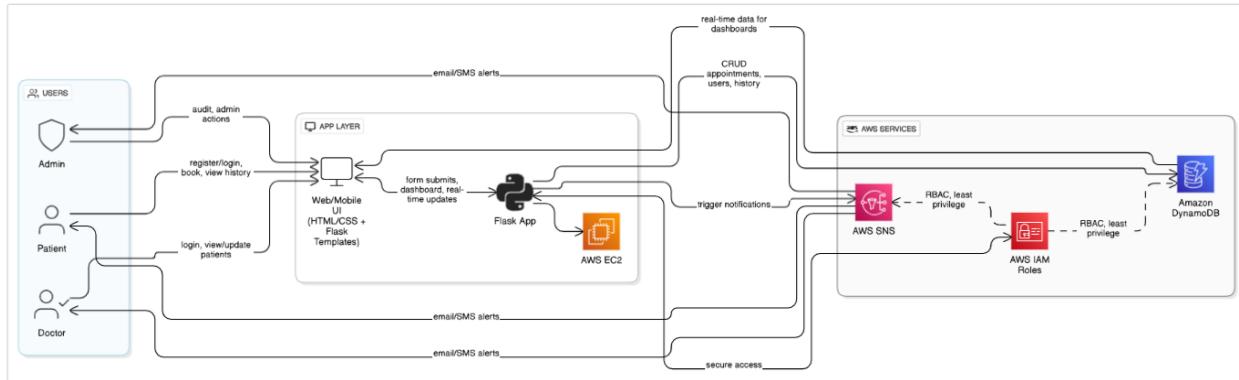
MedTrack uses IAM roles for secure, role-based access control. Patients and doctors receive permissions specific to their roles, ensuring secure access to sensitive data.

### 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.

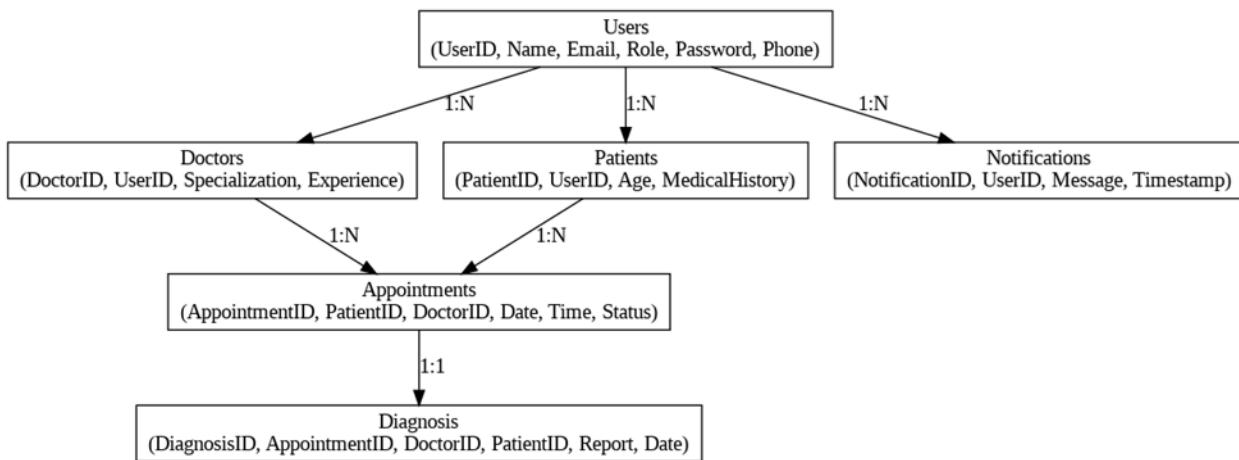
---

# 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.

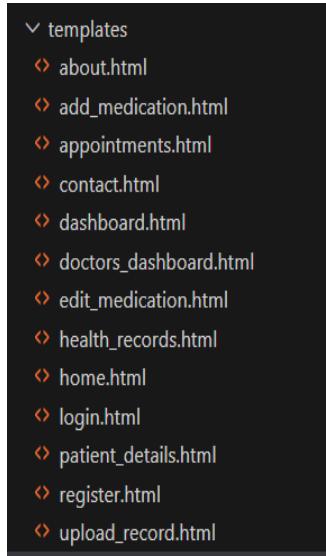
- Run Flask app: sudo flask run --host=0.0.0.0 --port=5000

## 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.

```

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        role = request.form.get('role') or request.args.get('role')
        name = request.form['name']
        email = request.form['email']
        gender = request.form['gender']
        problem = request.form.get('health_problem', '')
        phone = request.form['phone']
        password = generate_password_hash(request.form['password'])

        existing = users_table.get_item(Key={'email': email}).get('Item')
        if existing:
            flash("User already exists!", "danger")
            return redirect(url_for('register'))

        user_item = {
            'email': email,
            'name': name,
            'password': password,
            'gender': gender,
            'phone': phone,
            'role': role
        }
    
```

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

```

from flask import Flask, render_template, request, redirect, url_for, session, flash
import boto3
from werkzeug.security import generate_password_hash, check_password_hash
import uuid

```

## Milestone 2: AWS Account Setup

- Access AWS via Troven Labs.

**AWS - MedTrack Cloud based Patient Medication Tracker**

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

**Overview**

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.

**Skills**

- EC2
- Database on AWS
- IAM
- Monitoring and Observability

**Lab**

**AWS Final Deployment**

AWS - MedTrack Cloud based Patient Medication Tracker

AWS      Expired

easy	5	240 mins
Difficulty	Task	Duration

Kanithi  
22A51A4425@odityatekoll.edu.in

- 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 search interface with the query 'dynamodb' entered in the search bar. The results are categorized into 'Services' and 'Features'.

