

## **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.

### **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 page, 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.

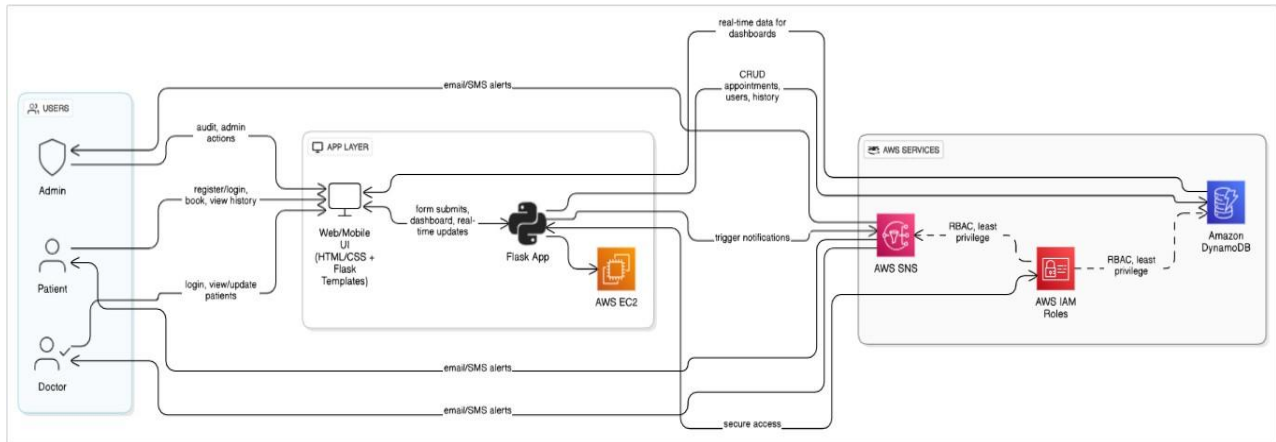
### **Scenario 2: Secure User Management with IAM**

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.

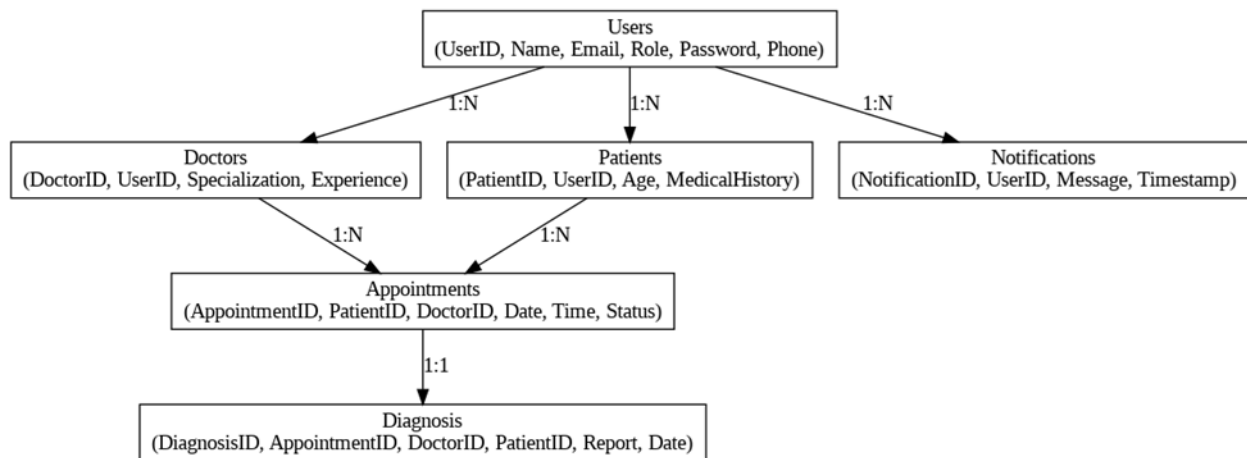
### **Scenario 3: Easy Access to Medical History and Resources**

The MedTrack system provides doctors and patients with easy access to medical histories and relevant resources. For example, a doctor logs in to view a patient's medical history and upcoming appointments. They can quickly access, and update records as needed. 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.

## AWS ARCHITECTURE



## Entity Relationship (ER)Diagram:



## Pre-requisites:

1. **AWS Account Setup:** [AWS Account Setup](#)
2. **Understanding IAM:** [IAM Overview](#)
3. **Amazon EC2 Basics:** [EC2 Tutorial](#)
4. **DynamoDB Basics:** [DynamoDB Introduction](#)
5. **SNS Overview:** [SNS Documentation](#)
6. **Git Version Control:** [Git Documentation](#)

## **Project WorkFlow:**

### **Milestone 1. Web Application Development and Setup**

- Develop the Backend Using Flask.
- Integrate AWS Services Using boto3.

### **Milestone 2. AWS Account Setup and Login**

- Set up an AWS account if not already done.
- Login to AWS Management Console.

### **Milestone 3. DynamoDB Database Creation and Setup**

- Create a DynamoDB Table.
- Configure Attributes for User Data and Book Requests.

### **Milestone 4. SNS Notification Setup**

- Create SNS topics for book request notifications.
- Subscribe users and library staff to SNS email notifications.

### **Milestone 5. IAM Role Setup**

- Create IAM Role
- Attach Policies

### **Milestone 6. EC2 Instance Setup**

- Launch an EC2 instance to host the Flask application.
- Configure security groups for HTTP, and SSH access.

### **Milestone 7. Deployment using EC2**

- Upload Flask Files
- Run the Flask App

### **Milestone 8. Testing and Deployment**

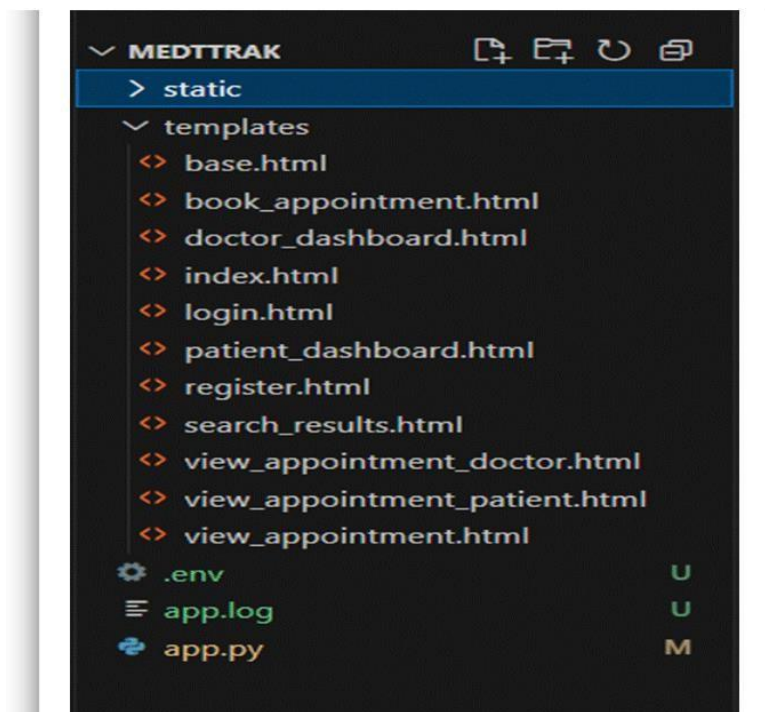
- Conduct functional testing to verify user registration, login, book requests, and notifications

## Milestone 1: Web Application Development and Setup

Backend Development and Application Setup focuses on establishing the core structure of the application. This includes configuring the backend framework, setting up routing, and integrating database connectivity. It lays the groundwork for handling user interactions, data management, and secure access

### FLASK DEPLOYMENT

File Explorer Structure



### Flask App Initialization:

In the MedTrack project, the Flask app is initialized to establish the backend infrastructure, enabling it to handle multiple user interactions such as patient registration, appointment booking, and submission of medical reports. The Flask framework processes incoming requests, communicates with the DynamoDB database for storing user data, and integrates seamlessly with AWS services. Additionally, the routes and APIs are defined to manage different functionalities like secure login, appointment scheduling, and medical history retrieval. This initialization sets up the foundation for smooth, real-time communication between patients and doctors while ensuring the app is scalable and secure.

Use boto3 to connect to DynamoDB for handling user registration, book requests database operations and also mention region\_name where Dynamodb tables are created.

```

app.py
app.py > ...
15 # -----
16 # App Configuration (Inline or via .env)
17 # -----
18 AWS_REGION_NAME = os.environ.get('AWS_REGION_NAME', 'us-east-1')
19 DYNAMODB_USERS_TABLE = os.environ.get('DYNAMODB_USERS_TABLE', 'UsersTable')
20 DYNAMODB_APPOINTMENTS_TABLE = os.environ.get('DYNAMODB_APPOINTMENTS_TABLE', 'A
21 SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')
22 ENABLE_EMAIL = os.environ.get('ENABLE_EMAIL', 'True').lower() == 'true'
23 SMTP_SERVER = os.environ.get('SMTP_SERVER', 'smtp.gmail.com')
24 SMTP_PORT = int(os.environ.get('SMTP_PORT', 587))
25 SENDER_EMAIL = os.environ.get('SENDER_EMAIL', 'your@email.com')
26 SENDER_PASSWORD = os.environ.get('SENDER_PASSWORD', 'your-app-password')
27
28 # -----
29 # AWS Resources
30 # -----
31 dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION_NAME)
32 sns = boto3.client('sns', region_name=AWS_REGION_NAME)
33 user_table = dynamodb.Table(DYNAMODB_USERS_TABLE)
34 appointment_table = dynamodb.Table(DYNAMODB_APPOINTMENTS_TABLE)
35

