

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.

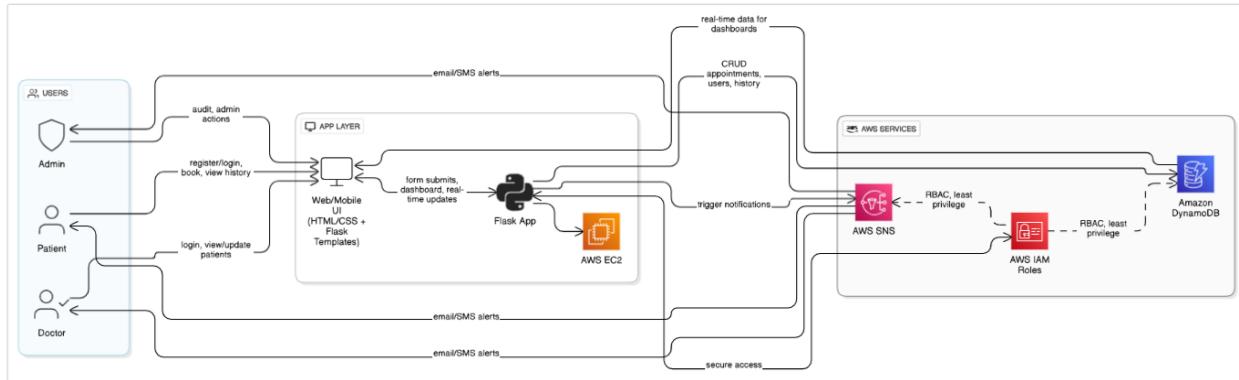
Scenario 2: Secure User Management with IAM

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

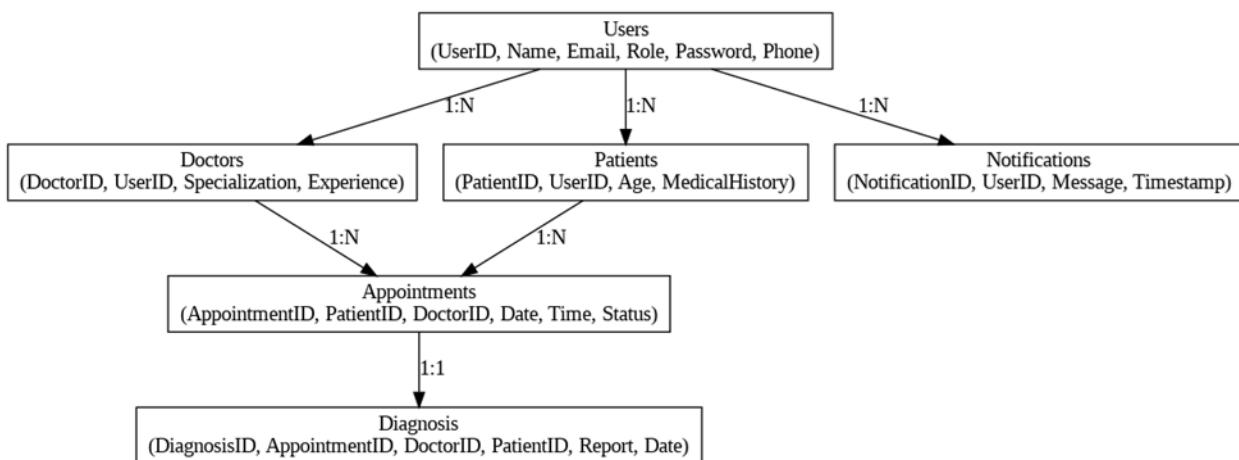
Doctors can retrieve patient records and update diagnoses in real time. Flask and DynamoDB integration enables high-speed access and data updates, ensuring seamless operation during peak usage.

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.

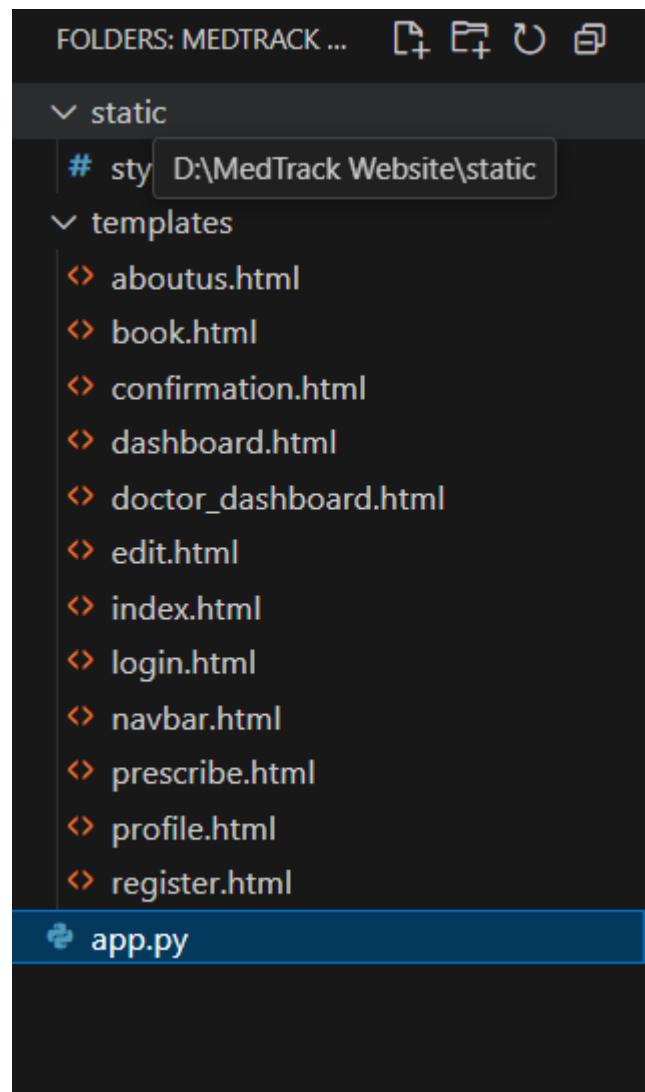
- Run Flask app: sudo flask run --host=0.0.0.0 --port=5000

