

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.

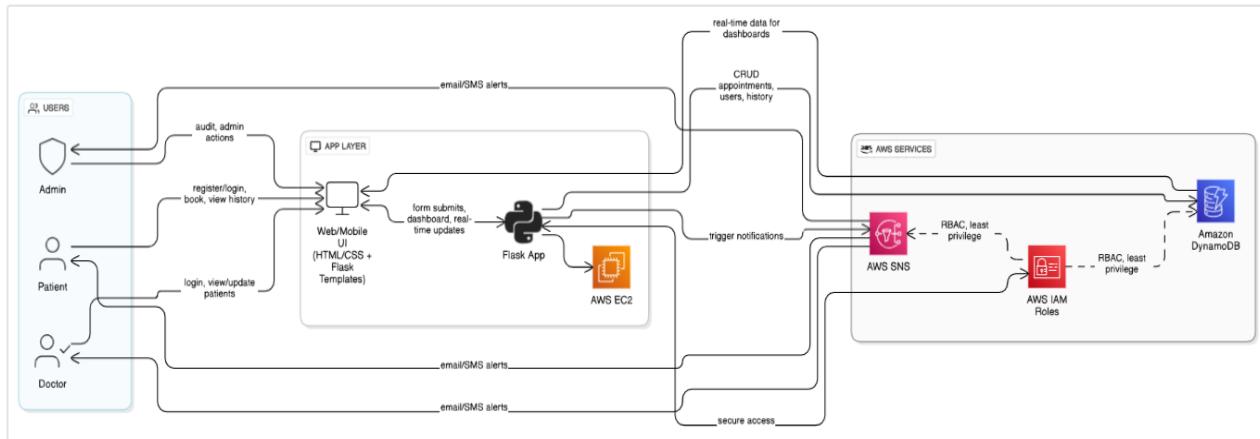
Scenario 2: Secure User Management with IAM

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.

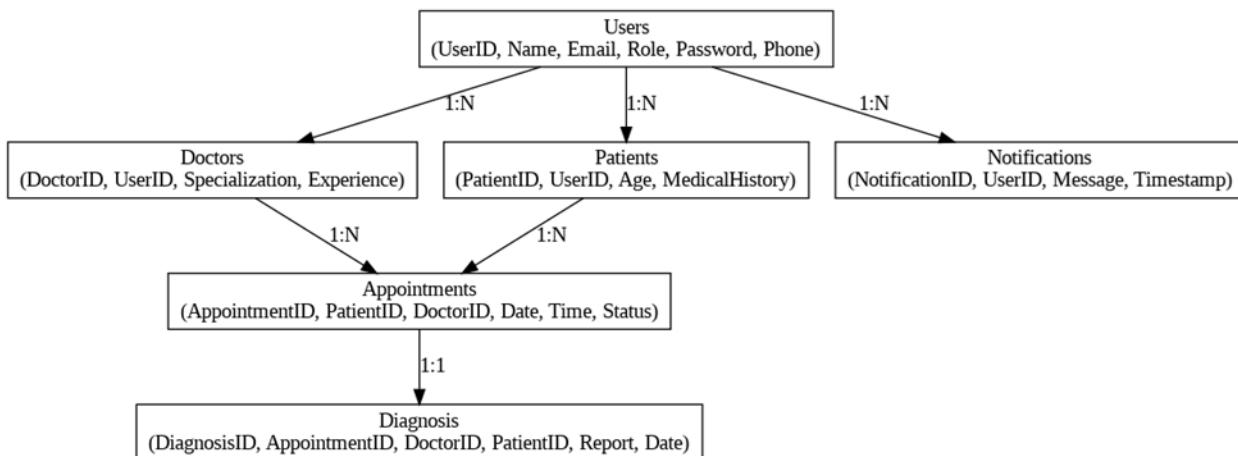
Scenario 3: Easy Access to Medical History and Resources

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

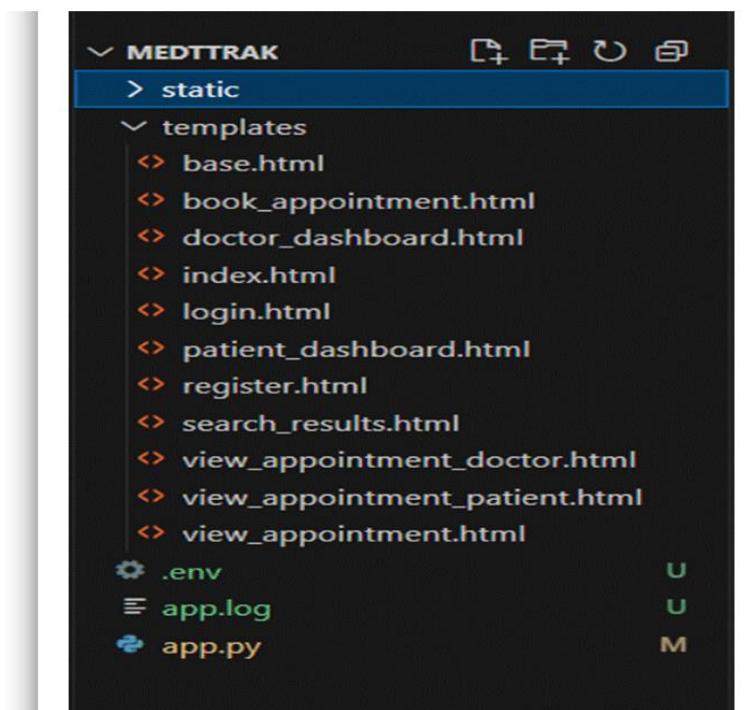
- Conduct functional testing to verify user registration, login, book requests, and notifications

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

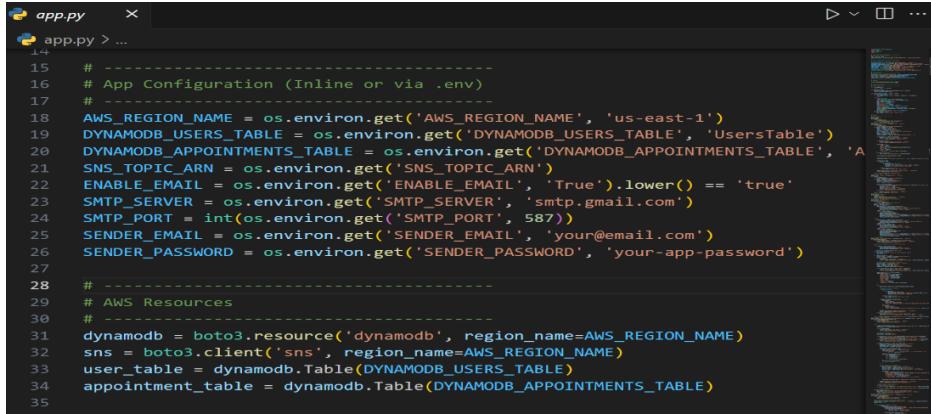
File Explorer Structure



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.

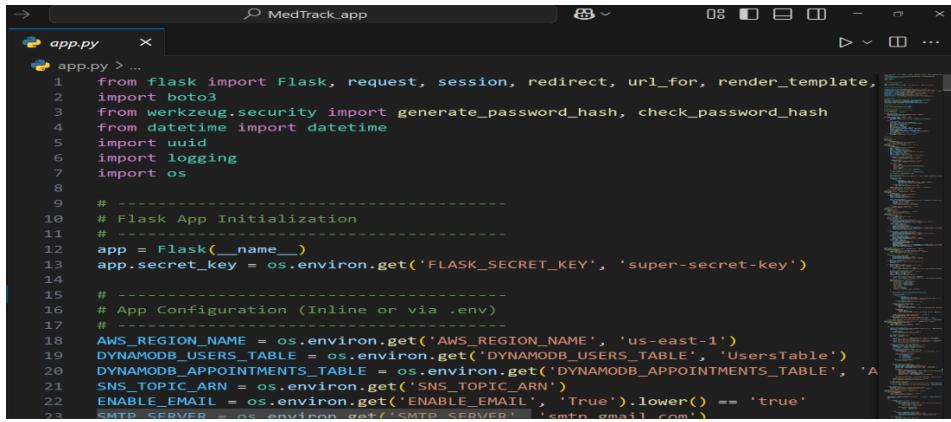


```

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')
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

```

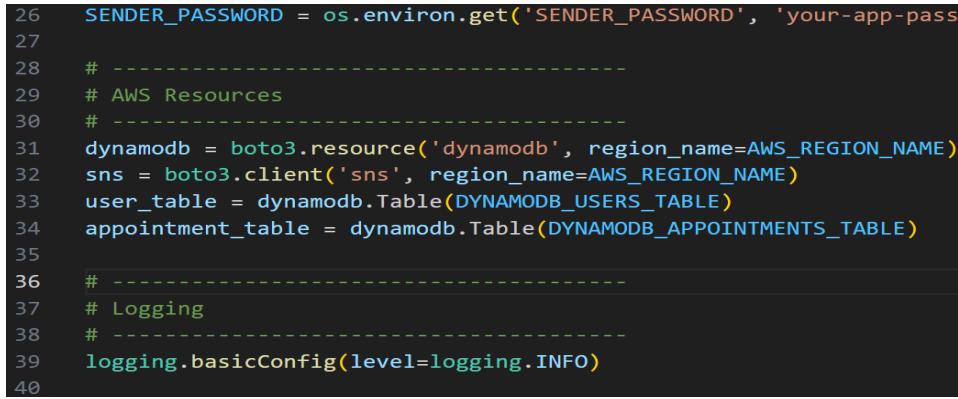
In the MedTrack project, AWS SNS sends real-time notifications to patients and doctors about appointments and updates. DynamoDB stores user data, medical records, and appointments securely, offering fast, scalable access. Both services are integrated with Flask to ensure smooth communication and efficient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



```

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')
24
25 # -----
26 # AWS Resources
27 #
28 dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION_NAME)
29 sns = boto3.client('sns', region_name=AWS_REGION_NAME)
30 user_table = dynamodb.Table(DYNAMODB_USERS_TABLE)
31 appointment_table = dynamodb.Table(DYNAMODB_APPOINTMENTS_TABLE)
32
33 # -----
34 # Logging
35 #
36 logging.basicConfig(level=logging.INFO)
37
38
39
40

```



```

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.

