

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 enabling diagnosis submissions. To address these challenges, the project utilizes Flask for backend development, AWS EC2 for hosting, and DynamoDB for managing data. MedTrack allows patients to register, log in, book appointments, and submit diagnosis reports online. The system ensures real-time notifications, enhancing communication between doctors and patients regarding appointments and medical submissions. Additionally, AWS Identity and Access Management (IAM) is employed to ensure secure access control to AWS resources, allowing only authorized users to access sensitive data. This cloud-based solution improves accessibility and efficiency in healthcare services for all users.

Scenarios:

Scenario 1: Efficient Appointment Booking System for Patients

In the MedTrack system, AWS EC2 provides a reliable infrastructure to manage multiple patients accessing the platform simultaneously. For example, a patient can log in, navigate to the appointment booking section, and easily submit a request for an appointment. Flask handles backend operations, efficiently retrieving and processing user data in real-time. The cloud-based architecture allows the platform to handle a high volume of appointment requests during peak periods, ensuring smooth operation without delays.

Scenario 2: Secure User Management with IAM

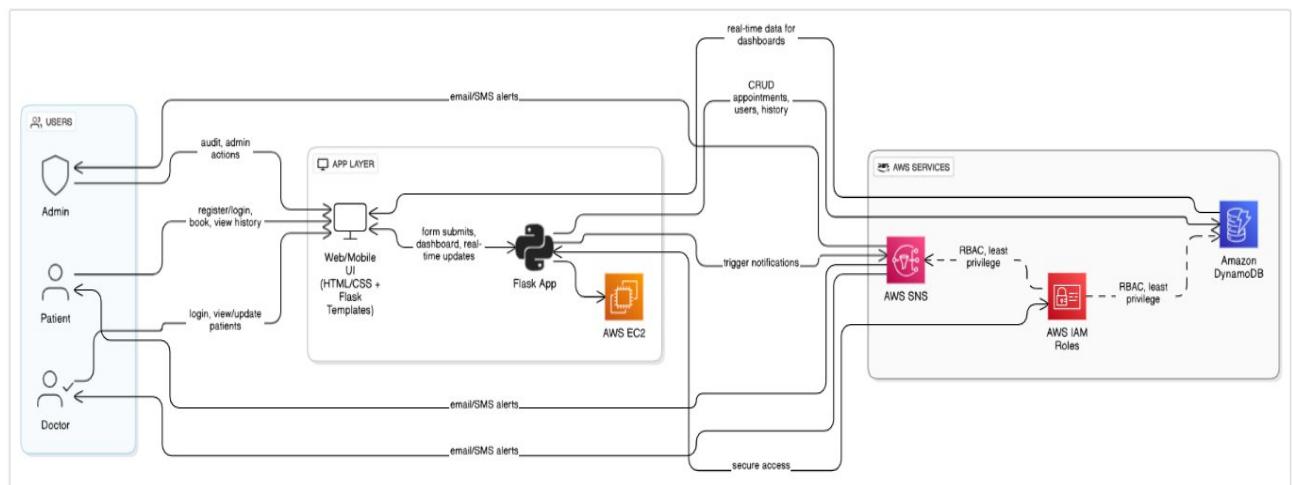
MedTrack utilizes AWS IAM to manage user permissions and ensure secure access to the system. For instance, when a new patient registers, an IAM user is created with specific roles and permissions to access only the features relevant to them. Doctors have their own IAM configurations, allowing them access to patient records and appointment details while maintaining strict security protocols. This setup ensures that sensitive data is accessible only to authorized users.

Scenario 3: Easy Access to Medical History and Resources

The MedTrack system provides doctors and patients with easy access to medical histories and relevant resources. For example, a patient logs in to view their medical history and upcoming appointments. Doctors diagnose conditions and prescribe necessary medications to their patients. Flask manages real-time data fetching from DynamoDB, while EC2 hosting ensures the platform performs seamlessly even when multiple users access it simultaneously, offering a smooth and uninterrupted user experience.

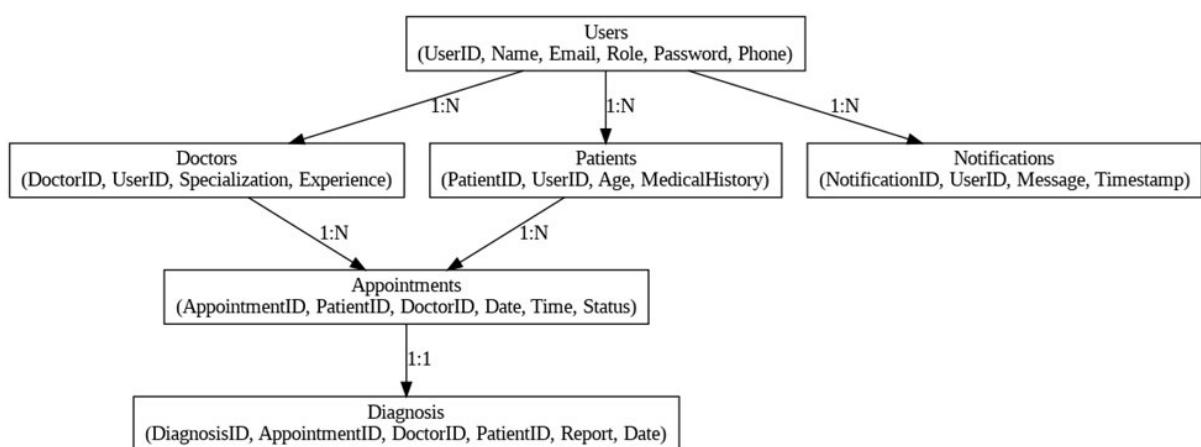
Architecture

This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for hosting the backend, with a lightweight framework like Flask handling core logic. Application data is stored in Amazon DynamoDB, ensuring fast, reliable access, while user access is managed through AWS IAM for secure authentication and control. Real-time alerts and system notifications are enabled via Amazon SNS, enhancing communication and user engagement.



Entity Relationship (ER) Diagram

An ER (Entity-Relationship) diagram visually represents the logical structure of a database by defining entities, their attributes, and the relationships between them. It helps organize data efficiently by illustrating how different components of the system interact and relate. This structured approach supports effective database normalization, data integrity, and simplified query design.



Pre-requisites

- AWS Account Setup:
<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
- AWS IAM (Identity and Access Management):
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
- AWS EC2 (Elastic Compute Cloud):
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
- AWS DynamoDB:
<https://docs.aws.amazon.com/amazondynamodb/Introduction.html>
- Amazon SNS:
<https://docs.aws.amazon.com/sns/latest/dg/welcome.html>
- Git Documentation:
<https://git-scm.com/doc>
- VS Code Installation: (download the VS Code using the below link or you can get that in Microsoft store)
<https://code.visualstudio.com/download>

Project WorkFlow

Milestone 1. AWS Account Setup and Login

Activity 1.1: Set up an AWS account if not already done.

Activity 1.2: Log in to the AWS Management Console.

Milestone 2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.

Activity 2.2: Configure Attributes for User Data and Book Requests.

Milestone 3. SNS Notification Setup

Activity 3.1: Create SNS topics for book request notifications.

Activity 3.2: Subscribe users and library staff to SNS email notifications.

Milestone 4. Backend Development and Application Setup

Activity 4.1: Develop the Backend Using Flask.

Activity 4.2: Integrate AWS Services Using boto3.

Milestone 5. IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

Milestone 6. EC2 Instance Setup

Activity 6.1: Launch an EC2 instance to host the Flask application.

Activity 6.2: Configure security groups for HTTP, and SSH access.