```

In the MedTrack project, AWS SNS sends real-time notifications to patients and doctors about appointments and updates. DynamoDB stores user data, medical records, and appointments securely, offering fast, scalable access. Both services are integrated with Flask to ensure smooth communication and efficient data management

```

MedTrack_app
app.py
app.py > ...
1 from flask import Flask, request, session, redirect, url_for, render_template,
2 import boto3
3 from werkzeug.security import generate_password_hash, check_password_hash
4 from datetime import datetime
5 import uuid
6 import logging
7 import os
8
9 # -----
10 # Flask App Initialization
11 # -----
12 app = Flask(__name__)
13 app.secret_key = os.environ.get('FLASK_SECRET_KEY', 'super-secret-key')
14
15 # -----
16 # App Configuration (Inline or via .env)
17 # -----
18 AWS_REGION_NAME = os.environ.get('AWS_REGION_NAME', 'us-east-1')
19 DYNAMODB_USERS_TABLE = os.environ.get('DYNAMODB_USERS_TABLE', 'UsersTable')
20 DYNAMODB_APPOINTMENTS_TABLE = os.environ.get('DYNAMODB_APPOINTMENTS_TABLE', 'A
21 SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')
22 ENABLE_EMAIL = os.environ.get('ENABLE_EMAIL', 'True').lower() == 'true'
23 SMTP_SERVER = os.environ.get('SMTP_SERVER', 'smtp.gmail.com')

```

```

26 SENDER_PASSWORD = os.environ.get('SENDER_PASSWORD', 'your-app-pass
27
28 # -----
29 # AWS Resources
30 # -----
31 dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION_NAME)
32 sns = boto3.client('sns', region_name=AWS_REGION_NAME)
33 user_table = dynamodb.Table(DYNAMODB_USERS_TABLE)
34 appointment_table = dynamodb.Table(DYNAMODB_APPOINTMENTS_TABLE)
35
36 # -----
37 # Logging
38 # -----
39 logging.basicConfig(level=logging.INFO)
40

```

## SNS Connection

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. Also, specify the chosen email service in `SMTP_SERVER` (e.g., Gmail, Yahoo, etc.) and enter the subscribed email in the `SENDER_EMAIL` section. Create an 'App password' for the email ID and store it in.

```

3 # -----
4 def is_logged_in():
5     return 'email' in session
6
7 def get_user(email):
8     response = user_table.get_item(Key={'email': email})
9     return response.get('Item')
10
11 def send_email(to_email, subject, body):
12     if not ENABLE_EMAIL:
13         app.logger.info(f"[Email Skipped] {subject} to {to_email}")
14         return
15     try:
16         import smtplib
17         from email.mime.text import MIMEText
18         from email.mime.multipart import MIMEMultipart
19         msg = MIMEMultipart()
20         msg['From'] = SENDER_EMAIL
21         msg['To'] = to_email
22         msg['Subject'] = subject
23         msg.attach(MIMEText(body, 'plain'))
24         server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
25
26         from email.mime.text import MIMEText
27         from email.mime.multipart import MIMEMultipart
28         msg = MIMEMultipart()
29         msg['From'] = SENDER_EMAIL
30         msg['To'] = to_email
31         msg['Subject'] = subject
32         msg.attach(MIMEText(body, 'plain'))
33         server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
34         server.starttls()
35         server.login(SENDER_EMAIL, SENDER_PASSWORD)
36         server.sendmail(SENDER_EMAIL, to_email, msg.as_string())
37         server.quit()
38         app.logger.info(f"Email sent to {to_email}")
39     except Exception as e:
40         app.logger.error(f"Email failed: {e}")

```

## Routes for Web Pages:

### Home Page

```

@app.route('/')
def index():
    if is_logged_in():
        return redirect(url_for('dashboard'))
    return render_template('index.html')

```

### Register User (Doctor/Patient)

The login route handles user input for authentication, verifying credentials against stored data in DynamoDB. On successful login, it increments the login count and redirects the user to the appropriate dashboard. This process ensures secure and efficient access to the platform



**Register Route (GET/POST):** Verifies user credentials, increments login count, and redirects to the dashboard on success.

```

# Register User (Doctor/Patient)
@app.route('/register', methods=['GET', 'POST'])
def register():
    if is_logged_in():
        return redirect(url_for('dashboard'))
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        password = request.form['password']
        confirm_password = request.form['confirmPassword']
        age = request.form['age']
        gender = request.form['gender']
        role = request.form['role']
        specialization = request.form.get('specialization', '')

        if password != confirm_password:
            flash('Passwords do not match!', 'danger')
            return render_template('register.html')

        if get_user(email):
            flash('Email already exists', 'danger')
            return render_template('register.html')

```

### Login router

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. This ensures secure access to the platform while maintaining user activity logs.

```

36 return render_template('register.html')
37 #login User
38 @app.route('/login', methods=['GET', 'POST'])
39 def login():
40     if is_logged_in():
41         return redirect(url_for('dashboard'))
42     if request.method == 'POST':
43         email = request.form['email']
44         password = request.form['password']
45         role = request.form['role']
46
47         user = get_user(email)
48         if user and check_password_hash(user['password'], password) and user['role'] == role:
49             session['email'] = email
50             session['role'] = role
51             session['name'] = user['name']
52             flash('Login successful.', 'success')
53             return redirect(url_for('dashboard'))
54         else:
55             flash('Invalid credentials.', 'danger')
56     return render_template('login.html')
57 # Logout User

```

### Logout Route:

The logout functionality allows users to securely end their session, clearing any session data and redirecting them to the login page. The dashboard provides users with an overview of their activities, such as upcoming appointments for patients or patient records for doctors, with relevant actions based on user roles.

```

# Logout User
@app.route('/logout')
def logout():
    session.clear()
    flash('You have been logged out.', 'success')
    return redirect(url_for('login'))

```

## Dashboard for both Doctors and Patients Router

```

53 # Dashboard for both Doctors and Patients
54 @app.route('/dashboard')
55 def dashboard():
56     if not is_logged_in():
57         return redirect(url_for('login'))
58     role = session['role']
59     email = session['email']
60     if role == 'doctor':
61         appointments = appointment_table.scan(
62             FilterExpression="#doctor_email = :email",
63             ExpressionAttributeNames={"#doctor_email": "doctor_email"},
64             ExpressionAttributeValues={"email": email}
65         ).get('Items', [])
66         return render_template(
67             'doctor_dashboard.html',
68             appointments=appointments,
69             doctor_name=session.get('name', ''),
70             pending_count=sum(1 for a in appointments if a.get('status') == 'p'),
71             completed_count=sum(1 for a in appointments if a.get('status') == 'c'),
72             total_count=len(appointments)
73         )
74     else:

```

## Book Appointment Router

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. This ensures smooth scheduling and timely updates for both patients and doctors.

```

# Book Appointment
@app.route('/book_appointment', methods=['GET', 'POST'])
def book_appointment():
    if not is_logged_in() or session['role'] != 'patient':
        flash('Only patients can book appointments.', 'danger')
        return redirect(url_for('login'))

    # Get list of doctors (for both GET and POST)
    try:
        response = user_table.scan(
            FilterExpression="#role = :role",
            ExpressionAttributeNames={"#role": "role"},
            ExpressionAttributeValues={"role": 'doctor'}
        )
        doctors = response.get('Items', [])
    except Exception as e:
        app.logger.error(f"Failed to fetch doctors: {e}")
        doctors = []

    if request.method == 'POST':
        doctor_email = request.form.get('doctor_email')
        doctor = get_user(doctor_email)

```

## View Appointment Route

The view appointment route allows users to access a list of their upcoming appointments. It retrieves appointment details from DynamoDB and displays them on the user's dashboard, providing information like date, time, and doctor. This ensures users can easily track their scheduled appointments.



```
# view appointment details
@app.route('/view_appointment/<appointment_id>', methods=['GET', 'POST'])
def view_appointment(appointment_id):
    if not is_logged_in():
        flash('Please log in to continue.', 'danger')
        return redirect(url_for('login'))

    try:
        # Fetch appointment by ID
        response = appointment_table.get_item(Key={'appointment_id': appointment_id})
        appointment = response.get('Item')

        if not appointment:
            flash('Appointment not found.', 'danger')
            return redirect(url_for('dashboard'))

        # Authorization check
        user_email = session['email']
        user_role = session['role']

        if user_role == 'doctor' and appointment['doctor_email'] != user_email:
            flash('You are not authorized to view this appointment.', 'danger')
```

## Search Appointment Router

The search appointment route enables users to search for specific appointments by date, doctor, or status. It queries DynamoDB to fetch relevant appointment details and displays the results in real-time, allowing users to quickly find specific appointments from their history or upcoming schedule.

```
# search appointments
@app.route('/search_appointments', methods=['GET', 'POST'])
def search_appointments():
    if not is_logged_in():
        flash('Please log in to continue.', 'danger')
        return redirect(url_for('login'))

    # Get the search term from either POST or GET
    search_term = request.form.get('search_term', '').strip() if request.method == 'POST' else request.args.get('search_term', '').strip()
    appointments = []

    try:
        if session['role'] == 'doctor':
            if search_term:
                response = appointment_table.scan(
                    FilterExpression="#doctor_email = :email AND contains(#patient_name, :search_term)",
                    ExpressionAttributeNames={
                        "#doctor_email": "doctor_email",
                        "#patient_name": "patient_name"
                    },
                    ExpressionAttributeValues={
                        ":email": session['email'],
                        ":search_term": search_term
                    }
                )
                appointments = response.get('Items', [])
```

## Profile Router

The profile route allows users to view and update their personal information, such as name, contact details, and medical history. It retrieves user data from DynamoDB and displays it on the dashboard. Users can also make changes to their profile, which are then saved securely in the database.

```
#profile page
@app.route('/profile', methods=['GET', 'POST'])
def profile():
    if not is_logged_in():
        flash('Please log in to continue.', 'danger')
        return redirect(url_for('login'))

    email = session['email']
    try:
        user = user_table.get_item(Key={'email': email}).get('Item', {})
        if not user:
            flash('User not found.', 'danger')
            return redirect(url_for('dashboard'))

        if request.method == 'POST':
            name = request.form.get('name')
            age = request.form.get('age')
            gender = request.form.get('gender')

            update_expression = "SET #name = :name, age = :age, gender = :gender"
            expression_values = {
                ":name": name,
                ":age": age,
                ":gender": gender
            }
```

## Deployment Code

The health routing feature in the MedTrack project checks the system's status by sending a request to a specific endpoint, ensuring the backend services are functioning properly. The `__name__ == '__main__'` block is used in the Flask app to ensure that the application runs only if the script is executed directly, not when imported as a module, enabling local development or deployment on a server. This setup ensures that the app runs smoothly and is self-contained during execution.