```

35 # Register User (Doctor/Patient)
36 @app.route('/register', methods=['GET', 'POST'])
37 def register():
38     if is_logged_in():
39         return redirect(url_for('dashboard'))
40     if request.method == 'POST':
41         name = request.form['name']
42         email = request.form['email']
43         password = request.form['password']
44         confirm_password = request.form['confirmPassword']
45         age = request.form['age']
46         gender = request.form['gender']
47         role = request.form['role']
48         specialization = request.form.get('specialization', '')
49
50         if password != confirm_password:
51             flash('Passwords do not match!', 'danger')
52             return render_template('register.html')
53
54         if get_user(email):
55             flash('Email already registered.', 'danger')
56             return render_template('register.html')
57
58         user = User(name=name, email=email, password=password, role=role, age=age, gender=gender, specialization=specialization)
59         db.session.add(user)
60         db.session.commit()
61         session['email'] = email
62         session['role'] = role
63         session['name'] = user.name
64         flash('User registered successfully.', 'success')
65         return redirect(url_for('login'))
66
67     return render_template('register.html')
68 
```

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
58 
```

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', '')
    appointments = []

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

            user_table.update_item(
                Key={'email': email},
                UpdateExpression=update_expression,
                ExpressionAttributeNames={'#name': 'name'},
                ExpressionAttributeValues=expression_values
            )
    
```

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.

```

        flash('Diagnosis submitted successfully.', 'success')
        return redirect(url_for('dashboard'))

    except Exception as e:
        app.logger.error(f"Submit diagnosis error: {e}")
        flash('An error occurred while submitting the diagnosis. Please try again.')
        return redirect(url_for('view_appointment', appointment_id=appointment_id))

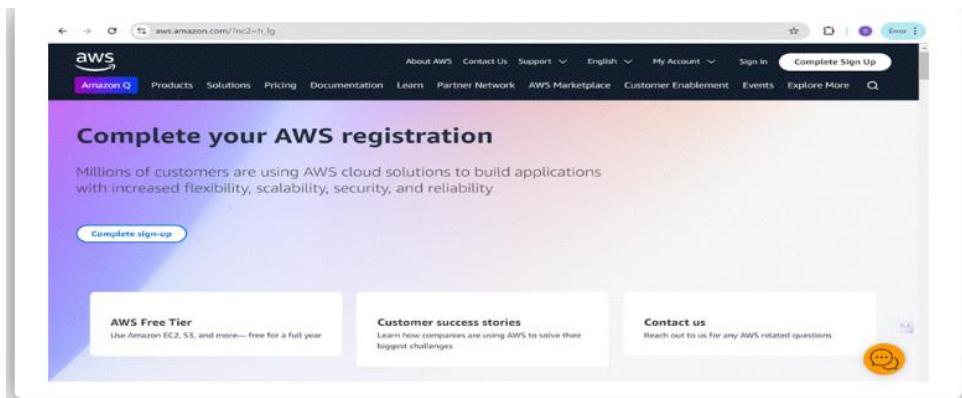
# -----
# Entrypoint for WSGI (production servers)
# -----
application = app # For gunicorn, uWSGI, etc.

# For local development
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5000))
    debug_mode = os.environ.get('FLASK_DEBUG', 'False').lower() == 'true'
    app.run(host='0.0.0.0', port=port, debug=debug_mode)

