

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:

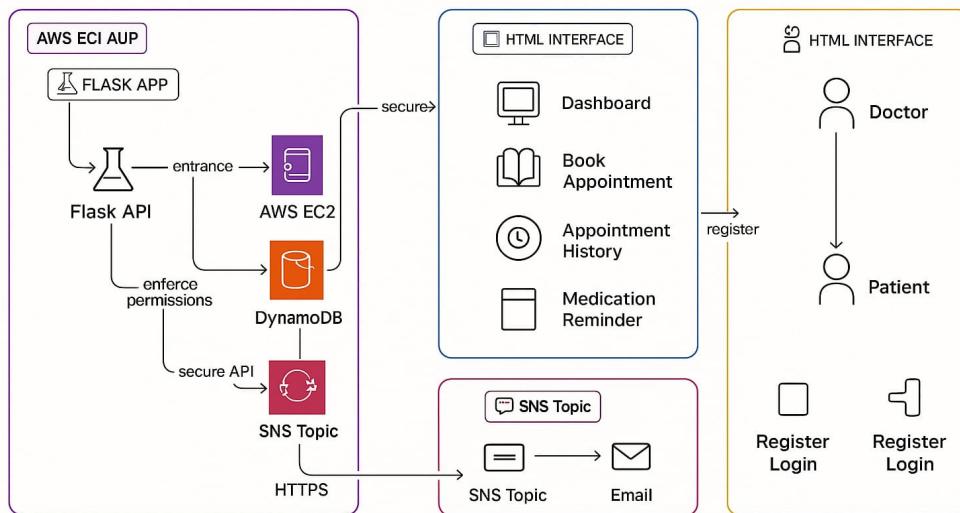
MedTrack utilizes AWS IAM to manage user permissions and ensure secure access to the system. For instance, when a new patient registers, an IAM user is created with specific roles and permissions to access only the features relevant to them. Doctors have their own IAM configurations, allowing them access 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:

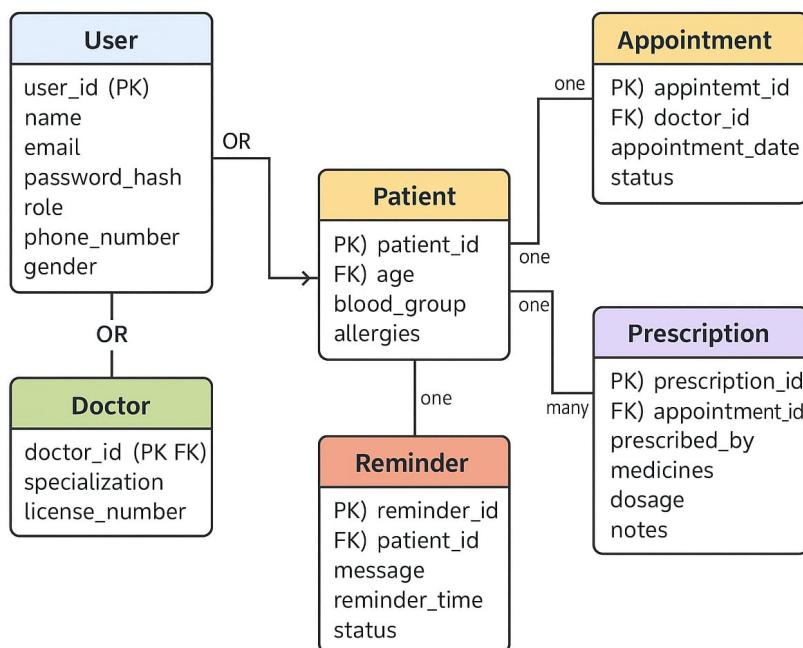
The MedTrack system provides doctors and patients with easy access to medical histories and relevant resources. For example, a doctor logs in to view a patient's medical history and upcoming appointments. They can quickly access, and update records as needed. Flask manages real-time data fetching from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

Medtack Infrastructure



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 WORK FLOW

1.AWS Account Setup and Login:

Activity1.1: Set up an AWS account if not already done.

Activity1.2: Log into the AWS Management Console

2.DynamoDB Database Creation and Setup:

Activity2.1: Create a DynamoDB Table.

Activity2.2: Configure Attributes for User Data and Book Requests.

3.SNS Notification Setup:

Activity3.1: Create SNS Topics for book request notifications.

Activity3.2: Subscribe users and library staff to SNS email notifications.

4.Backend Development and Application Setup:

Activity4.1: Develop the Backend Using Flask.

Activity4.2: Integrate AWS Services Using boto3.

5.IAM Role Setup:

Activity5.1:CreateIAMRole

Activity5.2:AttachPolicies

6.EC2 Instance Setup

Activity6.1:LaunchanEC2instancetohosttheFlaskapplication.

Activity6.2:ConfiguresecuritygroupsforHTTP, andSSHaccess.

7.Deployment on EC2

Activity7.1:UploadFlaskFiles

Activity7.2:RuntheFlaskApp

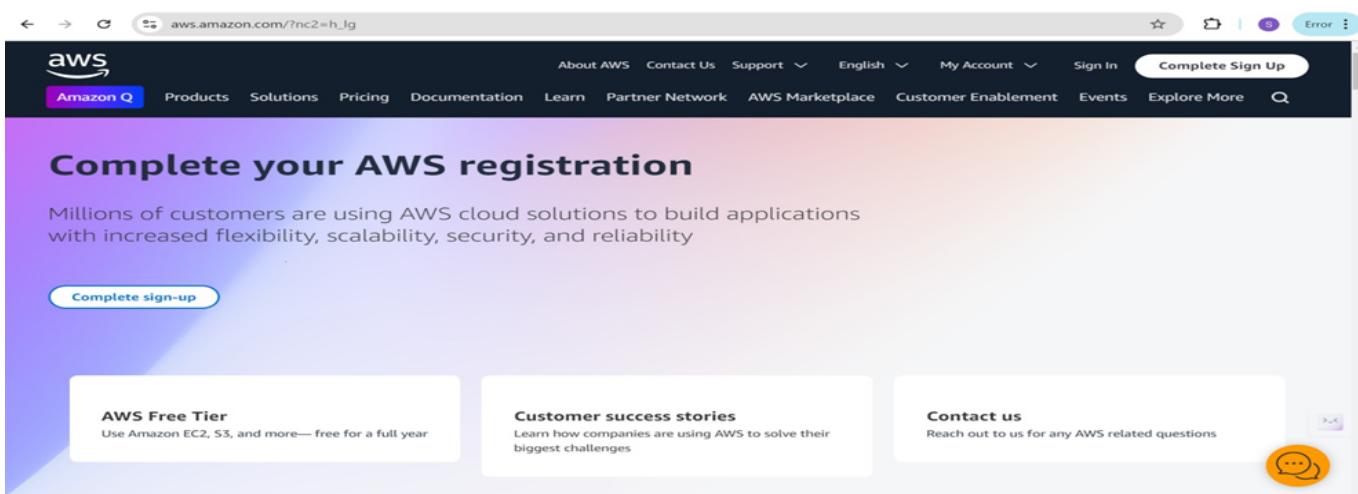
8. Testing and Deployment

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

Milestone 1: AWS Account Setup and Login

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

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



- **Activity 1.2: Log in to the AWS Management Console**

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

Sign in

Root user
 Account owner that performs tasks requiring unrestricted access. [Learn more](#)

IAM user
 User within an account that performs daily tasks. [Learn more](#)

Root user email address

username@example.com

Next

By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our [Cookie Notice](#) for more information.

AI Use Case Explorer

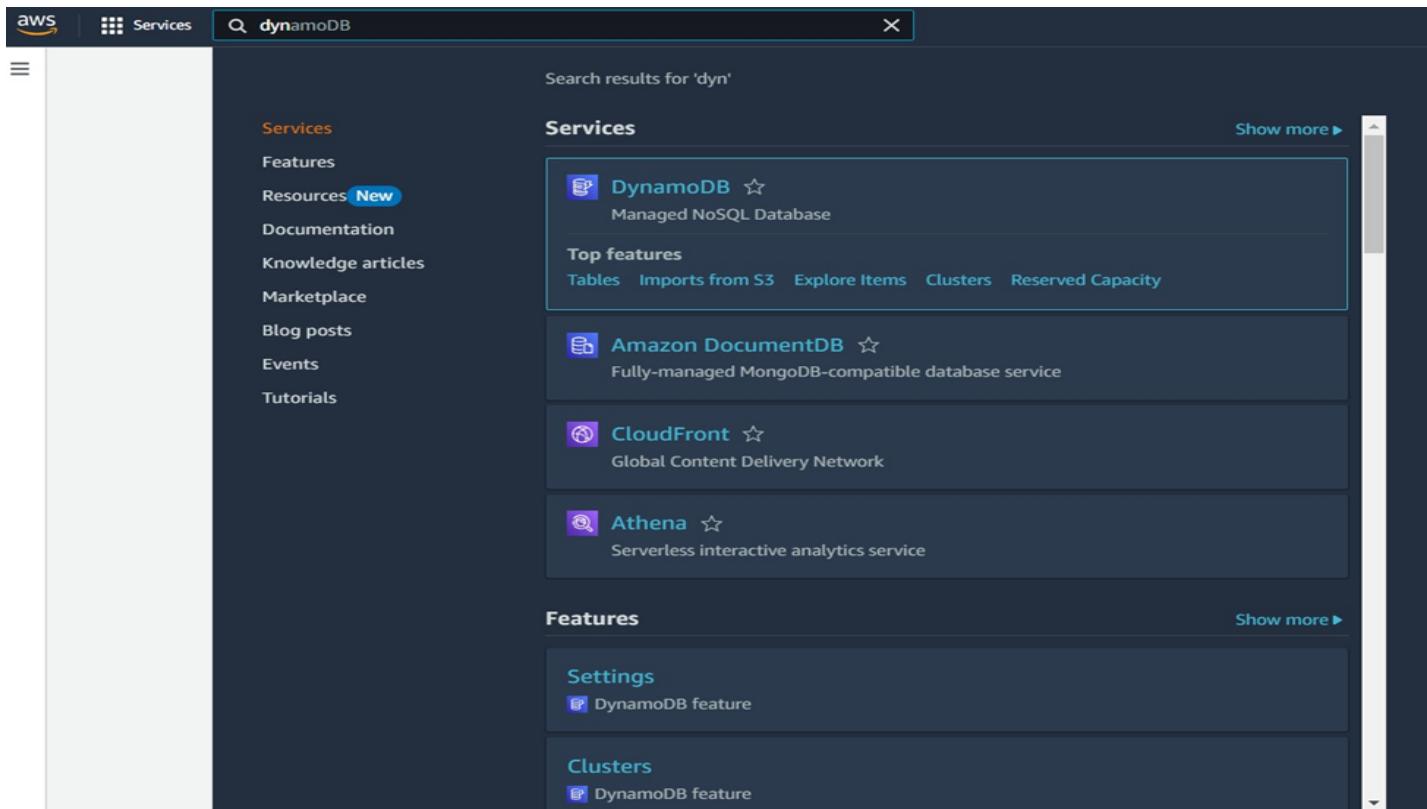
Discover AI use cases, customer success stories, and expert-curated implementation plans

[Explore now >](#)

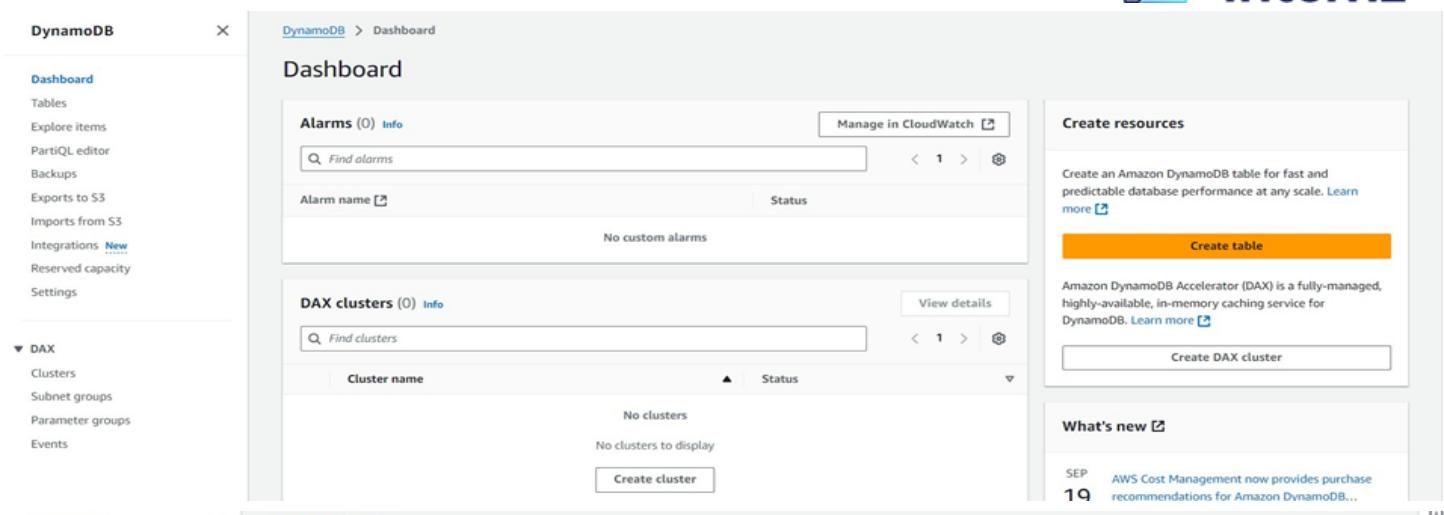
Milestone 2: DynamoDB Database Creation and Setup

- **Activity 2.1: Navigate to the DynamoDB**

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



The screenshot shows the AWS search interface. The search bar at the top contains the query "dynamodb". The left sidebar has a "Services" section with links to Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main content area displays search results for "dyn". Under the "Services" heading, there is a card for "DynamoDB" (Managed NoSQL Database) with a star icon and a link to "Tables". Below it are cards for "Amazon DocumentDB" (Fully-managed MongoDB-compatible database service), "CloudFront" (Global Content Delivery Network), and "Athena" (Serverless interactive analytics service). Under the "Features" heading, there is a card for "Settings" (DynamoDB feature) and another for "Clusters" (DynamoDB feature).