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.

```
# ----- REGISTER -----
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        data = request.form
        email = data['email']
        username = data['username']
        password = generate_password_hash(data['password'])
        role = data['role']
        disease = data.get('disease', '')
        specialization = data.get('specialization', '')

        # Check if user exists
        response = users_table.get_item(Key={'email': email})
        if 'Item' in response:
            return render_template('register.html', error='Email already registered')

        users_table.put_item(Item={
            'email': email,
            'username': username,
            'password': password,
            'role': role,
            'disease': disease,
            'specialization': specialization
        })

        return redirect('/login')

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

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

```
from flask import Flask, render_template, request, redirect, url_for, session
import boto3
import uuid
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import date
import os|
```

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

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. The search bar at the top contains the query 'dynamodb'. Below the search bar, there are two main sections: '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 has a 'DynamoDB' header and lists various options under 'Tables' and 'DAX'.

The main content area is titled 'Dashboard' and contains the following sections:

- Alarms (0) Info:** Manage in CloudWatch. A search bar labeled 'Find alarms' and a table with columns 'Alarm name' and 'Status'.
- DAX clusters (0) Info:** View details. A search bar labeled 'Find clusters' and a table with columns 'Cluster name' and 'Status'.
- Create resources:** Buttons for 'Create table' and 'Create DAX cluster'.
- What's new:** A section with a message from SEP 1: 'AWS Cost Management now provides 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 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. The left sidebar has 'Tables' selected under 'DynamoDB'. The main area displays two tables:

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

At the top, there is a feedback banner: "Share your feedback on Amazon DynamoDB. 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." Below it are notification settings and a "Create table" button.

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 sidebar with links like Dashboard, Topics, Subscriptions, Mobile, Push notifications, and Text messaging (SMS). The main area has a header 'Topics (0)' with a search bar and buttons for Edit, Delete, Publish message, and Create topic. Below this is a table with columns Name, Type, and ARN. A message at the bottom 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. Both options have associated descriptions and bullet points. The 'Standard' section includes a note: 'Topic type cannot be modified after topic is created'.

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

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

The screenshot shows the AWS SNS console with a successful subscription creation message. The URL in the address bar 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 main content area displays a green success message: "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, a section titled "Subscription: 837e417d-317b-4b43-97c3-054566ac3bbb" shows the subscription details:

Details	Status
ARN arn:aws:sns:us-east-1:715841346262:Medtrack:837e417d-317b-4b43-97c3-054566ac3bbb	Pending confirmation
Endpoint 22a51a4425@adityatekkali.edu.in	Protocol EMAIL
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 browser status bar at the bottom shows "CloudShell Feedback 29°C Cloudy 12:22 04-07-2025".

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

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

Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN
 X

Protocol
The type of endpoint to subscribe
 ▼

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

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

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

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

Cancel Create subscription

After the subscription request for the mail confirmation.

us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/topic/arnaws:sns:us-east-1:715841346262:Medtrack

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

Amazon SNS > Topics > Medtrack

ⓘ New Feature Amazon SNS now supports High Throughput FIFO topics. Learn more [\[More\]](#)

ⓘ Topic Medtrack created successfully. You can create subscriptions and send messages to them from this topic. Publish message X

Medtrack Edit Delete Publish message

Details	
Name	Medtrack
ARN	arn:aws:sns:us-east-1:715841346262:Medtrack
Type	Standard
Display name	-
Topic owner	715841346262

Subscriptions Access policy Data protection policy Delivery policy (HTTP/S) Delivery status logging Encryption Tags >

Subscriptions (0) Edit Delete Request confirmation Confirm subscription Create subscription

CloudShell Feedback 29°C Cloudy Search ENG IN 12:14 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>
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 shows search results for "iam": "Search results for 'iam'" followed by a list of services: "Services" (Show more ▾), "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), and "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) and 'Access reports' (Access Analyzer, Resource analysis, Unused access, Analyzer settings, Credential report). The main area is titled 'IAM Dashboard' and contains a 'IAM resources' section with a red-bordered box highlighting an 'Access denied' error. The error message states: 'You don't have permission to iam:GetAccountSummary. To request access, copy the following text and send it to your AWS administrator.' It includes a snippet of JSON policy code:

```
User: arn:aws:sts::715841346262:assumed-role/rsoaccount-new/68034028d2
Action: iam:GetAccountSummary
Context: no identity-based policy allows the action
```

Below this is a 'What's new' section with a 'View all' link. On the right, there are 'AWS Account' and 'Tools' sections, and at the bottom, standard browser navigation and status bars.

● Create IAM Roles:

The screenshot shows the 'Create role' wizard in the AWS IAM console. The left sidebar shows steps: Step 1 (Select trusted entity), Step 2 (Add permissions), and Step 3 (Name, review, and create). The current step is Step 3. The main area is titled 'Name, review, and create' and contains a 'Role details' section. In the 'Role name' field, 'EC2_MedTrack_Role' is entered. 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 document:

```
1+ [ {  
2     "Version": "2012-10-17",  
3     "Statement": [  
4         {  
5             "Effect": "Allow",  
6             "Action": [  
7                 "sts:AssumeRole"  
8             ]  
9         }  
10    ]  
11}  
12}
```

The bottom of the screen shows a Windows taskbar with various icons and a status bar indicating 'CloudShell Feedback' and the date '04-07-2025'.

- **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. The main section displays a table of roles, with one row selected:

Role name	Trusted entities	Last activity
EC2_MedTrack_Role	AWS Service: ec2	-

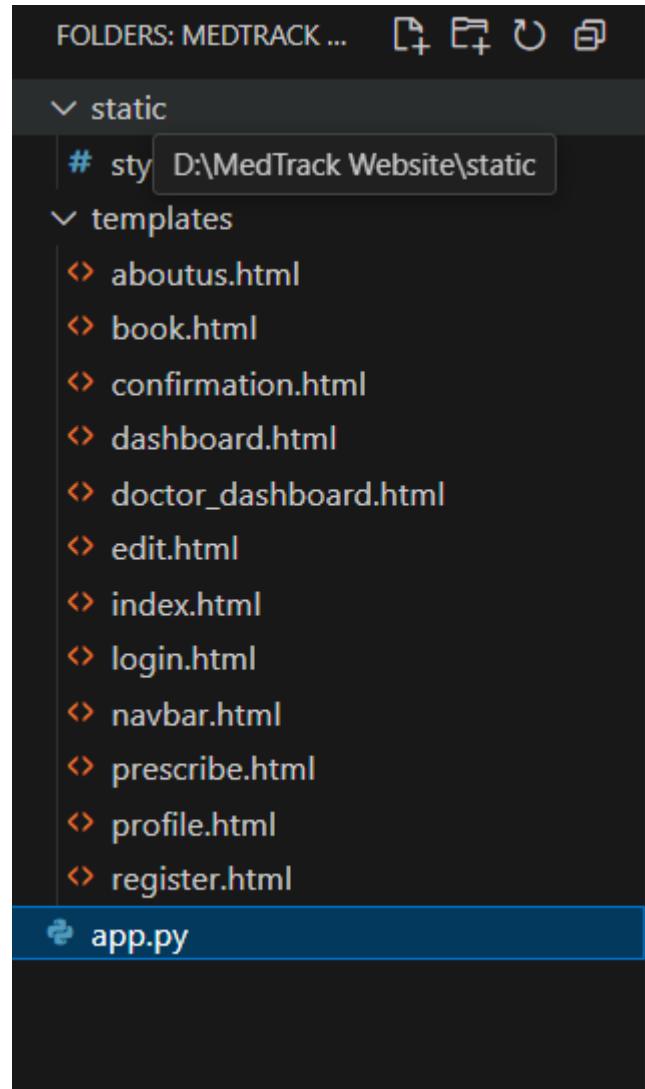
The status bar at the bottom right shows the date as 04-07-2025 and the time as 11:36.