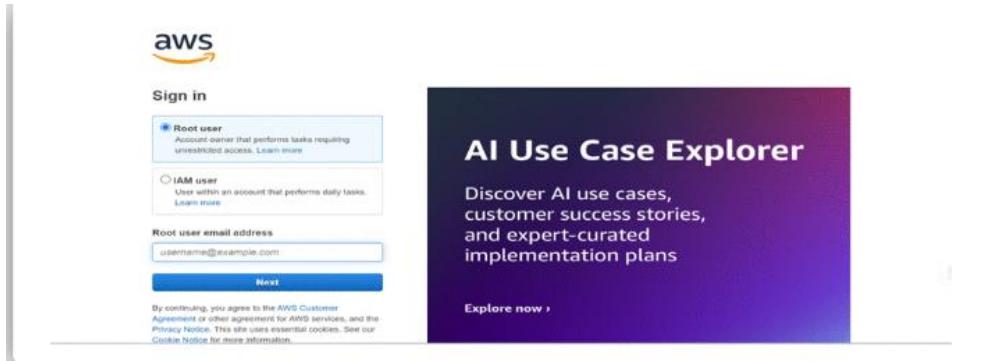
```

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](#).

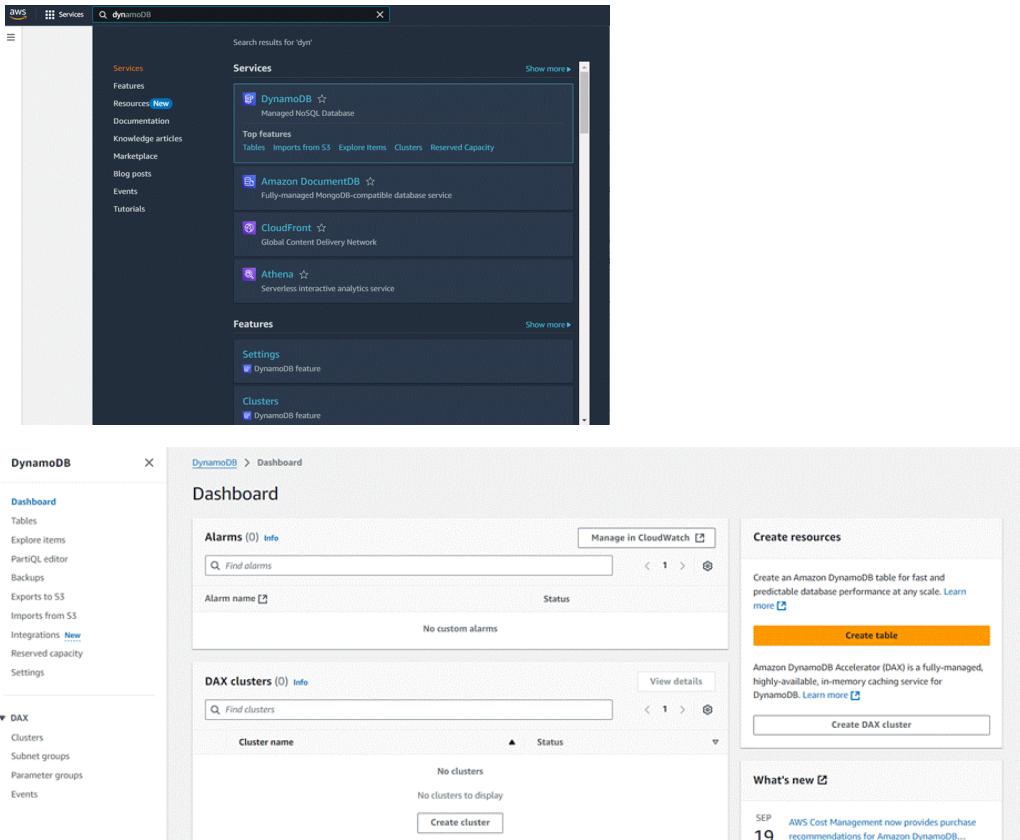


Milestone 3: DynamoDB Database Creation and Setup

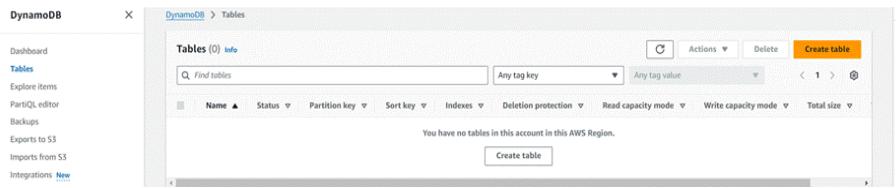
Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high-performance data storage for seamless backend operations.

Navigate to the DynamoDB

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

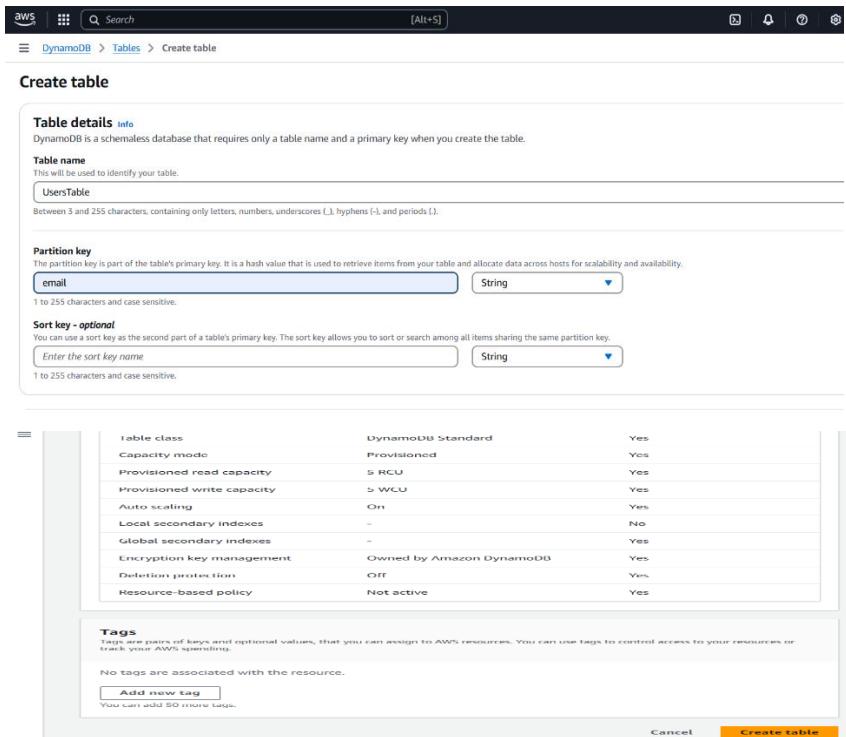


The screenshot shows the AWS DynamoDB dashboard. On the left, there is a sidebar with links for Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations (New), Reserved capacity, Settings, DAX (Clusters, Subnet groups, Parameter groups, Events), and a What's new section. The main area is titled 'DynamoDB > Dashboard' and shows sections for 'Alarms (0)', 'DAX clusters (0)', and 'What's new'. On the right, there is a 'Create resources' section with a prominent orange 'Create table' button. Below it, there is information about Amazon DynamoDB Accelerator (DAX) and a 'Create DAX cluster' button. A 'What's new' section at the bottom right mentions 'AWS Cost Management now provides purchase recommendations for Amazon DynamoDB...'.



Create a DynamoDB table for storing data

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

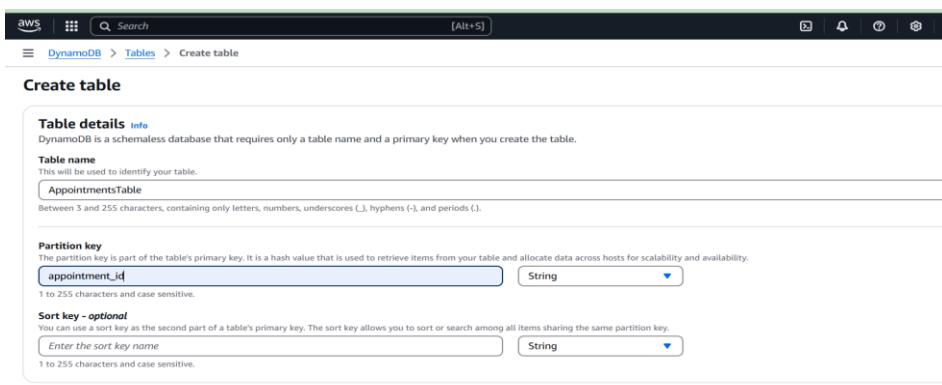


This screenshot shows the 'Create table' dialog for a new table named 'UsersTable'. It includes sections for 'Table details', 'Partition key', 'Sort key - optional', 'Table class', and 'Tags'. The 'Table class' section shows the following configuration:

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

The 'Tags' section indicates that no tags are associated with the resource.

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



This screenshot shows the 'Create table' dialog for a new table named 'AppointmentsTable'. It includes sections for 'Table details', 'Partition key', and 'Sort key - optional'. The 'Partition key' section shows the following configuration:

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	Type: String

The 'Sort key - optional' section indicates that no sort key is defined.

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](#)

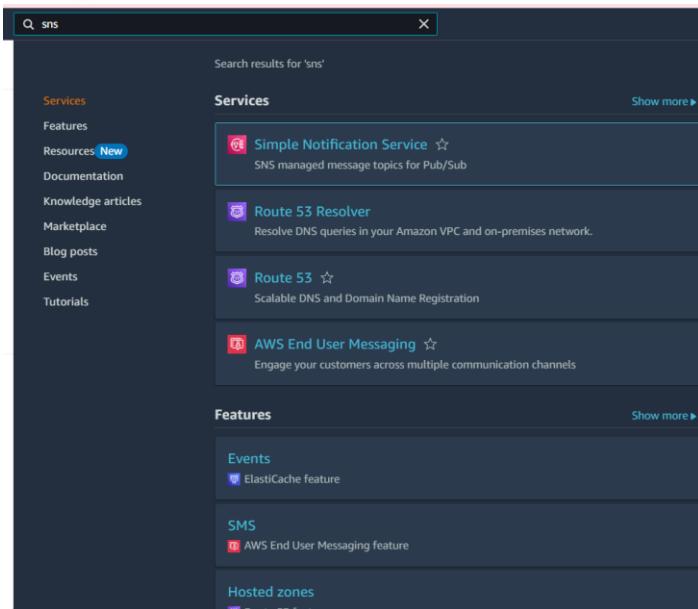
<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection
<input checked="" type="checkbox"/>	AppointmentsTable		Active	appointment_id (\$)	-	0 0	
<input type="checkbox"/>	NextGenHospital_Appointments		Active	appointment_id (\$)	-	0 0	
<input type="checkbox"/>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

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.



Search results for 'sns'

Services

- Simple Notification Service
- Route 53 Resolver
- Route 53
- AWS End User Messaging

Features

- Events
- SMS
- Hosted zones



New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Application Integration

Amazon Simple Notification Service

Pub/sub messaging for microservices and serverless applications.

Amazon SNS is a highly available, durable, secure, fully-managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.

Create topic

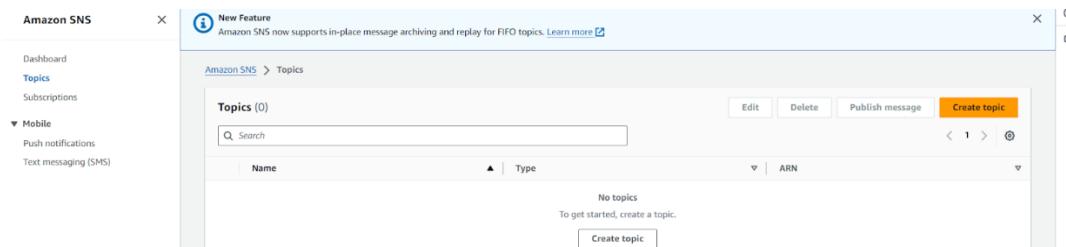
Topic name: MyTopic

Next step

[Start with an overview](#)

Pricing

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



New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Amazon SNS > Topics

Name	Type	ARN
No topics		
To get started, create a topic.		
Create 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, Lambda, Data Firehose, HTTP/SMS, email, mobile application endpoints

Standard

- Best-effort message ordering
- All-at-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 SMS 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**.

Amazon SNS > Topics > Medtrack

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