Milestone 7. Deployment on EC2

Activity 7.1: Upload Flask Files

Activity 7.2: Run the Flask App

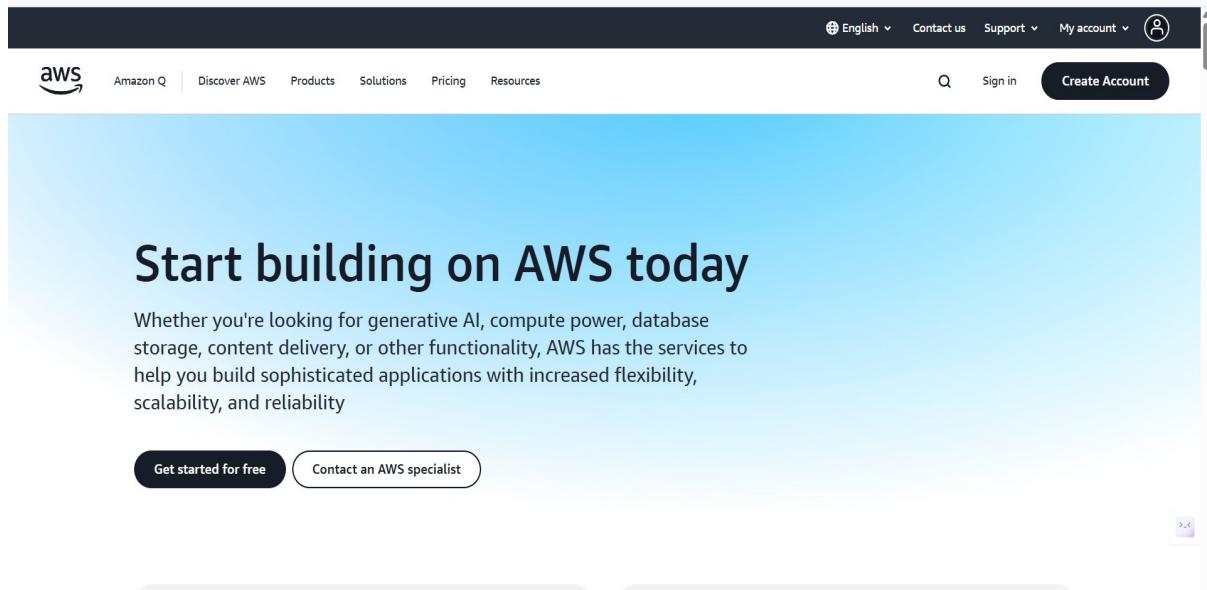
Milestone 8. Testing and Deployment

Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.

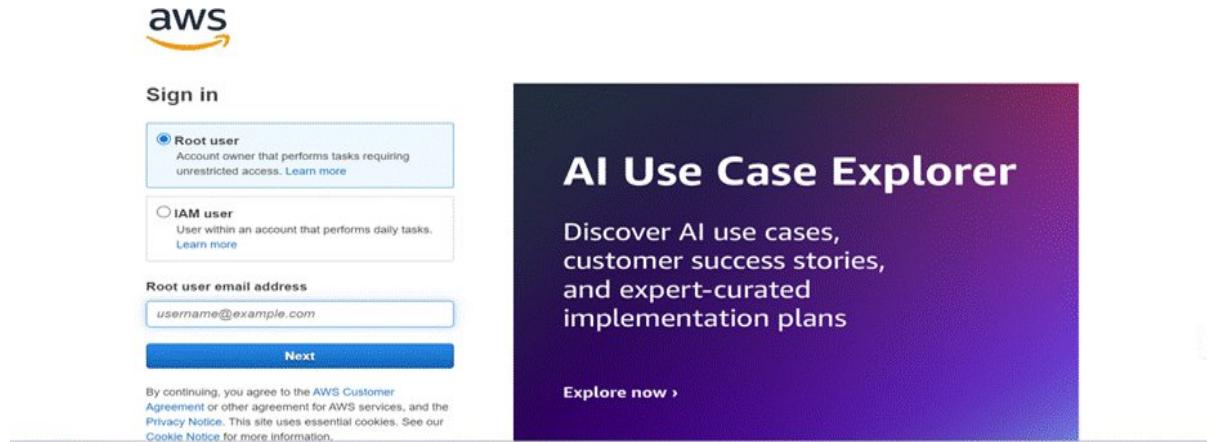
Milestone 1. AWS Account Setup and Login

Activity 1.1: Set up an AWS account if not already done.

- Sign up for an AWS account and configure billing settings.



- **Activity 1.2:** Log in to the AWS Management Console
 - After setting up your account, log in to the AWS Management Console.



Milestone 2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.

- In the AWS Console, navigate to DynamoDB and click on create tables.

The screenshot shows the AWS EC2 Services page with a search bar at the top containing 'dynamodb'. The results section is titled 'Services' and lists three services: 'DynamoDB' (Managed NoSQL Database), 'Amazon DocumentDB' (Fully-managed MongoDB-compatible database service), and 'CloudFront' (Global Content Delivery Network). Below this, under 'Features', there are sections for 'Settings' (DynamoDB feature), 'Clusters' (DynamoDB feature), and 'DHCP options sets' (VPC feature). On the left sidebar, the 'Instances' section is expanded, showing various EC2-related options like Instance Type, Launch Temp, and Events. At the bottom, there are 'Yes' and 'No' buttons for a feedback survey, along with the URL 'https://us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1'.

The screenshot shows the DynamoDB Dashboard. The left sidebar has sections for 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations', 'Reserved capacity', and 'Settings'. Under 'DAX', there are sections for 'Clusters', 'Subnet groups', 'Parameter groups', and 'Events'. The main area is titled 'Dashboard' and contains two sections: 'Alarms (0)' and 'DAX clusters (0)'. Both sections have search bars and status indicators. To the right, there is a 'Create resources' section for creating a new DynamoDB table, a 'Create DAX cluster' button, and a 'What's new' section with a note about AWS Cost Management.

The screenshot shows the 'Tables' page under the 'DynamoDB' navigation. The left sidebar includes 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations', and 'Reservations'. The main content area shows a table header with columns: Name, Status, Partition key, Sort key, Indexes, Deletion protection, Read capacity mode, Write capacity mode, and Total size. A message at the bottom states 'You have no tables in this account in this AWS Region.' There is a 'Create table' button at the bottom center.

- **Activity 2.2:** Create a DynamoDB table for storing registration details and book requests.

- Create medtrack_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 table name is 'medtrack_users'. The partition key is 'email' of type String. There is no sort key defined. Under 'Table settings', the 'Default settings' option is selected, which is the recommended choice. The table class is set to 'DynamoDB Standard' with provisioned capacity at 5 RCU and 5 WCU. Auto scaling is enabled. Local and global secondary indexes are not present. Resource-based policy is not active. In the 'Tags' section, there are no tags assigned. At the bottom right, there are 'Cancel' and 'Create table' buttons.

Table class	DynamoDB Standard	
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag
You can add 50 more tags.

Cancel Create table

- Create medtrack_appointments Table with partition key “appointment_id” with type String and click on create tables.

Screenshot of the AWS DynamoDB 'Create table' wizard.

Table details

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.
medtrack_appointments

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.

appointment_id	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.

Table settings

Default settings This feature helps you quickly create your table. You can modify most of these settings after your table has been created. To learn more about these settings, see Table settings.

Customize settings Use these advanced features to make DynamoDB work better for your needs.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Table class

Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag
You can add 50 more tags.

Cancel Create table

- Create medtrack_prescriptions Table with partition key “prescription_id” with type String and click on create tables.

Screenshot of the AWS DynamoDB 'Create table' wizard.

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.

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.
 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.
 String
1 to 255 characters and case sensitive.