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

The screenshot shows the DynamoDB dashboard. The left sidebar includes links for 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations', 'Reserved capacity', and 'Settings'. The main area displays the 'Dashboard' with sections for 'Alarms' (0), 'DAX clusters' (0), and a 'Create resources' section for creating a new table or DAX cluster.

## 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 'Create table'. The main title 'Create table' is displayed. Below it, the 'Table details' section is active, indicated by a blue border. A sub-section titled 'Info' is visible. A descriptive text 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 is set to 'Users', which is highlighted with a blue border. A note below the field specifies: 'This will be used to identify your table.' and 'Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).'. The 'Partition key' section is also visible, with a field labeled 'email' and a type dropdown set to 'String'. A note for the partition key states: '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 includes a field labeled 'Enter the sort key name' and a type dropdown set to 'String'. A note for the sort key states: '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.' Below the sort key field, it says '1 to 255 characters and case sensitive.'

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

The screenshot shows the AWS DynamoDB console with the URL [us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables](https://us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables). The left sidebar has sections for Dashboard, Tables (selected), Explore items, PartQL editor, Backups, Exports to S3, Imports from S3, Integrations (New), Reserved capacity, and Settings. Below that is a DAX section with Clusters, Subnet groups, Parameter groups, and Events. The main area shows a table titled 'Tables (2)'. The table has columns: Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read capac. Two rows are listed: 'AppointmentsTable' (Status: Active, Partition key: appointment\_id \$, Sort key: -, Indexes: 0, Replication Regions: 0, Deletion protection: Off, Favorite: On-demand) and 'UsersTable' (Status: Active, Partition key: email \$, Sort key: -, Indexes: 0, Replication Regions: 0, Deletion protection: Off, Favorite: On-demand). A blue banner at the top says 'Share your feedback on Amazon DynamoDB' with a 'Share feedback' button. The bottom of the screen shows the Windows taskbar with various pinned icons like CloudShell, Feedback, Trending videos, and The Lost Bus off... The status bar shows ENG IN, 12:01, and 04-07-2025.

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 - Route 53 feature

The screenshot shows the Amazon Simple Notification Service (SNS) homepage. A blue banner at the top announces a new feature: "Amazon SNS now supports High Throughput FIFO topics. Learn more [↗]".

The main heading is "Amazon Simple Notification Service" with the subtitle "Pub/sub messaging for microservices and serverless applications".

A sidebar on the right contains a "Create topic" form with a "Topic name" input field containing "MyTopic" and a "Next step" button. Below it is a link "Start with an overview".

A "Pricing" callout box states: "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...".

At the bottom, there are links for "CloudShell", "Feedback", "Privacy", "Terms", and "Cookie preferences". The status bar shows the date "04-07-2025" and time "12:09".

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

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

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

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

Type	Info
FIFO (first-in, first-out)	Topic type cannot be modified after topic is created
<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 SNS console with a successful subscription creation message. The URL in the browser is `us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/subscription/arm:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb`. The message says "Subscription to Medtrack created successfully. The ARN of the subscription is arm:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb." Below this, there is a "Subscription" card for the ARN `837e417d-317b-4b43-97c3-054566ac3bbb`, which includes details like ARN, Endpoint, Topic, and Status (Pending confirmation). The status bar at the bottom shows the date as 04-07-2025.

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

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

Amazon SNS > Subscriptions > Create subscription

## Create subscription

**Details**

**Topic ARN**  
 X

**Protocol**  
The type of endpoint to subscribe  
 ▼

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

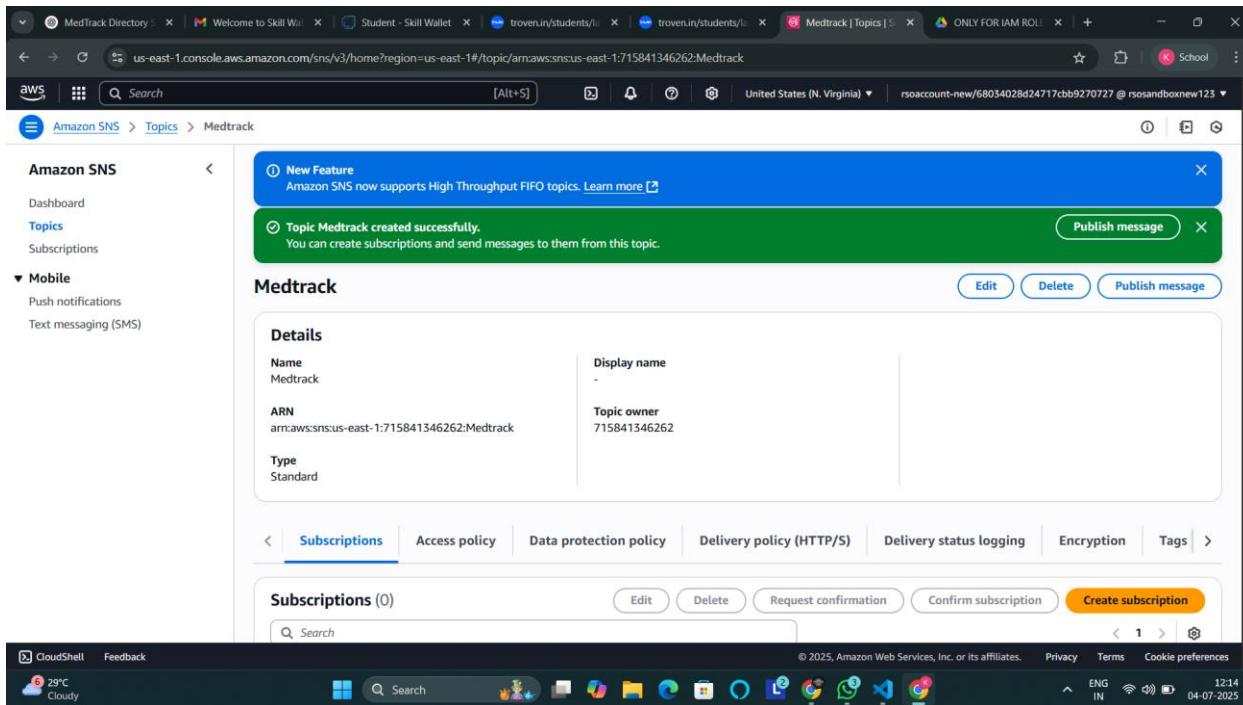
ⓘ After your subscription is created, you must confirm it. [Info](#)