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

Medtrack

[Edit](#) [Delete](#) [Publish message](#)

Details	
Name	Medtrack
ARN	arn:aws:sns:ap-south-1:1940482422578:Medtrack
Type	Standard

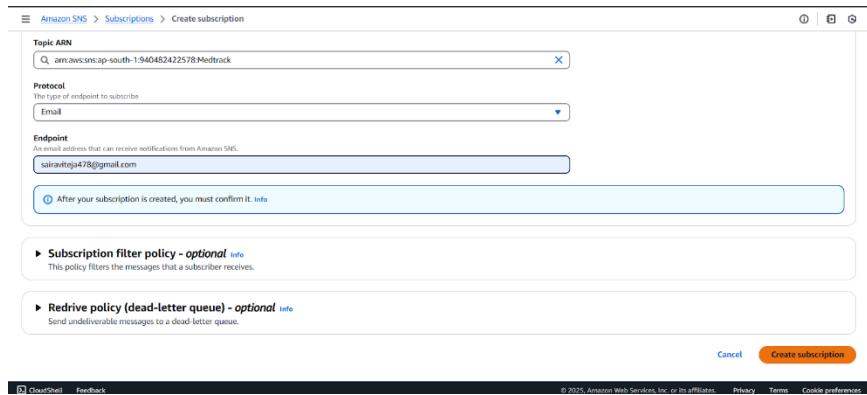
[Subscriptions](#) [Access policy](#) [Data protection policy](#) [Delivery policy \(HTTP/S\)](#) [Delivery status logging](#) [Encryption](#) [Tags](#)

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

Search

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.



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

Protocol
The type of endpoint to subscribe
Email

Endpoint
An email address that can receive notifications from Amazon SNS.
sairaviteja478@gmail.com

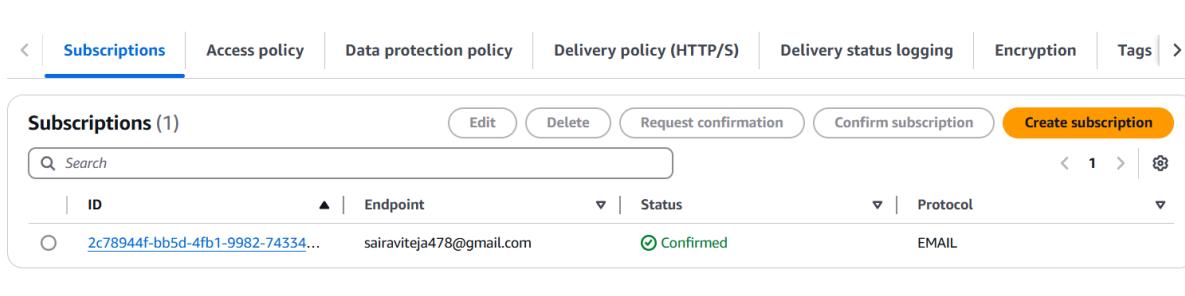
After your subscription is created, you must confirm it. [Info](#)

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 (1)

ID	Endpoint	Status	Protocol
2c78944f-bb5d-4fb1-9982-74334...	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 ▾

9

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!

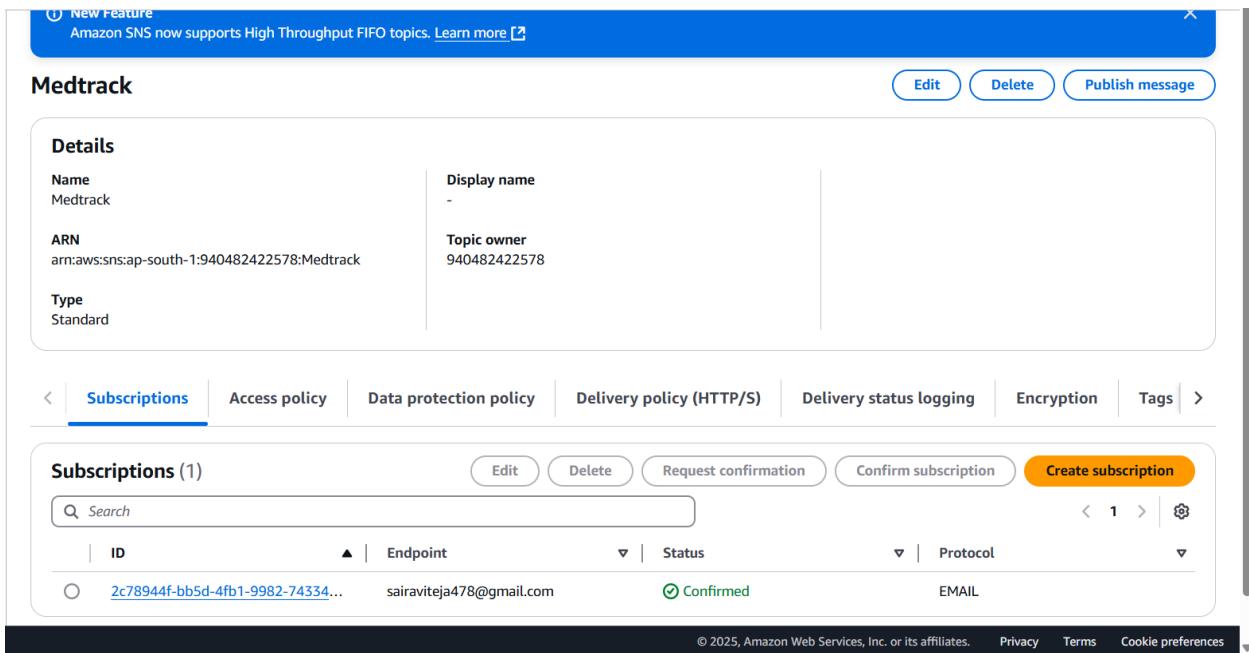
You have successfully subscribed.

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.



The screenshot shows the AWS SNS console for the 'Medtrack' topic. At the top, there is a blue banner with a 'New Feature' message: 'Amazon SNS now supports High Throughput FIFO topics. [Learn more](#)'. Below the banner, the topic name 'Medtrack' is displayed, along with 'Edit', 'Delete', and 'Publish message' buttons. The 'Details' section shows the topic's name, ARN, and type (Standard). The ARN is listed as 'arn:aws:sns:ap-south-1:940482422578:Medtrack'. The 'Subscriptions' tab is selected, showing one subscription entry:

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

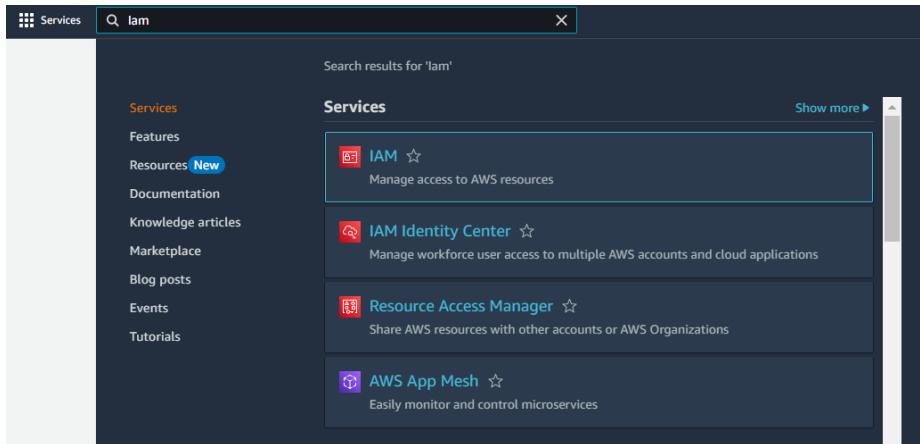
Other tabs available include 'Access policy', 'Data protection policy', 'Delivery policy (HTTP/S)', 'Delivery status logging', 'Encryption', and 'Tags'. At the bottom, there are links for 'Create subscription', 'Search', and navigation controls.

Milestone 5: IAM Role Setup

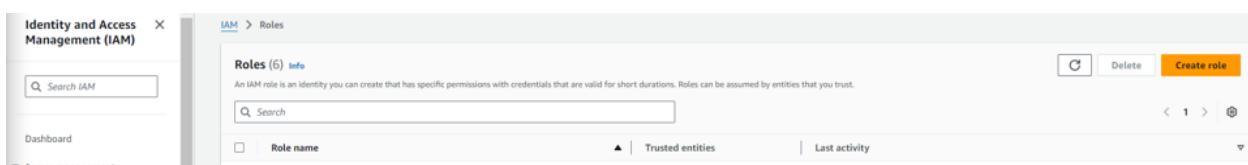
IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows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

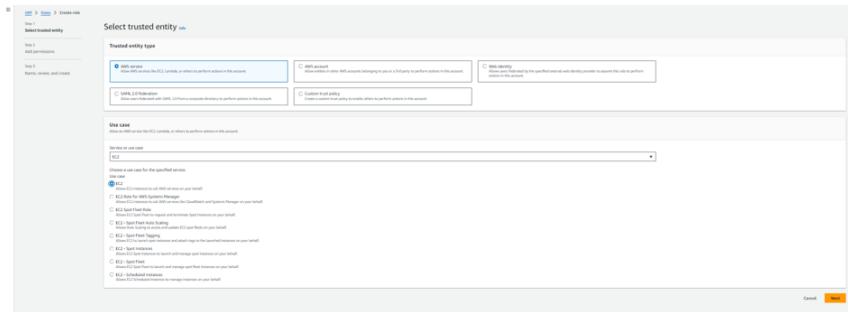


The screenshot shows the AWS IAM search results for the term 'Iam'. The left sidebar has a 'Services' section with links to Features, Resources (marked as New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main content area displays four services: IAM, IAM Identity Center, Resource Access Manager, and AWS App Mesh. Each service has a small icon, a name, and a brief description.

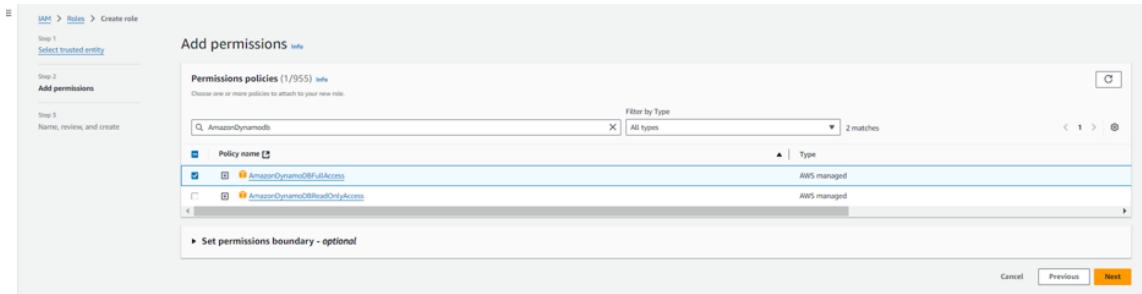


The screenshot shows the 'Roles' page in the AWS IAM console. It lists six roles. The top role, 'EC2', is selected. Other roles listed are 'AmazonSNSFullAccess', 'AmazonDynamoDBFullAccess', 'AmazonDynamoDBReadOnlyAccess', 'AmazonSQSFullAccess', and 'AmazonS3FullAccess'. The interface includes a search bar, a 'Create role' button, and navigation controls.

To create and select DynamoDBFullAccess and SNSFullAccess, go to the AWS IAM console, create a new role, and assign the respective policies. DynamoDBFullAccess allows full access to DynamoDB resources, while SNSFullAccess enables sending notifications via SNS. Attach the role to the relevant services to ensure proper integration with the project.



The screenshot shows the 'Select trusted entity' step in the 'Create role' wizard. It asks for a 'Trusted entity type' and provides options: 'AWS service', 'AWS account', 'AWS identity', 'AWS Lambda function with IAM role', or 'Custom trust policy'. Below this, it asks for a 'Service or user' and lists 'EC2' as the selected option. A large list of AWS services and their trust policies is shown below the selection fields.