Table settings

Default settings
The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

Customize settings
Use these advanced features to make DynamoDB work better for your needs.

- Similarly Create medtrack_medication_remainders Table with partition key “reminder_id” with type String and click on create tables.

Screenshot of the AWS DynamoDB 'Tables' page.

DynamoDB

- Dashboard
- Tables**
- Explore items
- PartiQL editor
- Backups
- Exports to S3
- Imports from S3
- Integrations New
- Reserved capacity
- Settings

DAX

- Clusters
- Subnet groups
- Parameter groups
- Events

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.

Tables (4) Info

Create table

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite
<input type="checkbox"/>	medtrack_appointments	Active	appointment_id (\$)	-	0 0	Off	Off	
<input type="checkbox"/>	medtrack_medication_reminders	Active	reminder_id (\$)	-	0 0	Off	Off	
<input type="checkbox"/>	medtrack_prescriptions	Active	prescription_id (\$)	-	0 0	Off	Off	
<input type="checkbox"/>	medtrack_users	Active	email (\$)	-	0 0	Off	Off	

Milestone 3. SNS Notification Setup

Activity 3.1: Create SNS topics for book request notifications.

- 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. On the left, there is a sidebar with links for various AWS services like DynamoDB, Route 53, and DAX. The main search results area has two sections: "Services" and "Features". Under "Services", the "Simple Notification Service" is listed with a brief description: "SNS managed message topics for Pub/Sub". Below it are "Route 53 Resolver" and "Route 53". Under "Features", there are sections for "Events" (ElastiCache feature) and "SMS" (AWS End User Messaging feature). At the bottom, there is a "Resources" section and a feedback card asking "let us know how we're doing." with a "Share feedback" button.

The screenshot shows the Amazon SNS dashboard. On the left, there is a sidebar with links for "Dashboard", "Topics", "Subscriptions", "Push notifications", and "Text messaging (SMS)". The main content area features a "New Feature" banner about in-place message archiving and replay for FIFO topics. Below it, there is a section titled "Application Integration" and a large heading "Amazon Simple Notification Service" with the subtext "Pub/sub messaging for microservices and serverless applications.". A detailed description of Amazon SNS follows. On the right, there is a "Create topic" form where the "Topic name" field is filled with "MyTopic". Below the form are "Next step" and "Start with an overview" buttons. At the bottom right, there is a "Pricing" link.

- 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 reads "New Feature" with the subtext "Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more". Below the banner, the page title is "Amazon SNS > Topics". On the left, a sidebar menu includes "Dashboard", "Topics", "Subscriptions", "Mobile" (with "Push notifications" and "Text messaging (SMS)" listed under it), and "Create topic". The main content area is titled "Topics (0)". It features a search bar, a table header with columns "Name" and "Type", and a message "No topics 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" wizard on the "Details" step. At the top, the AWS logo and navigation bar are visible. The main form has a "Type" section with two options: "FIFO (first-in, first-out)" and "Standard". "Standard" is selected and highlighted with a blue border. Below the type section is a "Name" field containing "Medtrack". A note says "Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_)." Below the name is a "Display name - optional" field with "My Topic". A note says "To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message." A note also says "Maximum 100 characters." Further down are sections for "Encryption - optional" (with a note about in-transit encryption) and "Access policy - optional" (with a note about defining access). A progress bar at the bottom indicates "3/5" completed.

► **Access policy - optional** [Info](#)
 This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

► **Data protection policy - optional** [Info](#)
 This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

► **Delivery policy (HTTP/S) - optional** [Info](#)
 The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

► **Delivery status logging - optional** [Info](#)
 These settings configure the logging of message delivery status to CloudWatch Logs.

► **Tags - optional**
 A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)

► **Active tracing - optional** [Info](#)
 Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

[Cancel](#)

[Create topic](#)

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

Amazon SNS

Dashboard

Topics

Subscriptions

▼ Mobile

Push notifications

Text messaging (SMS)

Medtrack

Details

Name	Medtrack	Display name	-
ARN	arn:aws:sns:us-east-1:418272775181:Medtrack	Topic owner	418272775181
Type	Standard		

Subscriptions (0)

ID	Endpoint	Status	Protocol

[Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#) [Create subscription](#)

- **Activity 3.2:** Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.
 - Subscribe users (or admin staff) to this topic via email. When a book request is made, notifications will be sent to the subscribed emails.

Create subscription

Details

Topic ARN
arn:aws:sns:us-east-1:418272775181:Medtrack

Protocol
The type of endpoint to subscribe
Email

Endpoint
An email address that can receive notifications from Amazon SNS.
manishwamsi1@gmail.com

ⓘ 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](#)

- After subscription request for the mail confirmation
- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.

in:spam

← Delete forever | Not spam | 📲 📤 ⚙️ ⚙️

1 of 2 < >

AWS Notification - Subscription Confirmation [Spam](#)

AWS Notifications <no-reply@sns.amazonaws.com>
to me

12:22 (0 minutes ago)

Why is this message in spam? This message is similar to messages that were identified as spam in the past.

Report as not spam

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:418272775181: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](#)



Simple Notification Service

Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

arn:aws:sns:us-east-1:418272775181:Medtrack:911baee0-b2d6-4c3d-b735-a29a602727f1

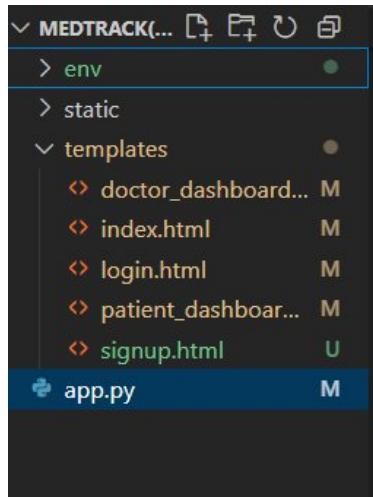
If it was not your intention to subscribe, [click here to unsubscribe](#).

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

Milestone 4. Backend Development and Application Setup

Activity 4.1: Develop the Backend Using Flask.

- File Explorer Structure



- **Description of the code:**

- **Flask App Initialization:**

```
from flask import Flask, render_template, request, redirect, url_for
import boto3
from boto3.dynamodb.conditions import Key
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from bcrypt import hashpw, gensalt, checkpw
```

Description: import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification

```
app = Flask( name )
```

- #### • **Dynamodb Setup:**

```
6
7     try:
8         dynamodb = boto3.resource(
9             'dynamodb',
10            region_name=AWS_REGION
11        )
12        sns_client = boto3.client(
13            'sns',
14            region_name=AWS_REGION
15        )
16
17        You, 1 second ago • Uncommitted changes
18        USERS_TABLE = dynamodb.Table('medtrack_users')
19        APPOINTMENTS_TABLE = dynamodb.Table('medtrack_appointments')
20        PRESCRIPTIONS_TABLE = dynamodb.Table('medtrack_prescriptions')
21        MEDICATION_REMINDERS_TABLE = dynamodb.Table('medtrack_medication_reminders')
22        logger.info("Boto3 clients and DynamoDB tables initialized successfully, assuming IAM Role credentials.")
```

Description: Initialize the DynamoDB resource for the ap-south-1 region and set up access to the Users and Requests tables for storing user details and book requests.