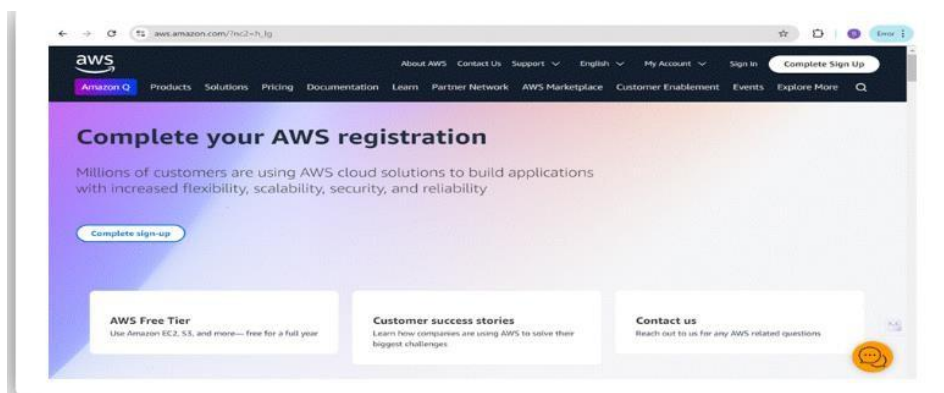
```

5         flash('Diagnosis submitted successfully.', 'success')
6         return redirect(url_for('dashboard'))
7
8
9     except Exception as e:
10        app.logger.error(f"Submit diagnosis error: {e}")
11        flash('An error occurred while submitting the diagnosis. Please try ag
12        return redirect(url_for('view_appointment', appointment_id=appointment
13
14    # -----
15    # Entrypoint for WSGI (production servers)
16    # -----
17    application = app # For gunicorn, uWSGI, etc.
18
19    # For local development
20    if __name__ == '__main__':
21        port = int(os.environ.get('PORT', 5000))
22        debug_mode = os.environ.get('FLASK_DEBUG', 'False').lower() == 'true'
23        app.run(host='0.0.0.0', port=port, debug=debug_mode)

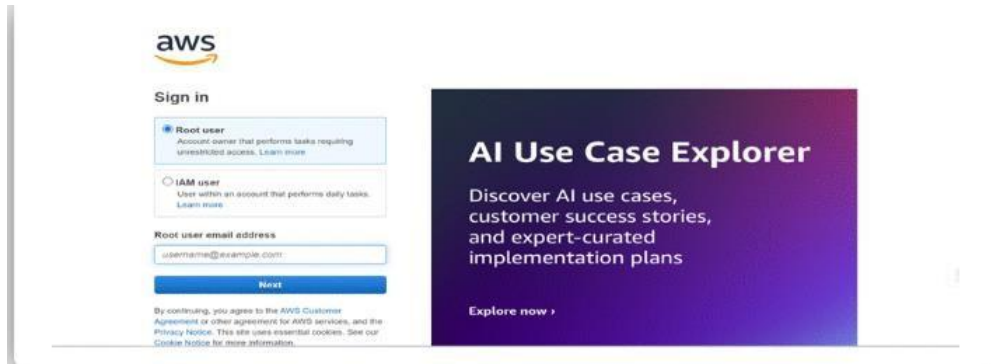
```

## Milestone 2. AWS Account Setup and Login

- Go to the AWS website (<https://aws.amazon.com/>).
- Click on the "Create an AWS Account" button.
- Follow the prompts to enter your email address and choose a password.
- Provide the required account information, including your name, address, and phone number.
- Enter your payment information. (Note: While AWS offers a free tier, a credit card or debit card is required for verification.)
- Complete the identity verification process.
- Choose a support plan (the basic plan is free and sufficient for starting).
- Once verified, you can sign in to your new AWS accounts.



- Log in to the AWS Management Console
- After setting up your account, log in to the [AWS Management Console](#).

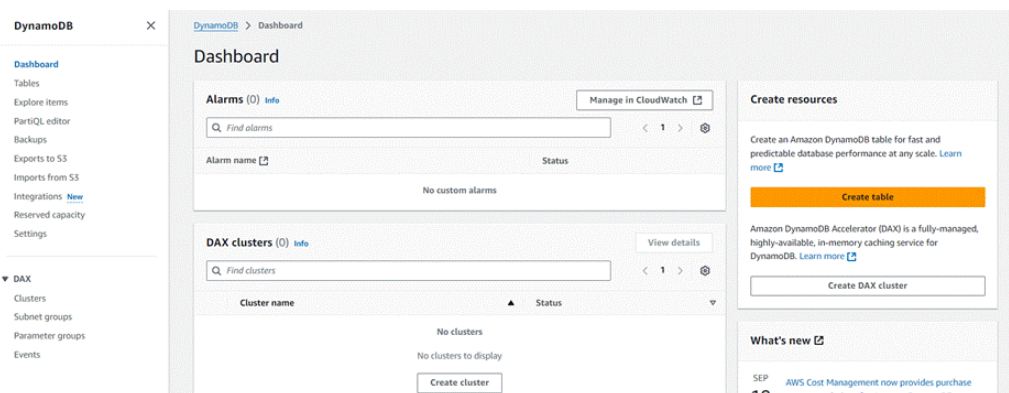
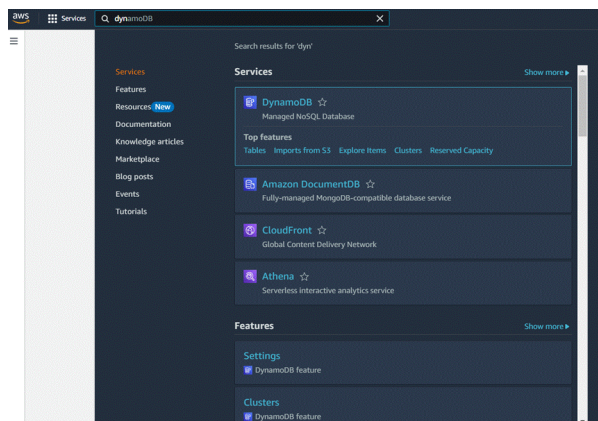


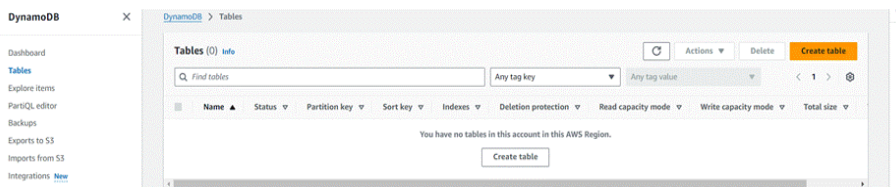
## Milestone 3: DynamoDB Database Creation and Setup

Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high-performance data storage for seamless backend operations.

### Navigate to the DynamoDB

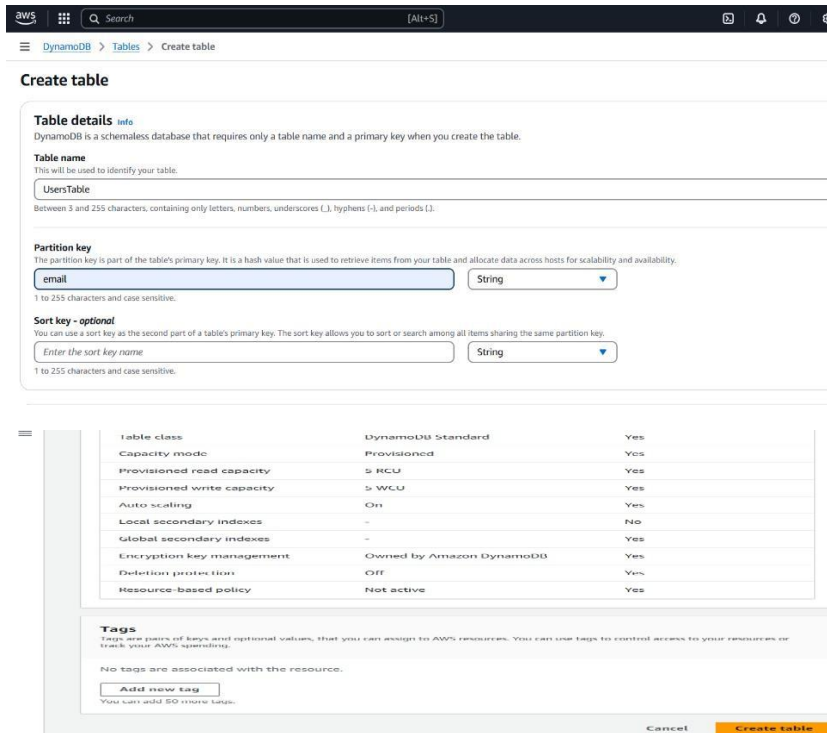
- In the AWS Console, navigate to DynamoDB and click on create tables.





## Create a DynamoDB table for storing data

- Create Users table with partition key “Email” with type String and click on create tables.



**Create table**

**Table details** [info](#)  
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
This will be used to identify your table.  
  
Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).

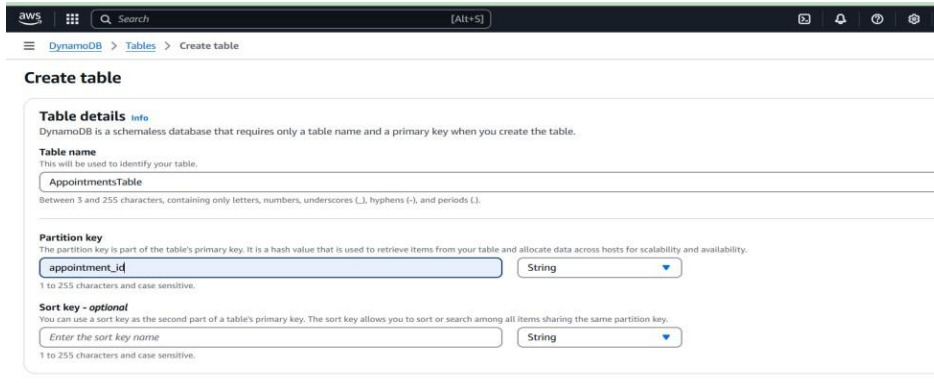
**Partition key**  
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.  
   
1 to 255 characters and case sensitive.

**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.  
   
1 to 255 characters and case sensitive.

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.  
  
You can add 50 more tags.

- Create Appointments Table with partition key “appointment\_id” with type String and click on create tables.



**Create table**

**Table details** [info](#)  
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
This will be used to identify your table.  
  
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.  
   
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.  
   
1 to 255 characters and case sensitive.