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

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

Cancel Create subscription

After the subscription request for the mail confirmation.



The screenshot shows the AWS SNS Topics page. On the left, there's a navigation sidebar with links for Dashboard, Topics (which is selected), Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and CloudShell. The main content area has a blue header bar with the text "New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more". Below this, a green success message states "Topic Medtrack created successfully. You can create subscriptions and send messages to them from this topic." There are "Edit", "Delete", and "Publish message" buttons. The "Subscriptions" tab is selected, showing a table with one row: "Subscriptions (0)". At the bottom, there are buttons for "Edit", "Delete", "Request confirmation", "Confirm subscription", and "Create subscription". The status bar at the bottom shows "CloudShell Feedback" and various system icons.

**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>  
to me ▾ 12:22PM (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](#)

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



Simple Notification Service

## 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 a subscription details page. The URL in the address bar is `us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/subscription/arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb`. The page title is "Subscription: 837e417d-317b-4b43-97c3-054566ac3bbb". On the left, there's a sidebar with "Amazon SNS" navigation: Dashboard, Topics, Subscriptions (selected), and Mobile (Push notifications, Text messaging (SMS)). A blue banner at the top says "New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more". The main content area shows the subscription details: ARN (arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb), Endpoint (22a51a4425@adityatekkali.edu.in), Topic (Medtrack), Status (Confirmed), and Protocol (EMAIL). Below this, there are tabs for "Subscription filter policy" (selected) and "Redrive policy (dead-letter queue)". The "Subscription filter policy" section includes a link to "Info" and a note: "This policy filters the messages that a subscriber receives." At the bottom, there's a toolbar with CloudShell, Feedback, a search bar, and system status indicators.

## 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 the term "iam". The search bar at the top has "Q iam". The sidebar on the left lists "Services", "Features", "Resources New", "Documentation", "Knowledge articles", "Marketplace", "Blog posts", "Events", and "Tutorials". The main content area displays search results for "iam":

- IAM** ☆ Manage access to AWS resources
- 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
- AWS App Mesh** ☆ Easily monitor and control microservices

The screenshot shows the AWS IAM Dashboard. On the left, there's a sidebar with 'Identity and Access Management (IAM)' and sections for 'Access management' (User groups, Users, Roles, Policies, Identity providers, Account settings, Root access management), 'Access reports' (Access Analyzer, Resource analysis, Unused access, Analyzer settings, Credential report), and 'CloudShell' and 'Feedback' buttons. The main area has a blue banner at the top stating 'New access analyzers available' and 'Create new analyzer'. Below it, the 'IAM resources' section shows an 'Access denied' error for the user 'arn:aws:sts::715841346262:assumed-role/rsoaccount-new/68034028d2'. The error message says: 'You don't have permission to iam:GetAccountSummary. To request access, copy the following text and send it to your AWS administrator.' It includes the User, Action, and Context details. There's also a 'Diagnose with Amazon Q' button. To the right, there are sections for 'AWS Account' (with an 'Access denied' error for iam>ListAccountAliases) and 'Tools' (with a 'Policy simulator' link). The bottom of the screen shows the Windows taskbar with various icons.

## ● Create IAM Roles:

The screenshot shows the 'Create role' wizard in the AWS IAM console. On the left, a vertical navigation bar shows 'Step 1: Select trusted entity', 'Step 2: Add permissions', and 'Step 3: Name, review, and create' (which is selected). The main area is titled 'Name, review, and create' and contains a 'Role details' section. In this section, the 'Role name' field is filled with 'EC2\_MedTrack\_Role'. The 'Description' field contains the text 'Allows EC2 instances to call AWS services on your behalf.'. Below this is the 'Step 1: Select trusted entities' section, which displays a 'Trust policy' with the following JSON code:

```

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

```

The bottom of the screen shows the Windows taskbar with various icons.