The screenshot shows the 'Modify IAM role' page for the EC2 instance 'i-01e69ab66b8e6b6a4'. The 'Instance ID' dropdown is set to 'i-01e69ab66b8e6b6a4 (medtrack-server)'. The 'IAM role' dropdown is set to 'EC2_MedTrack_Role'. A button for 'Create new IAM role' is visible. At the bottom, there are 'Cancel' and 'Update IAM role' buttons.

The status bar at the bottom right shows the date as 04-07-2025 and the time as 11:44.

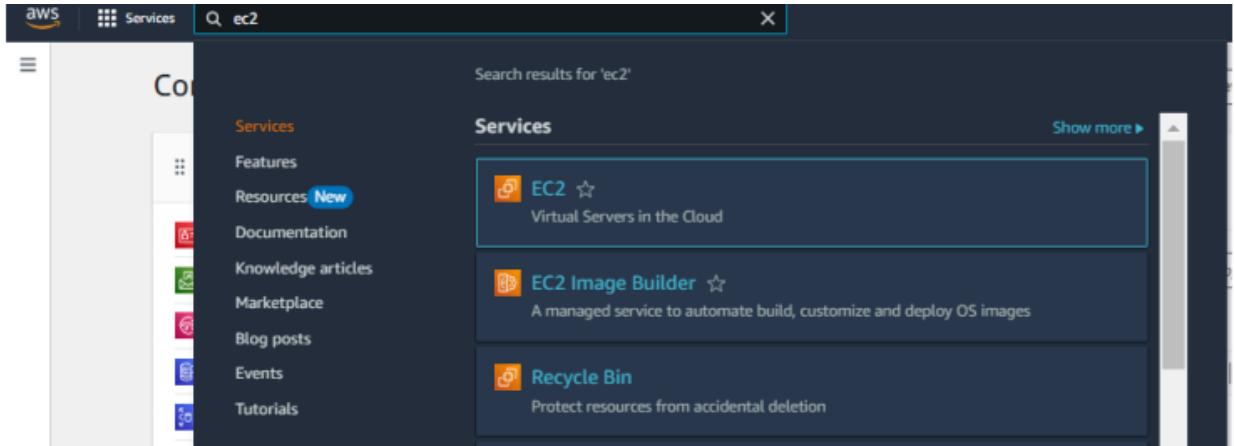
Milestone 6: EC2 Instance Setup

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

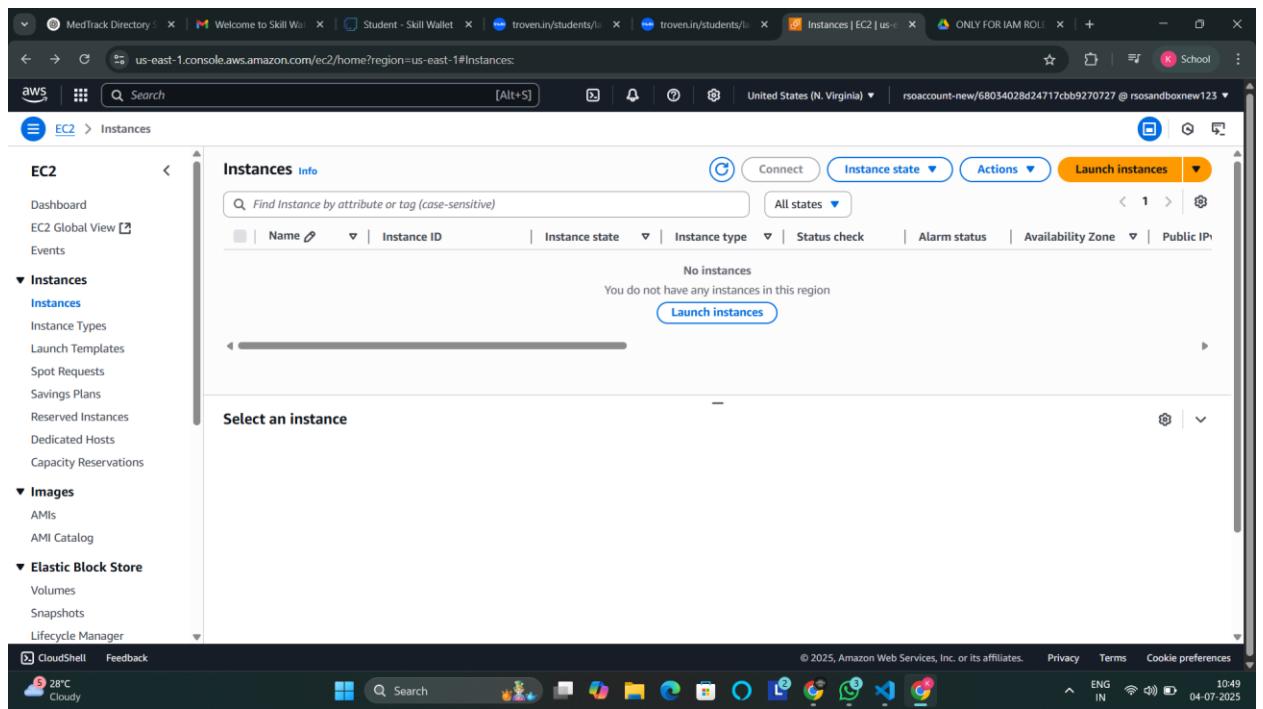


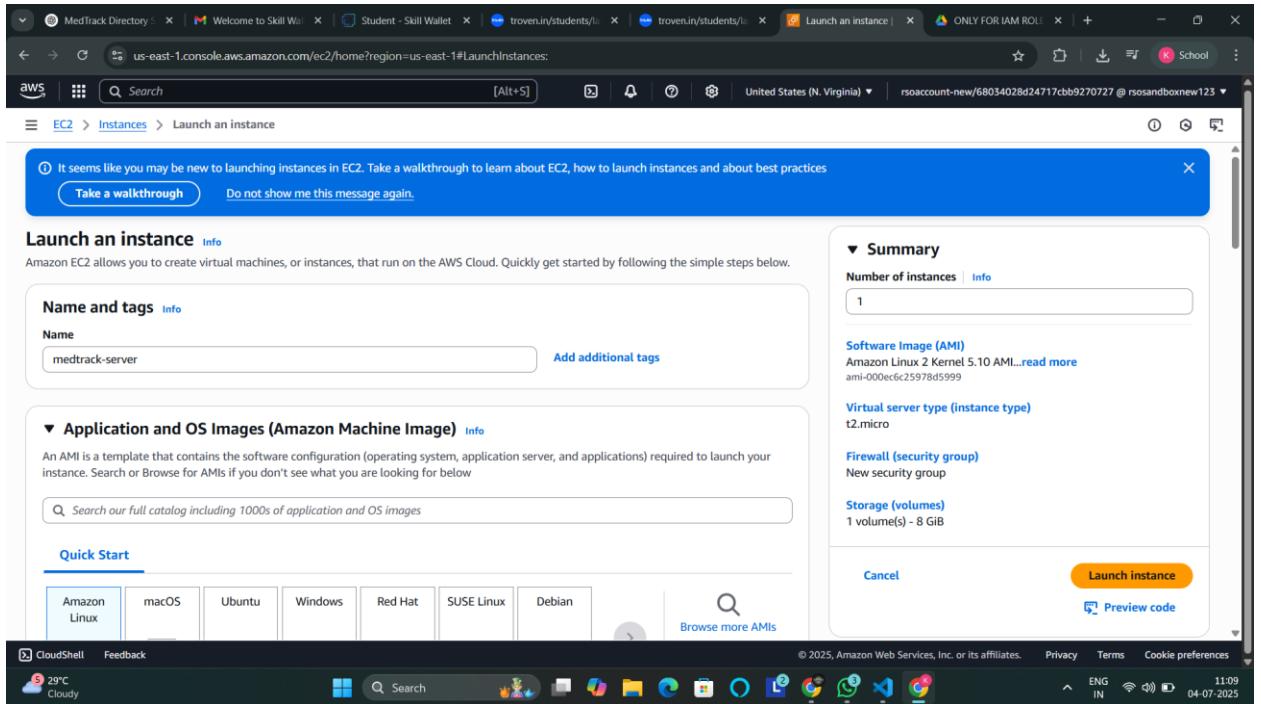
Launch an EC2 instance to host the Flask application.

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

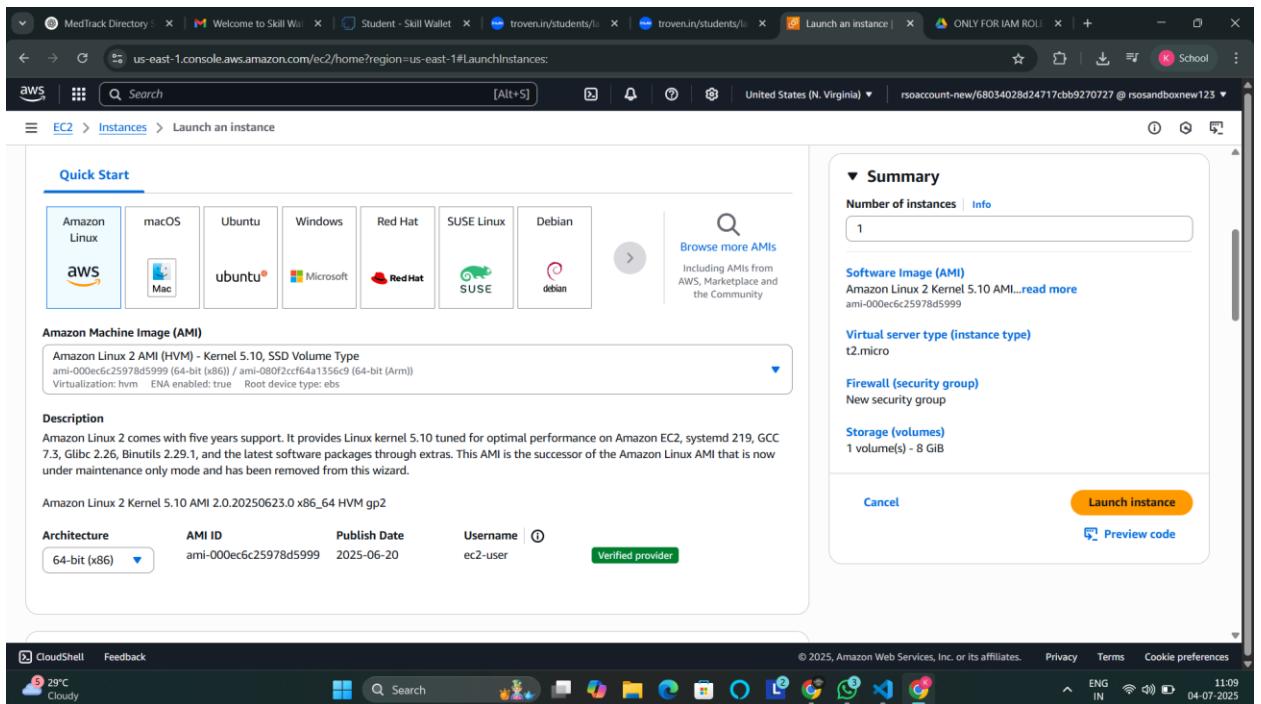