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](#)

Tables (2/10) Info Actions Delete Create table

Q Status

Any tag key

Any tag value

< 1 >

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection
<input checked="" type="checkbox"/>	<a href="#">AppointmentsTable</a>	Active	appointment_id (S)	-	0 0		Off
<input type="checkbox"/>	<a href="#">NextGenHospital_Appointments</a>	Active	appointment_id (S)	-	0 0		Off
<input type="checkbox"/>	<a href="#">NextGenHospital_ContactMessages</a>	Active	message_id (S)	-	0 0		Off
<input type="checkbox"/>	<a href="#">NextGenHospital_Doctors</a>	Active	doctor_id (S)	-	0 0		Off
<input type="checkbox"/>	<a href="#">NextGenHospital_PatientRecords</a>	Active	patient_id (S)	-	0 0		Off
<input type="checkbox"/>	<a href="#">NextGenHospital_Users</a>	Active	email (S)	-	0 0		Off
<input type="checkbox"/>	<a href="#">SalonAppointments</a>	Active	appointment_id (S)	user_email (S)	0 0		Off
<input type="checkbox"/>	<a href="#">SalonStylists</a>	Active	stylist_id (S)	-	0 0		Off
<input type="checkbox"/>	<a href="#">SalonUsers</a>	Active	email (S)	-	0 0		Off
<input checked="" type="checkbox"/>	<a href="#">UsersTable</a>	Active	email (S)	-	0 0		Off

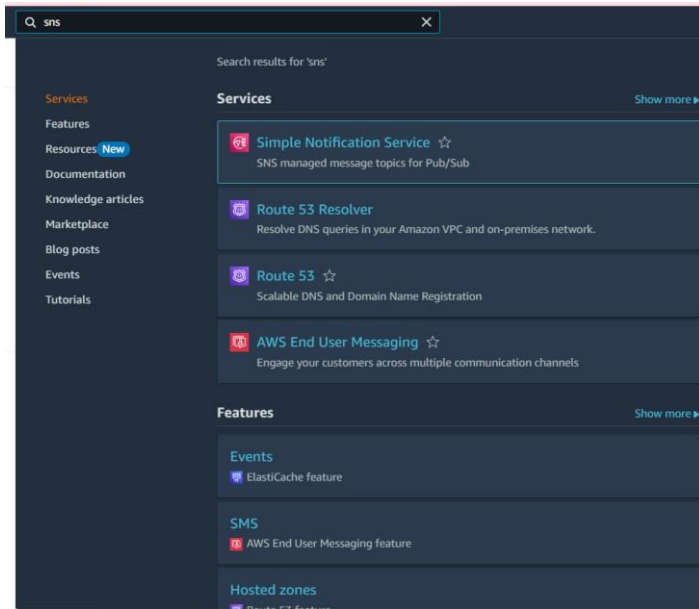
## Milestone 4 : SNS Notification Setup

Amazon SNS is a fully managed messaging service that enables real-time notifications through channels like SMS, email, or app endpoints. You create topics, configure subscriptions, and integrate SNS into your app to send notifications based on specific events.

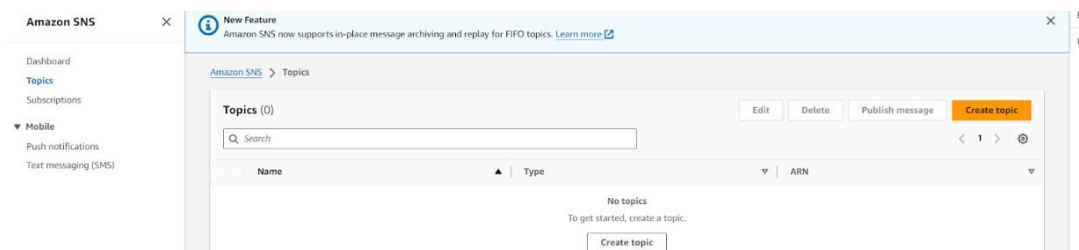
### SNS topics for email notifications

- In the AWS Console, search for SNS and navigate to the SNS Dashboard.





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



- Choose **Standard** type for general notification use cases and Click on **Create Topic**.

Amazon SNS > Topics > Create topic

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

## Create topic

### Details

#### Type [info](#)

Topic type cannot be modified after topic is created.

##### ☐ FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- Subscription protocols: SQS

##### ☒ Standard

- Best-effort message ordering
- At-least-once message delivery
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

#### Name

Medtrack

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (\_).

#### Display name - optional [info](#)

To use this topic with SNS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

My Topic

Maximum 100 characters.

#### ► 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](#)

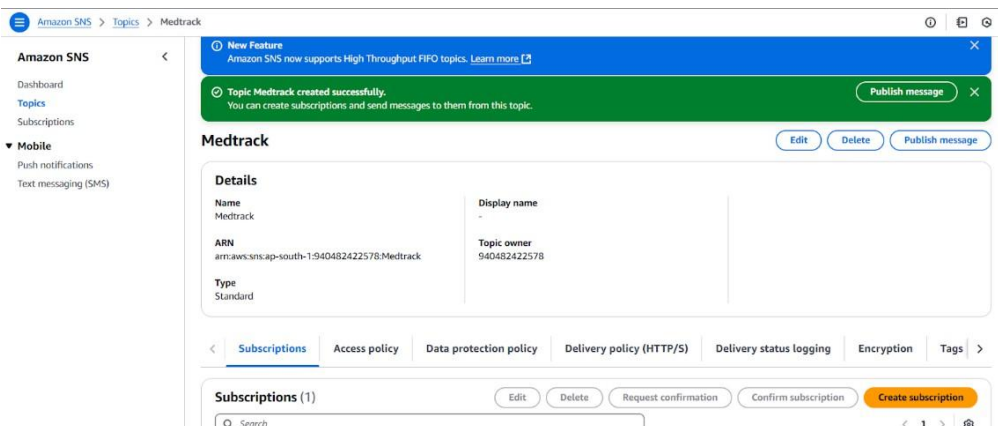
#### ► Active tracing - optional [info](#)

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**.



The screenshot shows the Amazon SNS console interface. On the left, there's a navigation menu with 'Amazon SNS', 'Dashboard', 'Topics', 'Subscriptions', and 'Mobile'. The main content area shows the 'Medtrack' topic details. At the top, there's a blue banner for a new feature and a green banner indicating the topic was created successfully. Below these, the 'Medtrack' topic details are shown, including its name, display name, ARN, topic owner, and type (Standard). A tabbed interface at the bottom allows switching between 'Subscriptions', 'Access policy', 'Data protection policy', 'Delivery policy (HTTP/S)', 'Delivery status logging', 'Encryption', and 'Tags'. The 'Subscriptions' tab is active, showing a list of subscriptions with a search bar and buttons for 'Edit', 'Delete', 'Request confirmation', 'Confirm subscription', and 'Create subscription'.

## Subscribe users and Admin

- Subscribe users (or admin staff) to this topic via email. When a book request is made, notifications will be sent to the subscribed emails.

Amazon SNS > Subscriptions > Create subscription

Topic ARN  
arn:aws:sns:ap-south-1:540482422578:Medtrack

Protocol  
Email

Endpoint  
sairaviteja478@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](#)  
Send undeliverable messages to a dead-letter queue.

[Cancel](#) [Create subscription](#)

- After subscription request for the mail confirmation

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

Subscriptions (1) [Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#) [Create subscription](#)

ID	Endpoint	Status	Protocol
<a href="#">2c78944f-bb5d-4fb1-9982-74334...</a>	sairaviteja478@gmail.com	Confirmed	EMAIL

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

## AWS Notification - Subscription Confirmation Inbox x

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

You have chosen to subscribe to the topic:

**arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications**

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](#)

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

You have chosen to subscribe to the topic:

**arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications**

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:ap-south-1:557690616836:BookRequestNotifications:d78e0371-9235-404d-952c-85c2743607c4

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.

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

Medtrack

Edit
Delete
Publish message

Details

Name  
Medtrack

ARN  
arn:aws:sns:ap-south-1:940482422578:Medtrack

Type  
Standard

Display name  
-

Topic owner  
940482422578

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

Subscriptions (1)

Edit
Delete
Request confirmation
Confirm subscription
Create subscription

Search

ID	Endpoint	Status	Protocol
2c78944f-bb5d-4fb1-9982-74334...	sairaviteja478@gmail.com	Confirmed	EMAIL

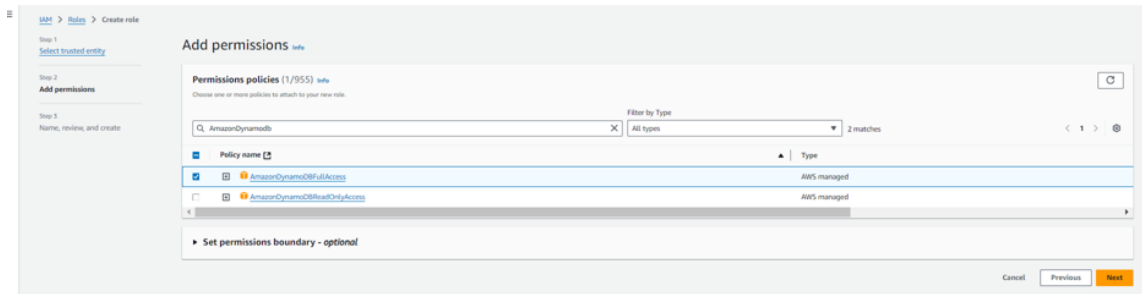
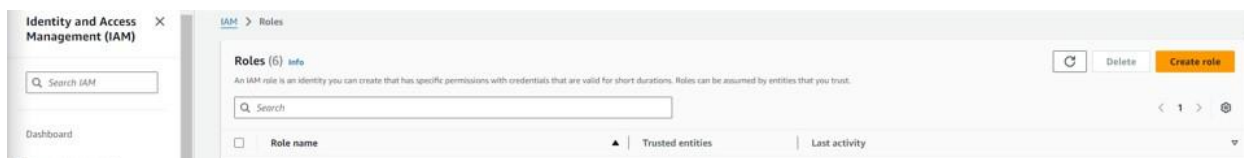
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

## Milestone 5: IAM Role Setup

IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows controlled access and ensures security best practices in managing AWS resources.