- **SNS Connection**

```
message = (f"Appointment cancelled: Patient {patient_email}'s appointment "
           f"with Dr. {appointment['doctor_name']} on {appointment['date']} "
           f"at {appointment['time']} has been cancelled.")
try:
    sns_client.publish(TopicArn=SNS_TOPIC_ARN, Message=message, Subject="Medtrack Appointment Cancelled")
    logger.info(f"SNS notification sent for appointment cancellation: {appointment_id}.")
except Exception as sns_e:
    logger.error(f"Failed to send SNS notification for cancellation: {sns_e}")

message = (f"New appointment booked: Patient {patient_name} ({patient_email}) "
           f"with Dr. {doctor_name} on {appointment_date} at {appointment_time} "
           f"for reason: {reason}.")
try:
    sns_client.publish(TopicArn=SNS_TOPIC_ARN, Message=message, Subject="New Medtrack Appointment")
    logger.info(f"SNS notification sent for new appointment: {new_appointment['appointment_id']} .")
except Exception as sns_e:
    logger.error(f"Failed to send SNS notification for appointment booking: {sns_e}")

message = (f"Your appointment with Dr. {appointment['doctor_name']} "
           f"on {appointment['date']} at {appointment['time']} has been updated to: {new_status}.")
try:
    sns_client.publish(TopicArn=SNS_TOPIC_ARN, Message=message, Subject="Medtrack Appointment Update")
    logger.info(f"SNS notification sent for appointment status update: {appointment_id}.")
except Exception as sns_e:
    logger.error(f"Failed to send SNS notification for status update: {sns_e}")
```

Description: Configure SNS to send notifications when a book request is submitted. Paste your stored ARN link in the sns_topic_arn space, along with the region_name where the SNS topic is created.

- **Routes for Web Pages:**

- **Index Route:**

```
@app.route('/')
def index():
    if 'user_email' in session:
        if session['user_type'] == 'patient':
            return redirect(url_for('patient_dashboard'))
        elif session['user_type'] == 'doctor':
            return redirect(url_for('doctor_dashboard'))
    return render_template('index.html')
```

- **Register Route:**

Verifies user credentials, increments login count, and redirects to the dashboard on success.

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

        if password != confirm_password:
            flash('Passwords do not match. Please try again.', 'error')
            return redirect(url_for('register'))

        try:
            response = USERS_TABLE.get_item(Key={'email': email})
            if 'Item' in response:
                flash('Email already registered. Please login or use a different email.', 'error')
                return redirect(url_for('register'))

            new_user = {
                'email': email,
                'name': name,
                'password': password,
                'user_type': user_type
            }
            if user_type == 'doctor':
                new_user['specialization'] = request.form.get('specialization', '')
                new_user['location'] = request.form.get('location', '')
                new_user['medical_license'] = request.form.get('medical_license', '')
            elif user_type == 'patient':
                new_user['age'] = request.form.get('age', '')
                new_user['gender'] = request.form.get('gender', '')

            USERS_TABLE.put_item(Item=new_user)
            flash('Account created successfully! Please login.', 'success')
            return redirect(url_for('login'))
        except Exception as e:
            logger.error(f"Error during user registration in DynamoDB: {e}")
            flash('An error occurred during registration. Please try again.', 'error')
            return redirect(url_for('register'))
    return render_template('signup.html')
```

- **Login Route (GET/POST):** The **login route** handles user authentication by verifying credentials stored in **DynamoDB**. Upon successful login, it increments the **login count** and redirects the user to their dashboard.

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

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

        if user and user['password'] == password:
            session['user_email'] = user['email']
            session['username'] = user['name']
            session['user_type'] = user['user_type']
            flash(f'Welcome, {user["name"]}!', 'success')

            if user['user_type'] == 'patient':
                return redirect(url_for('patient_dashboard'))
            elif user['user_type'] == 'doctor':
                return redirect(url_for('doctor_dashboard'))
            else:
                flash('Invalid email or password.', 'error')
    except Exception as e:
        logger.error(f"Error during login from DynamoDB: {e}")
        flash('An error occurred during login. Please try again.', 'error')
    return render_template('login.html')
```

- **Logout Route:** The logout functionality allows users to securely end their session, clearing any session data and redirecting them to the login page.

```
@app.route('/logout')
def logout():
    session.pop('user_email', None)
    session.pop('username', None)
    session.pop('user_type', None)
    flash('You have been logged out.', 'info')
    return redirect(url_for('index'))
```

- **Book Appointment Route:** The book appointment route allows users to select a date, time, and doctor for their appointment. Upon submission, the system stores the appointment details in DynamoDB and sends a confirmation notification via SNS.

```
@app.route('/book_appointment', methods=['POST'])
def book_appointment():
    if 'user_email' not in session or session['user_type'] != 'patient':
        flash('Unauthorized access.', 'error')
        return redirect(url_for('login'))

    patient_email = session['user_email']
    patient_name = session['username']

    doctor_name = request.form['doctor_name']
    appointment_date = request.form['appointment_date']
    appointment_time = request.form['appointment_time']
    reason = request.form['reason']  Manish Vamsi, 5 days ago • first commit ...
    try:
        new_appointment = {
            'appointment_id': str(uuid.uuid4()),
            'patient_email': patient_email,
            'patient_name': patient_name,
            'doctor_name': doctor_name,
            'date': appointment_date,
            'time': appointment_time,
            'reason': reason,
            'status': 'Pending'
        }
        APPOINTMENTS_TABLE.put_item(Item=new_appointment)

        message = (f"New appointment booked: Patient {patient_name} ({patient_email}) "
                   f"with Dr. {doctor_name} on {appointment_date} at {appointment_time} "
                   f"for reason: {reason}.")
        try:
            sns_client.publish(TopicArn=SNS_TOPIC_ARN, Message=message, Subject="New Medtrack Appointment")
            logger.info(f"SNS notification sent for new appointment: {new_appointment['appointment_id']}.")
        except Exception as sns_e:
            logger.error(f"Failed to send SNS notification for appointment booking: {sns_e}")

        flash('Appointment booked successfully! Awaiting doctor\'s approval.', 'success')
        return redirect(url_for('patient_dashboard', section='patient-appointments-section'))
    except Exception as e:
        logger.error(f"Error booking appointment to DynamoDB: {e}")
        flash('An error occurred while booking the appointment. Please try again.', 'error')
        return redirect(url_for('patient_dashboard', section='patient-book-appointment-section'))
```

- **Medication Remainder Route:** This route allows patients to set new medication reminders. It saves the medication details, schedule, and optional information to DynamoDB and sends an SNS notification.

```
@app.route('/add_medicationReminder', methods=['POST'])
def addMedicationReminder():
    if 'user_email' not in session or session['user_type'] != 'patient':
        flash('Unauthorized access.', 'error')
        return redirect(url_for('login'))

    patient_email = session['user_email']
    medication = request.form['medication']
    dosage = request.form['dosage']
    frequency = request.form['frequency']
    times = request.form.getlist('times[]')
    start_date_str = request.form['start_date']
    end_date_str = request.form.get('end_date')
    prescribed_by = request.form.get('prescribed_by')
    instructions = request.form.get('instructions')
    is_active = request.form.get('is_active') == 'on'

    try:
        datetime.strptime(start_date_str, '%Y-%m-%d').date()
        if end_date_str:
            datetime.strptime(end_date_str, '%Y-%m-%d').date()
    except ValueError:
        flash('Invalid date format for reminder. Please use YYYY-MM-DD.', 'error')
        return redirect(url_for('patient_dashboard', section='patient-medication-reminders-section'))
```

```
try:
    newReminder = {
        'reminder_id': str(uuid.uuid4()),
        'patient_email': patient_email,
        'medication': medication,
        'dosage': dosage,
        'frequency': frequency,
        'times': times,
        'date': start_date_str,
        'end_date': end_date_str if end_date_str else None,
        'prescribed_by': prescribed_by if prescribed_by else None,
        'instructions': instructions if instructions else None,
        'is_active': is_active,
        'status': 'Upcoming',
        'taken_today': False,
        'last_checked_date': datetime.now().strftime('%Y-%m-%d')
    }
    MEDICATION_REMINDERS_TABLE.put_item(Item=newReminder)

    message = (f"New medication reminder set: {medication} ({dosage}) "
               f"at {', '.join(times)} starting {start_date_str} (Frequency: {frequency.capitalize()}).")
    try:
        sns_client.publish(TopicArn=SNS_TOPIC_ARN, Message=message, Subject="Medtrack Medication Reminder Set")
        logger.info(f"SNS notification sent for new medication reminder: {newReminder['reminder_id']}.")
    except Exception as sns_e:
        logger.error(f"Failed to send SNS notification for reminder creation: {sns_e}")

    flash(f'Medication reminder for {medication} added successfully!', 'success')
    return redirect(url_for('patient_dashboard', section='patient-medication-reminders-section'))
except Exception as e:
    logger.error(f"Error adding medication reminder to DynamoDB: {e}")
    flash('An error occurred while adding the reminder. Please try again.', 'error')
    return redirect(url_for('patient_dashboard', section='patient-medication-reminders-section'))
```