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

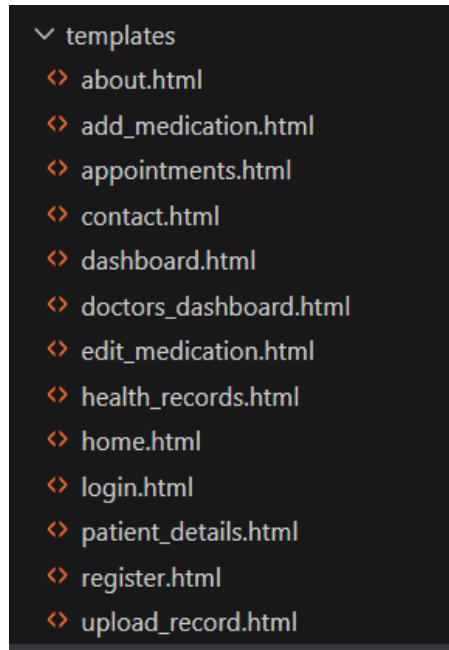
The screenshot shows the AWS IAM Roles page. A green banner at the top indicates that the role 'EC2\_MedTrack\_Role' has been created. Below this, a table lists 11 existing roles, including the newly created one. The table columns are 'Role name', 'Trusted entities', and 'Last activity'. The newly created role is listed as 'EC2\_MedTrack\_Role' with 'AWS Service: ec2' as the trusted entity and '58 minutes ago' as the last activity.

Role name	Trusted entities	Last activity
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service)	-
AWSServiceRoleForAutoScaling	AWS Service: autoscaling (Service)	144 days ago
AWSServiceRoleForECS	AWS Service: ecs (Service-Linked Role)	144 days ago
AWSServiceRoleForOrganizations	AWS Service: organizations (Service)	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)	-
EC2_MedTrack_Role	AWS Service: ec2	-
OrganizationAccountAccessRole	Account: 058264256896	58 minutes ago

The screenshot shows the 'Modify IAM role' page for the EC2 instance 'i-01e69ab66b8e6b6a4'. The instance ID is 'i-01e69ab66b8e6b6a4 (medtrack-server)'. A dropdown menu under 'IAM role' shows the selected role 'EC2\_MedTrack\_Role'. There is also a link to 'Create new IAM role'. At the bottom right are 'Cancel' and 'Update IAM role' buttons.

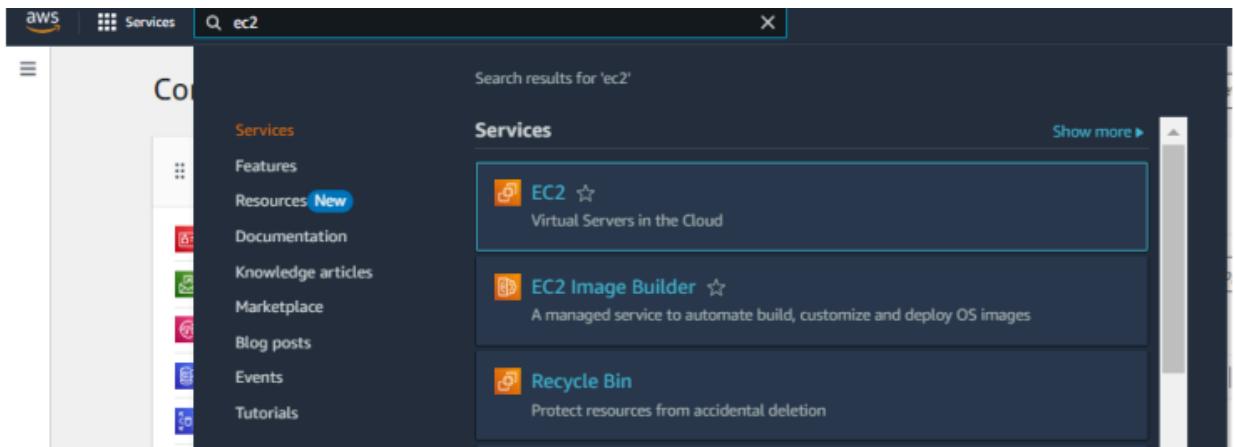
## Milestone 6: EC2 Instance Setup

- 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.