- Click on Launch instance to launch EC2 instance

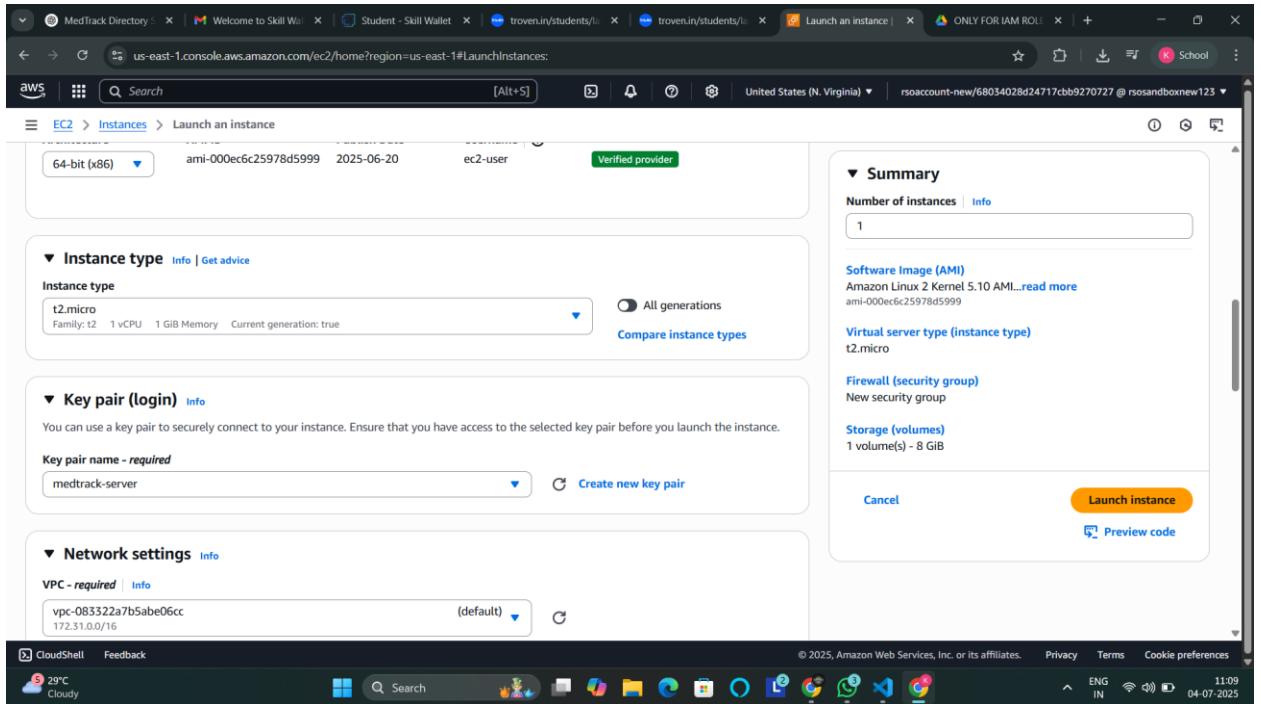




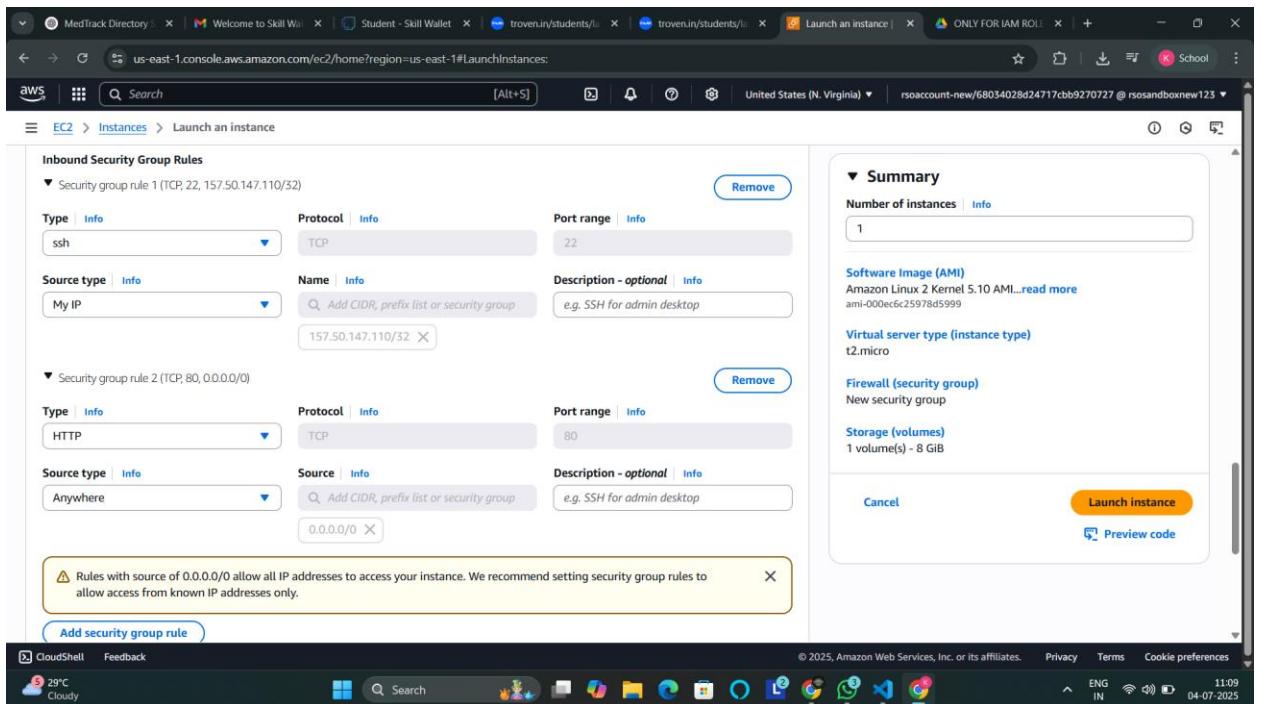
- Launch EC2 instance (Amazon Linux 2/Ubuntu, t2.micro).



- Assign an IAM Role and key pair.

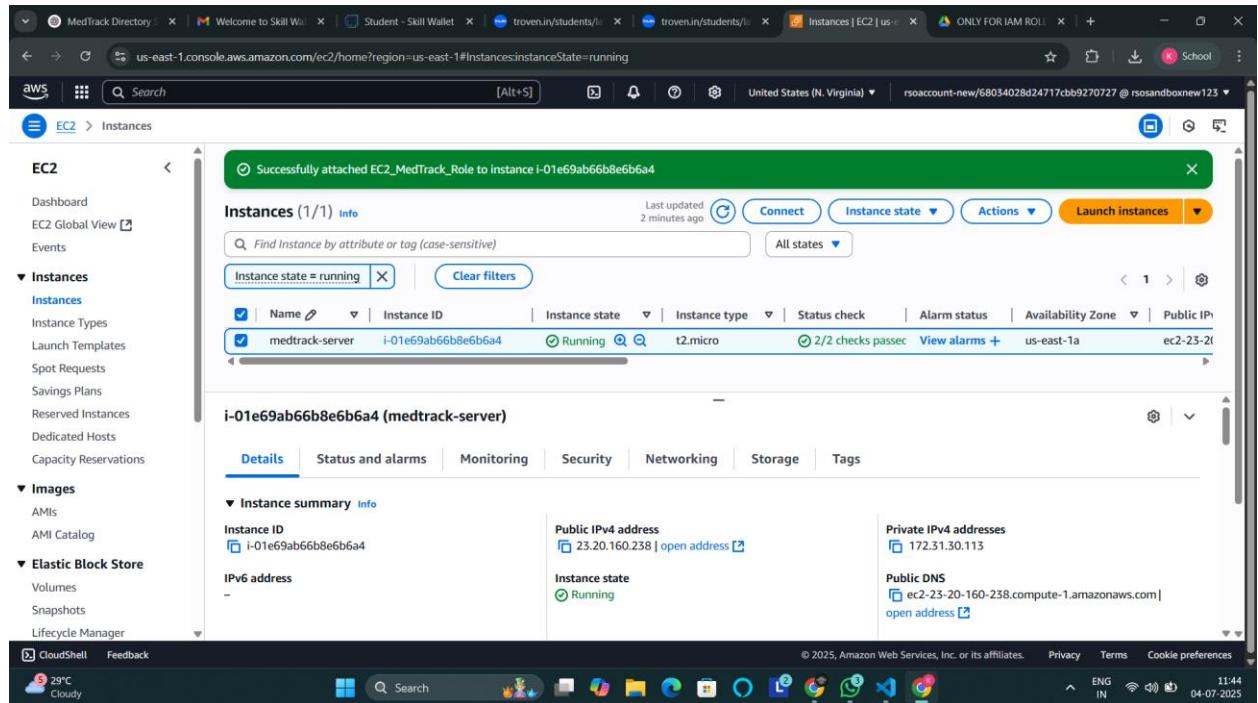


- Configure security groups for HTTP/SSH.

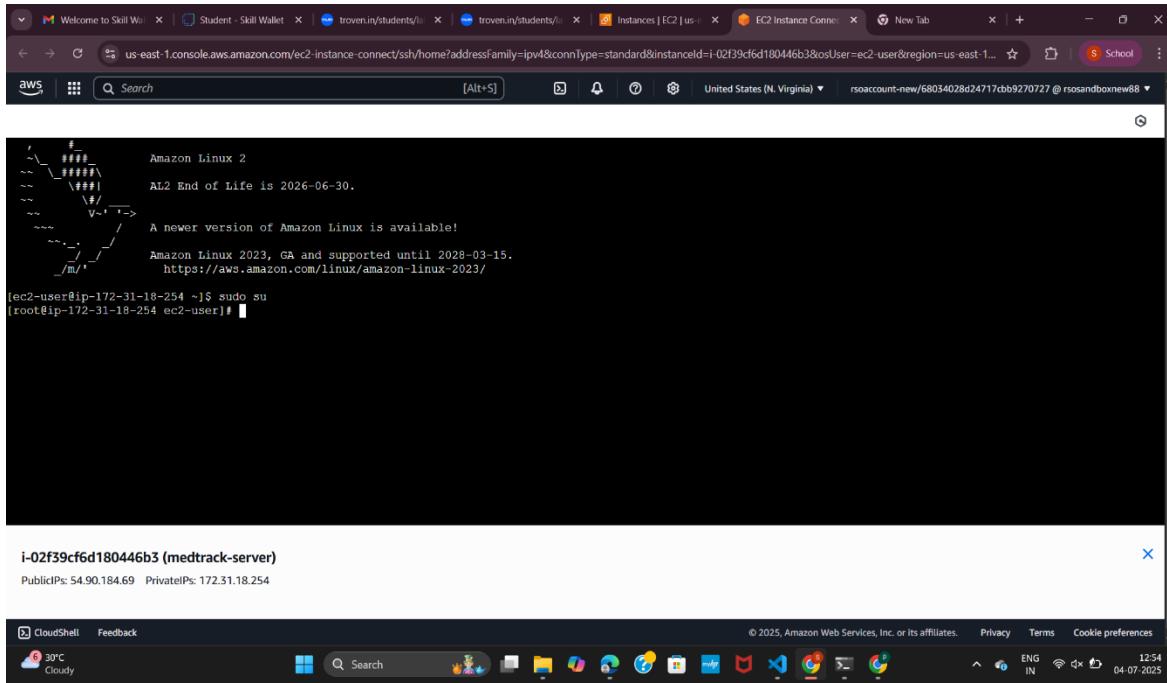


- To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance,

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



- Now connect the EC2 with the files.



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/Sattaru-Praveena/Medtrack-Website.git>

Note: change your-github-username and your-repository-name with your credentials.
here: 'git clone https://github.com/Sattaru-Praveena/Medtrack-Website.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 - esc[33mPress CTRL+C to quitesc[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] "esc[36mGET /static/css/custom.css HTTP/1.1esc[0m" 304 -
5-06-22 14:30:05,104 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:30:05] "esc[36mGET /static/js/custom.js HTTP/1.1esc[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] "esc[36mGET /static/css/custom.css HTTP/1.1esc[0m" 304 -
5-06-22 14:30:12,054 - werkzeug - INFO - 127.0.0.1 - - [22/Jun/2025 14:30:12] "esc[36mGET /static/js/custom.js HTTP/1.1esc[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://54.90.184.69:5000>

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():
    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.

