

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.

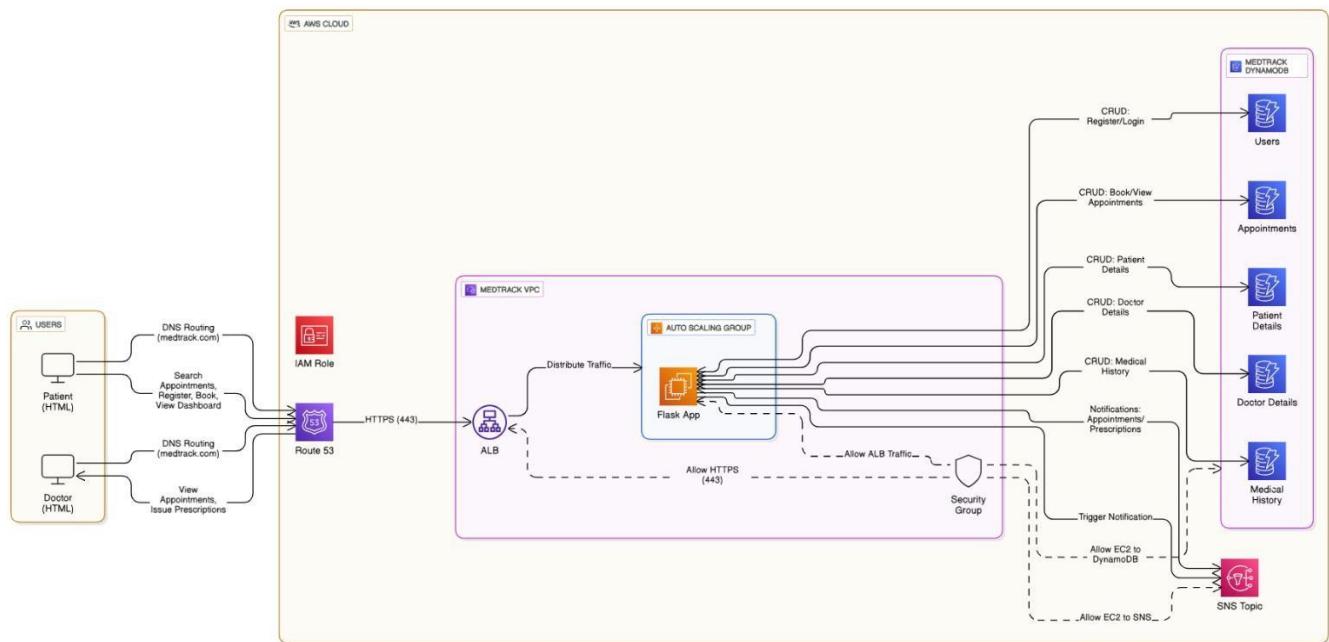
Scenario 2: Seamless Book Request Notifications for Library Staff

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.

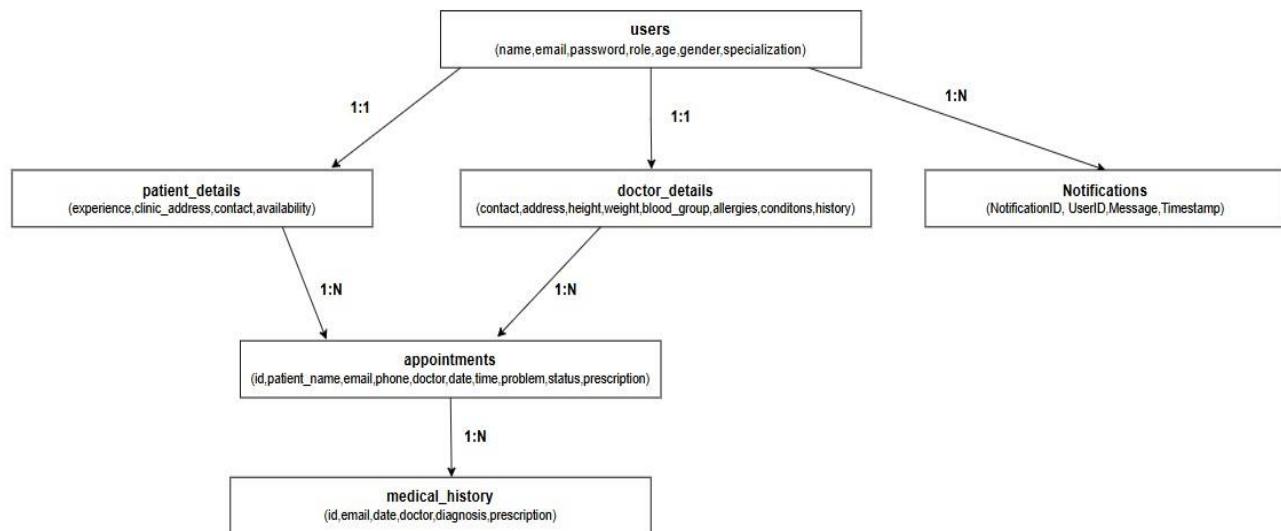
Scenario 3: Easy Access to Library Resources

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:**
<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
2. **Understanding IAM:**
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
3. **Amazon EC2 Basics:**
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
4. **DynamoDB Basics:**
<https://docs.aws.amazon.com/amazondynamodb/Introduction.html>
5. **SNS Overview:**
<https://docs.aws.amazon.com/sns/latest/dg/welcome.html>
6. **Git Version Control:** <https://git-scm.com/doc>

Project Work Flow:

1. Backend Development and Application Setup

Activity 1.1: Develop the Backend Using Flask.