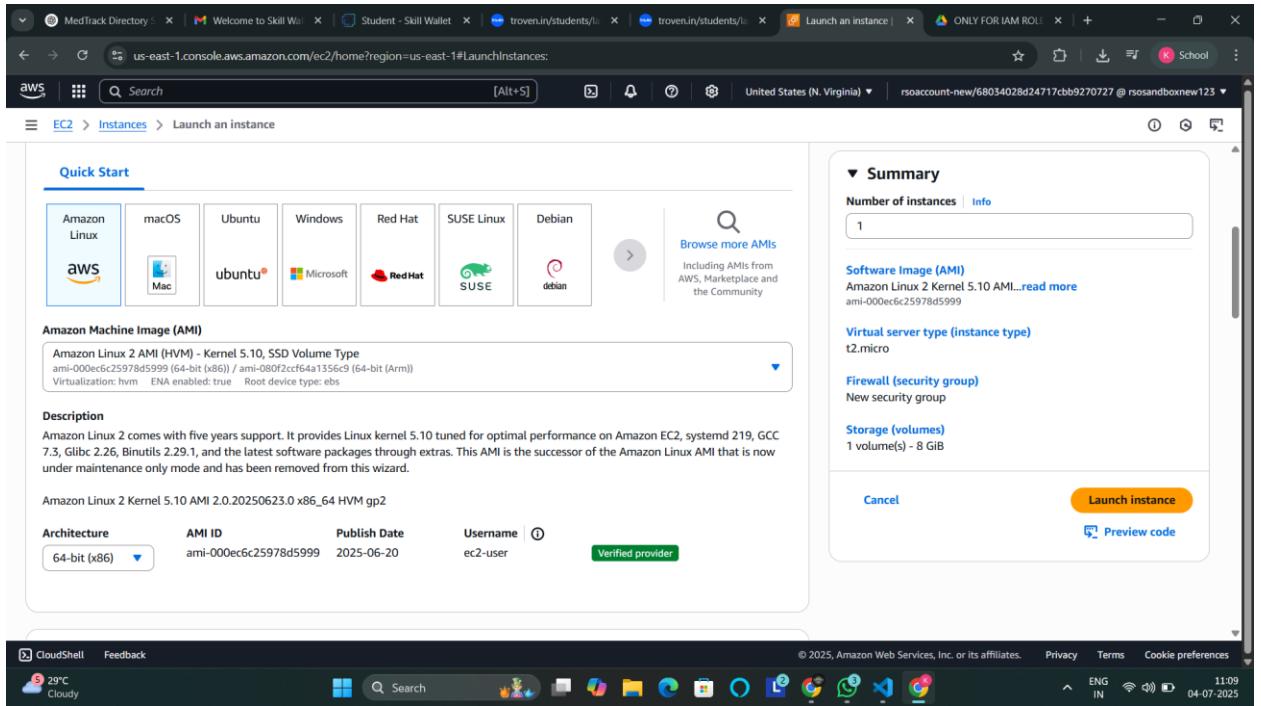


- Click on Launch instance to launch EC2 instance

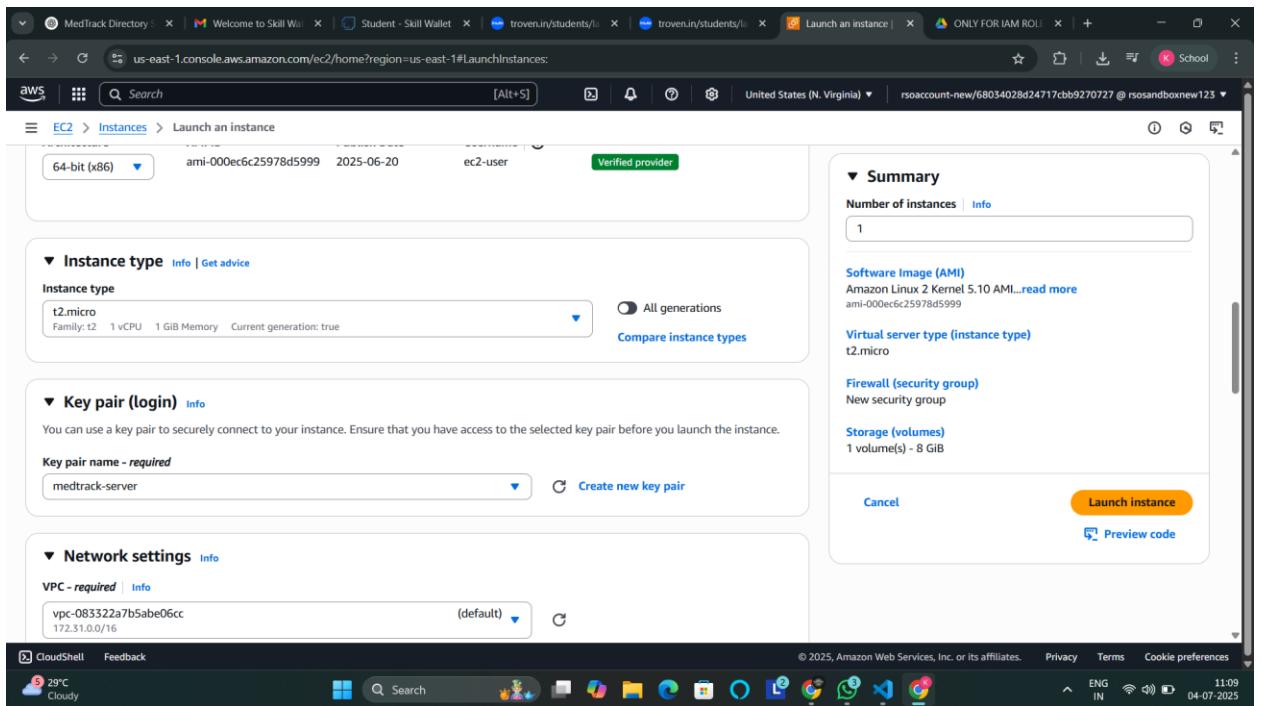
The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed, and the main area displays the 'Instances Info' section. A search bar at the top allows filtering by 'Name' or 'Instance ID'. Below it, a table header includes columns for 'Instance state', 'Instance type', 'Status check', 'Alarm status', 'Availability Zone', and 'Public IP'. A message states 'No instances' and 'You do not have any instances in this region'. A prominent blue button labeled 'Launch instances' is centered below the message. The bottom section is titled 'Select an instance'.

The screenshot shows the 'Launch an instance' wizard. A blue header bar at the top contains a message: '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'. Below this are two buttons: 'Take a walkthrough' and 'Do not show me this message again.' The main content area is titled 'Launch an instance' and includes sections for 'Name and tags', 'Application and OS Images (Amazon Machine Image)', and 'Quick Start'. In the 'Name and tags' section, the name 'medtrack-server' is entered. Under 'Application and OS Images (Amazon Machine Image)', 'Amazon Linux 2' is selected. On the right side, there's a summary panel showing 'Number of instances' (1), 'Software Image (AMI)' (Amazon Linux 2 Kernel 5.10 AMI...), 'Virtual server type (instance type)' (t2.micro), 'Firewall (security group)' (New security group), and 'Storage (volumes)' (1 volume(s) - 8 GiB). At the bottom right is a large orange 'Launch instance' button.