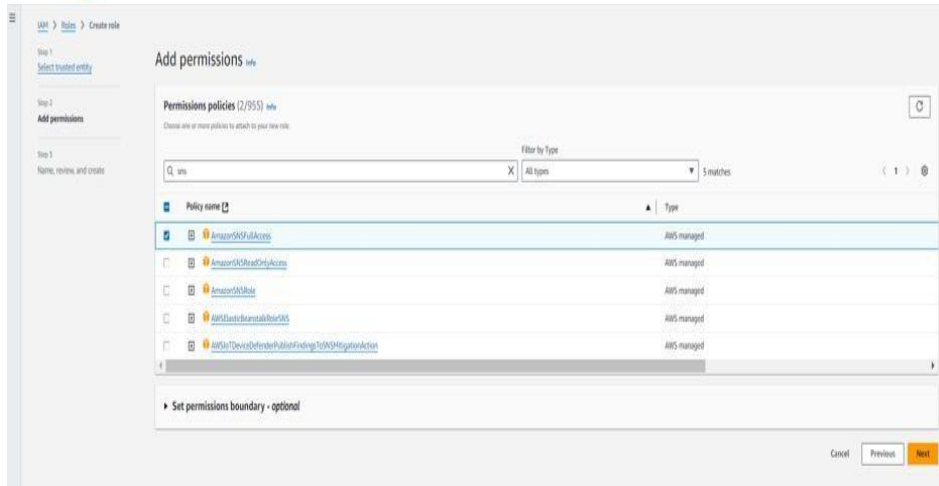
### Create IAM Role.

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

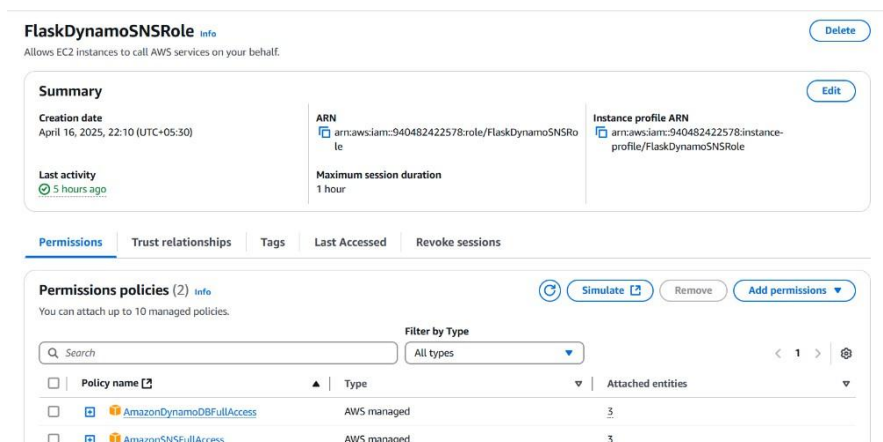


- **AmazonDynamoDBFullAccess:** Allows EC2 to perform read/write operations on DynamoDB.
- **AmazonSNSFullAccess:** Grants EC2 the ability to send notifications via SNS.





To create a role named `flaskdynamodbsns`, go to the AWS IAM console, create a new role, and assign `DynamoDBFullAccess` and `SNSFullAccess` policies. Name the role `flaskdynamodbsns` and attach it to the necessary AWS services. This role will allow your Flask app to interact with both DynamoDB and SNS seamlessly.

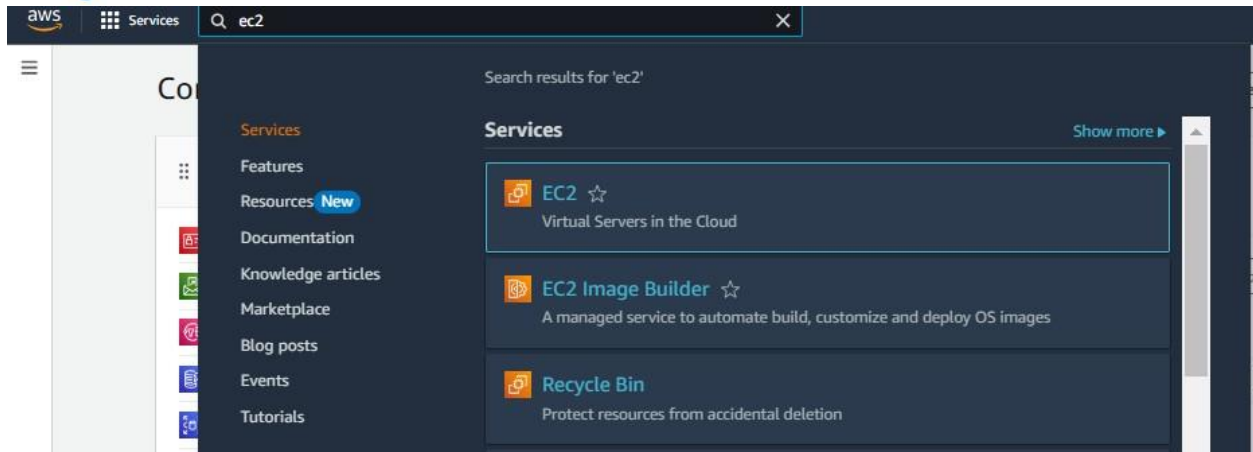


## Milestone 6: EC2 Instance setup

To set up a public EC2 instance, choose an appropriate Amazon Machine Image (AMI) and instance type. Ensure the security group allows inbound traffic on necessary ports (e.g., HTTP/HTTPS for web applications). After launching the instance, associate it with an Elastic IP for consistent public access, and configure your application or services to be publicly accessible.

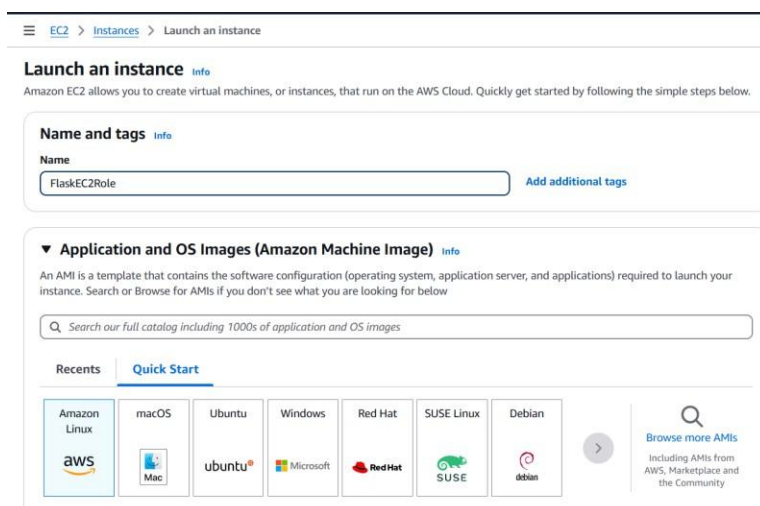
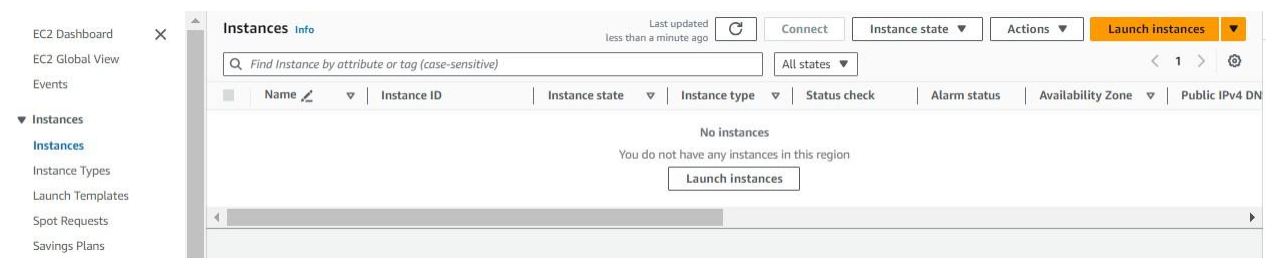
### Launch an EC2 instance to host the Flask application.

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



To launch an EC2 instance with the name **flaskec2role**, follow these steps:

1. Go to the **AWS EC2 Dashboard** and click on **Launch Instance**.
2. Select your desired AMI, instance type, configure instance details, and under **IAM role**, choose the role **flaskec2role**. Finally, launch the instance.



To launch an EC2 instance with **Amazon Linux 2** or **Ubuntu** as the AMI and **t2.micro** as the instance type (free-tier eligible):

1. In the **Launch Instance** wizard, choose **Amazon Linux 2** or **Ubuntu** from the available AMIs.

2. Select **t2.micro** as the instance type, which is free-tier eligible, and continue with the configuration and launch steps.

Amazon Linux  
aws

macOS  
Mac

Ubuntu  
ubuntu

Windows  
Microsoft

Red Hat  
Red Hat

[Browse more AMIs](#)  
 Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI  
 ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)  
 Virtualization: hvm ENA enabled: true Root device type: ebs

Free tier eligible

Description

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture	Boot mode	AMI ID	
64-bit (x86)	uefi-preferred	ami-02b49a24cfb95941c	Verified provider

To create and download the key pair for server access:

1. In the **Launch Instance** wizard, under the **Key Pair** section, click **Create a new key pair**.
2. Name your key pair (e.g., **flaskkeypair**) and click **Download Key Pair**. This will download the .pem file to your system, which you will use to access the EC2 instance securely via SSH.