DynamoDB

Dashboard

Alarms (0) Info

DAX clusters (0) Info

Create resources

What's new

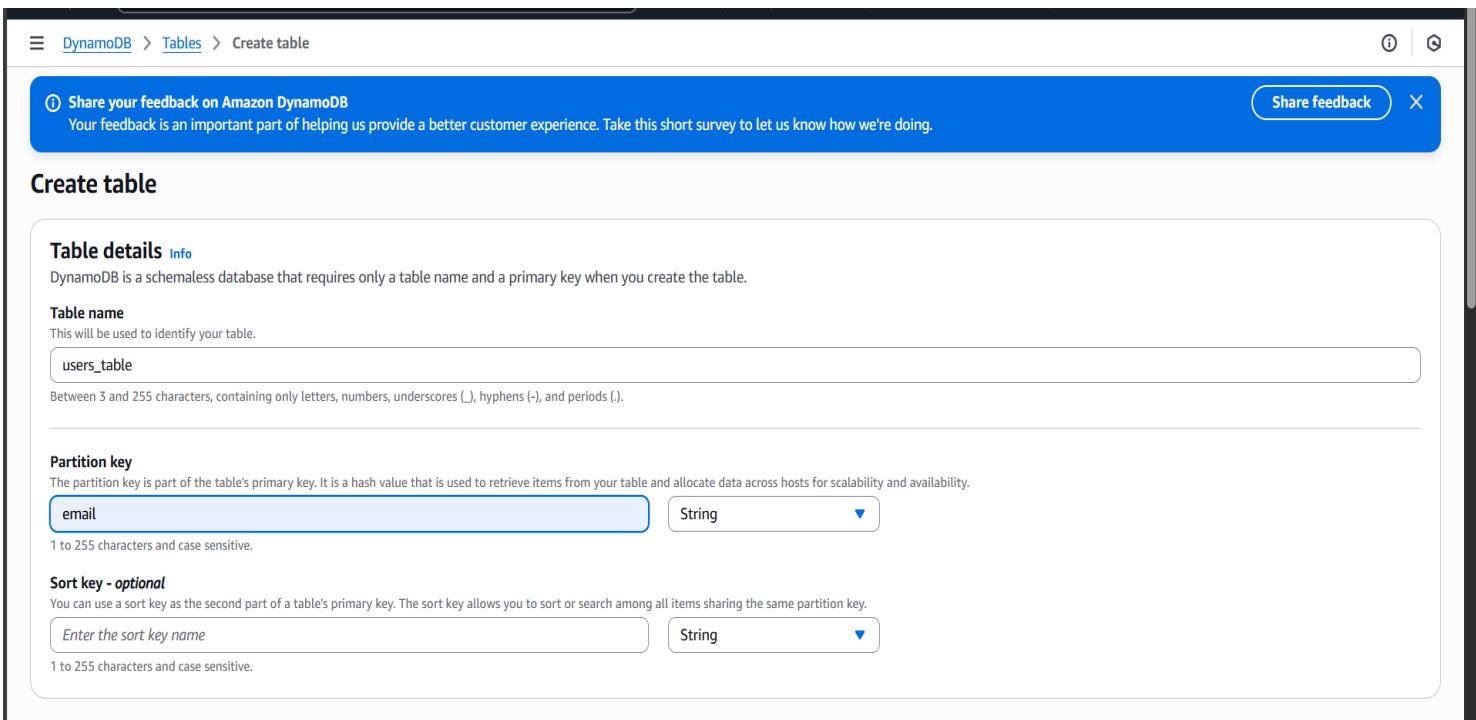
DynamoDB

Tables

Tables (0) Info

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

- 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.

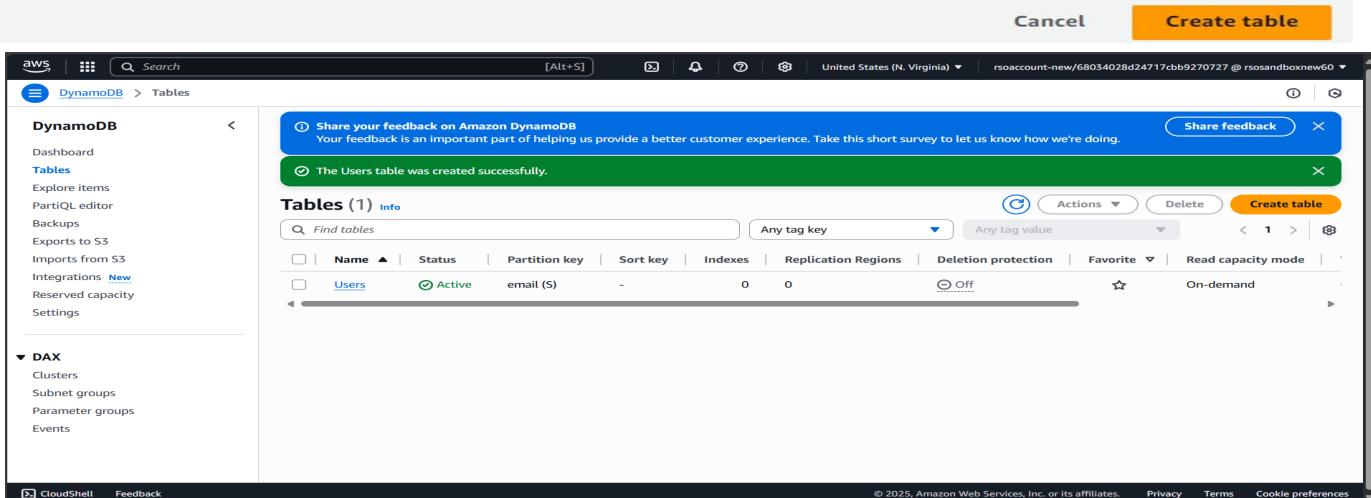
Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based 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.

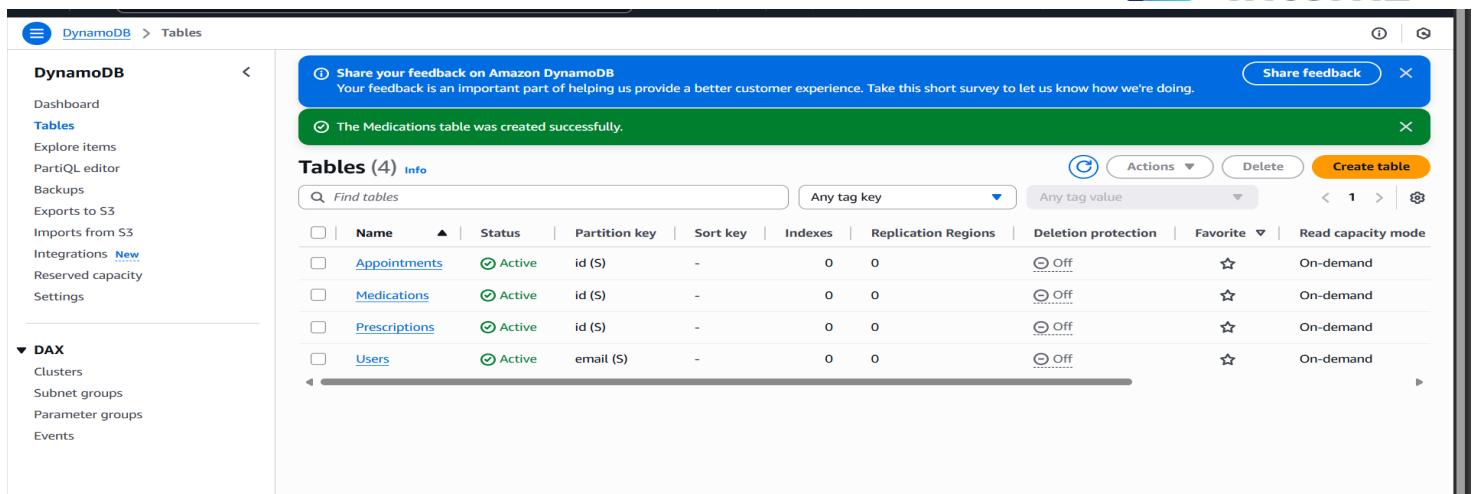
Add new tag

You can add 50 more tags.



The screenshot shows the AWS DynamoDB Tables page. On the left, there's a navigation sidebar with 'DynamoDB' selected. The main area displays a table titled 'Tables (1)'. A success message at the top says 'The Users table was created successfully.' The table has columns: Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read capacity mode. One row is shown for the 'Users' table, which is Active, uses 'email (\$)' as the partition key, and has 'Off' deletion protection. At the top right, there are 'Cancel' and 'Create table' buttons.

- Follow the same steps to create an Appointments,Prescriptions,Medications tablewith ID as the primary key for book requests data.



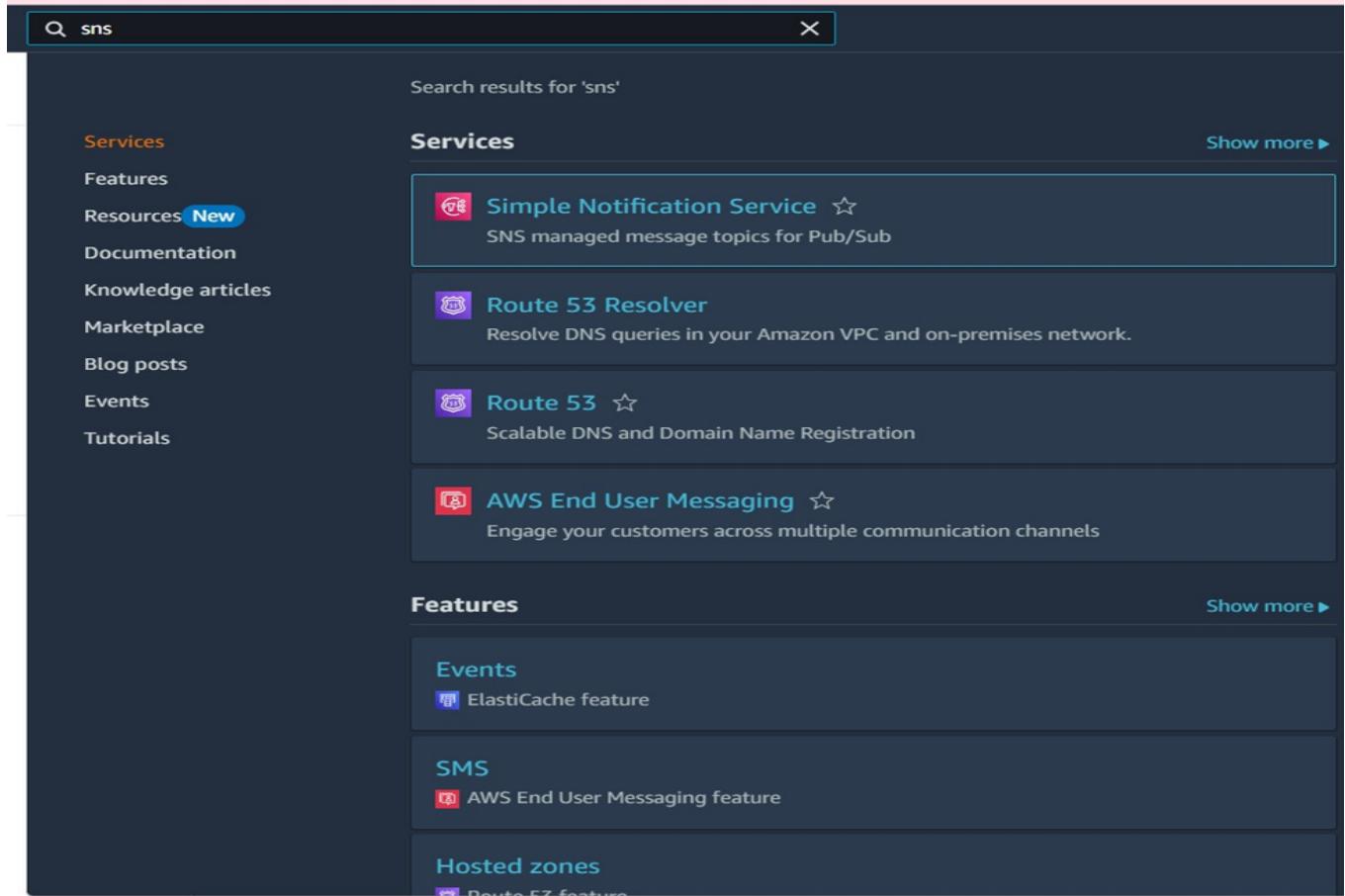
The screenshot shows the AWS DynamoDB Tables page. On the left, there's a sidebar with 'DynamoDB' selected. The main area displays a success message: 'The Medications table was created successfully.' Below this is a table titled 'Tables (4) Info' with the following data:

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity mode
Appointments	Active	id (S)	-	0	0	Off	☆	On-demand
Medications	Active	id (S)	-	0	0	Off	☆	On-demand
Prescriptions	Active	id (S)	-	0	0	Off	☆	On-demand
Users	Active	email (S)	-	0	0	Off	☆	On-demand

Milestone 3: SNS Notification Setup

● Activity 3.1: Create SNS topics for sending email notifications to users and library staff.

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



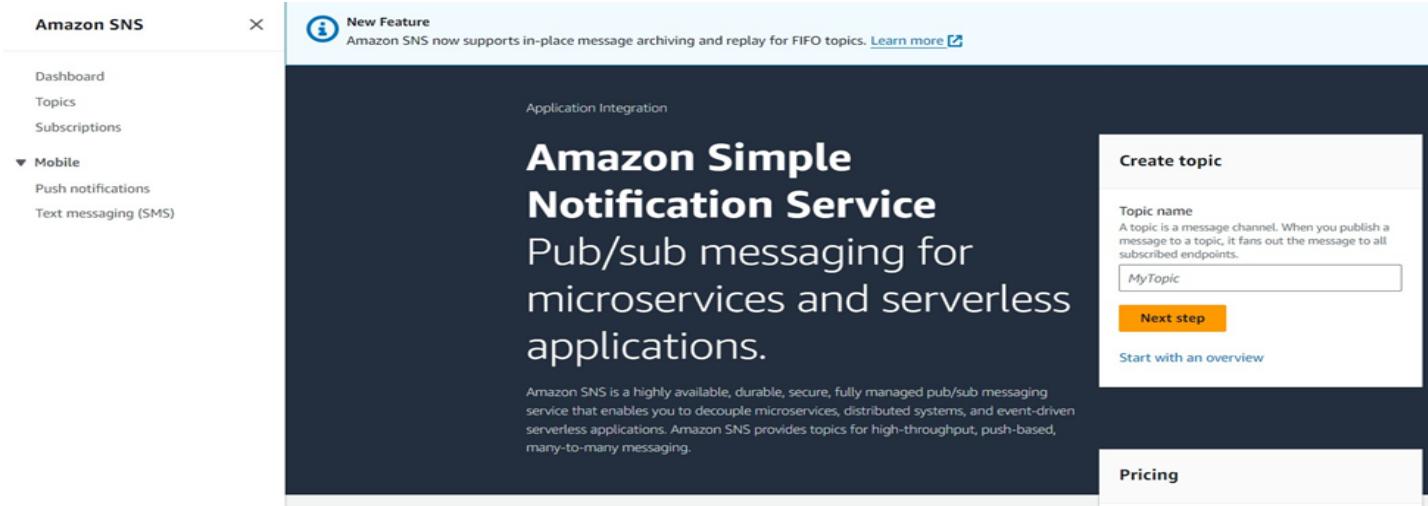
The screenshot shows the AWS search results for 'sns'. The search bar at the top contains 'sns'. The results are categorized under 'Services' and 'Features'.