- **Issue Prescription Route:** This route enables doctors to issue new prescriptions to patients. It stores the prescription details in DynamoDB and sends an SNS notification to inform the patient.

```

@app.route('/issue_prescription', methods=['POST'])
def issue_prescription():
    if 'user_email' not in session or session['user_type'] != 'doctor':
        flash('Unauthorized access.', 'error')
        return redirect(url_for('login'))

    doctor_name = session['username']
    patient_email = request.form['patient_email_prescribe']
    medication = request.form['medication']
    dosage = request.form['dosage']
    instructions = request.form['instructions']

    try:
        response = USERS_TABLE.get_item(Key={'email': patient_email})
        patient_user = response.get('Item')

        if not patient_user or patient_user.get('user_type') != 'patient':
            flash('Patient with this email does not exist or is not a patient user type.', 'error')
            return redirect(url_for('doctor_dashboard', section='doctor-issue-prescription-section'))

        new_prescription = {
            'prescription_id': str(uuid.uuid4()),
            'doctor_name': doctor_name,
            'patient_email': patient_email,
            'patient_name': patient_user['name'],
            'medication': medication,
            'dosage': dosage,
            'instructions': instructions,
            'date_prescribed': datetime.now().strftime('%Y-%m-%d')
        }
        PRESCRIPTIONS_TABLE.put_item(Item=new_prescription)

        message = f"New prescription issued: Dr. {doctor_name} prescribed {medication} ({dosage}) " \
                  f"for {patient_user['name']} ({patient_email}). Instructions: {instructions}"
        try:
            sns_client.publish(TopicArn=SNS_TOPIC_ARN, Message=message, Subject="Medtrack New Prescription")
            logger.info(f"SNS notification sent for new prescription: {new_prescription['prescription_id']}.")
        except Exception as sns_e:
            logger.error(f"Failed to send SNS notification for prescription: {sns_e}")

        flash(f'Prescription for {patient_user["name"]} issued successfully!', 'success')
        return redirect(url_for('doctor_dashboard', section='doctor-prescriptions-section'))
    except Exception as e:
        logger.error(f"Error issuing prescription to DynamoDB: {e}")
        flash('An error occurred while issuing the prescription. Please try again.', 'error')
        return redirect(url_for('doctor_dashboard', section='doctor-issue-prescription-section'))

```

- **Deployment Code:**

```

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

```

Milestone 5. IAM Role Setup

Activity 5.1: Create IAM Role

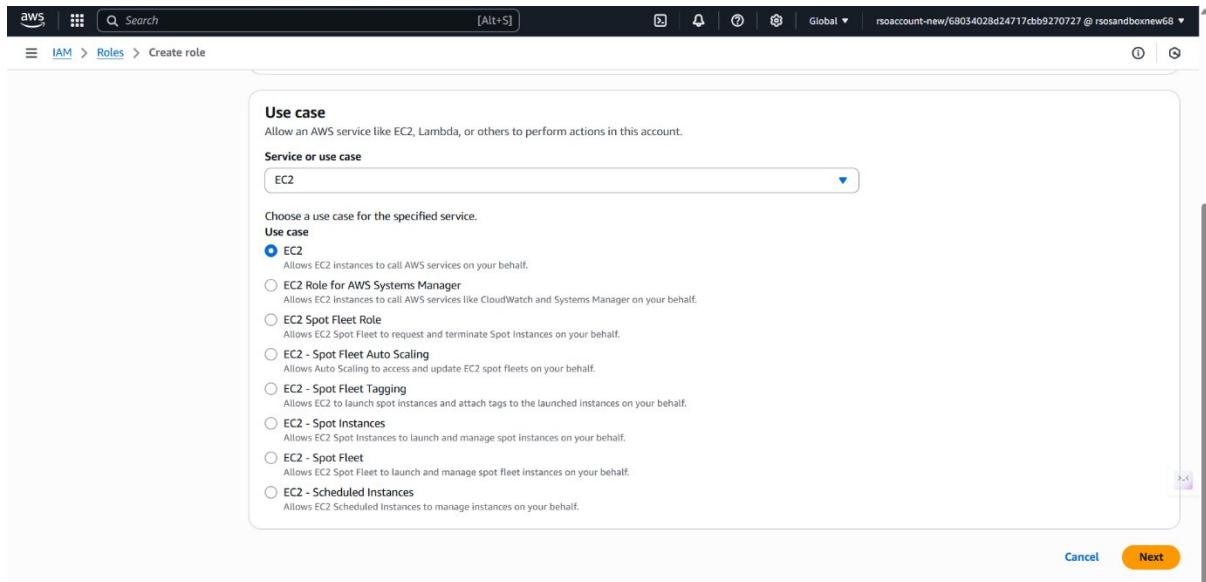
- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.

The screenshot shows the AWS Services search results for 'iam'. The search bar at the top contains 'iam'. Below it, a sidebar on the left lists various services and features: Services, Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main area displays a list of services under 'Services':

- 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 'Create role' wizard in the AWS IAM console. The current step is 'Step 1: Select trusted entity'. The navigation bar at the top shows 'aws' and the path 'IAM > Roles > Create role'. The main content area has the following sections:

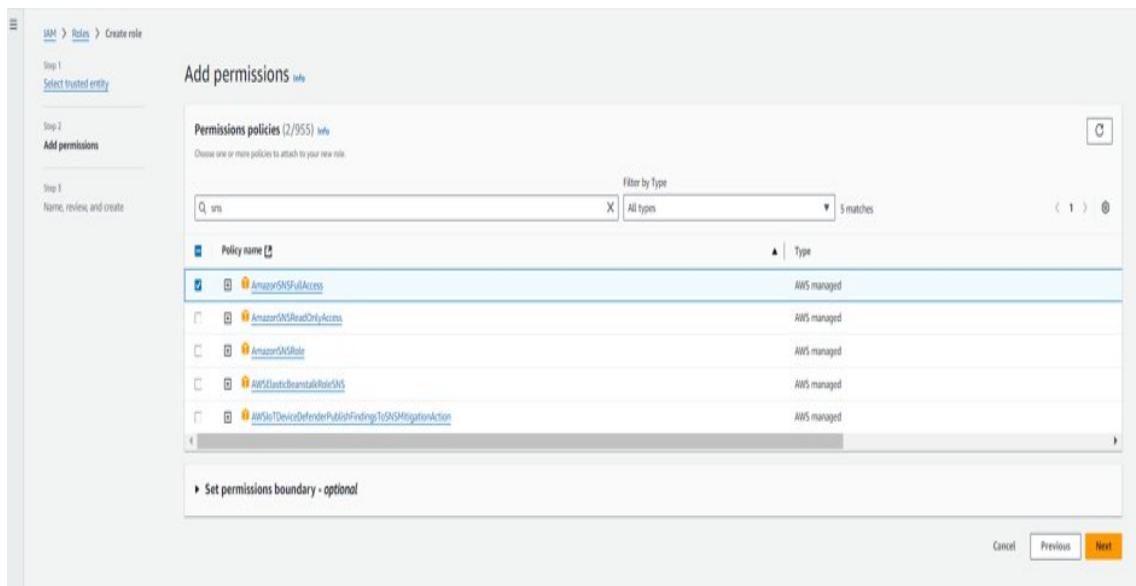
- Step 1**:
 - Select trusted entity** (radio button selected)
 - Step 2: Add permissions
 - Step 3: Name, review, and create
- Select trusted entity** (Info icon): This section is titled 'Trusted entity type'. It contains four options:
 - AWS service** (radio button selected): Allows AWS services like EC2, Lambda, or others to perform actions in this account.
 - AWS account**: Allows 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**: Allows 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**: Allows an AWS service like EC2, Lambda, or others to perform actions in this account. A dropdown menu labeled 'Choose a service or use case' is shown.