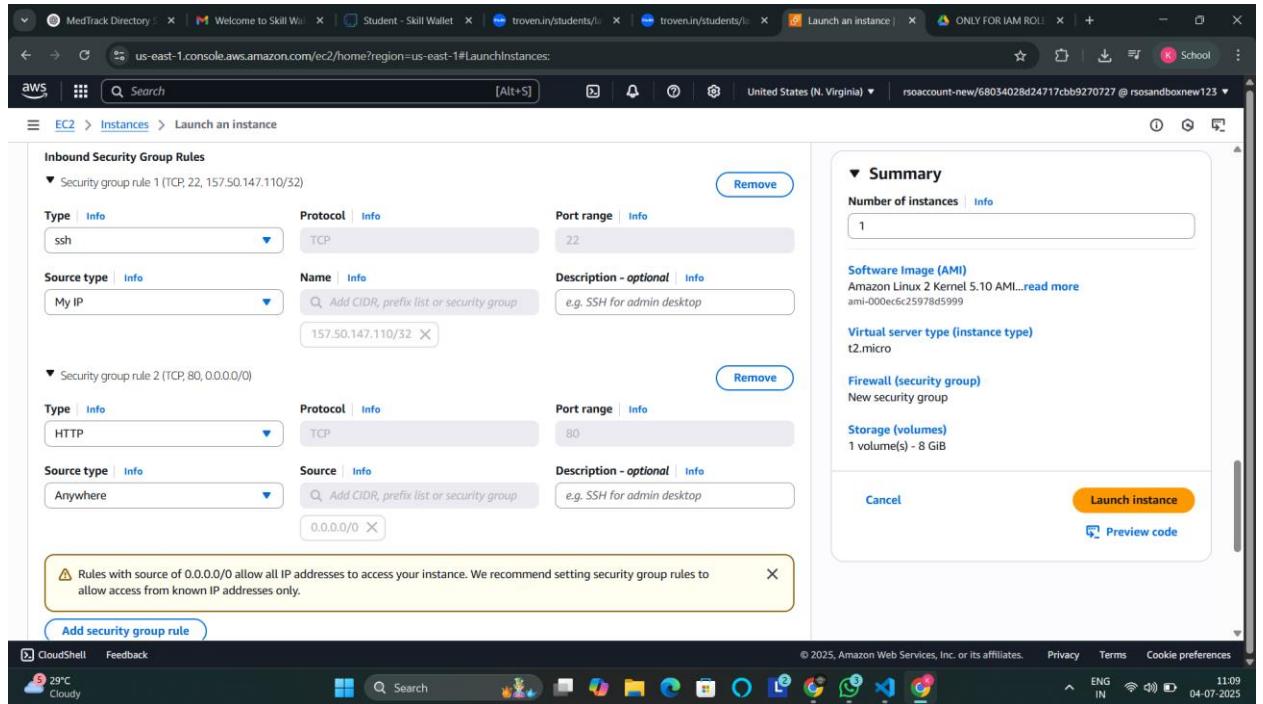
- **Launch EC2 instance (Amazon Linux 2/Ubuntu, t2.micro).**



- Assign an IAM Role and key pair.



- Configure security groups for HTTP/SSH.



- To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected, navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the EC2 Dashboard, click the Connect button. From the connection methods presented, choose EC2 Instance Connect. Finally, click Connect again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

The screenshot shows the AWS Management Console with the EC2 Instances page open. The left sidebar shows navigation options like Dashboard, EC2 Global View, Events, Instances (with sub-options like Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes, Snapshots, Lifecycle Manager). The main content area displays a table of instances. A green success message at the top says "Successfully attached EC2\_MedTrack\_Role to instance i-01e69ab66b8e6b6a4". The table has columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IP. One row is selected for the instance 'medtrack-server' with ID i-01e69ab66b8e6b6a4, which is 'Running'. Below the table, a detailed view for the instance 'i-01e69ab66b8e6b6a4 (medtrack-server)' is shown, with tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. Under Details, the Instance summary section shows the Instance ID (i-01e69ab66b8e6b6a4), Public IPv4 address (23.20.160.238), Private IPv4 addresses (172.31.30.113), and Instance state (Running).

## ● Now connect the EC2 with the files.

The screenshot shows the AWS CloudShell interface. The terminal window displays a session connected to the EC2 instance 'i-01e69ab66b8e6b6a4'. The session starts with a welcome message from Amazon Linux 2, followed by a note about the end of life. It then shows the user installing Python and Git packages using the 'yum' command. The terminal output includes package names like 'python3-3.7.16-1.amzn2.0.17.x86\_64' and dependency resolution details. The session ends with a message indicating the transaction was successful.

```

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 python
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(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

## **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/Prasanthilakshmi05/MedTrack.git>

Note: change your-github-username and your-repository-name with your credentials.  
here: ‘git clone https://github.com/Prasanthilakshmi05/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://127.0.0.1:5000>
- 

## 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 home():
    return render_template('home.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 request.method == 'POST':
        role = request.form.get('role') or request.args.get('role')
        name = request.form['name']
        email = request.form['email']
        gender = request.form['gender']
        problem = request.form.get('health_problem', '')
        phone = request.form['phone']
        password = generate_password_hash(request.form['password'])

        existing = users_table.get_item(Key={'email': email}).get('Item')
        if existing:
            flash("User already exists!", "danger")
            return redirect(url_for('register'))

        user_item = {
            'email': email,
            'name': name,
            'password': password,
            'gender': gender,
            'phone': phone,
            'role': role
        }
```

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

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password_input = request.form['password']

        user = users_table.get_item(Key={'email': email}).get('Item')
        if user and check_password_hash(user['password'], password_input):
            session['user'] = email
            session['role'] = user['role']
            return redirect(url_for('doctors_dashboard')) if user['role'].lower() == 'doctor' else url_for('dashboard'))

            flash("Invalid credentials", "danger")
    return render_template('login.html')
```

- **Logout:** End session.