Services

- Simple Notification Service** ☆
SNS managed message topics for Pub/Sub
- Route 53 Resolver**
Resolve DNS queries in your Amazon VPC and on-premises network.
- Route 53** ☆
Scalable DNS and Domain Name Registration
- AWS End User Messaging** ☆
Engage your customers across multiple communication channels

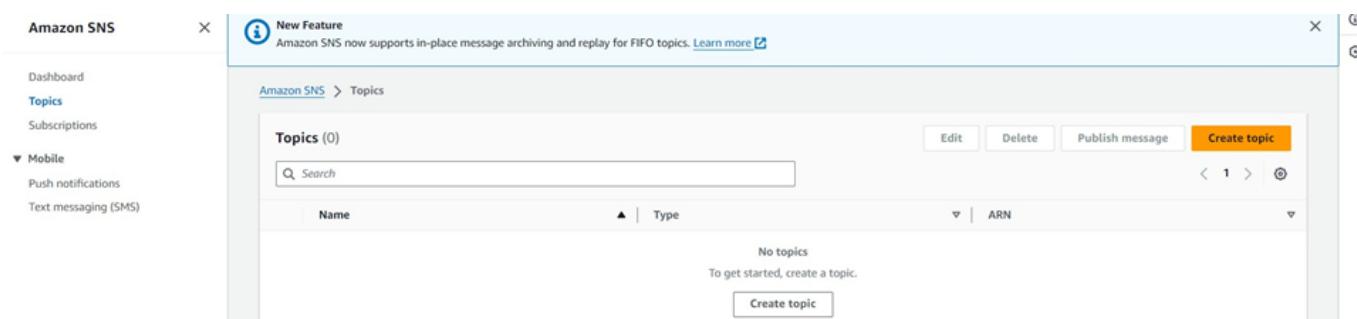
Features

- Events**
ElastiCache feature
- SMS**
AWS End User Messaging feature
- Hosted zones**
Route 53 feature



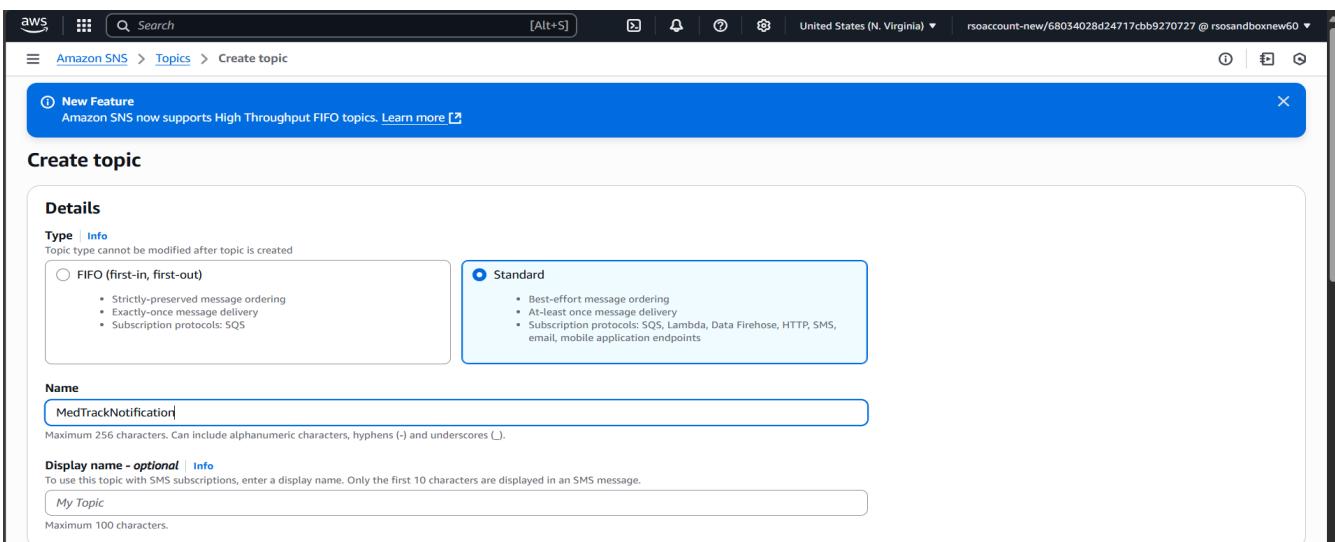
The screenshot shows the Amazon SNS dashboard. On the left, a sidebar menu includes: Dashboard, Topics, Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and a New Feature announcement about FIFO topics. The main content area features a large heading "Amazon Simple Notification Service" with the subtext "Pub/sub messaging for microservices and serverless applications." Below this is a detailed description of Amazon SNS and a "Create topic" form. The "Create topic" form has a "Topic name" field containing "MyTopic", a "Next step" button, and a "Start with an overview" link.

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



The screenshot shows the "Topics" page within the Amazon SNS service. The sidebar is identical to the dashboard. The main area displays a table titled "Topics (0)" with columns for Name, Type, and ARN. A search bar and buttons for Edit, Delete, Publish message, and Create topic are at the top right. A message at the bottom says "No topics To get started, create a topic." with a "Create topic" button.

- Choose Standardtype for general notification use cases and Click on Create Topic.



The screenshot shows the "Create topic" page in the AWS Management Console. The top navigation bar includes the AWS logo, search bar, and various icons. The main content area has a blue header "Create topic". Under "Details", there are two options: "FIFO (first-in, first-out)" and "Standard". The "Standard" option is selected and highlighted with a blue border. It lists features: Best-effort message ordering, At-least once message delivery, and Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints. Below this are fields for "Name" (MedTrackNotification) and "Display name - optional" (My Topic). Both fields have character limits and notes about allowed 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**.

AWS |  Search [Alt+S] |    United States (N. Virginia) ▾ rsoaccount-new/68034028d24717ccb9270727 @ rsosandboxnew60 ▾

Amazon SNS > Topics > MedTrackNotification

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

Details

Name	MedTrackNotification	Display name	-
ARN	arn:aws:sns:us-east-1:241533142623:MedTrackNotification	Topic owner	241533142623
Type	Standard		

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

[Subscriptions \(0\)](#) [Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#) [Create subscription](#)

- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

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

Create subscription

Details

Topic ARN
arn:aws:sns:us-east-1:241533142623:MedTrackNotification

Protocol
The type of endpoint to subscribe
Email

Endpoint
An email address that can receive notifications from Amazon SNS.
sreeusha790@gmail.com

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

► **Subscription filter policy - optional** [Info](#)

Amazon SNS

Subscription: 5b6ddd93-7912-437c-ba65-272244d944f5

Subscription: 5b6ddd93-7912-437c-ba65-272244d944f5

Details

ARN
arn:aws:sns:us-east-1:241533142623:MedTrackNotification:5b6ddd93-7912-437c-ba65-272244d944f5

Endpoint
sreeusha790@gmail.com

Topic
MedTrackNotification

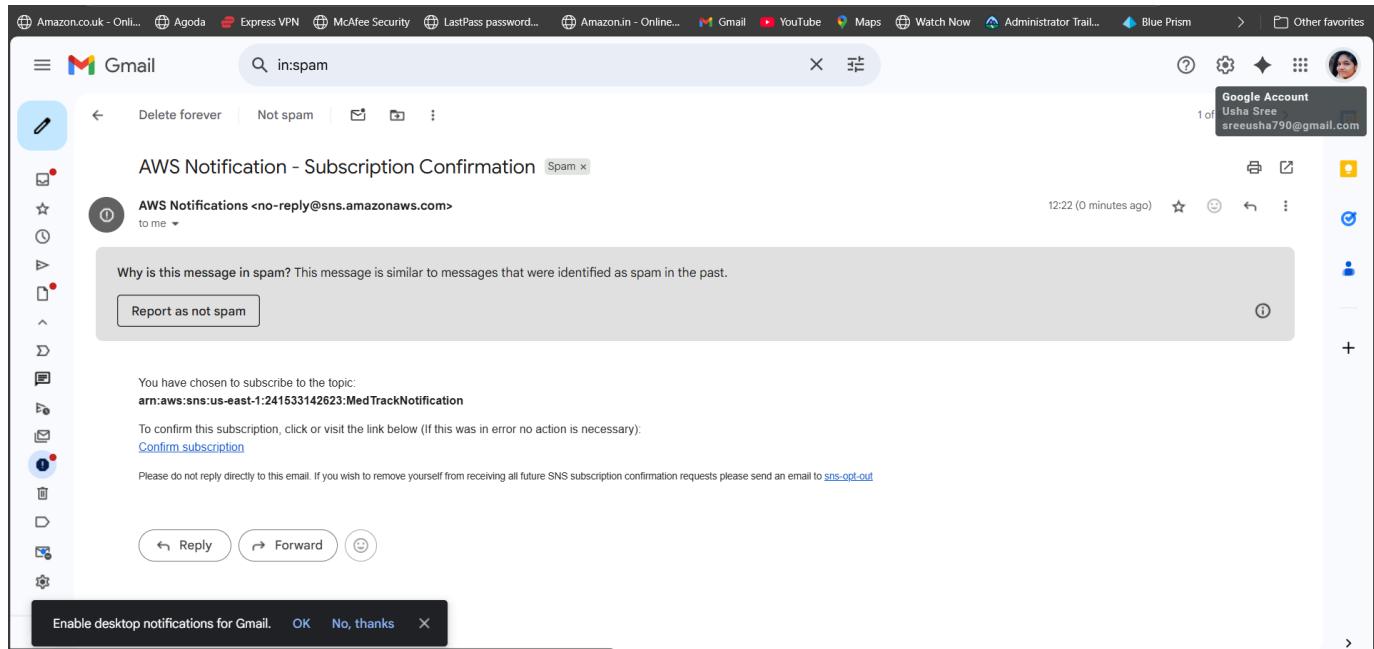
Subscription Principal
arn:aws:iam::241533142623:role/rsoaccount-new

Status
Pending confirmation

Protocol
EMAIL

Subscription filter policy | Redrive policy (dead-letter queue)

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



AWS Notification - Subscription Confirmation Spam

AWS Notifications <no-reply@sns.amazonaws.com>
to me 12:22 (0 minutes ago)

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

[Report as not spam](#)

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:241533142623:MedTrackNotification

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

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

Enable desktop notifications for Gmail. [OK](#) [No, thanks](#) [X](#)



Simple Notification Service

Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

arn:aws:sns:us-east-1:241533142623:MedTrackNotification:5b6ddd93-7912-437c-ba65-272244d944f5

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

Amazon SNS
Subscription: 5b6ddd93-7912-437c-ba65-272244d944f5
Edit
Delete

Details

ARN arn:aws:sns:us-east-1:241533142623:MedTrackNotification:5b6ddd93-7912-437c-ba65-272244d944f5	Status Confirmed
Endpoint sreeusha790@gmail.com	Protocol EMAIL
Topic MedTrackNotification	
Subscription Principal arn:aws:iam::241533142623:role/rsoaccount-new	

[Subscription filter policy](#) | [Redrive policy \(dead-letter queue\)](#)

Subscription filter policy Info

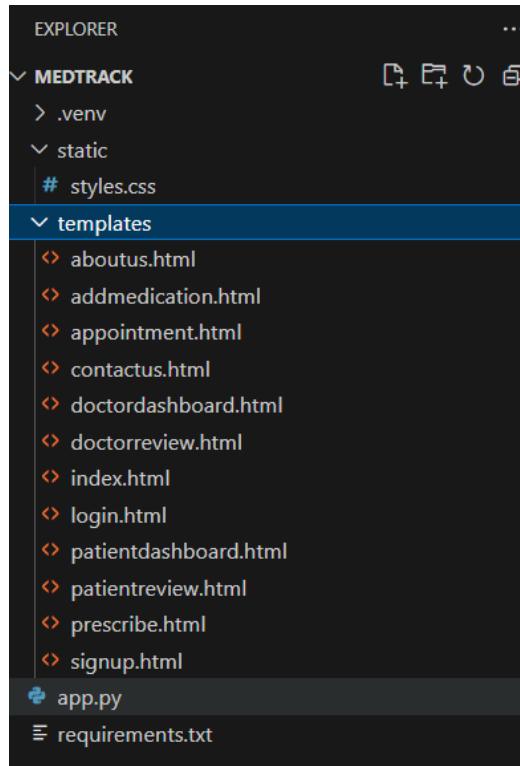
This policy filters the messages that a subscriber receives.

No filter policy configured for this subscription.
To apply a filter policy, edit this subscription.

Milestone 4: Backend Development and Application Setup

- **Activity 4.1: Develop the backend using Flask**

- File Explorer Structure



Description: set up the INSTANT LIBRARY project with an app.py file, a static/ folder for assets, and a templates/ directory containing all required HTML pages like index, login, signup, subject-specific pages (e.g., Doctordashboard.html, patientdashboard.html), and utility pages.

Description of the code :

- Flask App Initialization

```
app.py > ...
1  from flask import Flask, render_template, request, redirect, url_for, session, flash
2  import boto3
3  from boto3.dynamodb.conditions import Key
4  import os
5  import uuid
6  from datetime import datetime
7  import smtplib
8  from email.mime.text import MIMEText
9
10 app = Flask(__name__)
11 app.secret_key = 'sreeusha790'
12
13 USE_DYNAMODB = False # Set to True when deploying on AWS
14 AWS_REGION = 'us-east-1'
```

Description: This project begins by importing all essential libraries required for MedTrack. Flask utilities are used to handle routing, sessions, and template rendering. **Boto3** is used to interact with AWS **DynamoDB** for storing user, appointment, and prescription data. **SMTP** and the **email.mime** modules are utilized for sending custom email notifications to users. Additionally, libraries like `uuid` and `datetime` assist in generating unique IDs and managing date/time operations.

```
app = Flask(__name__)
```

Description: initialize the Flask application instance using `Flask(__name__)` to start building the web app.

- Dynamodb Setup:

```

USE_DYNAMODB = True # Set to True when deploying on AWS
AWS_REGION = 'us-east-1'

if USE_DYNAMODB:
    dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION)
    users_table = dynamodb.Table('Users')
    appointments_table = dynamodb.Table('Appointments')
    prescriptions_table = dynamodb.Table('Prescriptions')
    medications_table = dynamodb.Table('Medications')

```

Description: Initialize the **DynamoDB** resource for the **us-east-1** region and set up access to the Users, Appointments, Prescriptions, and Medications tables. These tables are used to securely store user information, appointment bookings, doctor-issued prescriptions, and patient medication details for the MedTrack healthcare application.