▼ Instance type Info | Get advice

Instance type  
 t2.micro  
 Family: t2 1 vCPU 1 GiB Memory Current generation: true  
 On-Demand Linux base pricing: 0.0124 USD per Hour  
 On-Demand Windows base pricing: 0.017 USD per Hour  
 On-Demand RHEL base pricing: 0.0268 USD per Hour  
 On-Demand SUSE base pricing: 0.0124 USD per Hour

Free tier eligible

☐ All generations  
[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

▼ 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

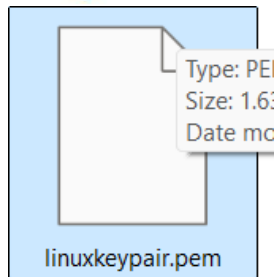
▼ Key pair (login) Info

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  

linuxkeypair

Create new key pair



## 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.

**Network settings** [Info](#)

VPC - required [Info](#)

vpc-03cdc7b6f19dd7211 (default) [Info](#)

Subnet [Info](#)

No preference [Info](#) [Create new subnet](#)

Auto-assign public IP [Info](#)

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group ☐ Select existing security group

Security group name - required

launch-wizard

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and .-:/#@![]+=&:()!\$\*

Description - required [Info](#)

launch-wizard created 2024-10-13T17:49:56.622Z

**Inbound Security Group Rules**

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0) [Remove](#)

Type [Info](#) ssh Protocol [Info](#) TCP Port range [Info](#) 22

Source type [Info](#) Anywhere Source [Info](#) 0.0.0.0/0 Description - optional [Info](#) e.g. SSH for admin desktop

▼ Security group rule 2 (TCP, 80, 0.0.0.0/0) [Remove](#)

Type [Info](#) HTTP Protocol [Info](#) TCP Port range [Info](#) 80

Source type [Info](#) Custom Source [Info](#) 0.0.0.0/0 Description - optional [Info](#) e.g. SSH for admin desktop

▼ Security group rule 3 (TCP, 5000, 0.0.0.0/0) [Remove](#)

Type [Info](#) Custom TCP Protocol [Info](#) TCP Port range [Info](#) 5000

Source type [Info](#) Custom Source [Info](#) 0.0.0.0/0 Description - optional [Info](#) e.g. SSH for admin desktop

[Add security group rule](#)

EC2 > Launch an instance

**Success**  
Successfully initiated launch of instance i-021861037f9a2700

Launch log

**Next Steps**

What would you like to do next with this instance, for example "create alarm" or "create backup"?

**Create billing and free tier usage alerts**

To manage costs and avoid surprise bills, set up email notifications for billing and free tier usage thresholds.

Create billing alerts

**Connect to your instance**

Once your instance is running, log into it from your local computer.

Connect to instance

Learn more

**Connect an RDS database**

Configure the connection between an EC2 instance and a database to allow traffic flow between them.

Connect an RDS database

Create a new RDS database

Learn more

**Create EBS snapshot policy**

Create a policy that automates the creation, retention, and deletion of EBS snapshots.

Create EBS snapshot policy

**Manage detailed monitoring**

Enable or disable detailed monitoring for the instance. If you enable detailed monitoring, the Amazon EC2 console displays monitoring graphs with a 5-minute period.

Manage detailed monitoring

**Create Load Balancer**

Create an application, network gateway, or classic Elastic Load Balancer.

Create Load Balancer

**Create AWS budget**

AWS Budgets allow you to create budgets, forecast spend, and take action on your costs and usage from a single location.

Create AWS budget

**Manage CloudWatch alarms**

Create or update Amazon CloudWatch alarms for the instance.

Manage CloudWatch alarms

**Disaster recovery for your instances**

Recover the instance you just launched into a different Availability Zone or a different Region using AWS Elastic Disaster Recovery (EDR).

Disaster recovery for your instances

**Monitor for suspicious runtime activities**

Amazon GuardDuty enables you to continuously monitor for malicious runtime activity and anomalous behavior, with near real-time visibility into on-host activities occurring across your Amazon EC2 workloads.

Monitor for suspicious runtime activities

**Get instance screenshot**

Capture a screenshot from the instance and view it as an image. This is useful for troubleshooting an unresponsive instance.

Get instance screenshot

**Get system log**

View the instance's system log to troubleshoot issues.

Get system log

[View all instances](#)

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.

Successfully initiated starting of i-0caa76d6126ba3c8f

**Instances (1)** Info Last updated less than a minute ago

Find Instance by attribute or tag (case-sensitive)

Running

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
<input type="checkbox"/>	FlaskEC2Role	i-0caa76d6126ba3c8f	Running	t2.micro	Initializing	View alarms	ap-south-1b	ec2-13-21

The EC2 instance you are launching is configured with Amazon Linux 2 or Ubuntu as the AMI, t2.micro as the instance type (free-tier eligible), and flaskec2role IAM role for appropriate permissions. The flaskkeypair key pair is created for secure server access via SSH, and the instance is set to auto-assign a public IP for internet accessibility. The security group is configured to allow SSH (port 22) access for remote login.

**Instance summary for i-0caa76d6126ba3c8f (FlaskEC2Role)**

Updated less than a minute ago

**Instance ID**  
i-0caa76d6126ba3c8f

**IP v6 address**  
-

**Hostname type**  
IP name: ip-172-31-14-197-ap-south-1-compute.internal

**Answer private resource DNS name**  
ipv4 (A)

**Auto-assigned IP address**  
-

**IAM Role**  
FlaskDynamoS3Role

**IMDSv2**  
Required

**Operator**  
-

**Public IPv4 address**  
-

**Instance state**  
Stopped

**Private IP DNS name (IPv4 only)**  
ip-172-31-14-197-ap-south-1-compute.internal

**Instance type**  
t2.micro

**VPC ID**  
vpc-086a8db7a0c178b58

**Subnet ID**  
subnet-0c7267d27d4663be9

**Instance ARN**  
arn:aws:ec2:ap-south-1:940482422578:instance/i-0caa76d6126ba3c8f

**Private IPv4 addresses**  
172.31.14.197

**Public IPv4 DNS**  
-

**Elastic IP addresses**  
-

**AWS Compute Optimizer finding**  
Opt-in to AWS Compute Optimizer for recommendation

**Auto Scaling Group name**  
-

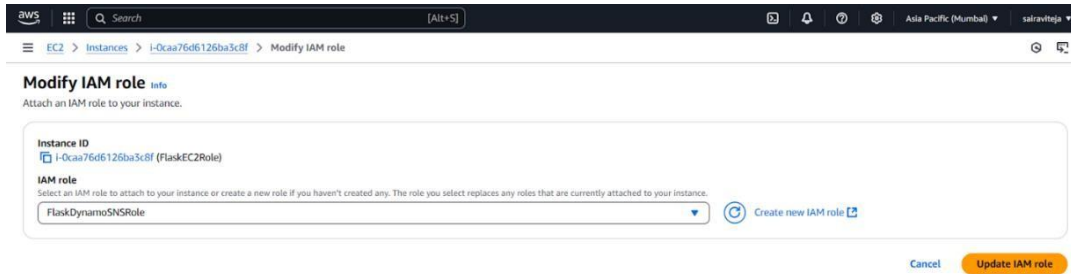
**Managed**  
false

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

- Go to the **AWS IAM Console**, select **Roles**, and find the **flaskec2role**.
- 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.



**Modify IAM role** Info

Attach an IAM role to your instance.

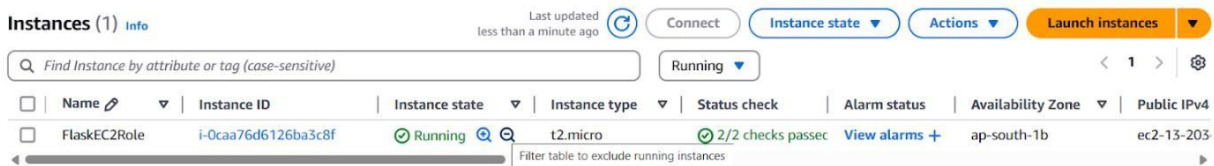
Instance ID  
 i-Ocaa76d6126ba3c8f (FlaskEC2Role)

IAM role  
 Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.  
 FlaskDynamoSNSRole [Create new IAM role](#)

[Cancel](#) [Update IAM role](#)

To connect to your EC2 instance:

- Go to the **EC2 Dashboard**, select your running instance, and click **Connect**.
- Follow the instructions provided in the **Connect To Your Instance** dialog, which will show the SSH command (e.g., `ssh -i flaskkeypair.pem ec2-user@<public-ip>`) to access your instance using the downloaded .pem key.

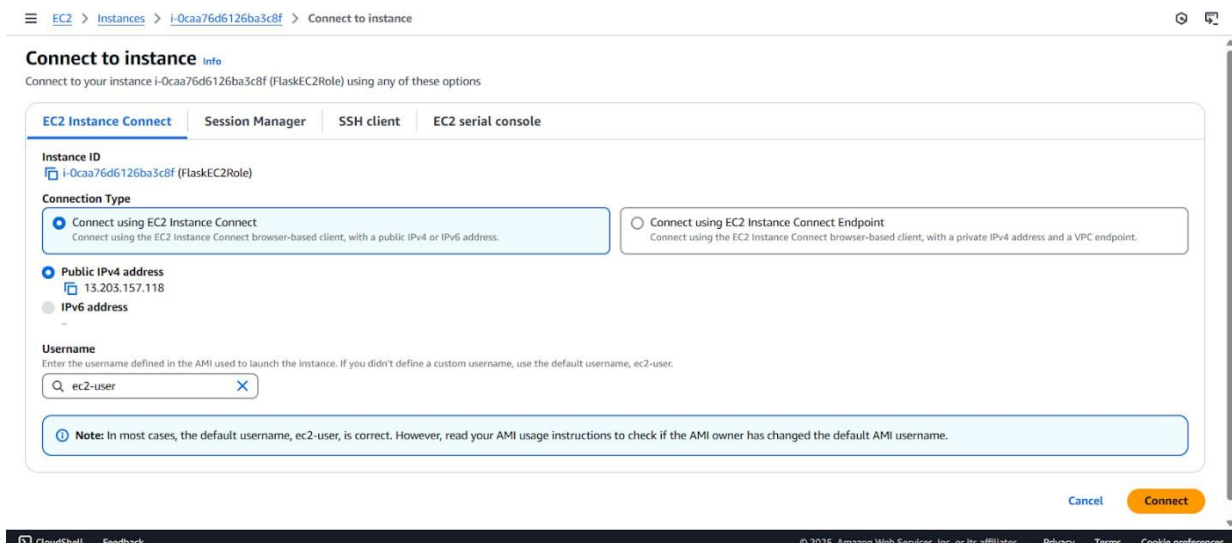


**Instances (1)** Info Last updated less than a minute ago [Connect](#) [Instance state](#) [Actions](#) [Launch instances](#)

[Running](#) < 1 > [Filter table to exclude running instances](#)

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
<input type="checkbox"/>	FlaskEC2Role	i-Ocaa76d6126ba3c8f	Running	t2.micro	2/2 checks passed	<a href="#">View alarms</a>	ap-south-1b	ec2-13-203-

- Now connect the EC2 with the files



**Connect to instance** Info

Connect to your instance i-Ocaa76d6126ba3c8f (FlaskEC2Role) using any of these options

[EC2 Instance Connect](#) [Session Manager](#) [SSH client](#) [EC2 serial console](#)

Instance ID  
 i-Ocaa76d6126ba3c8f (FlaskEC2Role)

Connection Type

☒ **Connect using EC2 Instance Connect**  
 Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.