```

@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))

```

- **Book Appointment:** Collect and store appointment details, trigger SNS.
- **View Appointments:** Retrieve data from DynamoDB.
- **Search:** Filter appointments.

```

@app.route('/appointments', methods=['GET', 'POST'])
def appointments():
    if 'user' not in session:
        return redirect(url_for('login'))

    if request.method == 'POST':
        item = {
            'id': str(uuid.uuid4()),
            'user_email': session['user'],
            'date': request.form['date'],
            'time': request.form['time'],
            'reason': request.form['reason'],
            'doctor_email': request.form['doctor_email']
        }
        appointments_table.put_item(Item=item)

        try:
            sns.publish(
                TopicArn=sns_topic_arn,
                Message=f"New appointment booked with Dr. {item['doctor_email']} on {item['date']} at {item['time']}.",
                Subject="New Appointment Notification"
            )
        except Exception as e:
            print(f"SNS Error: {e}")

        flash("Appointment booked and doctor notified.", "success")
        return redirect(url_for('dashboard'))

    return render_template('appointments.html')

```

### Profile: View/edit personal data:

```

@app.route('/patient_details')
def patient_details():
    if 'user' not in session:
        return redirect(url_for('login'))
    user = users_table.get_item(Key={'email': session['user']}).get('Item')
    return render_template('patient_details.html', patient=user)