● SNS Connection:

```

AWS_REGION = "ap-south-1"
SNS_TOPIC_ARN = "arn:aws:sns:ap-south-1:123456789012:MedTrackNotifications"
sns = boto3.client('sns', region_name=AWS_REGION)
def send sns_email(subject, message):
    try:
        sns.publish(
            TopicArn=SNS_TOPIC_ARN,
            Message=message,
            Subject=subject
        )
        print("■ SNS email sent successfully.")
    except Exception as e:
        print(f"✗ Failed to send SNS email: {e}")
def send_local_sms(phone, message):
    print(f"[SIMULATED SMS to {phone}]: {message}")
def send_local_email(to_email, subject, body):
    try:
        smtp_server = 'smtp.gmail.com'
        smtp_port = 587
        sender_email = os.getenv('SENDER_EMAIL')
        sender_password = os.getenv('SENDER_PASSWORD') # Use Gmail App Password (not regular password)
        msg = MIMEText(body)
        msg['Subject'] = subject
        msg['From'] = sender_email
        msg['To'] = to_email
        server = smtplib.SMTP(smtp_server, smtp_port)
        server.starttls()
        server.login(sender_email, sender_password)
        server.send_message(msg)
        server.quit()
        print(f"Email sent to {to_email}")
    except Exception as e:
        print(f"Email sending failed: {e}")

```

Description:

Configure **Amazon SNS** with your Topic ARN and region (e.g., us-east-1) to send medical alerts. Set up **SMTP** (like Gmail) in SENDER_EMAIL and SENDER_PASSWORD using an app password to send local email notifications for reminders and signups.

● Routes for Web Pages

- Home Route:

Description: Defines the home route / to render the **index page**, which contains the **Login** and **Sign Up** buttons for both patients and doctors.

```
@app.route('/')
def index():
    return render_template('index.html')
```

- Sign Up Route:

Description: Defines /signup route to **collect user details** (name, email, password, phone, role, and reminder frequency), **store them in DynamoDB or local DB**, and **send an SNS Welcome email** upon successful registration.

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        data = {k: request.form[k] for k in ['name', 'email', 'password', 'phone', 'role', 'reminder']}
        users = load_users()
        if data['email'] in users:
            flash("Email already exists", "error")
            return redirect(url_for('signup'))
        db['users'][data['email']] = data

        subject = ">Welcome to MedTrack!"
        message = f"Hi {data['name']},\n\nYou have successfully registered in the MedTrack app.\n\nStay healthy!"
        send sns_email(subject, message)

        flash("Signup successful! Now login.", "success")
        return redirect(url_for('login'))
    return render_template('signup.html')
```

- Login Route (GET/POST):

Description: Defines /login route to validate login credentials against the **user database**, initiate a session, and **redirect doctors to doctordashboard or patients to patientdashboard** based on their role.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email, password = request.form['email'], request.form['password']
        user = db['users'].get(email)
        if user and user['password'] == password:
            session.update({"email": email, "role": user['role'], "name": user['name'], "phone": user.get('phone', '')})
            return redirect(url_for(f"{user['role']}dashboard"))
        flash("Invalid credentials", "error")
    return render_template('login.html')
```

- Doctor Dashboard Route:

Description: Defines /doctordashboard route where doctors can **view, approve, or reject appointments, and prescribe medications**. Also displays **alerts for missed medications** by patients.

```
@app.route('/doctordashboard', methods=['GET', 'POST'])
def doctordashboard():
    if session.get('role') != 'doctor':
        return redirect(url_for('login'))

    name = session['name']
    doctor_appts = [a for a in db['appointments'] if a['doctor'].strip().lower() == name.strip().lower()]

    now = datetime.now()
    for appt in doctor_appts:
        appt_time_str = f'{appt["date"]}{appt["time"]}'
        try:
            appt_time = datetime.strptime(appt_time_str, "%Y-%m-%d %H:%M")
        except ValueError:
            continue
        if appt['status'] == 'Pending' and now > appt_time:
            appt['status'] = 'Missed'

    alerts = []
    for m in db['medications']:
        if m['status'].lower() == 'missed':
            alerts.append(f"Patient {m['patient']} missed dose of {m['medicine']} at {m['time']}")

    if request.method == 'POST':
        decision = request.form['decision']
        appt_id = request.form['appointment_id']
        for a in db['appointments']:
            if a['id'] == appt_id:
                a['status'] = decision
                break
        flash(f"Appointment {decision.lower()}ed.", "info")

    return render_template('doctordashboard.html',
                           name=name,
                           email=session['email'],
                           appointments=doctor_appts,
```

- **Patient Dashboard Route:**

Description: Defines /patientdashboard route where patients can **view appointments, prescriptions, and medication schedules, and mark doses as Taken, Missed, or Pending**

```

@app.route('/patientdashboard')
def patientdashboard():
    if session.get('role') != 'patient':
        return redirect(url_for('login'))

    name = session.get('name')
    email = session.get('email')

    user_appointments = [appt for appt in db['appointments'] if appt['patient'] == name]

    # Convert date + time to datetime object and check if it's passed
    now = datetime.now()
    for appt in user_appointments:
        appt_time_str = f"{appt['date']} {appt['time']}"
        try:
            appt_time = datetime.strptime(appt_time_str, "%Y-%m-%d %H:%M")
        except ValueError:
            appt_time = now # fallback if formatting is bad

        if appt['status'] == 'Pending' and now > appt_time:
            appt['status'] = 'Missed'

    user_medications = [med for med in db['medications'] if med['patient'] == name]
    user_prescriptions = [p for p in load_prescriptions() if p['patient'] == name]

    alerts = []
    for med in user_medications:
        med_time_str = f"{med.get('date', now.strftime('%Y-%m-%d'))} {med['time'].split()[0]}"
        try:
            med_time = datetime.strptime(med_time_str, "%Y-%m-%d %H:%M")
        except:
            continue
        if med['status'] == 'Pending' and now > med_time:
            med['status'] = 'Missed'
    
```

• Appointment Booking Route:

Description: Defines /appointment route to **allow patients to book appointments** by submitting required details. Bookings are saved in **DynamoDB or local DB** and shown on dashboards.

```

@app.route('/appointment', methods=['GET', 'POST'])
def appointment():
    if request.method == 'POST':
        f = request.form
        appt = {
            'id': str(uuid.uuid4()), 'patient': session.get('name'), 'name': f['name'], 'gender': f['gender'],
            'phone': f['phone'], 'age': f['age'], 'email': session.get('email'),
            'department': f['department'], 'problem': f['problem'], 'doctor': f['doctor'],
            'status': 'Pending', 'date': f['date'], 'time': f['time']
        }
        save_appointment(appt)
        flash("Appointment booked successfully!", "success")
    return redirect(url_for('appointment'))
    return render_template('appointment.html')

```

● Prescription Route:

Description: Defines /prescribe/<appt_id> route for doctors to **add prescriptions linked to a specific appointment**. Medications are saved and reminders are scheduled.

```

3 @app.route('/prescribe/<appt_id>', methods=['GET', 'POST'])
4 def prescribe(appt_id):
5     if request.method == 'POST':
6         f = request.form
7         time = f"{f['time']} {f['ampm']}" if 'ampm' in f else f['time']
8         presc = {
9             'id': str(uuid.uuid4()), 'appointment_id': appt_id, 'medicine': f['medicine'],
0             'dosage': f['dosage'], 'days_left': f['days_left'], 'time': time,
1             'status': 'Pending', 'patient': f['patient']
2         }
3         save_prescription(presc)
4         save_medication(presc)
5         flash("Prescription added & medication reminder set.", "success")
6         return redirect(url_for('doctordashboard'))
7     appt = next((a for a in db['appointments'] if a['id'] == appt_id), None)
8     return render_template('prescribe.html', appointment=appt)
9

```

● Add Medication Route:

Description: Defines /addmedication route for patients to **manually add a medication schedule**, which is later used for reminders.

```
@app.route('/addmedication', methods=['GET', 'POST'])
def addmedication():
    if session.get('role') != 'patient': return redirect(url_for('login'))
    if request.method == 'POST':
        f = request.form
        med = {
            'id': str(uuid.uuid4()), 'patient': session['name'], 'medicine': f['medicine'], 'date': f['date'],
            'dosage': f['dosage'], 'time': f['time'], 'days_left': f['days_left'], 'status': f.get('status', 'Pending')
        }
        save_medication(med)
        flash("Medication added successfully.", "success")
        return redirect(url_for('patientdashboard'))
    return render_template('addmedication.html')
```

- **Notification Route:**

Description: Defines /send_notifications route to **send SNS notifications** to patients for scheduled medications based on the current date and time.

```
@app.route('/send_notifications')
def send_notifications():
    now = datetime.now()
    current_time = now.strftime("%H:%M")
    today = now.strftime("%Y-%m-%d")
    reminders_sent = 0

    for med in db['medications']:
        med_time = datetime.strptime(med['time'], "%I:%M %p").strftime("%H:%M")
        med_date = med.get('date', today) # Optional date field
        if med['status'].lower() == 'pending' and med_time == current_time and med_date == today:
            # Send SNS email reminder
            subject = f"🔴 Medication Reminder: {med['medicine']}"
            message = f"Dear {med['patient']},\n\nThis is your MedTrack reminder to take {med['medicine']} at {med['time']} today.\n\nStay healthy!"
            send sns_email(subject, message)

            med['status'] = 'Sent' # Prevent duplicate notifications
            reminders_sent += 1

    return f"{reminders_sent} reminder(s) sent."
```

- **Update Medication Status Route:**

Description: Defines /update_medication_status route to **allow patients to update the status** of their medications—Taken, Missed, or Pending.

```

@app.route('/update_medication_status', methods=['POST'])
def update_medication_status():

    if session.get('role') != 'patient':
        return redirect(url_for('login'))

    for med in db['medications']:
        form_key = f"status_{med['id']}"
        if form_key in request.form:
            new_status = request.form[form_key]
            current_time = datetime.now().strftime("%H:%M")
            if new_status == "Not Yet" and current_time > med['time']:
                med['status'] = "Missed"
            else:
                med['status'] = "Completed" if new_status == "Taken" else "Pending"

    flash("Medication status updated.", "success")
    return redirect(url_for('patientdashboard'))

```

• Profile Update Route:

Description: Defines /update-profile route to let **users update their name, email, and phone**, which is reflected in the session and user table.

```

@app.route('/update-profile', methods=['POST'])
def update_profile():
    if 'email' not in session or 'role' not in session:
        return redirect(url_for('login'))

    email = session['email']
    role = session['role']

    # Support for local or DynamoDB users
    users = load_users()

    if email not in users:
        flash("User not found", "error")
        return redirect(url_for(f'{role}dashboard'))

    user = users[email]
    user['name'] = request.form['name']
    user['email'] = request.form['email']
    user['phone'] = request.form['phone']

    # Update session values
    session['name'] = user['name']
    session['email'] = user['email']
    session['phone'] = user['phone']

    if not USE_DYNAMODB:
        db['users'][email] = user
    else:
        users_table.put_item(Item=user)

    flash("Profile updated successfully.", "success")
    return redirect(url_for(f'{role}dashboard'))

```

- **Static Info Routes:**

/aboutus – Displays About Us page.

/contactus – Sends a confirmation message after user submits a contact form.

```
@app.route('/aboutus')
def aboutus():
    return render_template('aboutus.html')

@app.route('/contactus', methods=['GET', 'POST'])
def contactus():
    if request.method == 'POST':
        flash("Message sent successfully. Thank you for contacting MedTrack!", 'success')
        return redirect(url_for('contactus'))
    return render_template('contactus.html')
```

- **Logout Route:**

Description: Defines /logout to **clear session data** and safely redirect the user to the login page.

```
@app.route('/logout')
def logout():
    session.clear()
    flash("You have been logged out.", "info")
    return redirect(url_for('login'))
```

Deployment Code:

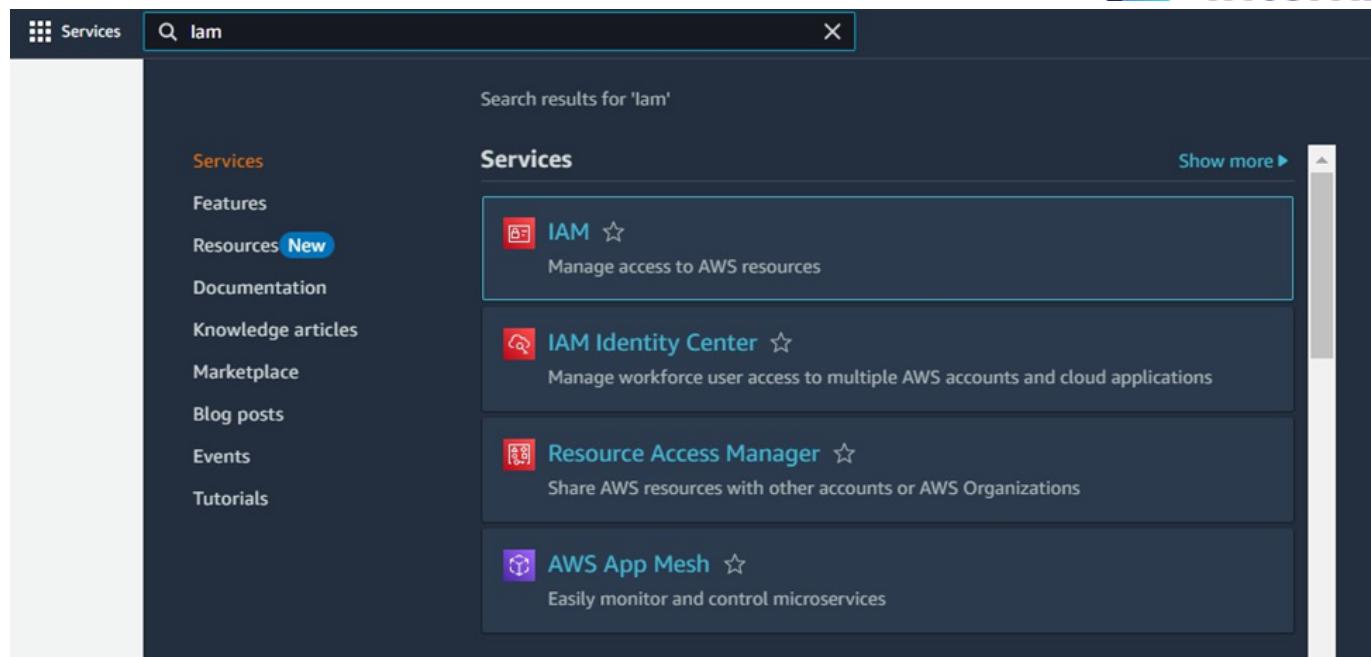
```
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Description: start the Flask server to listen on all network interfaces (0.0.0.0) at port 80 with debug mode enabled for development and testing.

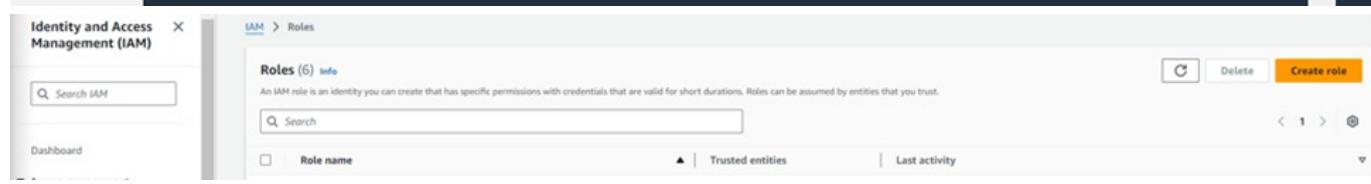
Milestone 5: IAM Role Setup

- **Activity 5.1: Create IAM Role.**

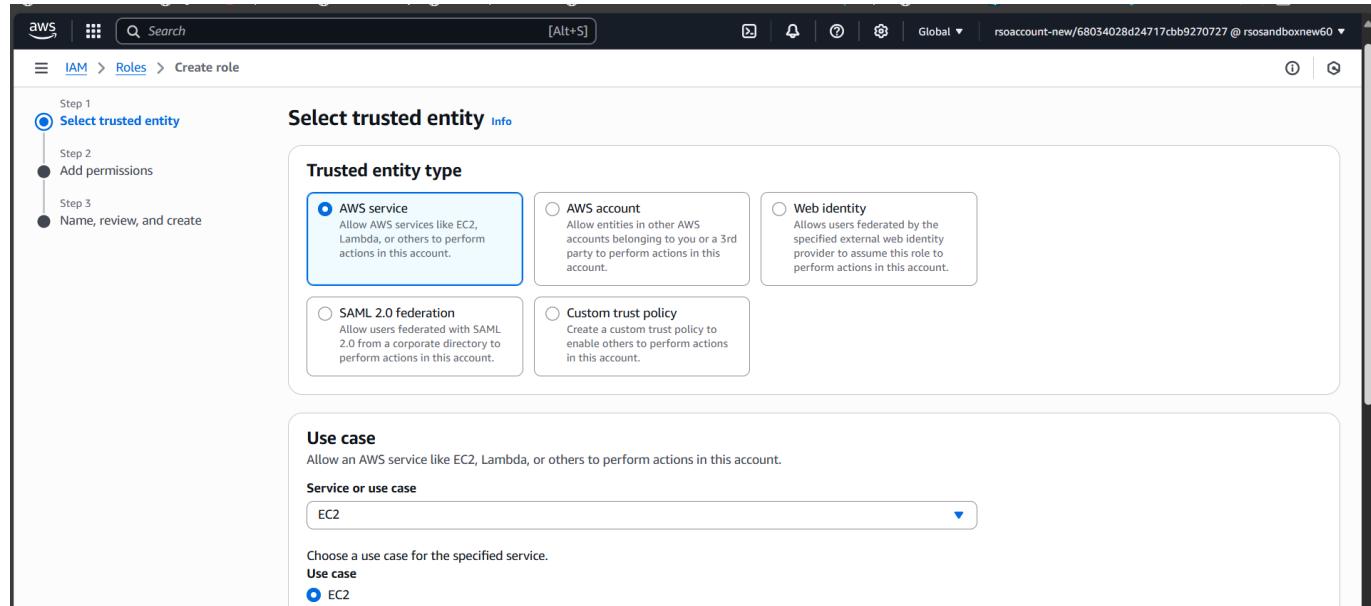
- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.



The screenshot shows a search results page for 'Iam'. The sidebar on the left includes links for Services, Features, Resources (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 thumbnail icon, a name, and a brief description.



This screenshot shows the 'Roles' section of the AWS IAM console. It lists six existing roles. The interface includes a search bar, a 'Create role' button, and filters for 'Role name', 'Trusted entities', and 'Last activity'.



This screenshot shows the first step of creating a new IAM role, titled 'Select trusted entity'. It includes a navigation bar with steps: Step 1 (Select trusted entity), Step 2 (Add permissions), and Step 3 (Name, review, and create). The 'AWS service' option is selected under 'Trusted entity type'. Other options include 'AWS account', 'Web identity', 'SAML 2.0 federation', and 'Custom trust policy'. A 'Use case' section allows selecting a service like EC2, with 'EC2' currently chosen. A note at the bottom states: 'Allow EC2 instances to call AWS services on your behalf'.

● Activity 5.2: Attach Policies: Attach the following policies to the role:

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

aws | Search [Alt+S] | Global | rsoaccount-new/68034028d24717ccb9270727 @ rsosandboxnew60 ▾

☰ IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions Info

Permissions policies (2/1063) Info

Choose one or more policies to attach to your new role.

Filter by Type All types 5 matches

Policy name	Type	Description
<input checked="" type="checkbox"/>  AmazonSNSFullAccess	AWS managed	Provides full access to Amazon SNS via...
<input type="checkbox"/>  AmazonSNSReadOnlyAccess	AWS managed	Provides read only access to Amazon S...
<input type="checkbox"/>  AmazonSNSSRole	AWS managed	Default policy for Amazon SNS service...
<input type="checkbox"/>  AWSElasticBeanstalkRoleSNS	AWS managed	(Elastic Beanstalk operations role) Allo...
<input type="checkbox"/>  AWSIoTDeviceDefenderPublishFindingsToSN...	AWS managed	Provides messages publish access to S...

▶ Set permissions boundary - optional

Cancel Previous Next

aws | Search [Alt+S] | Global | rsoaccount-new/68034028d24717ccb9270727 @ rsosandboxnew60 ▾

☰ IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions Info

Permissions policies (2/1063) Info

Choose one or more policies to attach to your new role.

Filter by Type All types 6 matches

Policy name	Type	Description
<input checked="" type="checkbox"/>  AmazonDynamoDBFullAccess	AWS managed	Provides full access to Amazon Dynam...
<input type="checkbox"/>  AmazonDynamoDBFullAccess_v2	AWS managed	Provides full access to Amazon Dynam...
<input type="checkbox"/>  AmazonDynamoDBFullAccesswithDataPipeline	AWS managed	This policy is on a deprecation path. Se...
<input type="checkbox"/>  AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only access to Amazon D...
<input type="checkbox"/>  AWSLambdaDynamoDBExecutionRole	AWS managed	Provides list and read access to Dynam...
<input type="checkbox"/>  AWSLambdaInvocation-DynamoDB	AWS managed	Provides read access to DynamoDB Str...

▶ Set permissions boundary - optional

aws | Search [Alt+S] | Global | rsoaccount-new/68034028d24717ccb9270727 @ rsosandboxnew60 ▾

☰ IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.

Maximum 64 characters. Use alphanumeric and '+,-,.,@-' characters.

Description
Add a short explanation for this role.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-!#\$%^&`~-`

Step 1: Select trusted entities

Trust policy