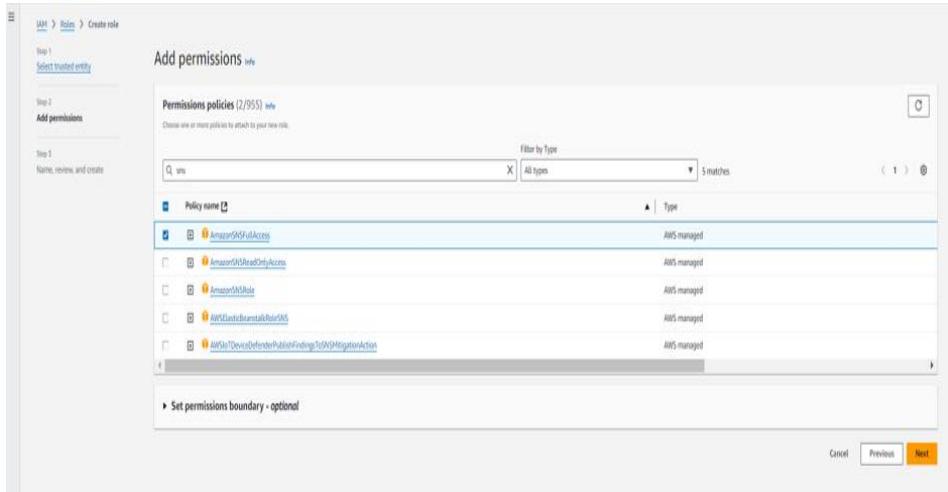


The screenshot shows the 'Add permissions' step in the 'Create role' wizard. It displays a list of 'Permissions policies (1/955)' with one item selected: 'AmazonDynamoDBFullAccess'. A 'Filter by Type' dropdown shows 'All types' with '2 matches'. At the bottom, there is a note about setting a 'permissions boundary - optional'.

Attach Policies.

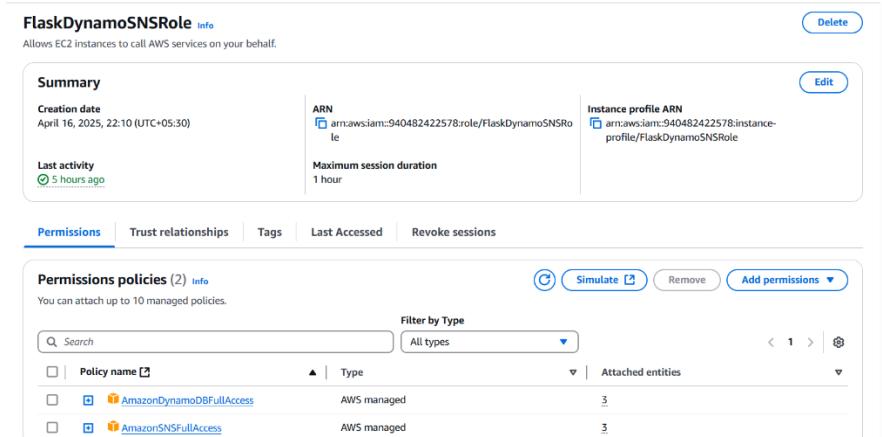
Attach the following policies to the role:

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



The screenshot shows the 'Add permissions' step in the AWS IAM console. It displays a search results page for 'Permissions policies (2/955)'. A search bar at the top contains 'sns'. Below it, a table lists several AWS managed policies, with 'AmazonSNSFullAccess' being the selected policy. The table includes columns for 'Policy name', 'Type', and a small icon. At the bottom of the table, there is a link 'Set permissions boundary - optional'.

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.



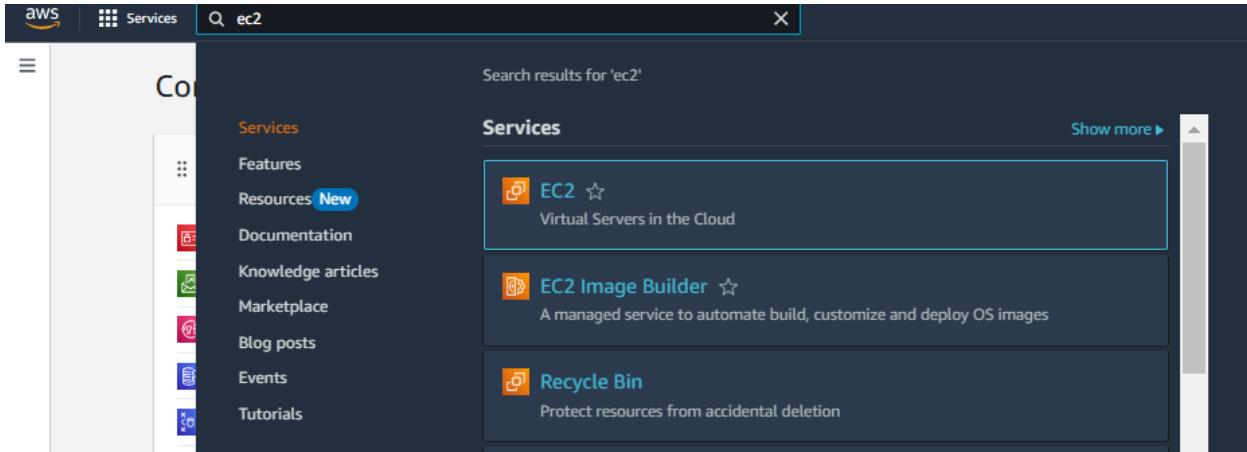
The screenshot shows the details of a role named 'FlaskDynamoSNSRole'. It includes sections for 'Summary' (Creation date: April 16, 2025, 22:10 (UTC+0:30), Last activity: 5 hours ago), 'ARN' (arn:aws:iam::940482422578:role/FlaskDynamoSNSRole), 'Instance profile ARN' (arn:aws:iam::940482422578:instance-profile/FlaskDynamoSNSRole), and 'Permissions' (Maximum session duration: 1 hour). The 'Permissions' section shows two attached policies: 'AmazonDynamoDBFullAccess' and 'AmazonSNSFullAccess', both listed under 'Attached entities'.

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.

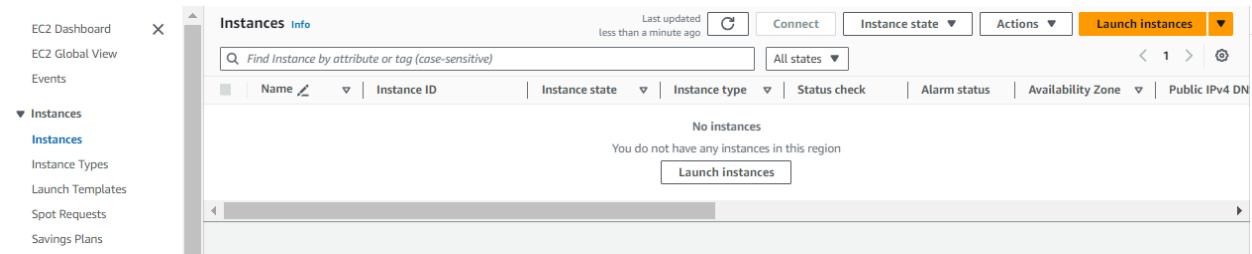


The screenshot shows the AWS CloudSearch interface with a search query of 'ec2'. The results are categorized under 'Services' and include:

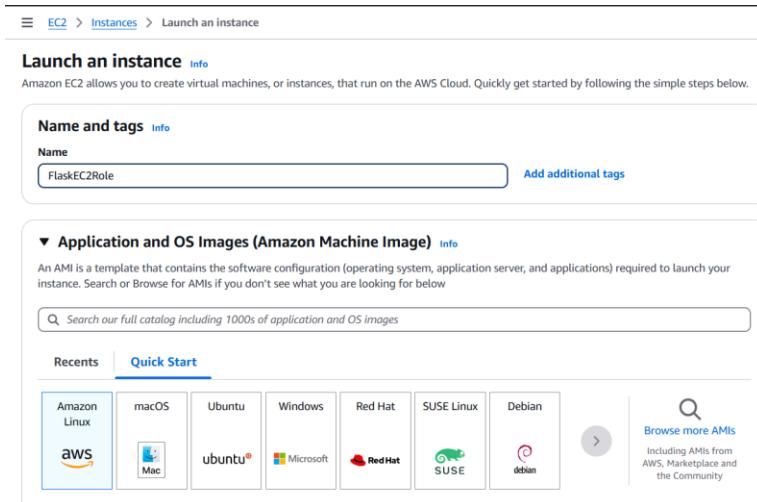
- EC2** ★ Virtual Servers in the Cloud
- EC2 Image Builder** ★ A managed service to automate build, customize and deploy OS images
- Recycle Bin** Protect resources from accidental deletion

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.



The screenshot shows the AWS EC2 Instances page. The sidebar includes options like EC2 Dashboard, EC2 Global View, Events, Instances (selected), Instances Types, Launch Templates, Spot Requests, and Savings Plans. The main area displays the 'Instances Info' section with a search bar, filters, and a message stating 'No instances' with a 'Launch instances' button.

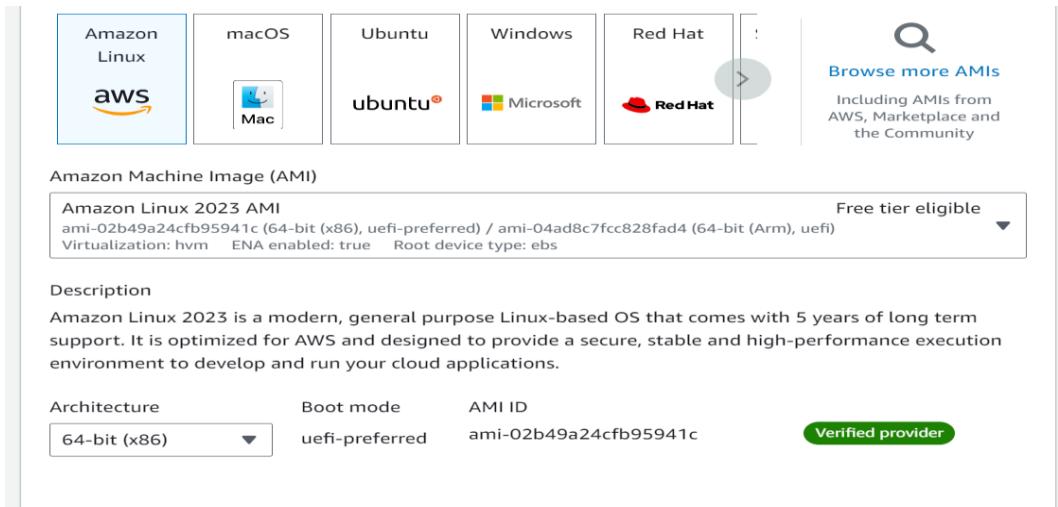


The screenshot shows the 'Launch an instance' wizard. It starts with the 'Name and tags' step, where 'FlaskEC2Role' is entered. The next step is 'Application and OS Images (Amazon Machine Image)', which lists various AMIs including Amazon Linux 2, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian. A search bar and a 'Quick Start' tab are also visible.

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.

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



Amazon Machine Image (AMI)

Amazon Linux 2023 AMI Free tier eligible

ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)

Virtualization: hvm ENA enabled: true Root device type: ebs

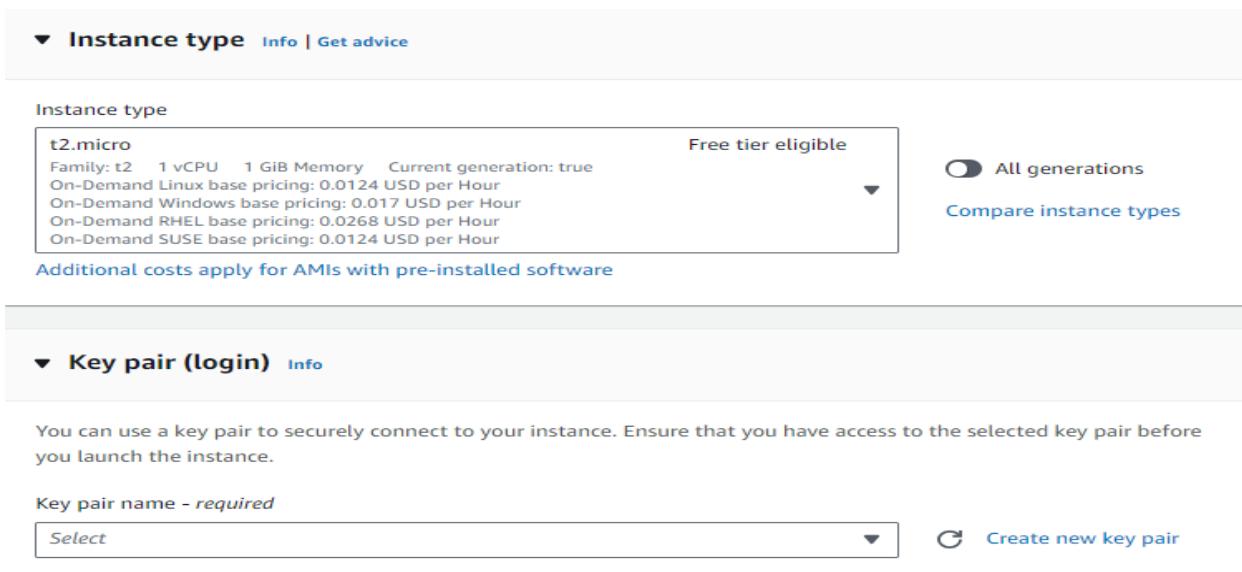
Description

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

To create and download the key pair for server access:

- In the **Launch Instance** wizard, under the **Key Pair** section, click **Create a new key pair**.
- 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	Free tier eligible
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	

All generations

Compare instance types

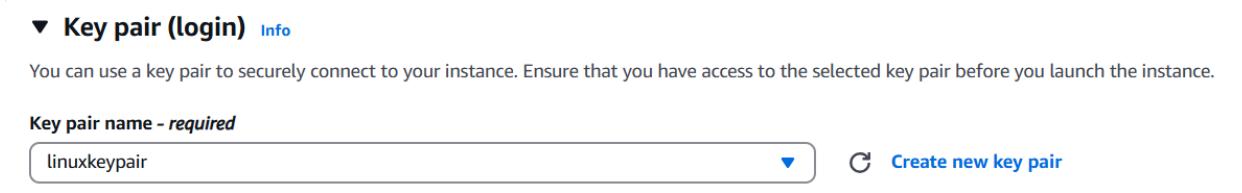
Additional costs apply for AMIs with pre-installed software

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](#)

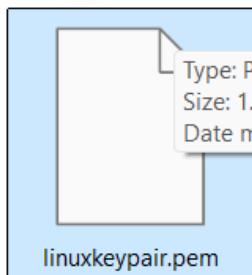


Key pair (login) [Info](#)