☐ **Connect using EC2 Instance Connect Endpoint**  
 Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

☒ **Public IPv4 address**  
 13.203.157.118

☐ **IPv6 address**

Username  
 Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

**Note:** In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

[Cancel](#) [Connect](#)

```
aws [Alt+S] Search
```

```
A newer release of "Amazon Linux" is available.  
Version 2023.7.20250414:  
Run "/usr/bin/dnf check-release-update" for full release and version update info  
  
#  
#####  
#####\n#####  
\\#\nV-.-> https://aws.amazon.com/linux/amazon-linux-2023  
m/\nLast login: Fri Apr 18 14:42:17 2025 from 13.233.177.3  
[ec2-user@ip-172-31-14-197 ~]$
```

i-Ocaa76d6126ba3c8f (FlaskEC2Role)

PublicIPs: 13.203.157.118 PrivateIPs: 172.31.14.197

## 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.

## 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

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

Run:

[https://github.com/Pavan-12393/MedTrack\\_app.git](https://github.com/Pavan-12393/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
```

```
aws | Search | [Alt+S] | Asia Pacific (Mumbai) | sairaviteja |
2025-04-18 08:02:36,630 - _main_ - ERROR - Failed to fetch appointments: An error occurred (ValidationException) when calling the Query operation: The table does not have the specified index: DoctorEmailIndex
2025-04-18 08:02:36,637 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:02:36] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:02:39,258 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:02:39] "GET /logout HTTP/1.1" 302 -
2025-04-18 08:02:39,289 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:02:39] "GET / HTTP/1.1" 200 -
2025-04-18 08:02:41,589 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:02:41] "GET /login HTTP/1.1" 200 -
2025-04-18 08:02:47,468 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:02:47] "POST /login HTTP/1.1" 302 -
2025-04-18 08:02:47,500 - _main_ - ERROR - Failed to query appointments: An error occurred (ValidationException) when calling the Query operation: The table does not have the specified index: PatientEmailIndex
2025-04-18 08:02:47,517 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:02:47] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:02:49,506 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:02:49] "GET /book_appointment HTTP/1.1" 200 -
2025-04-18 08:03:25,936 - _main_ - INFO - Email sent to gtharunasri19@gmail.com
2025-04-18 08:03:28,795 - _main_ - INFO - Email sent to sairaviteja478@gmail.com
2025-04-18 08:03:28,796 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:03:28] "POST /book_appointment HTTP/1.1" 302 -
2025-04-18 08:03:28,852 - _main_ - ERROR - Failed to query appointments: An error occurred (ValidationException) when calling the Query operation: The table does not have the specified index: PatientEmailIndex
2025-04-18 08:03:28,868 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:03:28] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:03:44,609 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:03:44] "GET /logout HTTP/1.1" 302 -
2025-04-18 08:03:44,654 - werkzeug - INFO - 110.235.236.48 -- [18/Apr/2025 08:03:44] "GET / HTTP/1.1" 200 -
C:\Users\ [ec2-user@ip-172-31-14-197 medtrack_app]$ python app.py
* Serving Flask app 'app'
* Debug mode: off
2025-04-18 08:04:27,745 - werkzeug - INFO - 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://172.31.14.197:5000
2025-04-18 08:04:27,746 - werkzeug - INFO - Press CTRL+C to quit
```

i-Ocaa76d6126ba3c8f (FlaskEC2Role)

PublicIPs: 13.201.72.132 PrivateIPs: 172.31.14.197

## Verify the Flask app is running:

<http://your-ec2-public-ip>

- Run the Flask app on the EC2 instance

```
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
* Debug mode: off
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://127.0.0.1:80
* Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```

## Access the website through:

PublicIPs: <https://13.201.72.132:5000/>

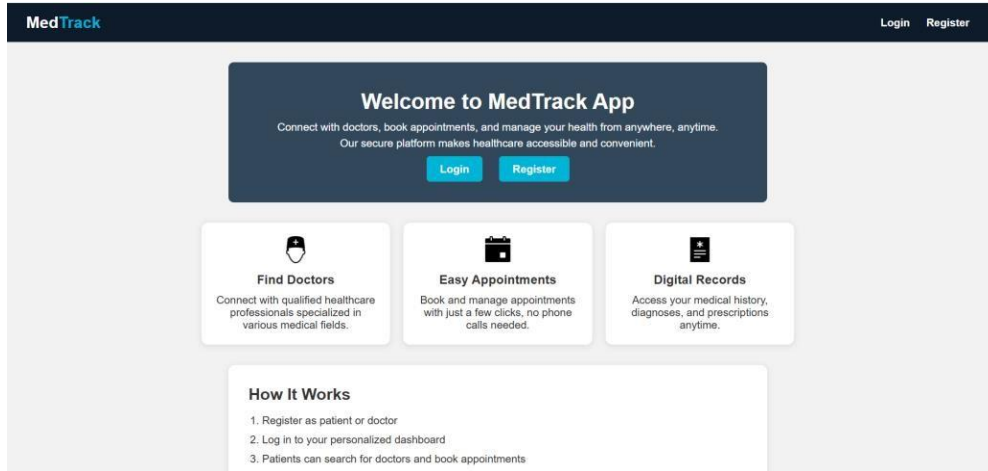
# Milestone 8: Testing and Deployment

## 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.

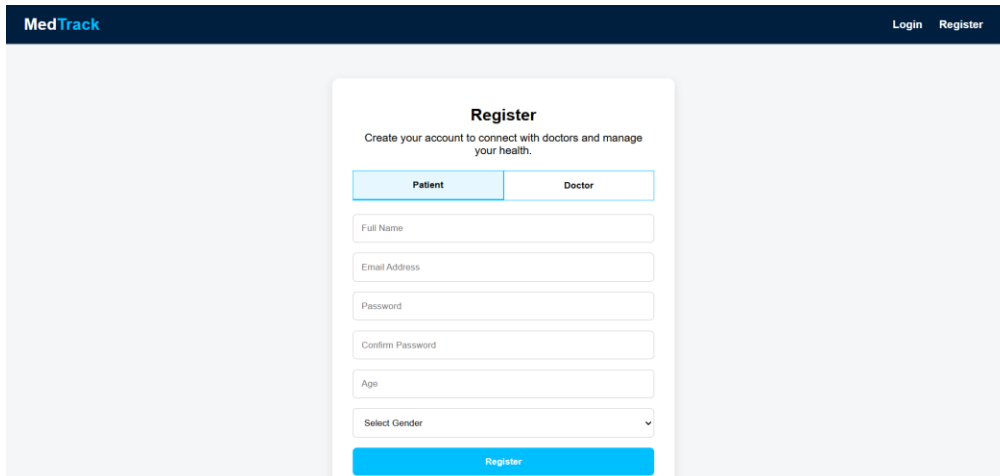
The Home Page serves as the initial interface that directs users to the key functionalities of your web application.



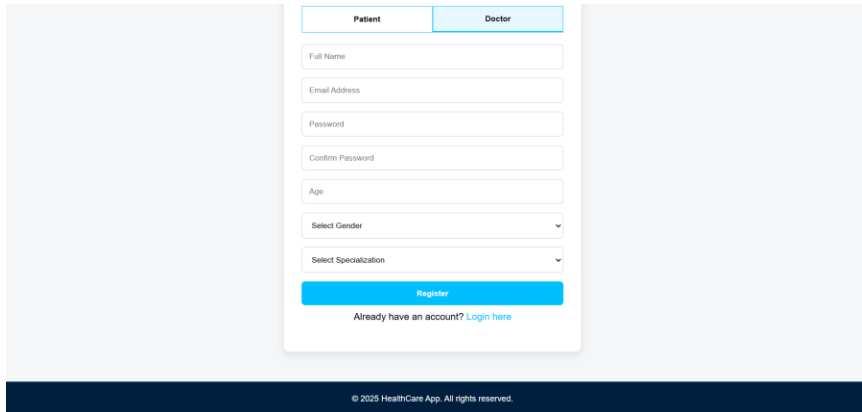
### Doctor and Patient register page:

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



The screenshot shows the MedTrack App register page. At the top, there's a dark blue header with the MedTrack logo on the left and 'Login Register' links on the right. Below the header is a white box with the title 'Register' and a subtext 'Create your account to connect with doctors and manage your health.' Below this are two tabs: 'Patient' and 'Doctor'. Under the 'Patient' tab, there are several input fields: 'Full Name', 'Email Address', 'Password', 'Confirm Password', 'Age', and a dropdown menu for 'Select Gender'. At the bottom of the form is a blue button labeled 'Register'.



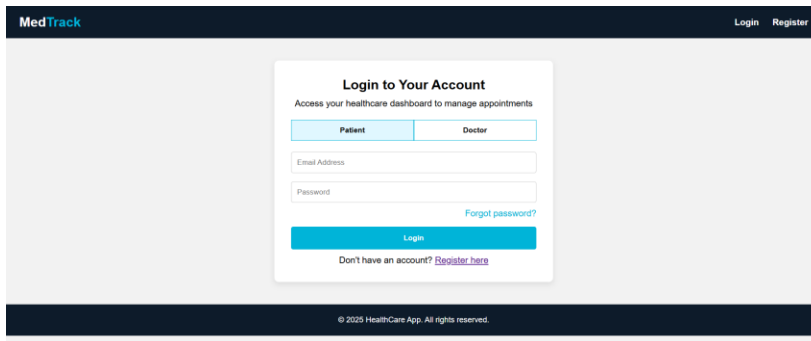
The registration form is titled 'Patient' and 'Doctor'. It includes fields for Full Name, Email Address, Password, Confirm Password, Age, Select Gender, and Select Specialization. A blue 'Register' button is at the bottom, followed by a link: 'Already have an account? [Login here](#)'. The footer contains the text: '© 2025 HealthCare App. All rights reserved.'