```

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

```

IAM > Roles

Identity and Access Management (IAM)

Roles (9) Info

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

Role name	Trusted entities	Last activity
OrganizationAccountAccessRole	Account: 058264256896	1 hour ago
rsoaccount-new	Account: 058264256896	17 minutes ago

Roles Anywhere

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

- Access AWS from your non AWS workloads**
- X.509 Standard**
- Temporary credentials**

EC2 > Instances

Instances (1/2) Info

Successfully attached EC2_MedTrack_Role to instance i-0c1e4d16fde784183

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
medtrack-server	i-0c1e4d16fde784183	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1d	ec2-3-2
medtrack-server	i-01215ffcc232615d2	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1d	ec2-54-2

i-0c1e4d16fde784183 (medtrack-server)

Details Status and alarms Monitoring Security Networking Storage Tags

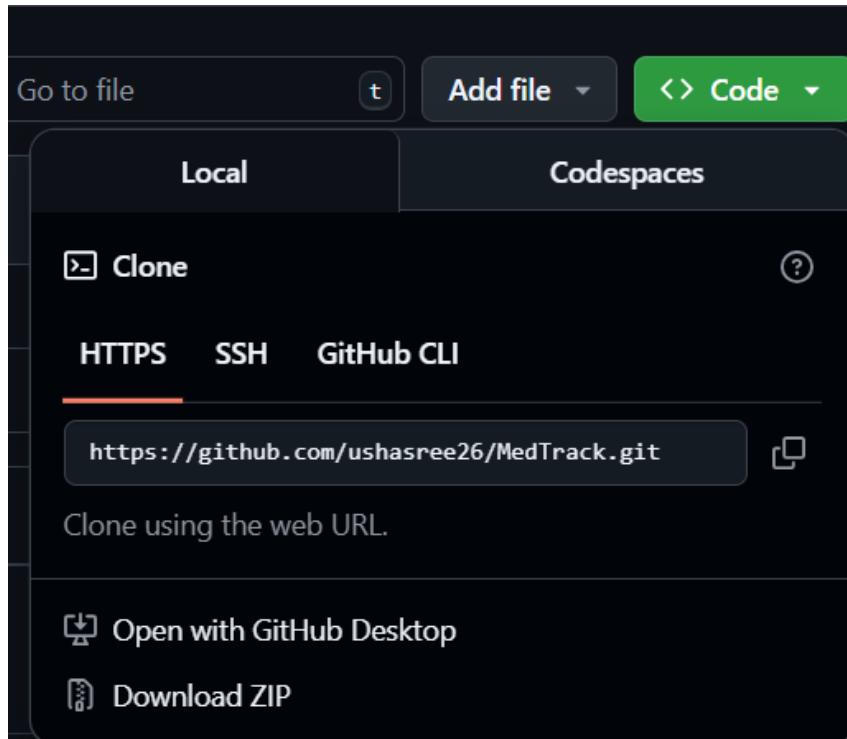
Instance summary

Instance ID i-0c1e4d16fde784183	Public IPv4 address 3.208.17.147 open address	Private IPv4 addresses 172.31.17.129
IPv6 address -	Instance state Running	Public DNS ec2-3-208-17-147.compute-1.amazonaws.com open address

Milestone 6: EC2 Instance Setup

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

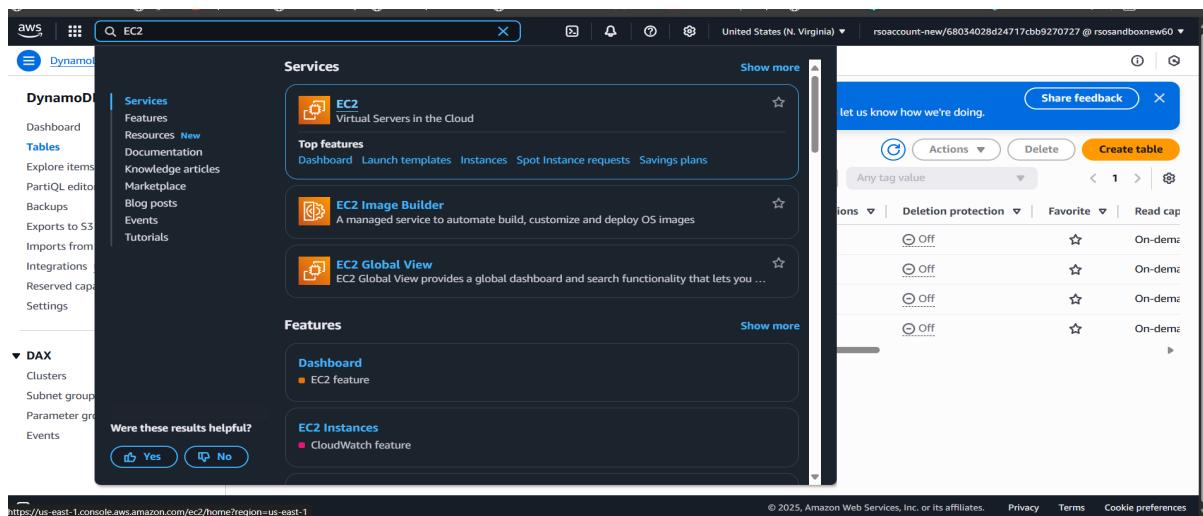
 static	Initial commit
 templates	Update statistics.html
 app.py	Update app.py



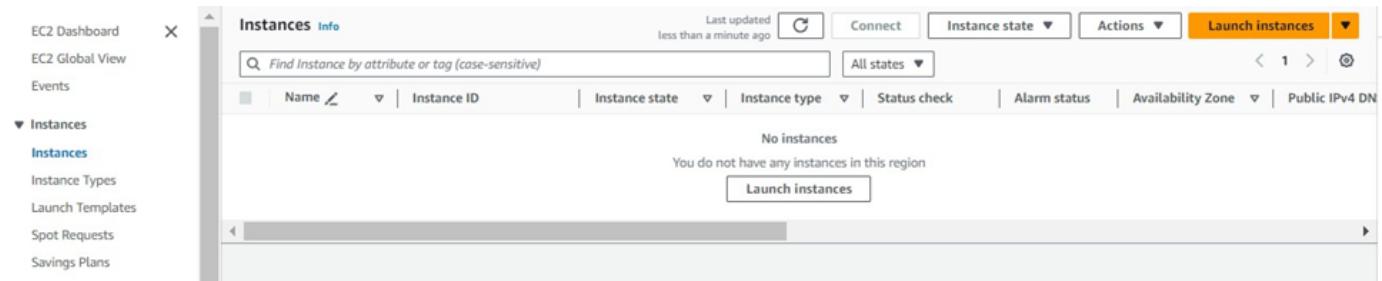
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- **Launch EC2 Instance**

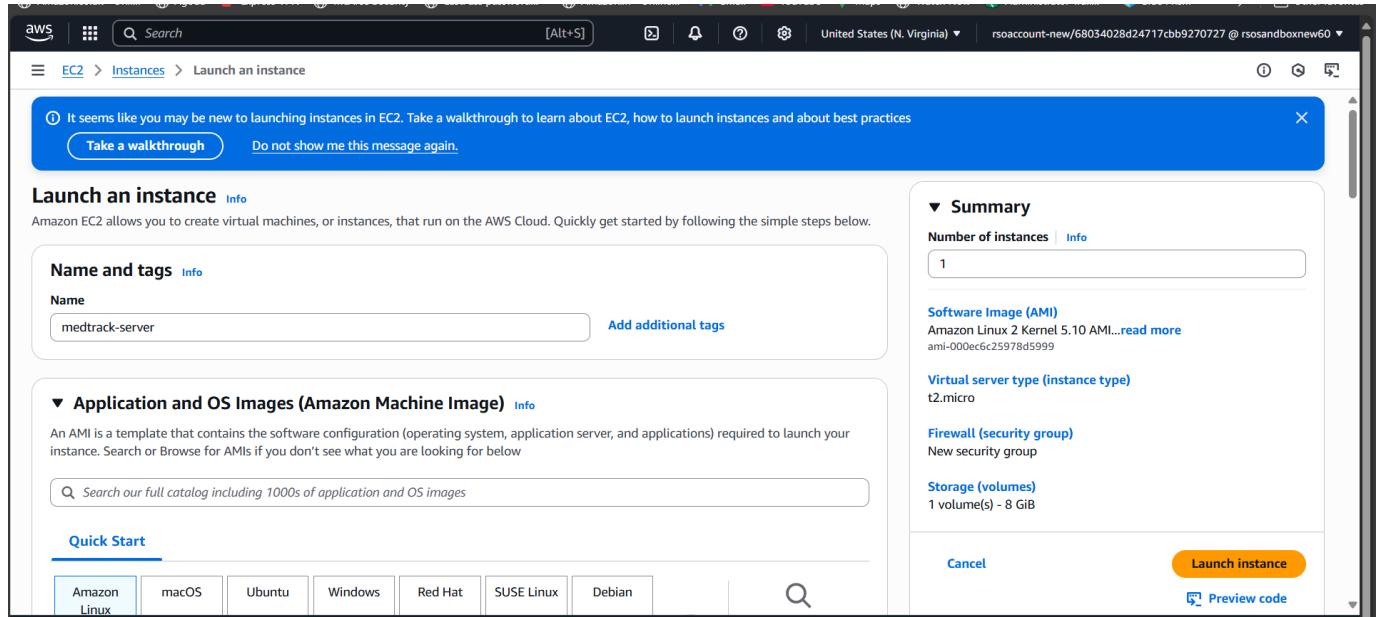
- In the AWS Console,navigate to EC2 and launcha new instance.
 -



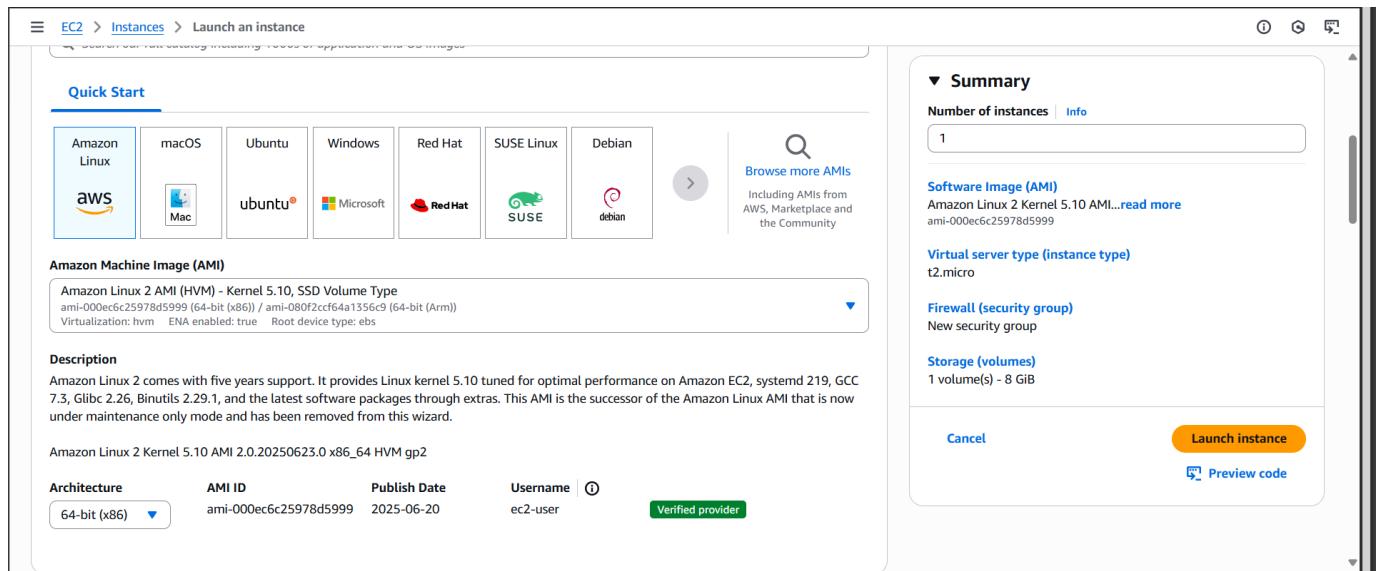
- Click on Launchinstance to launchEC2 instance



The screenshot shows the SmartBridge EC2 Instances dashboard. On the left, a sidebar lists navigation options: EC2 Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, and Savings Plans. The main area is titled "Instances Info" with a search bar and filters for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DN. A message states "No instances" and "You do not have any instances in this region". A "Launch instances" button is at the bottom.



The screenshot shows the "Launch an instance" wizard. Step 1: Set instance details. It includes fields for "Name and tags" (Name: medtrack-server, Add additional tags), "Application and OS Images (Amazon Machine Image)" (Search bar, Quick Start dropdown with Amazon Linux selected), and "Quick Start" (buttons for Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, Debian). On the right, the "Summary" section shows 1 instance, Software Image (AMI) as Amazon Linux 2 Kernel 5.10 AMI, Virtual server type as t2.micro, Firewall as New security group, and Storage (volumes) as 1 volume(s) - 8 GiB. Buttons for "Cancel", "Launch instance", and "Preview code" are at the bottom.



The screenshot shows the "Launch an instance" wizard. Step 2: Configure instance details. It includes a "Quick Start" section with a grid of operating system icons (Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, Debian) and a "Browse more AMIs" link. Below is the "Amazon Machine Image (AMI)" section for Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type. The description notes it's the successor to the Amazon Linux AMI. The "Description" section provides technical details like Virtualization: hvm, ENA enabled: true, Root device type: ebs. At the bottom, configuration fields show Architecture: 64-bit (x86), AMI ID: ami-000ec6c25978d5999, Publish Date: 2025-06-20, Username: ec2-user, and a "Verified provider" badge. The "Summary" section on the right is identical to the previous screenshot.

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as theinstancetype (free-tier eligible).
- Create and download the key pair for Server access.



EC2 > Instances > Launch an instance

▼ Instance type [Info](#) | [Get advice](#)

Instance type

t2.micro
Family: t2 1 vCPU 1 GiB Memory Current generation: true

All generations

[Compare instance types](#)

▼ 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

medtrack-server

[Create new key pair](#)

▼ Network settings [Info](#)

VPC - required [Info](#)

vpc-066d5cfe7b1b104b
172.31.0.0/16

(default)

Subnet [Info](#)

No preference

[Create new subnet](#)

Cancel

Launch instance

[Preview code](#)

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

EC2 > Instances > Launch an instance

Description - required [Info](#)

launch-wizard-1 created 2025-07-04T05:13:13.357Z

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 59.92.61.45/32)

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

Source type [Info](#): My IP
Name [Info](#):
Description - optional [Info](#): e.g. SSH for admin desktop
59.92.61.45/32

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

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

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

Summary

Number of instances [Info](#)

1

Software Image (AMI)
Amazon Linux 2 Kernel 5.10 AMI... [read more](#)

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Cancel

Launch instance

[Preview code](#)

medtrack-server.pem

EC2 > ... > Launch an instance

Success
Successfully initiated launch of instance i-001861022fbac290

Launch log

Next Steps

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

< 1 2 3 4 >

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	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 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 1-minute period. Manage detailed monitoring	Create Load Balancer Create a application, network gateway or classic Elastic Load Balancer. Create Load Balancer
Create AWS budget AWS Budgets allows 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 instances 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 unauthorized 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.

aws [Alt+S]

Search ...

United States (N. Virginia) rsoaccount-new/68034028d24717cbb9270727 @ rsosandboxnew60

EC2 > Instances

Instances (1/2) Info

Last updated 2 minutes ago [Connect](#) [Instance state](#) [Actions](#) [Launch instances](#)

Find Instance by attribute or tag (case-sensitive)

Instance state = running All states

Instance ID Name Instance state Instance type Status check Alarm status Availability Zone Public IP

i-0c1e4d16fde784183	medtrack-server	Running	t2.micro	2/2 checks passed	...	us-east-1d	ec2-3-21
i-01215ff2232615d2	medtrack-server	Running	t2.micro	2/2 checks passed	...	us-east-1d	ec2-54-1

i-0c1e4d16fde784183 (medtrack-server)

[Details](#) [Status and alarms](#) [Monitoring](#) [Security](#) [Networking](#) [Storage](#) [Tags](#)

Instance summary

Instance ID i-0c1e4d16fde784183	Public IPv4 address 3.208.17.147 open address	Private IPv4 addresses 172.31.17.129
IPv6 address -	Instance state Running	Public DNS ec2-3-208-17-147.compute-1.amazonaws.com open address

https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ConnectToInstance... © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

aws | Search [Alt+S] | United States (N. Virginia) | rsoaccount-new/68034028d24717ccb9270727 @ rsosandboxnew60

EC2 > Instances > i-0c1e4d16fde784183 > Connect to instance

Connect Info

Connect to an instance using the browser-based client.

EC2 Instance Connect Session Manager SSH client EC2 serial console

Instance ID
i-0c1e4d16fde784183 (medtrack-server)

Connect using a Public IP
Connect using a public IPv4 or IPv6 address

Connect using a Private IP
Connect using a private IP address and a VPC endpoint

Public IPv4 address
3.208.17.147

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.

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

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

Last login: Fri Jul 4 07:19:48 2025 from ec2-18-206-107-28.compute-1.amazonaws.com

```
'~\_\_###_
~~ \###\
~~ \### AL2 End of Life is 2026-06-30.
~~ \#/
~~ \/_-->
~~ .- / A newer version of Amazon Linux is available!
~~ .- / Amazon Linux 2023, GA and supported until 2028-03-15.
~~ .- / https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-17-129 ~]$ sudo su
```

i-0c1e4d16fde784183 (medtrack-server)
PublicIPs: 3.208.17.147 PrivateIPs: 172.31.17.129

Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git: On Amazon Linux 2:

```
sudo yum update-y
sudo yum install python3
git sudo pip3 install flaskboto3
```

Verify Installations:

```
flask --version git --version
```

Activity 7.2:Clone Your Flask Projectfrom GitHub

Clone your projectrepository from GitHubinto the EC2 instance using Git.

Run: 'git clone : '<https://github.com/ushasree26/MedTrack.git>'

- This will download your project to the EC2 instance.

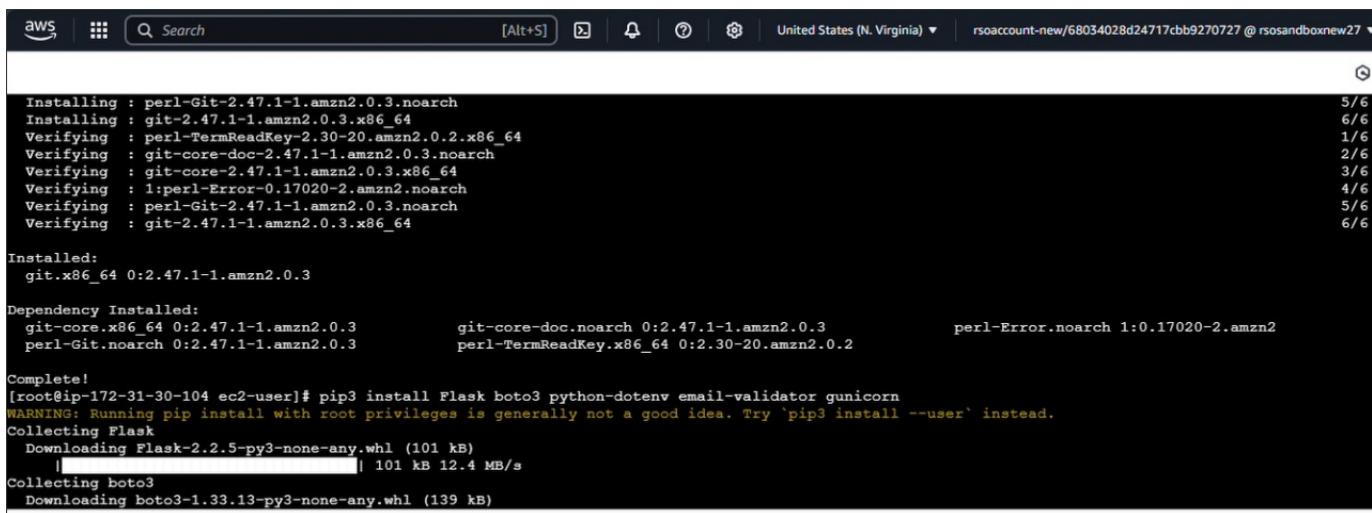
To navigateto the project directory, run the following command:

```
cd MedTrack
```

Once inside the projectdirectory, 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 | [Alt+Space] Search | United States (N. Virginia) | rsosandboxnew/68034028d24717cbb9270727 @ rsosandboxnew27 ▾
Installing : perl-Git-2.47.1-1.amzn2.0.3.noarch 5/6
Installing : git-2.47.1-1.amzn2.0.3.x86_64 6/6
Verifying  : perl-TermReadKey-2.30-20.amzn2.0.2.x86_64 1/6
Verifying  : git-core-doc-2.47.1-1.amzn2.0.3.noarch 2/6
Verifying  : git-core-2.47.1-1.amzn2.0.3.x86_64 3/6
Verifying  : 1:perl-Error-0.17020-2.amzn2.noarch 4/6
Verifying  : perl-Git-2.47.1-1.amzn2.0.3.noarch 5/6
Verifying  : git-2.47.1-1.amzn2.0.3.x86_64 6/6

Installed:
git.x86_64 0:2.47.1-1.amzn2.0.3

Dependency Installed:
git-core.x86_64 0:2.47.1-1.amzn2.0.3          git-core-doc.noarch 0:2.47.1-1.amzn2.0.3          perl-Error.noarch 1:0.17020-2.amzn2
perl-Git.noarch 0:2.47.1-1.amzn2.0.3           perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2

Complete!
[root@ip-172-31-30-104 ec2-user]# pip3 install Flask boto3 python-dotenv email-validator gunicorn
WARNING: Running pip install with root privileges is generally not a good idea. Try 'pip3 install --user' instead.
Collecting Flask
  Downloading Flask-2.2.5-py3-none-any.whl (101 kB)
|██████████| 101 kB 12.4 MB/s
Collecting boto3
  Downloading boto3-1.33.13-py3-none-any.whl (139 kB)
```