```
# ----- REGISTER -----
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        data = request.form
        email = data['email']
        username = data['username']
        password = generate_password_hash(data['password'])
        role = data['role']
        disease = data.get('disease', '')
        specialization = data.get('specialization', '')

        # Check if user exists
        response = users_table.get_item(Key={'email': email})
        if 'Item' in response:
            return render_template('register.html', error='Email already registered')

        users_table.put_item(Item={
            'email': email,
            'username': username,
            'password': password,
            'role': role,
            'disease': disease,
            'specialization': specialization
        })

    return redirect('/login')

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

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

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

        response = users_table.get_item(Key={'email': email})
        user = response.get('Item')

        if user and check_password_hash(user['password'], password):
            session['email'] = email
            session['username'] = user['username']
            session['role'] = user['role']
            return redirect('/dashboard')
        else:
            return render_template('login.html', error='Invalid credentials')

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

- **Logout:** End session.

```
# ----- LOGOUT -----
@app.route('/logout')
def logout():
    session.clear()
    return redirect('/')
```

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

```

# ----- BOOK APPOINTMENT -----
@app.route('/book', methods=['GET', 'POST'])
def book():
    if 'username' not in session:
        return redirect('/login')

    if request.method == 'POST':
        doctor = request.form['doctor']
        apt_date = request.form['date']
        time = request.form['time']
        reason = request.form['reason']

        appointments_table.put_item(Item={
            'id': str(uuid.uuid4()),
            'username': session['username'],
            'doctor': doctor,
            'date': apt_date,
            'time': time,
            'reason': reason
        })

    # Notify via SNS
    try:
        sns.publish(
            TopicArn=sns_topic_arn,
            Subject="New Appointment",
            Message=f"{session['username']} booked with Dr. {doctor} on {apt_date} at {time}"
        )
    except Exception as e:
        print("SNS error:", e)

    return render_template('confirmation.html', doctor=doctor, date=apt_date, time=time)

return render_template('book.html', current_date=date.today())

```

Profile: View/edit personal data:

```

# ----- EDIT -----
@app.route('/edit/<string:appt_id>', methods=['GET', 'POST'])
def edit_appointment(appt_id):
    if 'username' not in session:
        return redirect('/login')

    if request.method == 'POST':
        doctor = request.form['doctor']
        date_val = request.form['date']
        time_val = request.form['time']
        reason = request.form['reason']

        appointments_table.update_item(
            Key={'id': appt_id},
            UpdateExpression="SET doctor = :d, date = :dt, time = :t, reason = :r",
            ExpressionAttributeValues={
                ':d': doctor, ':dt': date_val, ':t': time_val, ':r': reason
            }
        )
        return redirect('/dashboard')

    response = appointments_table.get_item(Key={'id': appt_id})
    appt = response.get('Item')
    return render_template('edit.html', appt=appt, appt_id=appt_id, current_date=date.today())