- **Activity 5.2: Attach Policies.**

Attach the following policies to the role:

- **AmazonDynamoDBFullAccess:** Allows EC2 to perform read/write operations on DynamoDB.
- **AmazonSNSFullAccess:** Grants EC2 the ability to send notifications via SNS.



Screenshot of the AWS IAM 'Create role' wizard - Step 3: Name, review, and create.

Role details

Role name: EC2_MedTrack_Role

Description: Allows EC2 instances to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy:

```

1 "[
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "sts:AssumeRole"
8       ],
9       "Principal": [
10      {
11        "Service": [
12          "ec2.amazonaws.com"
13        ]
14      }
15    ]
16  ]
]

```

Screenshot of the AWS IAM 'Roles' page.

Identity and Access Management (IAM)

Roles (9) Info

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

Role name	Trusted entities	Last activity
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service)	-
EC2_MedTrack_Role	AWS Service: ec2	-
OrganizationAccountAccessRole	Account: 058264256896	53 minutes ago
rsoaccount-new	Account: 058264256896	22 minutes ago

Roles Anywhere Info

Authenticate your non AWS workloads and securely provide access to AWS services.

Access AWS from your non AWS workloads

Operate your non AWS workloads using the same authentication and authorization strategy that you use within AWS.

X.509 Standard

Use your own existing PKI infrastructure or use AWS Certificate Manager Private Certificate Authority to authenticate identities.

Temporary credentials

Use temporary credentials with ease and benefit from the enhanced security they provide.

- **Milestone 6. EC2 Instance Setup**

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

The screenshot shows a GitHub repository page for 'ManishVamsi/MedTrack'. At the top, there's a table of commits:

File	Commit Message	Time Ago
env	first commit	yesterday
templates	first commit	yesterday
app.py	Update app.py	3 hours ago

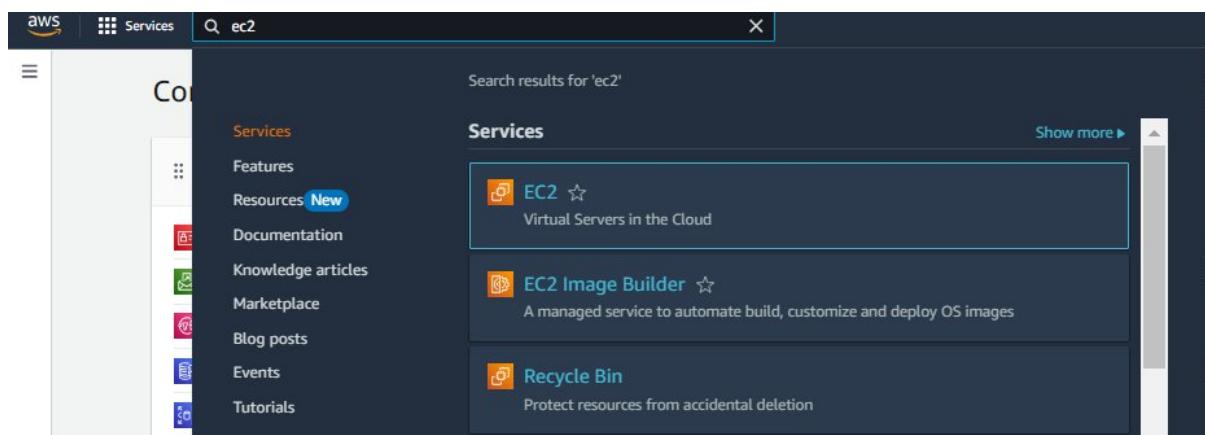
Below the commits, there are two tabs: 'Local' and 'Codespaces', with 'Codespaces' being active. Underneath are cloning options:

- Clone**: Includes links for HTTPS, SSH, and GitHub CLI.
- https://github.com/ManishVamsi/MedTrack.git**: A copy icon is next to this link.
- Clone using the web URL.**
- Open with GitHub Desktop**
- Download ZIP**

Activity 6.1: Launch an EC2 instance to host the Flask application.

- **Launch EC2 Instance**

- 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. On the left, there's a sidebar with options like EC2 Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, and Savings Plans. The main content area has a header 'Instances Info' with filters for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DN. A search bar at the top says 'Find Instance by attribute or tag (case-sensitive)'. Below the header, it says 'Last updated less than a minute ago' and 'All states'. A message 'No instances' and 'You do not have any instances in this region' is displayed, along with a 'Launch instances' button.

This screenshot shows the 'Launch an instance' wizard. Step 1: Name and tags. It asks for a name ('medtrack-server') and allows adding additional tags. Step 2: Application and OS Images (Amazon Machine Image). It lists various AMI categories: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian. A search bar is available. Step 3: Quick Start, showing a grid of AMI icons. Step 4: Summary, showing 1 instance selected. Step 5: Software Image (AMI) selected: Amazon Linux 2 Kernel 5.10 AMI. Step 6: Virtual server type (instance type) selected: t2.micro. Step 7: Firewall (security group) selected: New security group. Step 8: Storage (volumes) selected: 1 volume(s) - 8 GiB. Step 9: Buttons for 'Cancel', 'Launch instance' (highlighted in orange), and 'Preview code'.

This screenshot continues the 'Launch an instance' wizard. Step 4: Software Image (AMI) selected: Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type. Step 5: Virtual server type (instance type) selected: t2.micro. Step 6: Firewall (security group) selected: New security group. Step 7: Storage (volumes) selected: 1 volume(s) - 8 GiB. Step 8: Buttons for 'Cancel', 'Launch instance' (highlighted in orange), and 'Preview code'.