Verify the Flask app is running:

<http://3.208.17.147:5000/>

- Run the Flask app on the EC2 instance

Access the website through:

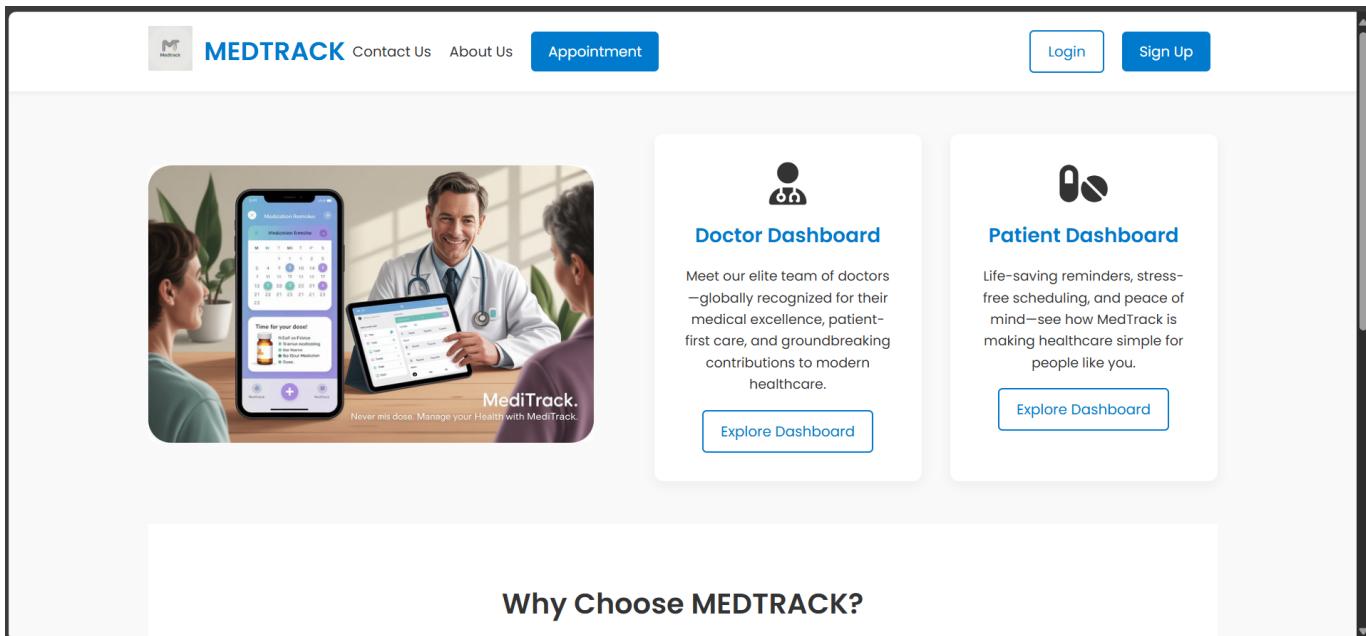
Public IPs: <http://3.208.17.147:5000/>

Milestone 8: Testing and Deployment:

• Activity 8.1:

Conduct functional testing to verify user sign-up, login, appointment booking, prescription generation, medication reminders, and SNS notifications.

Index Page:



The screenshot shows the MedTrack website homepage. At the top, there is a navigation bar with the MedTrack logo, 'Contact Us', 'About Us', a blue 'Appointment' button, and 'Login' and 'Sign Up' buttons. Below the navigation, there is a large image showing a doctor in a white coat interacting with a patient, with a smartphone and a tablet displaying the MedTrack app interface. The app screens show a calendar and a medication reminder. To the right of this image are two boxes: 'Doctor Dashboard' and 'Patient Dashboard'. Both boxes feature icons (a doctor silhouette and two pills respectively), brief descriptions of the service, and a 'Explore Dashboard' button. At the bottom of the page, the text 'Why Choose MEDTRACK?' is displayed.

Stay in the Loop

Subscribe for latest updates, features, and health tips from MedTrack.


Cloud-Powered Security
HIPAA-compliant data stored on AWS with top-level access control and monitoring.


Smart Reminders
Get automatic medication reminders via SMS, email, or app push notifications.


Doctor & Family Sharing
Share your schedule with caregivers or physicians with consent-based access.

© 2025 MEDTRACK. All rights reserved. | [Privacy Policy](#) | [Terms of Use](#)

Signup Page:

Welcome to MedTrack

Problem: Patients, especially those on multiple medications, often struggle with adherence, leading to missed doses or incorrect timings, impacting health outcomes.

Solution: MedTrack is a secure, cloud-based platform that helps patients track medication schedules, receive reminders, and manage prescription details.

- User login & registration
- Medication entry & dosage tracking
- Custom reminders via SMS/Email (AWS SNS)
- Secure storage using AWS DynamoDB
- Deployed via AWS EC2 & IAM secured

User Sign Up

Full Name

Email

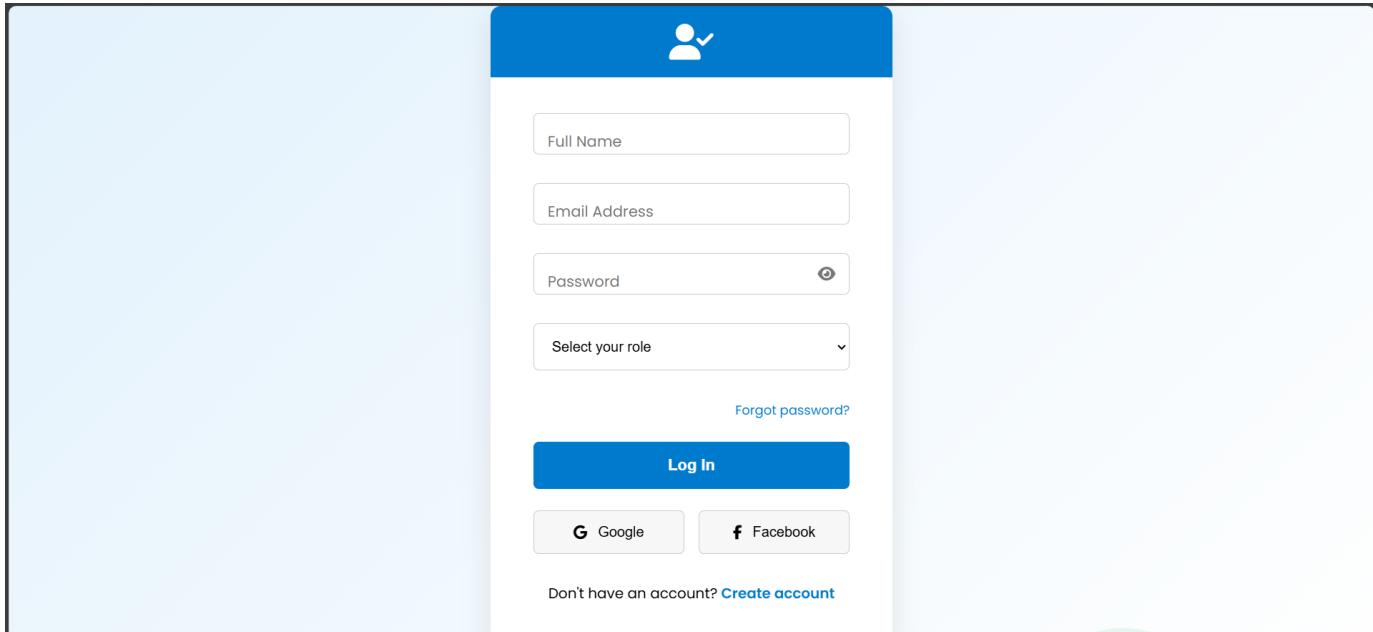
Password

Phone Number

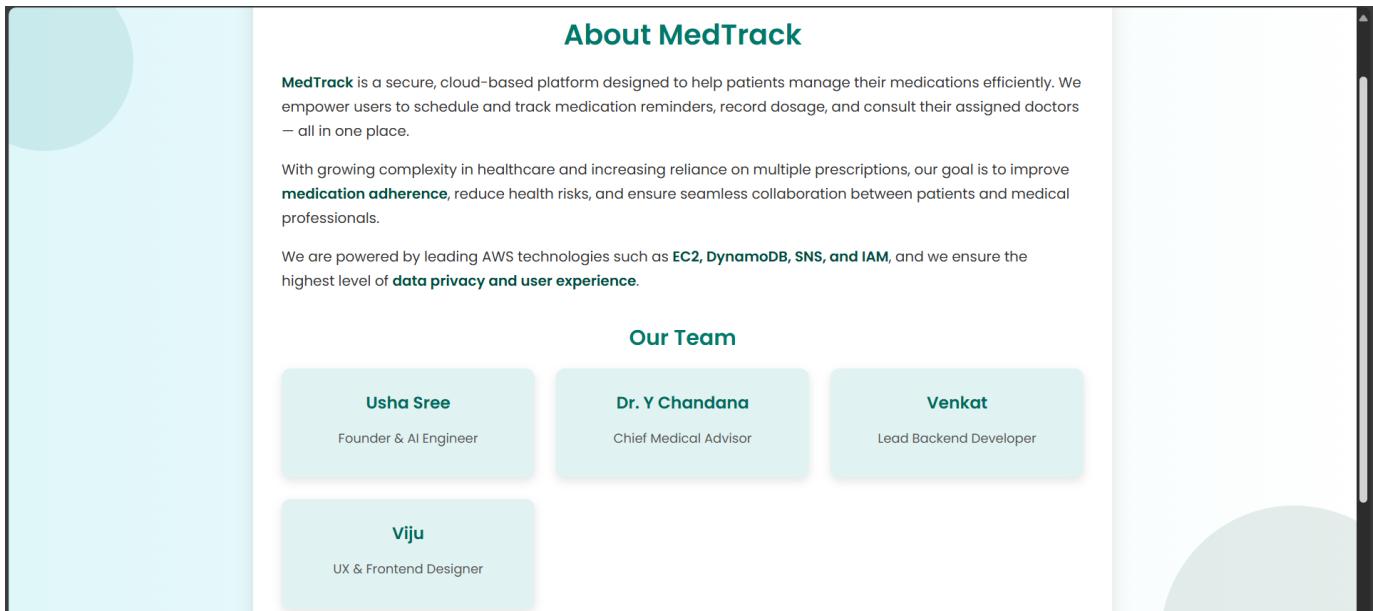
Role

Reminder Type

Already registered? [Login here](#)

Login Page:


The login page features a blue header with a user icon. Below it is a form with four input fields: 'Full Name', 'Email Address', 'Password' (with a visibility toggle), and 'Select your role'. Below the form are links for 'Forgot password?' and 'Log In'. At the bottom are social media login buttons for 'Google' and 'Facebook'. A 'Create account' link is also present.

About Us page:


About MedTrack

MedTrack is a secure, cloud-based platform designed to help patients manage their medications efficiently. We empower users to schedule and track medication reminders, record dosage, and consult their assigned doctors – all in one place.

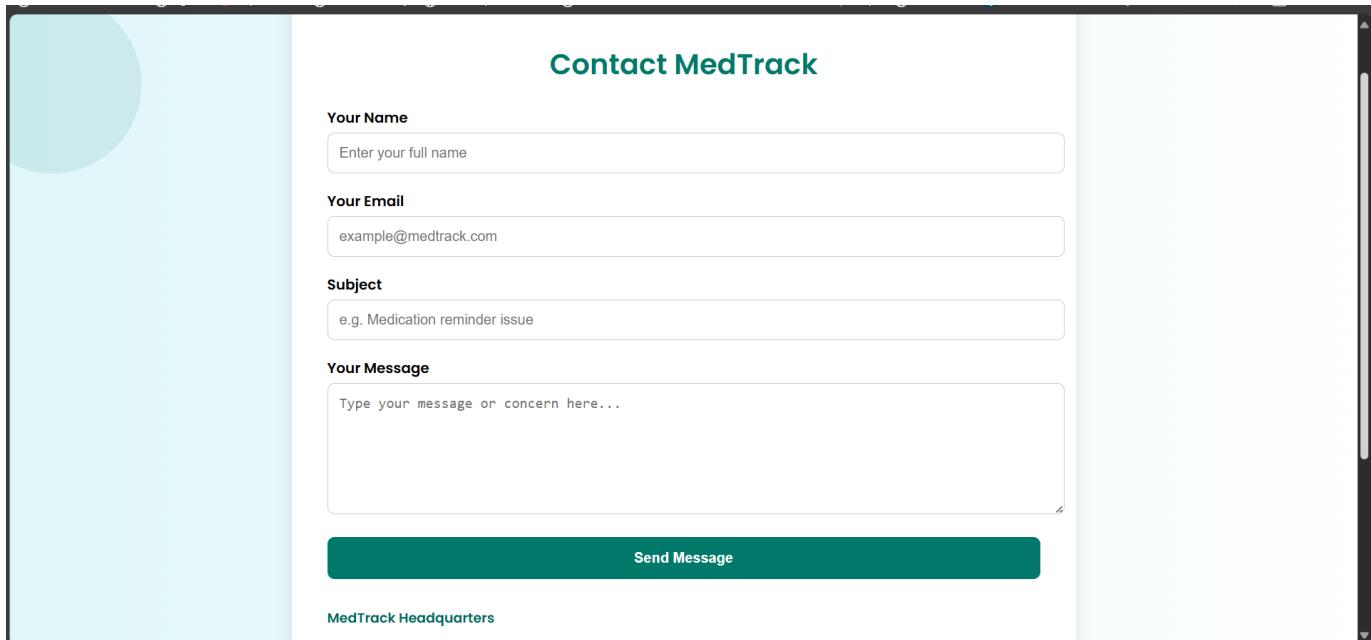
With growing complexity in healthcare and increasing reliance on multiple prescriptions, our goal is to improve **medication adherence**, reduce health risks, and ensure seamless collaboration between patients and medical professionals.

We are powered by leading AWS technologies such as **EC2, DynamoDB, SNS, and IAM**, and we ensure the highest level of **data privacy and user experience**.