```

Deployment Code:

```

# ----- RUN APP -----
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.

Welcome to MedTrack

Your personal cloud-based medication and appointment management system.

Get Started

Why Choose MedTrack?

Easy Appointments

Book doctor appointments and track all your visits in one place.

Smart Reminders

Stay on top of your medication schedule with timely alerts.

Doctor's Prescription

Doctors can view and respond to appointment requests and provide secure prescriptions.

Login

Email:

Enter your email

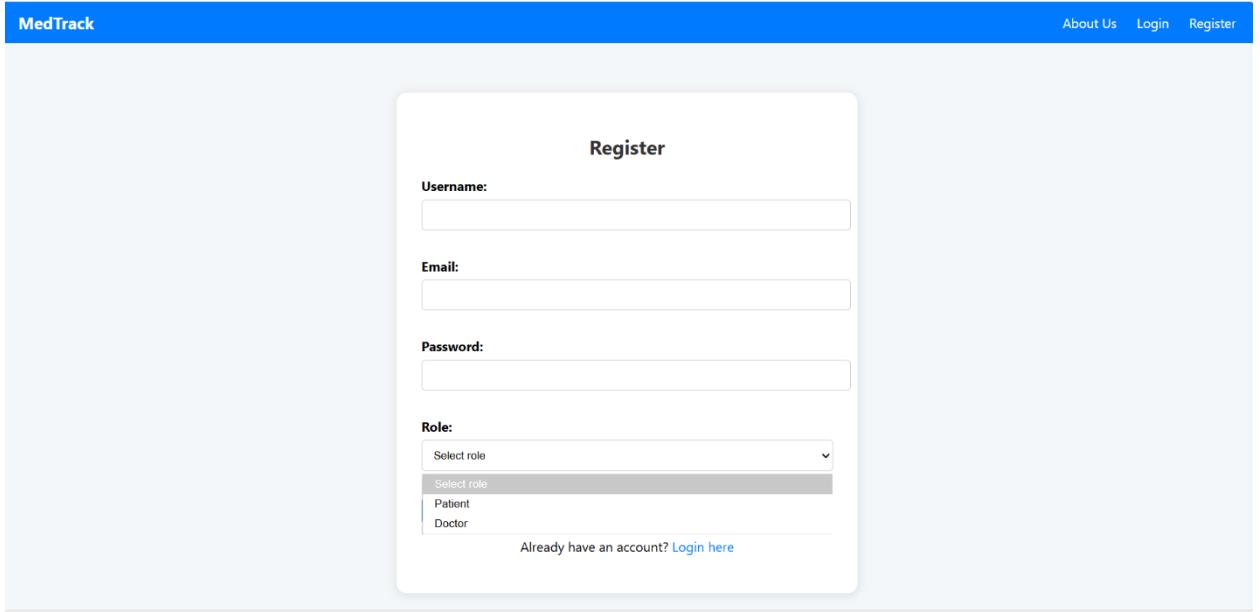
Password:

Enter your password

Login

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

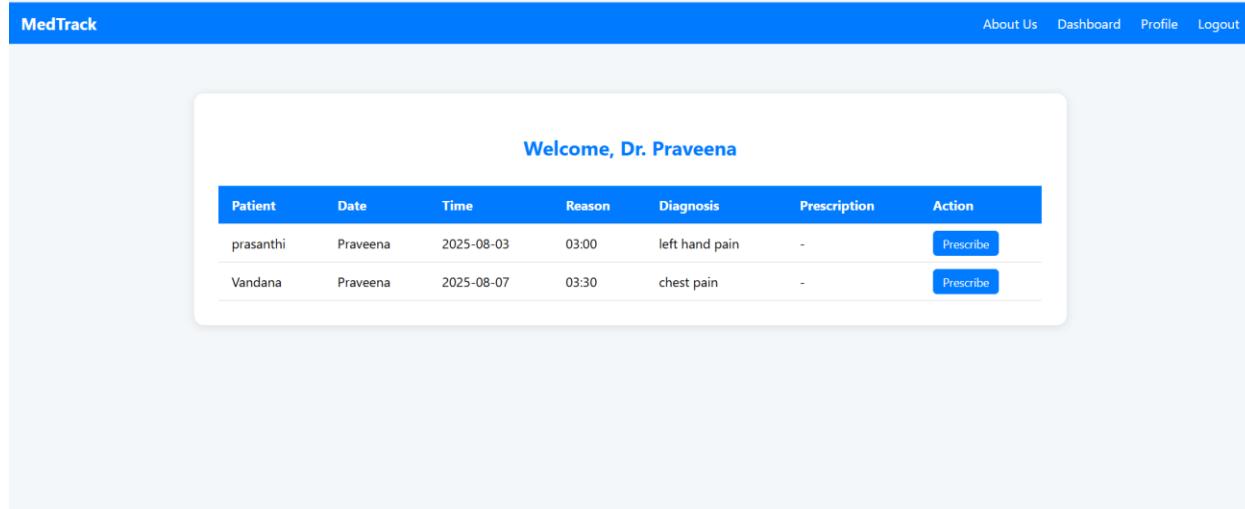
- **Doctor & Patient Registration:** Collects and validates credentials.



The screenshot shows the 'Register' page of the MedTrack application. At the top, there is a blue header bar with the 'MedTrack' logo on the left and 'About Us', 'Login', and 'Register' links on the right. Below the header is a large white registration form with a rounded rectangular border. The form has a title 'Register' at the top center. It contains four input fields: 'Username' (empty), 'Email' (empty), and 'Password' (empty). Below these is a 'Role' dropdown menu with 'Select role' as the placeholder. A dropdown list shows 'Patient' and 'Doctor', with 'Patient' currently selected. At the bottom of the form is a link 'Already have an account? [Login here](#)'.

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

→ Doctor Dashboard



The screenshot shows the 'Doctor Dashboard' page of the MedTrack application. At the top, there is a blue header bar with the 'MedTrack' logo on the left and 'About Us', 'Dashboard', 'Profile', and 'Logout' links on the right. Below the header is a large white dashboard area with a rounded rectangular border. At the top of this area, it says 'Welcome, Dr. Praveena'. Below this is a table with a blue header row containing columns: Patient, Date, Time, Reason, Diagnosis, Prescription, and Action. There are two rows of data:

Patient	Date	Time	Reason	Diagnosis	Prescription	Action
prasanthi	Praveena	2025-08-03	03:00	left hand pain	-	Prescribe
Vandana	Praveena	2025-08-07	03:30	chest pain	-	Prescribe

→ Patient Dashboard

The screenshot shows a web-based application named 'MedTrack'. At the top, there is a blue header bar with the 'MedTrack' logo on the left and navigation links for 'About Us', 'Dashboard', 'Profile', and 'Logout' on the right. Below the header, the main content area has a light gray background. In the center, it displays a 'Welcome, prasanthi' message. To the right of this message are three blue links: 'Logout', 'Book Appointment', and 'Profile'. Below the welcome message is a table with a light gray header row containing columns for 'Doctor', 'Date', 'Time', 'Reason', 'Status', and 'Actions'. The body of the table contains one row of data: 'prasanthi', 'Praveena', '2025-08-03', '03:00', 'left hand pain', and two buttons: 'Edit' (yellow) and 'Delete' (red). The entire interface is clean and modern, using a sans-serif font.

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.