- Create and download the key pair for server access:

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

Select ▼  [Create new key pair](#)



medtrack-server.pem

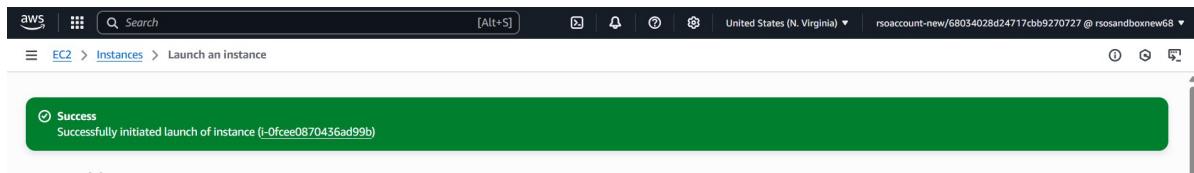
1,674 B • 8 minutes ago

[Full download history](#) 

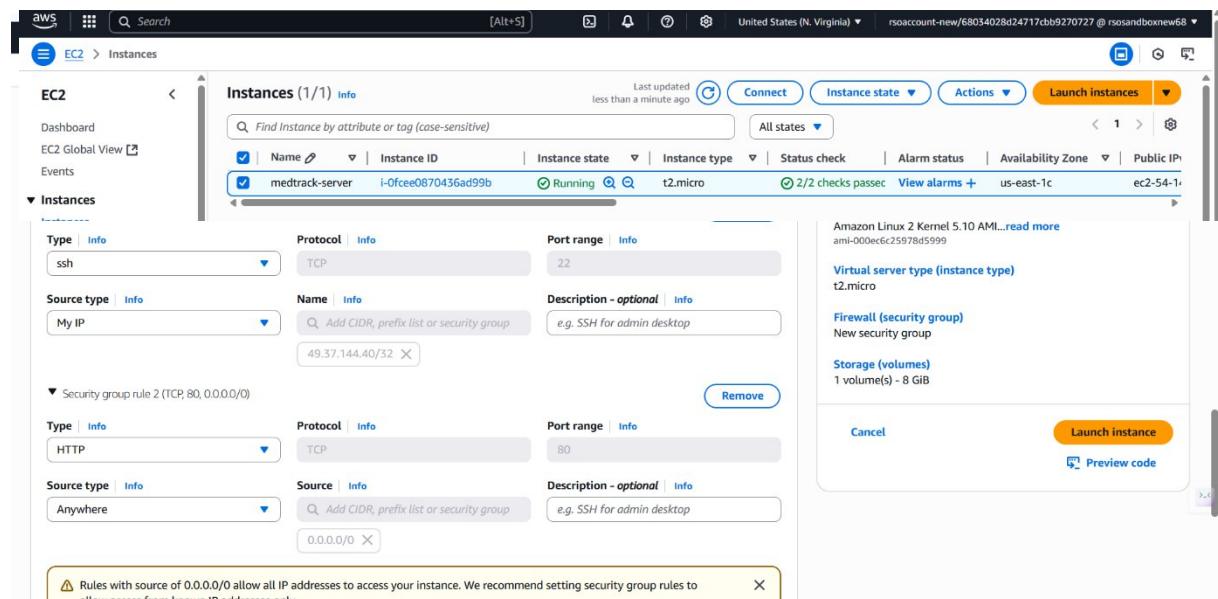
- **Activity 6.2: Configure security groups for HTTP, and SSH access.**

For network settings during EC2 instance launch:

1. In the **Network Settings** section, select the **VPC** and **Subnet** you wish to use (if unsure, the default VPC and subnet should work).
2. Ensure **Auto-assign Public IP** is enabled so your instance can be accessed from the internet.
3. In **Security Group**, either select an existing one or create a new one that allows SSH (port 22) access to your EC2 instance for remote login.



- 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.



To modify the **IAM role** for your EC2 instance:

- Go to the **AWS IAM Console**, select **Roles**.
- Click **Attach Policies**, then choose the required policies (e.g., **DynamoDBFullAccess**, **SNSFullAccess**) and click **Attach Policy**.
- If needed, update the instance to use this modified role by selecting the EC2 instance, clicking **Actions**, then **Security**, and **Modify IAM role** to select the updated role.

- Connect to your EC2 instance:

1. Go to the **EC2 Dashboard**, select your running instance, and click **Connect**.

- Now connect the EC2 with the files

AWS CloudShell terminal window showing a warning message about the end of life for Amazon Linux 2. The message indicates that AL2 End of Life is 2026-06-30, and a newer version of Amazon Linux is available, Amazon Linux 2023, GA and supported until 2028-03-15. A link to the documentation is provided: <https://aws.amazon.com/linux/amazon-linux-2023/>.

```
[ec2-user@ip-172-31-19-220 ~]$
```

Milestone 7: Deployment on EC2

Deployment on an EC2 instance involves launching a server, configuring security groups for public access, and uploading your application files. After setting up necessary dependencies and environment variables, start your application and ensure it's running on the correct port. Finally, bind your domain or use the public IP to make the application accessible online.

Activity 7.1: 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
```

Activity 7.2: Clone Your Flask Project from GitHub

Clone your project repository from GitHub into the EC2 instance using Git.

Run: ‘git clone <https://github.com/your-github-username/your-repository-name.git>’

Note: change your-github-username and your-repository-name with your credentials
here: ‘git clone https://github.com/Ravi-teja-777/medtrack_app.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