Our Team

Usha Sree Founder & AI Engineer	Dr. Y Chandana Chief Medical Advisor	Venkat Lead Backend Developer
Viju UX & Frontend Designer		

ContactUs Page:



Contact MedTrack

Your Name
Enter your full name

Your Email
example@medtrack.com

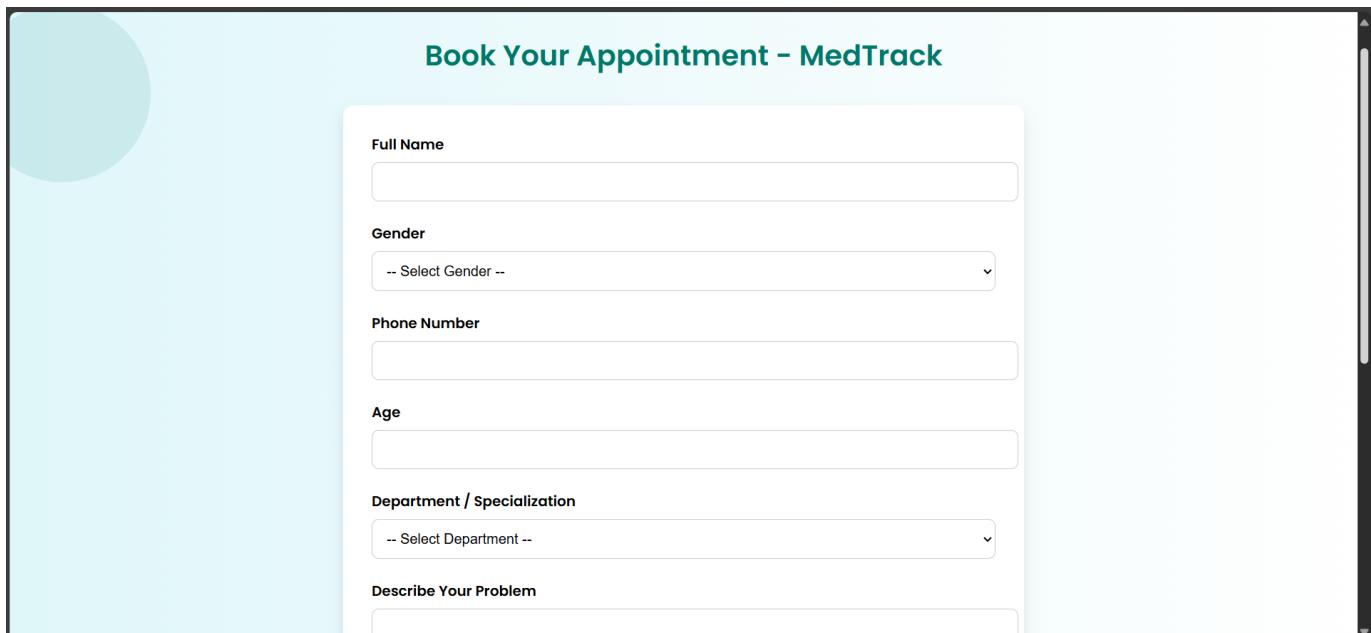
Subject
e.g. Medication reminder issue

Your Message
Type your message or concern here...

Send Message

MedTrack Headquarters

Book Appointment Page:



Book Your Appointment – MedTrack

Full Name

Gender
-- Select Gender --

Phone Number

Age

Department / Specialization
-- Select Department --

Describe Your Problem

Department:

-- Select Department --

Describe Your Problem

Select Available Doctor

-- Select Doctor --

Preferred Date

dd-mm-yyyy

Preferred Time

-- : -- AM

Book Appointment

(Doctor)Explore Dashboard Page:

Our Expert Medical Team

Certified. Compassionate. Committed to Your Health.

At MedTrack, we take pride in the team of elite professionals who bring passion and excellence to patient care. Each of our doctors is not only an expert in their field but also dedicated to building a better health experience. Discover the brilliant minds behind MedTrack and see why thousands trust us with their well-being.



(Patient)Explore Dashboard Page:

What Our Patients Say About MedTrack

Real patients. Real stories. Read what our users have to say about how MedTrack has helped them manage their medications and improve their health.



Priya Menon
★★★★★

"I used to miss my diabetes medications often, but with MedTrack's smart reminders and daily tracker, I haven't skipped a dose in months. Truly a life-saver!"



Ravi Teja
★★★★★

"The doctor communication through MedTrack is amazing. My doctor keeps track of my meds and progress. I feel safe and supported."



Megha Joshi
★★★★★

"MedTrack helped me build a strict schedule after my surgery. It's simple, intuitive, and even my 65-year-old mother uses it without issues."



Aman Kumar
★★★★★

"With AWS SMS reminders and easy logging, MedTrack is perfect for managing chronic illnesses. My asthma medication is always on schedule now."



Fatima Sheikh
★★★★★

"I love the privacy and control MedTrack gives. It's secure, cloud-based, and I never worry about data leaks. It's my health assistant."

Doctor Dashboard:


MedTrack

Dashboard

📅
Today's Appointments

💊
Medications Due

🔔
Alerts

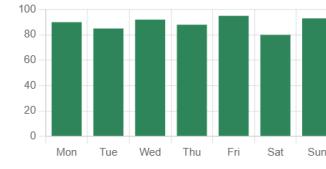
👤
Total Patients

🕒 Appointments
👤 Patients
⚙️ Settings

📅
Upcoming Appointments

Patient	Age	Department	Problem
No appointments yet.			

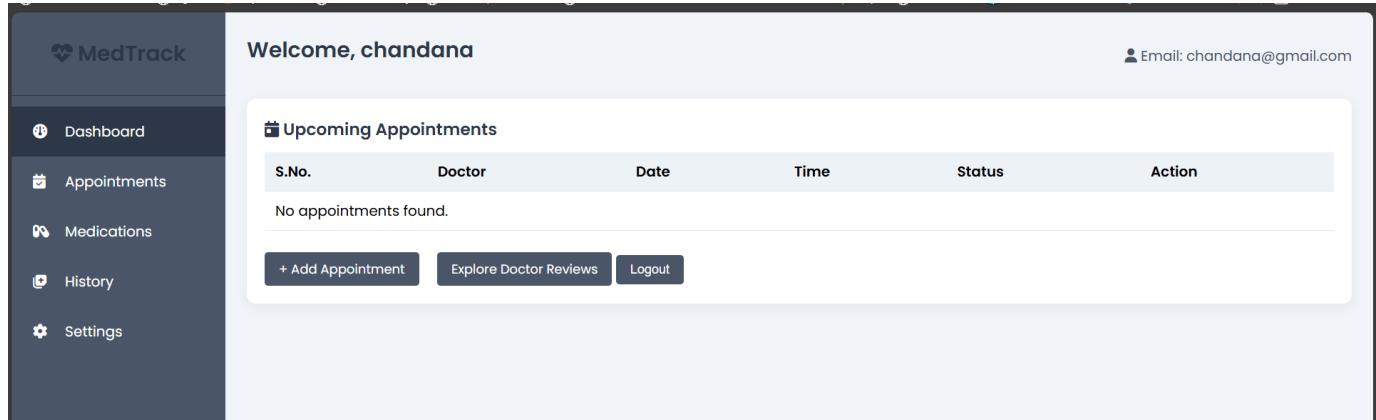
📊
Weekly Adherence



Day	Adherence (%)
Mon	~90
Tue	~85
Wed	~90
Thu	~90
Fri	~92
Sat	~80
Sun	~90

🔔
Alerts

No alerts for now.

Patient Dashboard:


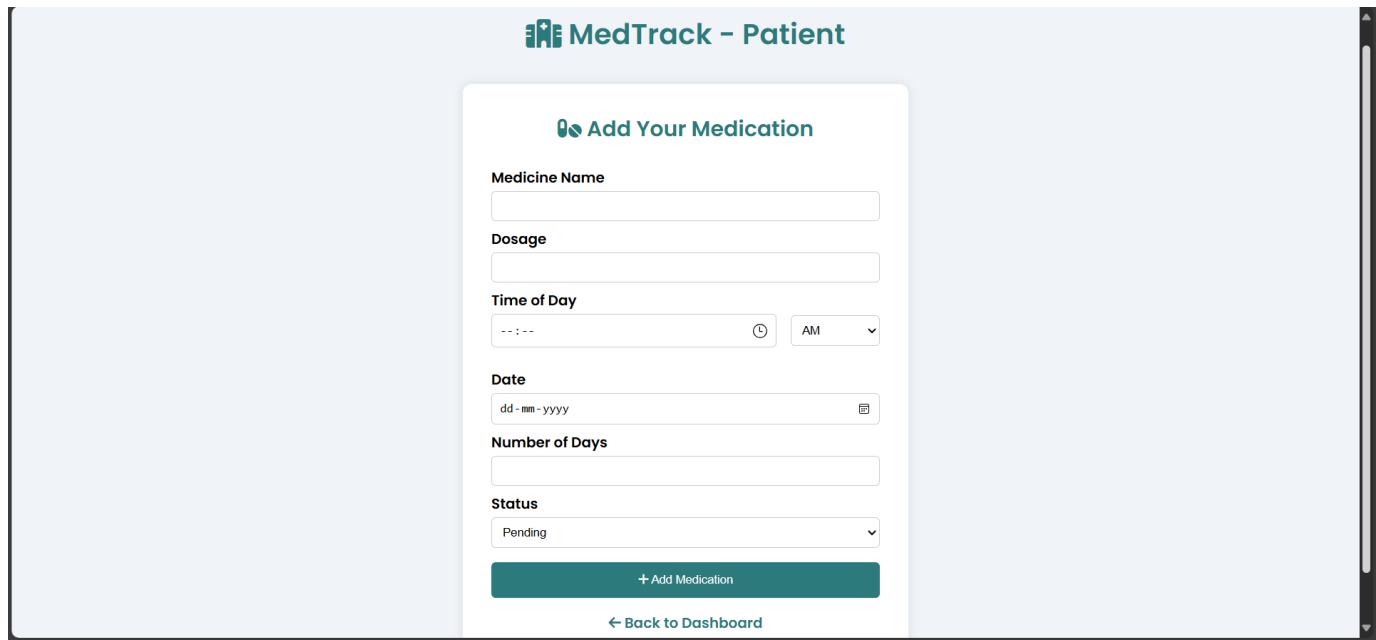
Welcome, chandana

Email: chandana@gmail.com

Upcoming Appointments

S.No.	Doctor	Date	Time	Status	Action
No appointments found.					

+ Add Appointment Explore Doctor Reviews Logout

Add Medication Page:


MedTrack – Patient

Add Your Medication

Medicine Name

Dosage

Time of Day

 :

Date

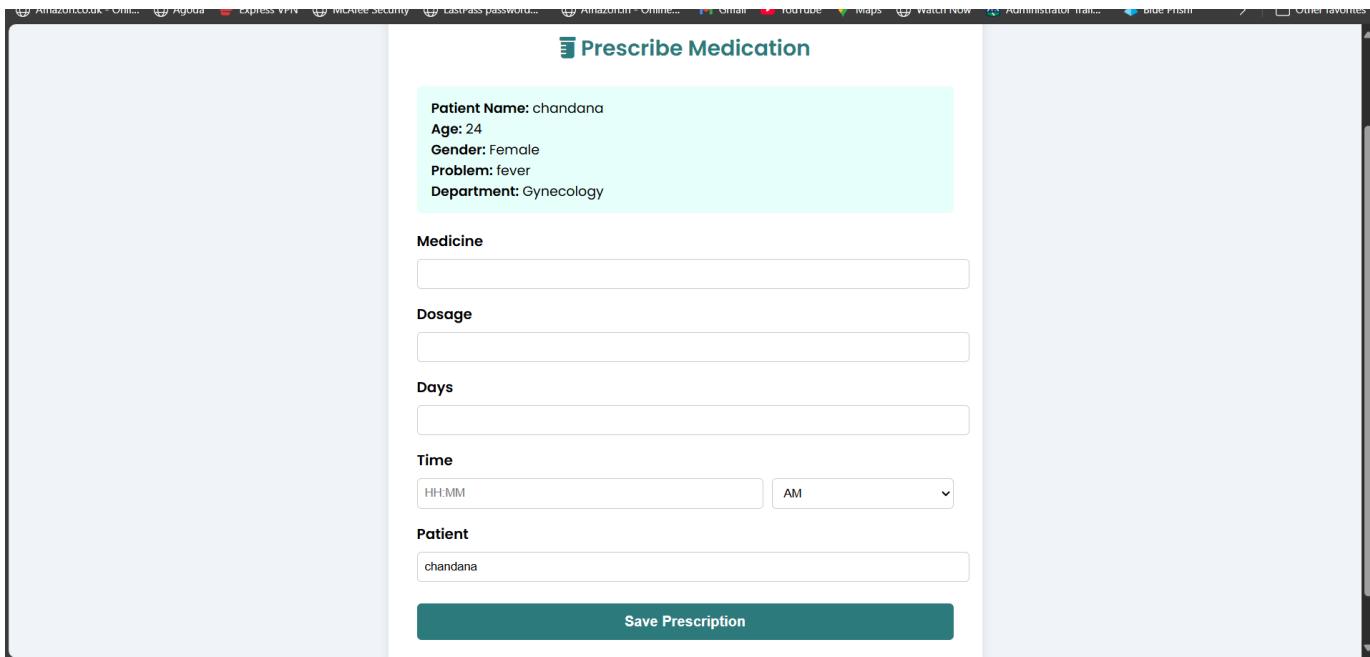
Number of Days

Status

+ Add Medication

← Back to Dashboard

Prescribe Medication:



The screenshot shows a web-based prescription form titled "Prescribe Medication". At the top, it displays the patient's information: Name: chandana, Age: 24, Gender: Female, Problem: fever, and Department: Gynecology. Below this, there are input fields for "Medicine", "Dosage", "Days", "Time" (with a dropdown for AM/PM), and "Patient" (set to chandana). A prominent teal button at the bottom right is labeled "Save Prescription".

Exit:

Session Ended

Please close this tab.

Conclusion:

The **MedTrack Healthcare Assistant** has been successfully developed and deployed using a robust cloud-based architecture. By leveraging **AWS services** such as **EC2 for hosting**, **DynamoDB for data storage**, **SNS for real-time email notifications**, and **IAM for secure access control**, the platform ensures reliable, secure, and scalable access to essential healthcare services for both patients and doctors.

This system addresses common healthcare challenges by allowing **patients to book appointments**, **receive timely medication reminders**, and track their prescriptions – all from a unified platform. **Doctors can manage appointments, issue prescriptions, and monitor patient adherence**, improving the overall quality of care.

The **cloud-native approach** ensures seamless scalability and high availability, making it adaptable to increasing user demand. The integration of **Flask with AWS services** guarantees efficient backend operations, including secure user authentication and notification handling.

During testing, all core functionalities – from user registration and login to **SNS-powered reminders and real-time medication tracking** – were verified for reliability and accuracy.

In conclusion, **MedTrack offers a modern, cloud-powered solution** for improving patient-doctor engagement, simplifying health management tasks, and enhancing the overall healthcare experience. This project showcases how cloud technologies can effectively address real-world challenges in the medical field.