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

VPC - required: **Info**
 VPC-03cdc7b6f19dd7211 (default)
 172.31.0.0/16

Subnet: **Info**
 No preference 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)

Type	Protocol	Port range	Action
ssh	TCP	22	Remove
Source type	Source	Description - optional	
Anywhere	Add CIDR, prefix list or security	e.g. SSH for admin desktop	
0.0.0.0/0 X			

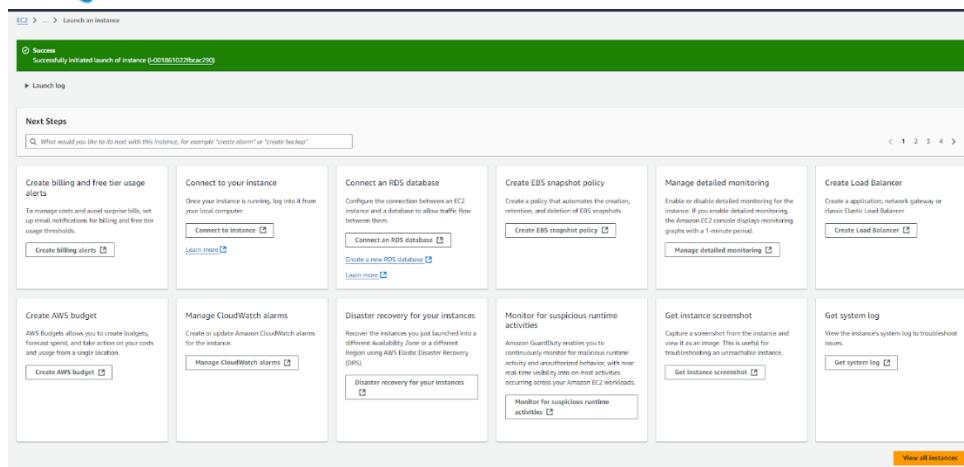
Security group rule 2 (TCP, 80, 0.0.0.0/0)

Type	Protocol	Port range	Action
HTTP	TCP	80	Remove
Source type	Source	Description - optional	
Custom	Add CIDR, prefix list or security	e.g. SSH for admin desktop	
0.0.0.0/0 X			

Security group rule 3 (TCP, 5000, 0.0.0.0/0)

Type	Protocol	Port range	Action
Custom TCP	TCP	5000	Remove
Source type	Source	Description - optional	
Custom	Add CIDR, prefix list or security	e.g. SSH for admin desktop	
0.0.0.0/0 X			

Add security group rule

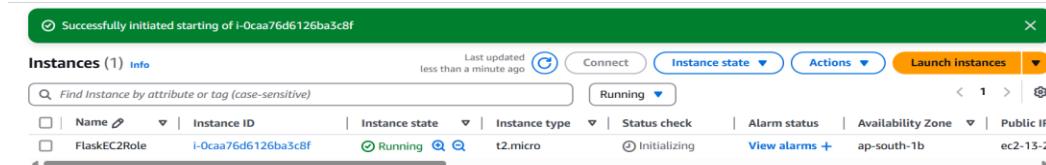


The screenshot shows the AWS EC2 instance launch success screen. At the top, a green banner says "Success Successfully initiated launch of instance i-00186102fbew296". Below it, a "Next Steps" section lists various AWS services with "Create" buttons:

- Create billing and free tier usage alerts
- Connect to your instance
- Connect an RDS database
- Create EBS snapshot policy
- Manage detailed monitoring
- Create Load Balancer
- Create AWS budget
- Manage CloudWatch alarms
- Disaster recovery for your instances
- Monitor for suspicious runtime activities
- Get instance screenshot
- Get system log

At the bottom right is an orange "View all instances" button.

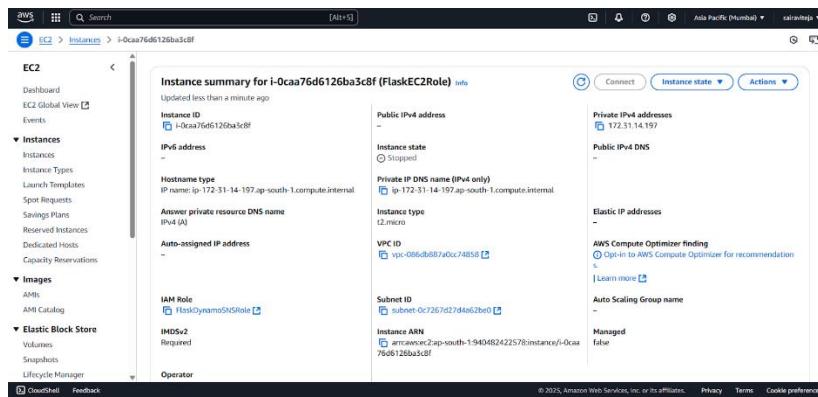
To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected,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



The screenshot shows the AWS EC2 Instances dashboard. A green banner at the top says "Successfully initiated starting of i-0caa76d6126ba3c8f". The main table shows one instance:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
FlaskEC2Role	i-0caa76d6126ba3c8f	Running	t2.micro	Initializing		ap-south-1b	ec2-13-2-

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.



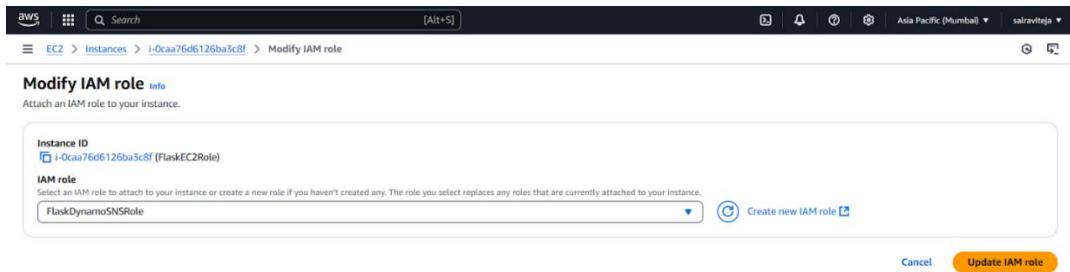
The screenshot shows the AWS EC2 Instance summary page for the instance i-0caa76d6126ba3c8f. It displays the following details:

- Instance summary for i-0caa76d6126ba3c8f (FlaskEC2Role)**
- Updated less than a minute ago
- Instance ID:** i-0caa76d6126ba3c8f
- Public IPv4 address:** 172.31.14.197
- Private IP DNS name (IPv4 only):** ip-172-31-14-197.ap-south-1.compute.internal
- Instance state:** Stopped
- Hostname type:** IP name: ip-172-31-14-197.ap-south-1.compute.internal
- Auto-assigned IP address:** -
- IAM Role:** FlaskGymnSNSRole
- IMDSv2:** Required
- Operator:** -
- Private IPv4 addresses:** 172.31.14.197
- Public IPv4 DNS name:** ip-172-31-14-197.ap-south-1.compute.internal
- Elastic IP addresses:** -
- AWS Compute Optimizer finding:** Opt-in to AWS Compute Optimizer for recommendation
- Subnet ID:** subnet-0c7267d27d4a62be0
- Instance ARN:** arn:aws:ec2:ap-south-1:940482422578:instance/i-0caa76d6126ba3c8f
- Managed:** false

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

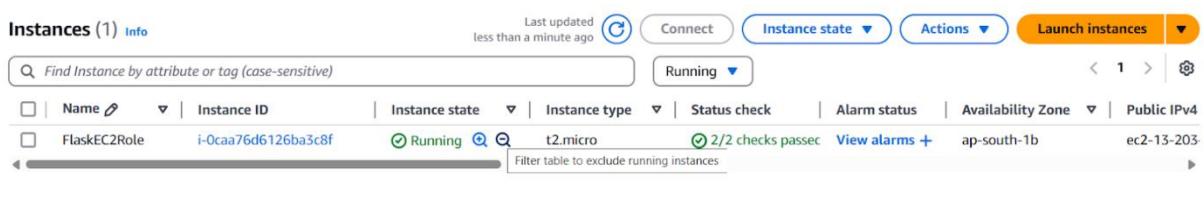
1. Go to the **AWS IAM Console**, select **Roles**, and find the **flaskec2role**.
2. Click **Attach Policies**, then choose the required policies (e.g., **DynamoDBFullAccess**, **SNSFullAccess**) and click **Attach Policy**.

3. 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.

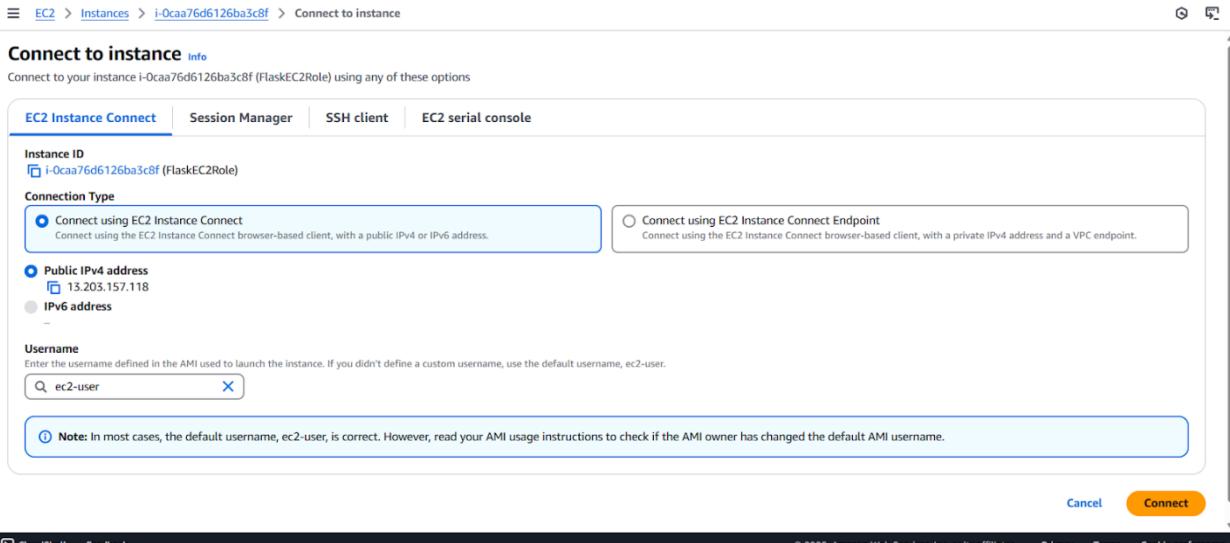


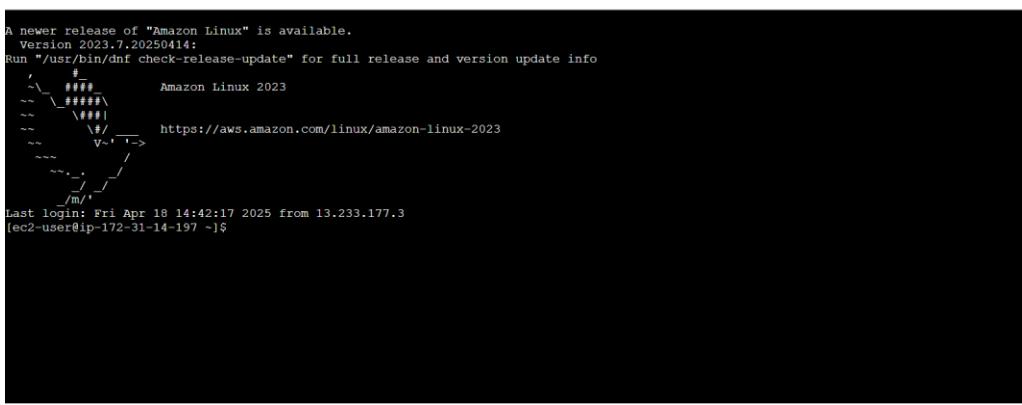
To connect to your EC2 instance:

1. Go to the **EC2 Dashboard**, select your running instance, and click **Connect**.
2. 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.



- Now connect the EC2 with the files