## Doctor and Patient login page:

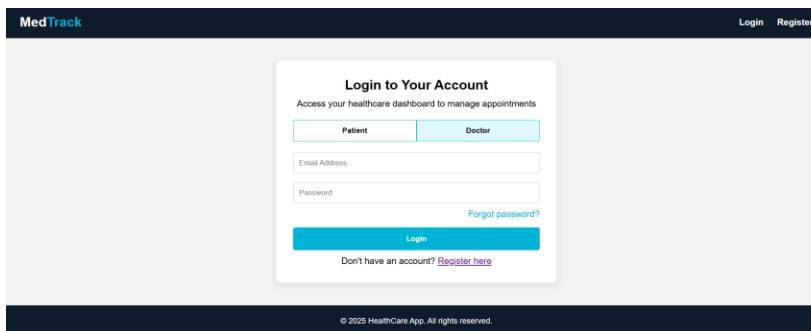
The Patient and Doctor Login Pages allow users to securely access their accounts on the platform. Each login page typically includes:

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.



The login form is titled 'Login to Your Account' and 'Access your healthcare dashboard to manage appointments'. It includes fields for Email Address and Password. A blue 'Login' button is at the bottom, followed by a link: 'Don't have an account? [Register here](#)'. The footer contains the text: '© 2025 HealthCare App. All rights reserved.'



The login form is titled 'Login to Your Account' and 'Access your healthcare dashboard to manage appointments'. It includes fields for Email Address and Password. A blue 'Login' button is at the bottom, followed by a link: 'Don't have an account? [Register here](#)'. The footer contains the text: '© 2025 HealthCare App. All rights reserved.'

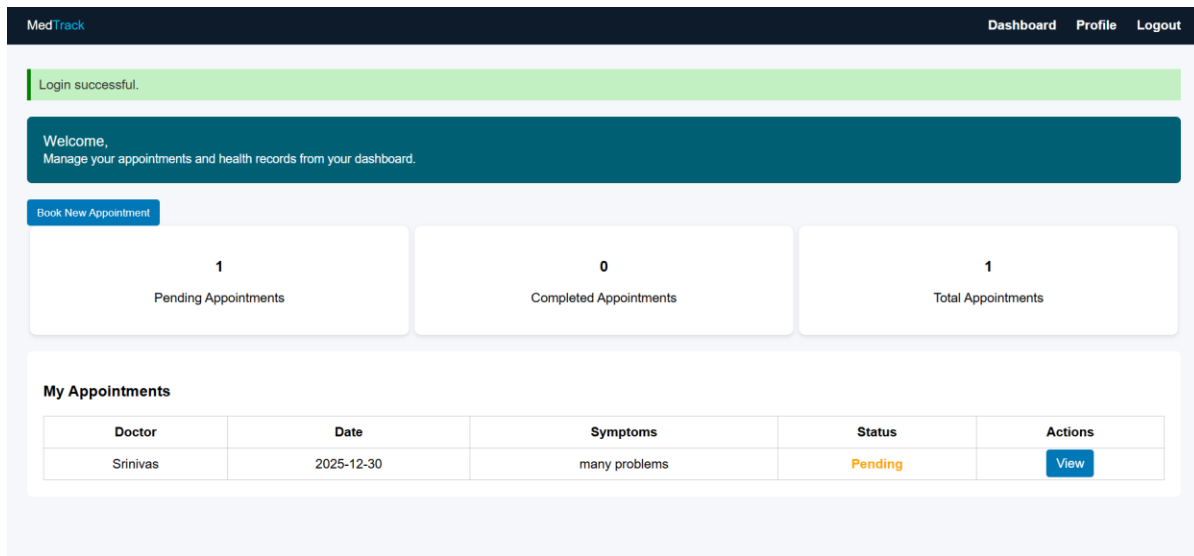


## User Dashboard:

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

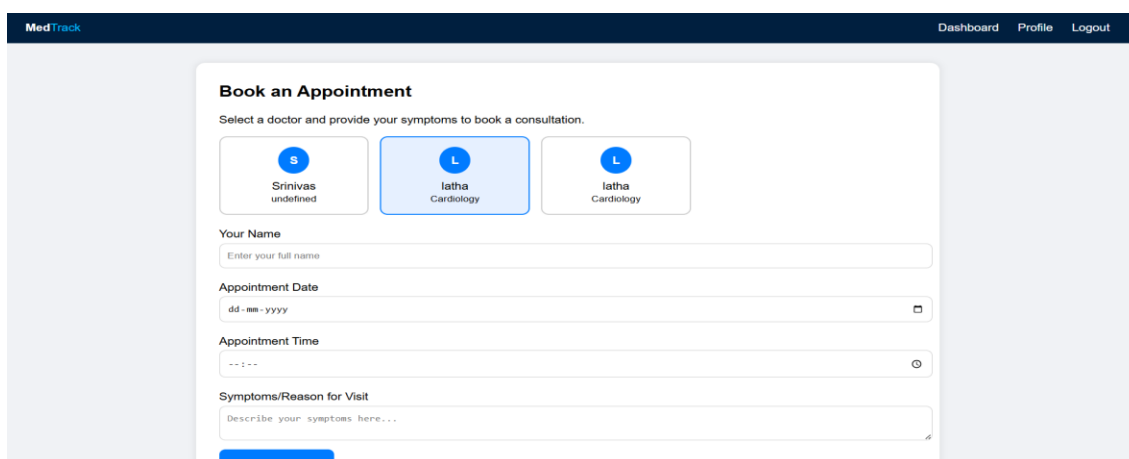
1. **Book Appointment Section:** A form for selecting a doctor, choosing an appointment time, and submitting the request.
2. **Appointment Status:** A section showing the current status of appointments (e.g., confirmed, pending, or completed) with options to view details or cancel.
3. **Upcoming Appointments:** A list of future appointments with relevant details such as doctor name, date, and time.

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



The screenshot shows the MedTrack user dashboard. At the top, there's a navigation bar with 'MedTrack' on the left and 'Dashboard', 'Profile', and 'Logout' on the right. Below the navigation bar, a green banner indicates 'Login successful.' followed by a teal welcome message: 'Welcome, Manage your appointments and health records from your dashboard.' A 'Book New Appointment' button is visible. The dashboard features three summary cards: '1 Pending Appointments', '0 Completed Appointments', and '1 Total Appointments'. Below these is a 'My Appointments' section with a table showing one appointment for 'Srinivas' on '2025-12-30' with 'many problems' and a 'Pending' status. A 'View' button is next to the appointment entry.

Doctor	Date	Symptoms	Status	Actions
Srinivas	2025-12-30	many problems	Pending	<a href="#">View</a>



The screenshot shows the 'Book an Appointment' form. It starts with the title 'Book an Appointment' and the instruction 'Select a doctor and provide your symptoms to book a consultation.' There are three doctor selection cards: 'Srinivas undefined', 'Iatha Cardiology' (selected), and 'Iatha Cardiology'. Below the doctor selection, there are input fields for 'Your Name', 'Appointment Date' (with a calendar icon), 'Appointment Time' (with a clock icon), and 'Symptoms/Reason for Visit' (with a text area). A 'Book Appointment' button is at the bottom.

MedTrack

Dashboard Profile Logout

**Appointment Details**  
Doctor: Srinivas  
Date: 2025-12-30  
Time: 10:00  
Symptoms: many problems  
Status: Pending

← Back to Dashboard

### Doctor Dashboard :

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

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

MedTrack Dashboard Profile Logout

Login successful.

Welcome, Dr. Srinivas  
Manage your appointments and patient consultations from your dashboard.

Search patient name... Search

0  
Pending Appointments

1  
Completed Appointments

1  
Total Appointments

Pending Appointments Completed Appointments All Appointments

**Appointments**

Patient Name	Date	Symptoms	Status	Actions
sruthi	2025-10-10	skin issue	Completed	<a href="#">View</a>

MedTrack

### Appointment Details

**Patient Information**

Name: sruthi  
Status: Completed  
Date: 2025-10-10  
Created: N/A

**Patient Symptoms**

skin issue

**Diagnosis**

Enter your diagnosis

**Treatment Plan**

Describe the treatment plan

MedTrack
Dashboard Profile Logout

### Appointment Details

**Patient Information**
Name: sruthi  
Status: Completed  
Date: 2025-10-10  
Created: N/A

**Patient Symptoms**
skin issue

**Diagnosis**
Enter your diagnosis

**Treatment Plan**
Describe the treatment plan

## Search Appointment:

The Search Appointment feature in the Doctor Dashboard allows doctors to quickly find and manage specific patient appointments. It typically includes:

1. **Search Bar:** A field where doctors can enter a patient's name, appointment date, or other relevant details to filter appointments.
2. **Appointment List:** Displays the search results, including patient names, appointment times, and statuses, for easy access.
3. **Action Options:** Options to view, update, or cancel appointments directly from the search results.

This feature streamlines appointment management, making it easier for doctors to find and handle patient requests.

MedTrack
Dashboard Profile Logout

### Search Results

Showing results for: "sruthi" (1 found)

Patient Name	Date	Symptoms	Status	Actions
sruthi	2025-10-10	skin issue	Completed	<button>View</button>

Back to Dashboard

## **DynamoDB Database updations :**

### **1. Users table :**

In the Users Table of DynamoDB, the data structure is designed to store user-related information for both patients and doctors. Typical updates include:

- **Add New Users:** When a new patient or doctor registers, their details such as name, email, role (patient/doctor), contact info, and password hash are added to the table.
- **Update User Info:** If a user updates their profile (e.g., changing contact details), the corresponding record in the table is modified.
- **Status Tracking:** Track the status of user accounts (active, inactive) based on their activity or admin updates.

The Users Table serves as the central repository for all user data, enabling quick access and modification of details when necessary

### **2. Appointment table :**

In the Appointment Table of DynamoDB, the data structure stores information related to patient appointments. Typical updates include:

- **Add New Appointment:** When a patient books an appointment, details such as patient ID, doctor ID, appointment date, time, and status (pending, confirmed, canceled) are stored.
- **Update Appointment Status:** As appointments are confirmed, rescheduled, or canceled, the status field in the table is updated accordingly.
- **Appointment History:** Historical data about completed appointments can also be stored to track past interactions between patients and doctors.

The Appointment Table allows for efficient management of appointments, ensuring accurate and up-to-date scheduling information for both doctors and patients

### **3. Mail to the User:**

- A confirmation email is automatically sent to the user upon successful appointment booking. This mail includes details such as appointment date, time, doctor's name, and status (confirmed or pending).
- In case of rescheduling or cancellation, users are notified instantly via email to ensure they stay updated on any changes.
- This ensures timely communication and improves the overall user experience within the MedTrack system.



## 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.