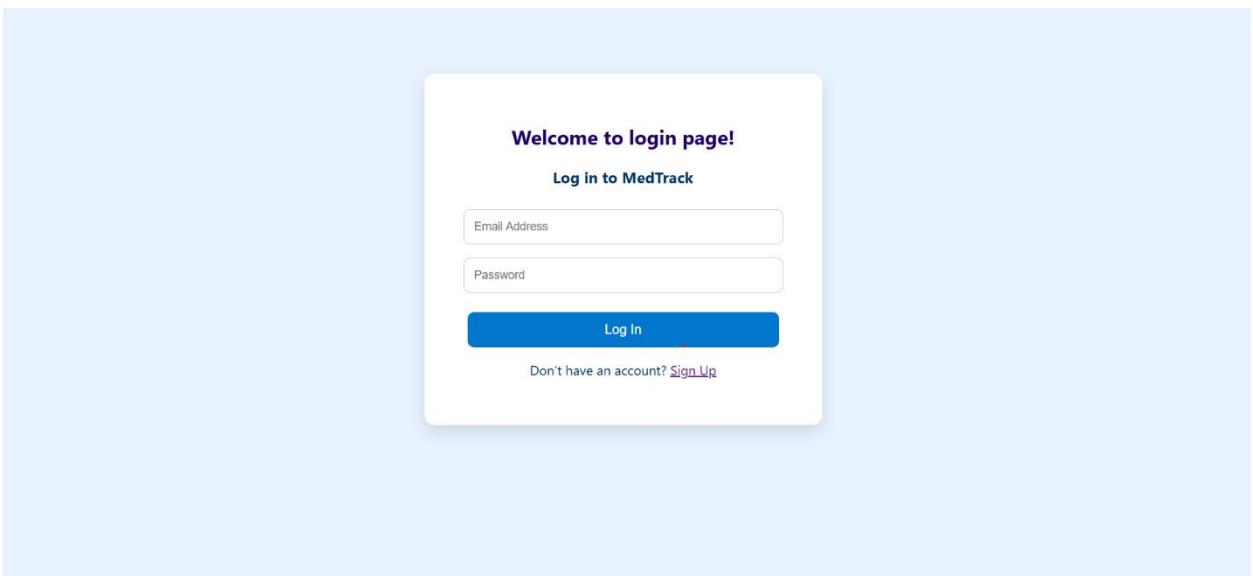
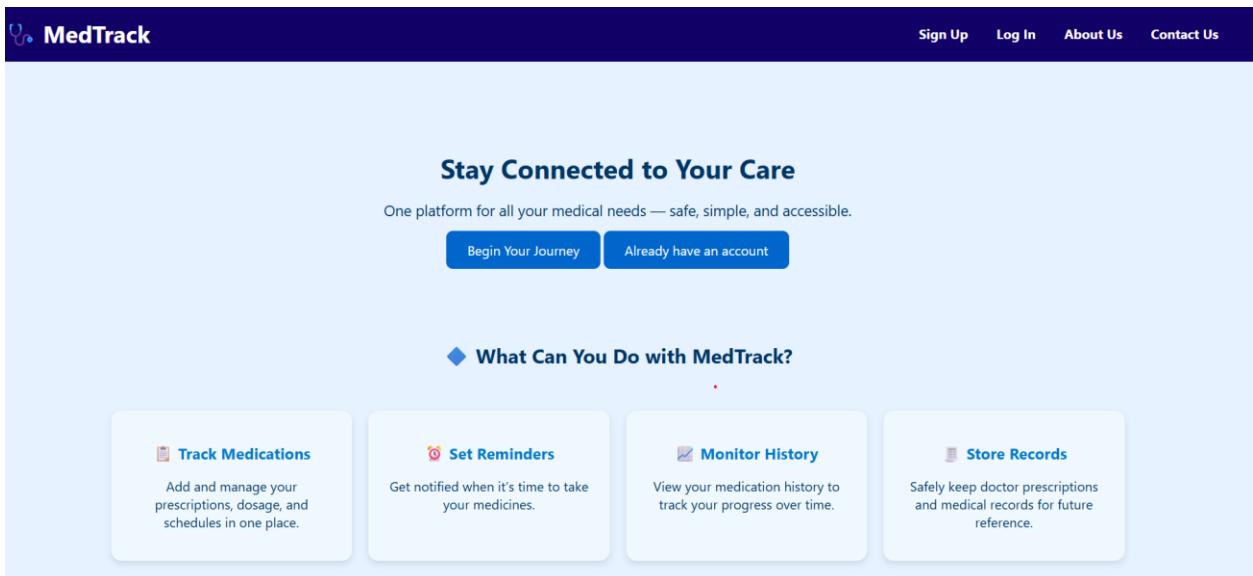
```

### Deployment Code:

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

## Functional Testing Summary

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



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

Registration Form

Patient

Doctor

Name:

Email:

Gender:

Health Problem:

Phone:

Password:

Confirm Password:

**Register as Patient**

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

MedTrack Dashboard

Logged in as: prasanthilakshmichappati@gmail.com

Welcome, Prasanthilakshmichappati!

Manage your medications, view health records, and more.

+ Add Medication    Appointments    Doctor's Dashboard    Patient Details    Logout

**Your Medications**  
You haven't added any medications yet.

**Your Appointments:**  
You haven't booked any appointments yet.

**Health Records Uploaded by Your Doctor**

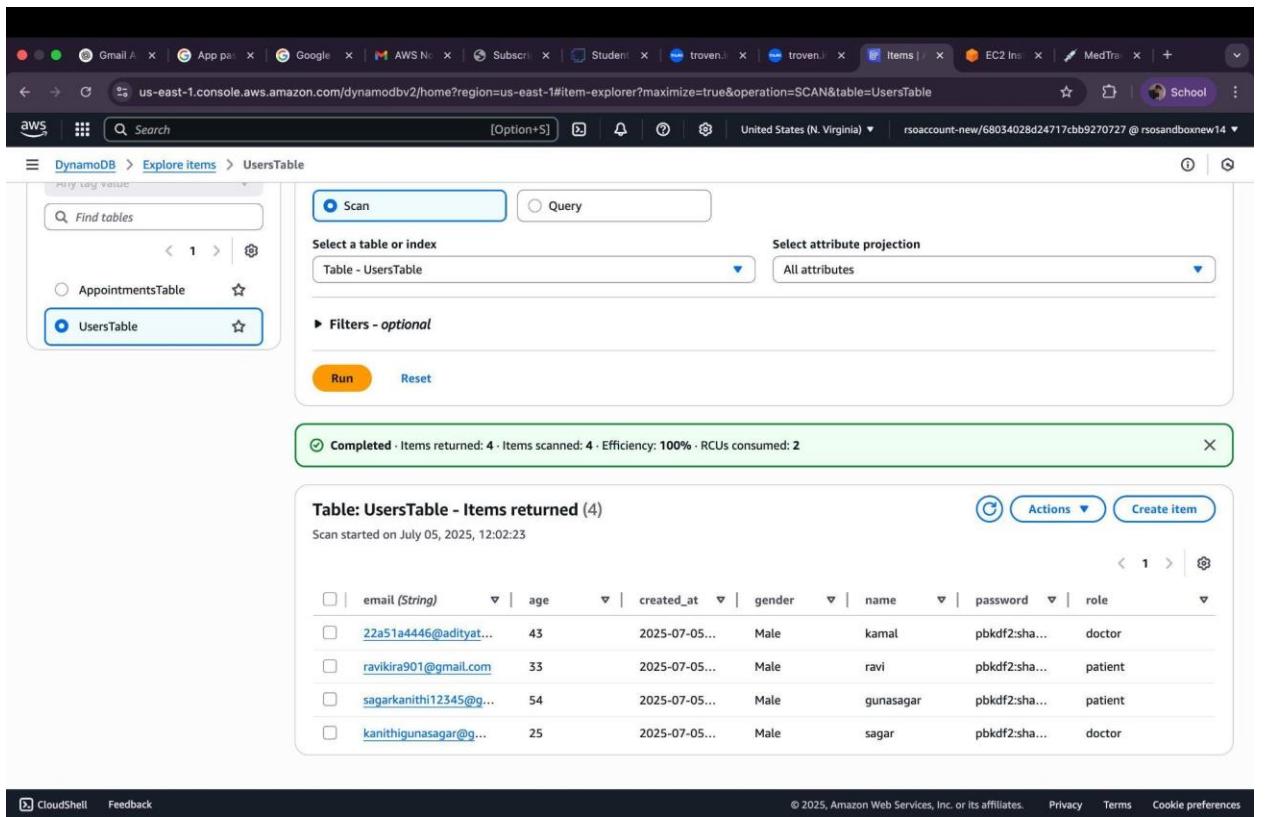
Logout

- **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.



**Completed · Items returned: 4 · Items scanned: 4 · Efficiency: 100% · RCU's consumed: 2**

	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 for the AppointmentsTable. The left sidebar lists tables: AppointmentsTable (selected) and UsersTable. The main area has a 'Scan or query items' section with 'Scan' selected, showing a table dropdown set to 'Table - AppointmentsTable' and an 'All attributes' dropdown. Below is a 'Filters - optional' section with a 'Run' button. A green status bar at the bottom indicates 'Completed - Items returned: 2 - Items scanned: 2 - Efficiency: 100% - RCU's consumed: 2'. The results table has columns: appointment\_id, appointment\_date, created\_at, doctor\_email, doctor\_name, patient\_email, and patient\_name. Two items are listed:

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

## 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.