```
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
# 
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Fri Apr 18 14:42:17 2025 from 13.233.177.3
[ec2-user@ip-172-31-14-197 ~]$
```

i-0caa76d6126ba3c8f (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/ArchanaPulakhandam/MedTrack_app'

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] |   0 |  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 - 110.235.236.49 - - [18/Apr/2025 08:02:36] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:02:39,258 - werkzeug - INFO - 110.235.236.49 - - [18/Apr/2025 08:02:39] "GET /logout HTTP/1.1" 302 -
2025-04-18 08:02:39,289 - werkzeug - INFO - 110.235.236.49 - - [18/Apr/2025 08:02:39] "GET /HTTP/1.1" 200 -
2025-04-18 08:02:41,589 - werkzeug - INFO - 110.235.236.49 - - [18/Apr/2025 08:02:41] "GET /login HTTP/1.1" 200 -
2025-04-18 08:02:47,468 - werkzeug - INFO - 110.235.236.49 - - [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 - 110.235.236.49 - - [18/Apr/2025 08:02:47] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:02:49,506 - werkzeug - INFO - 110.235.236.49 - - [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 gthamarasi19@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 - 110.235.236.49 - - [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 - 110.235.236.49 - - [18/Apr/2025 08:03:28] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:03:44,609 - werkzeug - INFO - 110.235.236.49 - - [18/Apr/2025 08:03:44] "GET /logout HTTP/1.1" 302 -
2025-04-18 08:03:44,654 - werkzeug - INFO - 110.235.236.49 - - [18/Apr/2025 08:03:44] "GET / HTTP/1.1" 200 -
^C[venv] [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-0caa76d6126ba3c8f(FlaskC2Role)

PublicIPs: 13.203.227.15 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/>

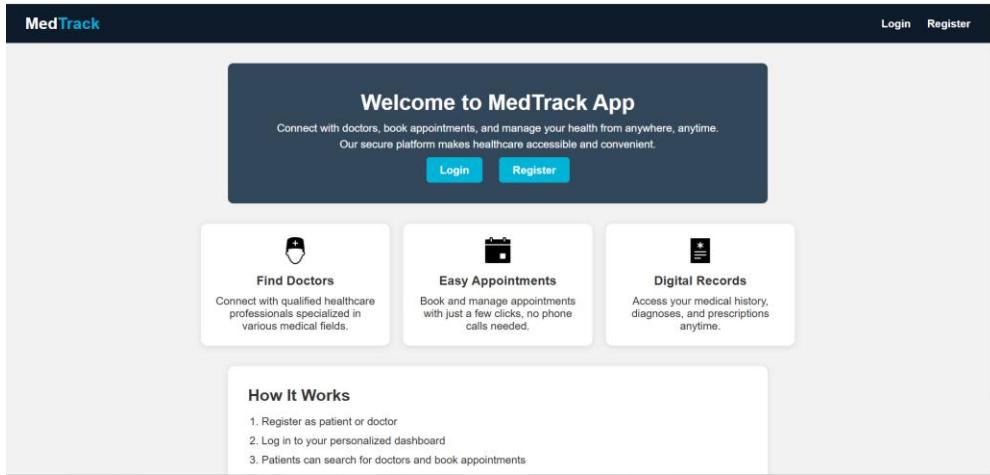
Milestone 8: Testing and Deployment

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.

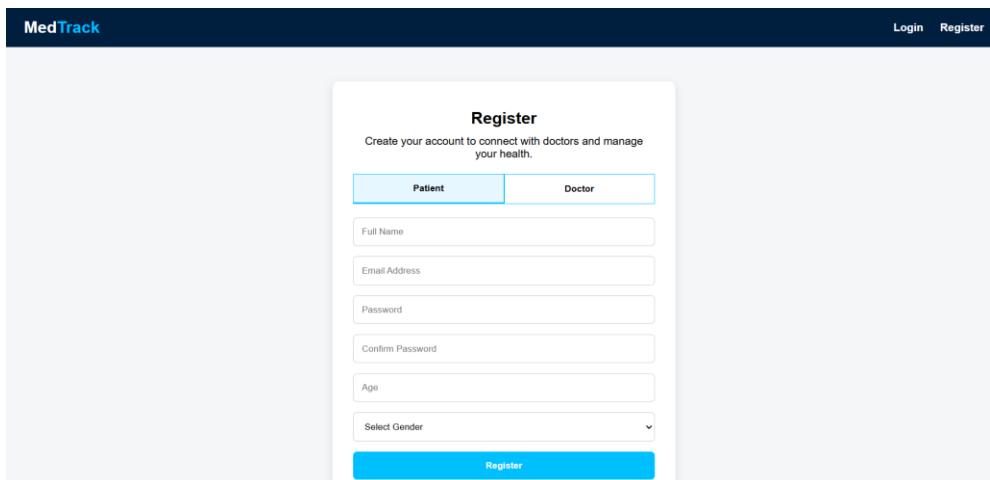


The screenshot shows the MedTrack Home Page. At the top, there is a dark header bar with the "MedTrack" logo on the left and "Login" and "Register" buttons on the right. Below the header is a large dark blue callout box with the title "Welcome to MedTrack App" and a subtext: "Connect with doctors, book appointments, and manage your health from anywhere, anytime. Our secure platform makes healthcare accessible and convenient." It features two buttons: "Login" and "Register". Below this are three white cards with icons and descriptions: "Find Doctors" (doctor icon), "Easy Appointments" (calendar icon), and "Digital Records" (database icon). At the bottom, there is a section titled "How It Works" with a numbered list: 1. Register as patient or doctor, 2. Log in to your personalized dashboard, 3. Patients can search for doctors and book appointments.

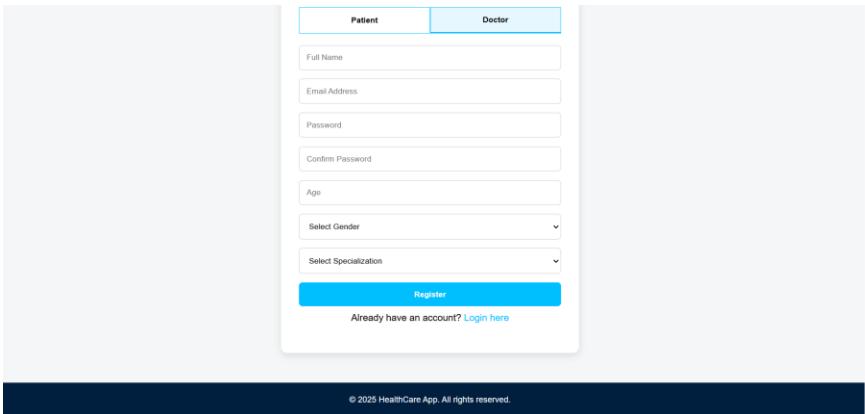
Doctor and Patient register page:

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 Register Page. At the top, there is a dark header bar with the "MedTrack" logo on the left and "Login" and "Register" buttons on the right. Below the header is a white registration form with a title "Register" and a subtitle: "Create your account to connect with doctors and manage your health." It features a tabbed input field between "Patient" and "Doctor". The form includes fields for "Full Name", "Email Address", "Password", "Confirm Password", "Age", and "Select Gender". At the bottom is a large blue "Register" button.



The registration form for SmartBridge is designed for patients. It includes fields for Full Name, Email Address, Password, Confirm Password, Age, Selected Gender, and Selected Specialization. A 'Register' button is at the bottom, followed by a link to 'Login here'.

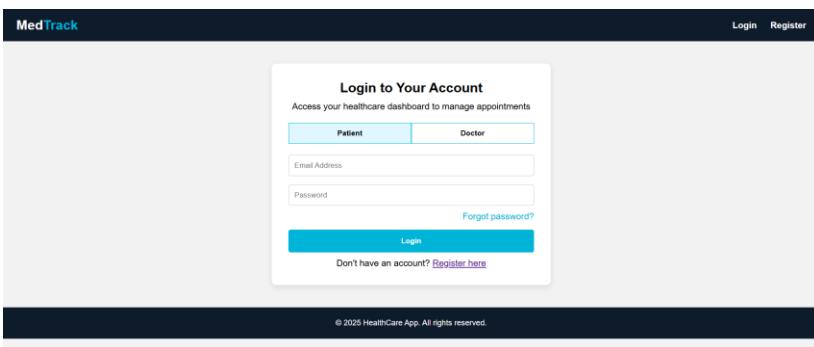
© 2025 HealthCare App. All rights reserved.

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:

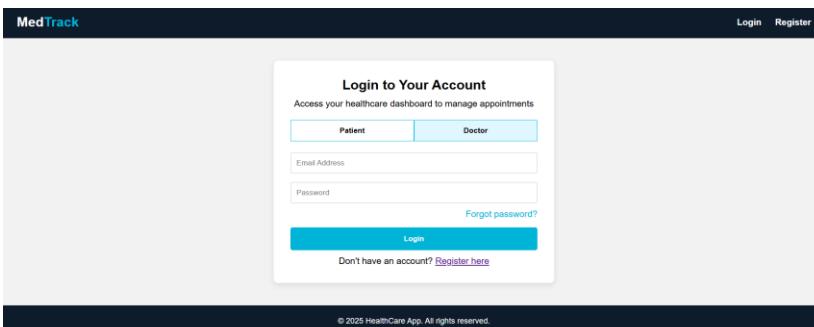
1. Username and Password Fields: Users enter their credentials (username and password) to authenticate their account.
2. Login Button: A button to submit login details and validate user access.

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



The login form for MedTrack is titled 'Login to Your Account' and includes a sub-instruction 'Access your healthcare dashboard to manage appointments'. It features a 'Patient' tab, an 'Email Address' field, a 'Password' field, a 'Forgot password?' link, and a 'Login' button. Below the button is a link to 'Register here'.

© 2025 HealthCare App. All rights reserved.



This version of the MedTrack login form is for doctors. It has a 'Doctor' tab instead of a 'Patient' tab. The rest of the interface is identical to the patient version, with fields for Email Address, Password, a 'Forgot password?' link, a 'Login' button, and a 'Register here' link.

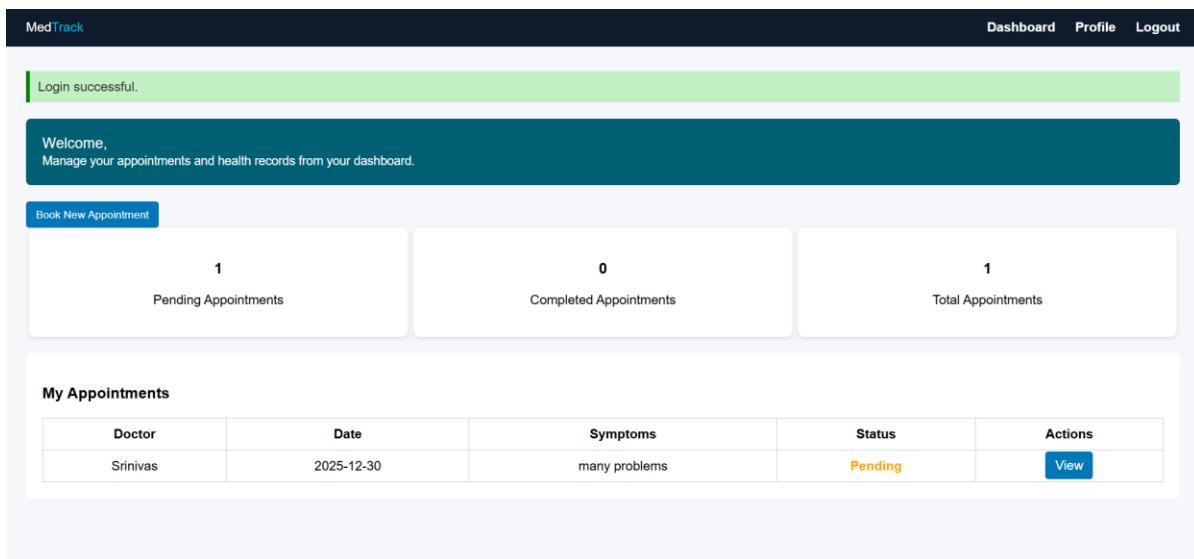
© 2025 HealthCare App. All rights reserved.

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.



MedTrack

Login successful.

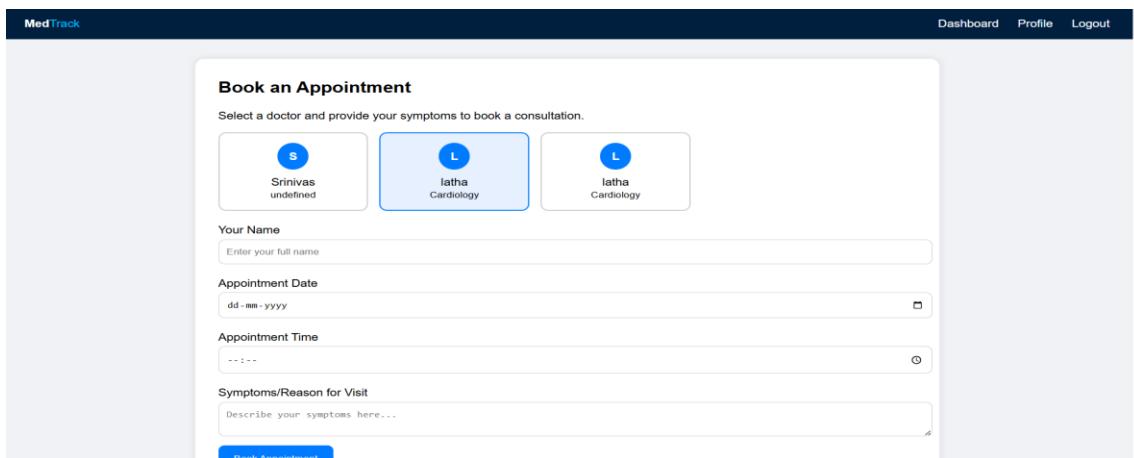
Welcome,
Manage your appointments and health records from your dashboard.

[Book New Appointment](#)

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

My Appointments

Doctor	Date	Symptoms	Status	Actions
Srinivas	2025-12-30	many problems	Pending	View



Book an Appointment

Select a doctor and provide your symptoms to book a consultation.

 Srinivas	 Iatha	 Iatha
undefined	Cardiology	Cardiology

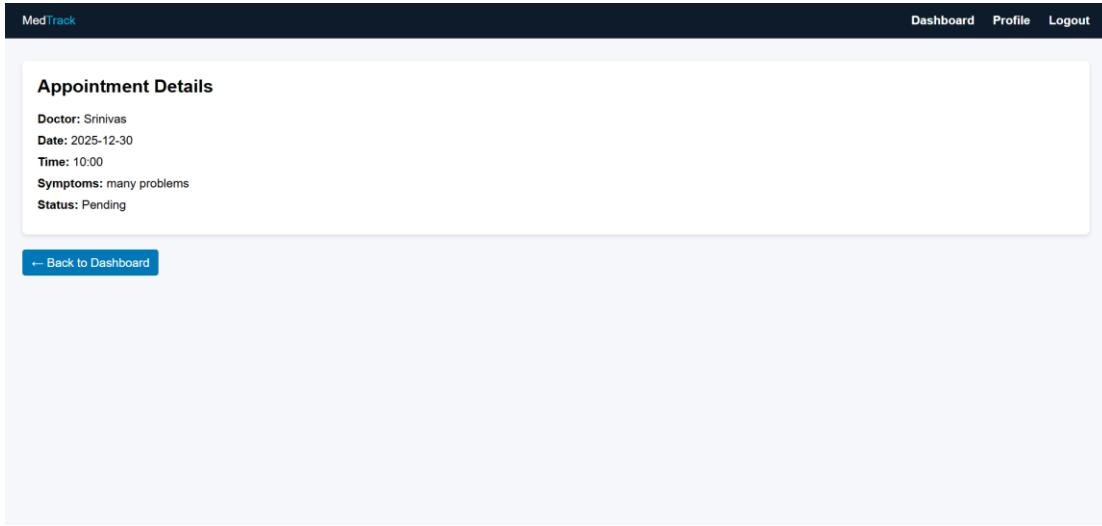
Your Name

Appointment Date

Appointment Time

Symptoms/Reason for Visit

[Book Appointment](#)



The screenshot shows a web-based application interface titled "MedTrack". At the top, there is a navigation bar with links for "Dashboard", "Profile", and "Logout". The main content area is titled "Appointment Details" and contains the following information:

Detail	Value
Doctor	Srinivas
Date	2025-12-30
Time	10:00
Symptoms	many problems
Status	Pending

At the bottom left of the content area, there is a blue button labeled "← 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...

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	<input type="button" value="View"/>

MedTrack

Appointment Details

Patient Information

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

Patient Symptoms

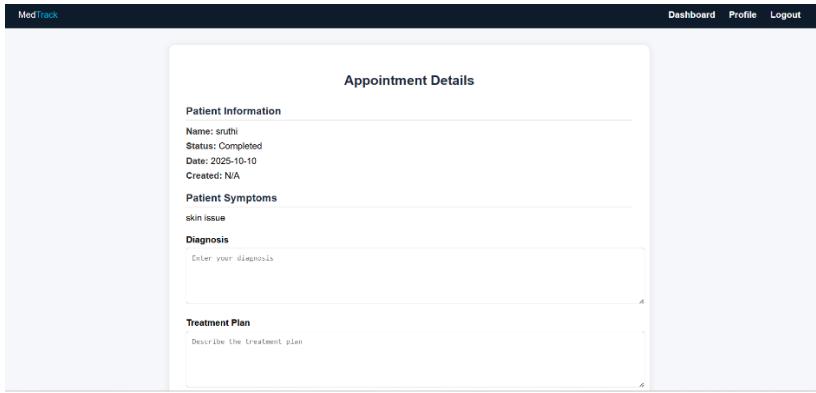
skin issue

Diagnosis

Enter your diagnosis

Treatment Plan

Describe the treatment plan



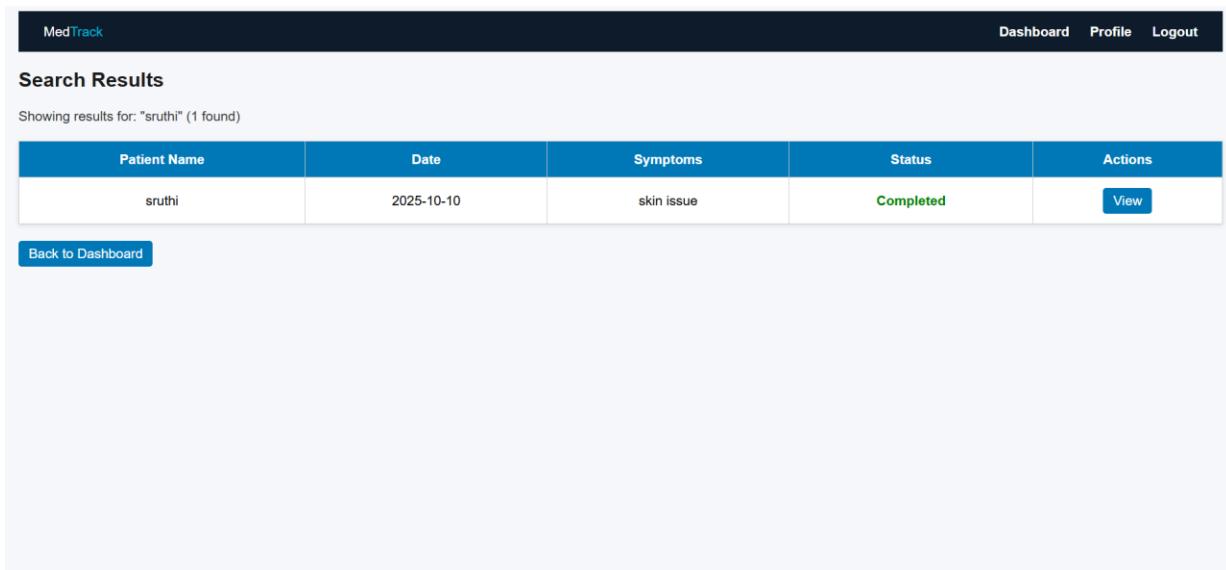
The screenshot shows the 'Appointment Details' section of the MedTrack Doctor Dashboard. At the top, there are navigation links: 'Dashboard', 'Profile', and 'Logout'. Below this, the 'Appointment Details' section is titled 'Appointment Details'. It contains three main sections: 'Patient Information', 'Patient Symptoms', and 'Treatment Plan'. Under 'Patient Information', the details are: Name: sruthi, Status: Completed, Date: 2025-10-10, Created: N/A. Under 'Patient Symptoms', the symptom listed is 'skin issue'. Under 'Treatment Plan', there is a placeholder text 'Enter your diagnosis'.

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.



The screenshot shows the 'Search Results' section of the MedTrack Doctor Dashboard. At the top, there are navigation links: 'Dashboard', 'Profile', and 'Logout'. Below this, the title 'Search Results' is displayed, followed by the message 'Showing results for: "sruthi" (1 found)'. A table is used to list the search results, with columns: Patient Name, Date, Symptoms, Status, and Actions. The single result is: Patient Name: sruthi, Date: 2025-10-10, Symptoms: skin issue, Status: Completed, Actions: View. At the bottom left, there is a 'Back to Dashboard' button.

DynamoDB Database updates :

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.