```
sudo flask run --host=0.0.0.0 --port=5000
```

```

[root@ip-172-31-19-228 MedTrack]# git pull origin master
fatal: couldn't find remote ref master
[root@ip-172-31-19-228 MedTrack]# git pull origin main
From https://github.com/ManishVamsi/Medtrack
 * branch      main      -> FETCH_HEAD
Already up to date.
[root@ip-172-31-19-228 MedTrack]# ls
app.py  config  templates
[root@ip-172-31-19-228 MedTrack]# python3 app.py
/usr/local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/devops/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn(warning, PythonDeprecationWarning)
INFO:botocore.credentials:Found credentials from IAM Role: EC2_MedTrack_Role
INFO:  main :Boto3 clients and dynamoDB tables initialized successfully, assuming IAM Role credentials.
 * Serving Flask app 'app'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.19.228:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
/usr/local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/devops/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn(warning, PythonDeprecationWarning)
INFO:botocore.credentials:Found credentials from IAM Role: EC2_MedTrack_Role
INFO:  main :Boto3 clients and dynamoDB tables initialized successfully, assuming IAM Role credentials.
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 841-459-071

```

i-0fcee0870436ad99b (medtrack-server)
PublicIPs: 54.145.226.169 PrivateIPs: 172.31.19.228

Verify the Flask app is running:

[http://your-ec2-public-ip](http://54.145.226.169:5000)

- Run the Flask app on the EC2 instance

Access the website through:

Public-IPs: <https://54.145.226.169:5000/>

Milestone 8: Testing and Deployment

- **Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.**

Functional testing to verify the project

Home Page:

The Home Page of your project is the main entry point for users, where they can interact with the system. It typically includes:

1. Input Fields: For users to enter basic information like appointment requests, diagnosis submissions, or service bookings.
2. Navigation: Links to other sections such as the login page, dashboard, or service options.
3. Responsive Design: Ensures the page is accessible across devices with a clean, user-friendly interface.

The Home Page serves as the initial interface that directs users to the key functionalities of your web application.

Revolutionizing Healthcare Access

Connect with doctors instantly, manage your medications seamlessly, and take control of your health journey.

Get Started Today!

CORE FEATURES

A smarter way to manage your health



Doctor Consultations

Book appointments with qualified doctors based on your health needs. Video consultations available for remote care.



Medication Reminders

Never miss a dose with our smart medication reminder system. Get alerts and track your medication history.



Appointment Management

Easily schedule, reschedule or cancel appointments. Receive automatic reminders before your appointments.



Secure Health Records

Your health data is encrypted and stored securely. Access your medical history anytime, anywhere.

DOCTOR AND PATIENT REGISTRATION PAGE:

The Doctor Registration Page allows doctors to register and create an account on the platform. It typically includes:

1. Input Fields: For doctor details such as name, specialty, qualifications, and contact information.
2. Login Credentials: Fields for setting a username and password for secure access.
3. Submit Button: A button to submit the registration details, which will then be stored in the database after validation.

PATIENT AND DOCTOR LOGIN PAGES:

The Patient and Doctor Login Pages allow users to securely access their accounts on the platform. Each login page typically includes:

1. Username and Password Fields: Users enter their credentials (username and password) to authenticate their account.
2. Login Button: A button to submit login details and validate user access.

Once logged in, patients and doctors are redirected to their respective dashboards to manage appointments, medical records, and other relevant tasks.

Create Your Account

Full Name

John Doe

Email

you@example.com

Password

Confirm Password

I am a:

Patient Doctor

Age

e.g., 30

Gender

-- Select Gender --

I agree to the Terms and Conditions

Create Account

Create Your Account

Full Name

John Doe

Please fill out this field

Email

you@example.com

Password

Confirm Password

I am a:

Patient Doctor

Specialization

e.g., Cardiology

Location (City, Clinic Name)

e.g., Srikakulam, City Hospital

Medical License Number

e.g., ML123456

I agree to the Terms and Conditions

Create Account

Login to Medtrack

Email

you@example.com

Password

Sign In

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

User Dashboard:

The User Dashboard (for patients) provides an easy interface to manage appointments and track their status. It typically includes:

1. Book Appointment Section: A form for selecting a doctor, choosing an appointment time, and submitting the request.
2. Appointment Status: A section showing the current status of appointments (e.g., confirmed, pending, or completed) with options to view details or cancel.
3. Medication Remainders: Patients can easily set up personalized reminders for their medications, including dosage, frequency, and specific times.

This dashboard helps patients book new appointments and keep track of their healthcare schedules.

The screenshot displays the Medtrack Patient Dashboard. At the top, there is a navigation bar with a heart icon, the brand name "Medtrack", and links for "Home", "Welcome, Tony", and "Logout". Below the navigation bar, a green header bar says "Welcome, Tony!". The main content area is titled "Patient Dashboard". A horizontal navigation menu at the top of this area includes "Appointments" (which is highlighted in blue), "Book Appointment", "Prescriptions", "Medication Reminders", and "All Doctors".

The "Your Appointments" section shows one entry for "Dr. Roy" with the details: Date: 2025-06-27 at 09:30, Reason: Fever and headache, and Status: Completed.

The "Book New Appointment" section contains fields for "Select Doctor" (a dropdown menu with an option "-- Select a Doctor --"), "Date" (a date input field with placeholder "dd-mm-yyyy"), "Time" (a time input field with placeholder "--:--"), and "Reason for Appointment or Describe your Symptoms" (a text input field with placeholder "e.g., Routine check-up, Fever, Consultation..."). A note "Please fill out this field." is displayed next to the symptoms input field. A blue "Book Appointment" button is located at the bottom of this section.

Your Medication Reminders

Add New Medication

Medication Name *

e.g., Metformin, Lisinopril

Dosage *

e.g., 500mg, 10mg, 1 tablet

Schedule

Frequency *

Once daily

Times *

--:--

🕒 + -

Start Date *

04-07-2025

📅

End Date

dd-mm-yyyy

📅

 Medication is currently active

Cancel

Add Medication

All Registered Doctors

Dr. Steven

Specialization: Cardiology

Location: Hyderabad, Yashodha Hospitals

Email: kmanishvamsi@gmail.com

Medical License: ML12345

Dr. Roy

Specialization: General Physician

Location: Delhi, Fortis Hospital

Email: 22a51a1295@adityatekkali.edu.in

Medical License: ML13456

Doctor Dashboard:

The **Doctor Dashboard** provides doctors with a comprehensive view of their upcoming appointments and patient details. It typically includes:

- Upcoming Appointments List:** A table or list showing patient names, appointment times, and appointment statuses (e.g., confirmed, pending).
- Patient Details:** Quick access to each patient's medical history, contact information, and previous visit records.
- Appointment Actions:** Options to view, confirm, or cancel appointments, ensuring efficient management.

The dashboard serves as the main interface for doctors to manage their schedules, track patient interactions, and provide timely care.

Welcome, Roy!

Doctor Dashboard

Appointments Issue Prescription View Prescriptions

Patient Appointments

PATIENT NAME	PATIENT EMAIL	DATE	TIME	REASON	STATUS	ACTIONS
Tony	manishvamsi1@gmail.com	2025-06-27	09:30	Fever and headache	Completed	Completed

Appointments Issue Prescription View Prescriptions

Issue New Prescription

Patient Email:

patient@example.com

Medication Name:

e.g., Amoxicillin

Dosage:

e.g., 250mg, 1 tablet daily

Instructions:

e.g., Take with food, Finish the course...

Please fill

Issue Prescription

Doctor Dashboard

Appointments Issue Prescription View Prescriptions

Issued Prescriptions

PATIENT DETAILS	MEDICATION	DOSAGE	INSTRUCTIONS	DATE ISSUED
Tony manishvamsi1@gmail.com	Dolo 650	Twice a Day	Take with Food For 4 days.	2025-06-27

DynamoDB Database:

1. medtrack_users:

This table stores all user profiles for the Medtrack application, including patient and doctor accounts. It contains essential login credentials, personal details, and user-specific attributes like specialization or age.

2. medtrack_appointments:

This table is dedicated to managing all scheduled medical appointments within the system. It records details such as the patient, doctor, date, time, reason for visit, and the current status of each appointment.

3. medtrack_prescriptions:

This table holds all digital prescriptions issued by doctors through the application. It details the prescribed medication, dosage, instructions, the prescribing doctor, and the patient it was issued for.

4. medtrack_medication_reminders:

This table stores personalized medication reminders set by patients to help them adhere to their treatment plans. It includes the medication name, dosage, frequency, times, and tracks whether a dose has been taken

Table: medtrack_users - Items returned (2/4)

Scan started on July 04, 2025, 13:57:18

	email (String)	age	gender	location	medical_license	name
<input type="checkbox"/>	ghgfhf@gmail.com	66	other			Hjj
<input checked="" type="checkbox"/>	kmanishvamsi@gmail...			Hyderabad,...	ML12345	Steve
<input checked="" type="checkbox"/>	manishvamsi1@gmail...	20	male			Manis
<input type="checkbox"/>	html@gmail.com	66	male			John

Table: medtrack_prescriptions - Items returned (1)

Scan started on July 04, 2025, 13:57:35

	prescription_id (String)	date_prescribed	doctor_name	dosage	instructions
<input type="checkbox"/>	61e8c798-0c09-4f35-ab44-4...	2025-07-04	Steven	1 tablet	Take with food

Table: medtrack_appointments - Items returned (2)

Scan started on July 04, 2025, 13:57:41

	appointment_id (String)	date	doctor_name	patient_email	patient_name
<input type="checkbox"/>	ec321f59-39c3-44c4-b2c8-d...	2025-07-04	Steven	html@gmail.com	John doe
<input type="checkbox"/>	15e91b04-d588-473...	2025-07-04	Steven	manishvamsi1@...	Manish

Conclusion:

The **MedTrack application** has been successfully developed and deployed using a robust cloud-based architecture tailored for modern healthcare environments. Leveraging AWS services such as EC2 for hosting, DynamoDB for secure and scalable patient data management, and SNS for real-time alerts, the platform ensures reliable and efficient access to essential medical tracking services. This system addresses critical challenges in healthcare such as managing patient records, monitoring medication schedules, and ensuring timely communication between healthcare providers and patients.

The cloud-native approach enables seamless scalability, allowing MedTrack to support increasing numbers of users and data without compromising performance or reliability. The integration of Flask with AWS ensures smooth backend operations, including patient registration, medication reminders, and health updates. Thorough testing has validated that all features—from user onboarding to alert notifications—function reliably and securely.

In conclusion, the MedTrack application delivers a smart, efficient solution for modernizing healthcare management, improving patient care, and streamlining communication between medical staff and patients. This project highlights the transformative power of cloud-based technologies in solving real-world challenges in the healthcare sector.