Activity 1.2: Integrate AWS Services Using boto3.

2. AWS Account Setup and Login

Activity 2.1: Set up an AWS account if not already done.

Activity 2.2: Log in to the AWS Management Console

3. DynamoDB Database Creation and Setup

Activity 3.1: Navigate to the DynamoDB

Activity 3.2: Create a DynamoDB table for storing data.

4. SNS Notification Setup

Activity 4.1: SNS topics for email notifications.

Activity 4.2: Subscribe users and Admin.

5.IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

6.EC2 Instance Setup

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

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

7.Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance.

Activity 7.2: Clone Your Flask Project from GitHub

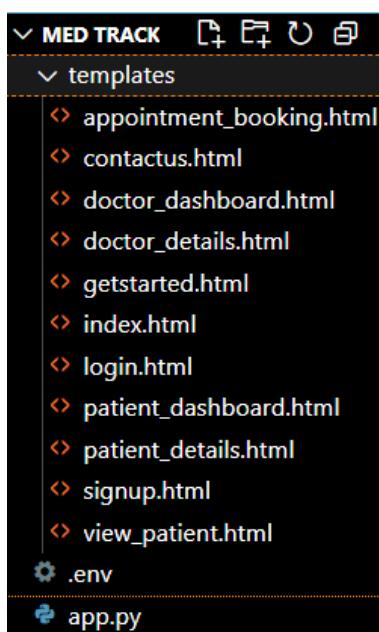
8.Testing and Deployment

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

Milestone 1:Backend Development and Application Setup

- **Activity 1.1: Develop the backend using Flask**

- File Explorer Structure



Description of Codes:

- Imports:

```
from flask import Flask, render_template, request, redirect, session, url_for, flash, g
from datetime import datetime, timedelta
import os
import uuid
from functools import wraps
import boto3
from dotenv import load_dotenv
from werkzeug.security import generate_password_hash, check_password_hash
```

Description: Import core Flask utilities, Boto3 for DynamoDB access, SNS for notifications, environment loading via dotenv, and werkzeug.security for password hashing.

- Flask: routes & web framework ○ render_template: to load HTML ○ request: handle form submissions ○ redirect, url_for: navigate between routes ○ session: manage user login state ○ flash: display alert ○ generate_password_hash / check_password_hash: secure passwords ○ boto3: AWS SDK for DynamoDB and SNS ○ dotenv: read environment variables ○ datetime, timedelta: timestamps ○ uuid: unique IDs
- functools.wraps: decorators

- **app.py** initializes the Flask server with:

```
app = Flask(__name__)
app.secret_key = os.environ.get('SECRET_KEY', os.urandom(24))
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(hours=1)
```

Description: initialize the Flask application instance using Flask(__name__) to start building the web app. Then initialize the Flask application with a random session secret key and configure the session lifetime.

- **DynamoDB Setup:**

```

AWS_REGION_NAME = os.environ.get('AWS_REGION_NAME', 'us-east-1')
USERS_TABLE_NAME = os.environ.get('USERS_TABLE_NAME', 'MedTrackUsers')
APPOINTMENTS_TABLE_NAME = os.environ.get('APPOINTMENTS_TABLE_NAME', 'MedTrackAppointments')
PATIENT_DETAILS_TABLE_NAME = os.environ.get('PATIENT_DETAILS_TABLE_NAME', 'MedTrackPatientDetails')
DOCTOR_DETAILS_TABLE_NAME = os.environ.get('DOCTOR_DETAILS_TABLE_NAME', 'MedTrackDoctorDetails')
MEDICAL_HISTORY_TABLE_NAME = os.environ.get('MEDICAL_HISTORY_TABLE_NAME', 'MedTrackMedicalHistory')

try:
    dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION_NAME)
    sns = boto3.client('sns', region_name=AWS_REGION_NAME)
    SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')
    print("AWS credentials loaded using IAM Role or environment.")
except Exception as e:
    print(f"Error initializing DynamoDB or SNS: {e}")
    dynamodb = None
    sns = None
    SNS_TOPIC_ARN = None

```

Description:

Initializes DynamoDB resource for us-east-1 region and sets up references to MedTrack tables for storing users, appointments, doctor/patient details, and medical histories.

- **SNS Connection:**

```

def publish_to_sns(message, subject="MedTrack Notification"):
    if sns and SNS_TOPIC_ARN:
        try:
            sns.publish(
                TopicArn=SNS_TOPIC_ARN,
                Message=message,
                Subject=subject
            )
        except Exception as e:
            print(f"Error publishing to SNS: {e}")
    else:
        print("SNS not configured, skipping publish.")

```

Description:

Configures SNS notifications for sending alerts about user registration and appointments, by referencing the topic ARN from environment variables.

- **Helper Functions:**

```

def get_user_table():
    if dynamodb:
        return dynamodb.Table(USERS_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def get_appointments_table():
    if dynamodb:
        return dynamodb.Table(APPOINTMENTS_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def get_patient_details_table():
    if dynamodb:
        return dynamodb.Table(PATIENT_DETAILS_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def get_doctor_details_table():
    if dynamodb:
        return dynamodb.Table(DOCTOR_DETAILS_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def get_medical_history_table():
    if dynamodb:
        return dynamodb.Table(MEDICAL_HISTORY_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_email' not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function

```

Description:

Defines helper methods to get specific DynamoDB tables and to wrap routes requiring login sessions.

- Before Request Hook:

```

@app.before_request
def load_logged_in_user():
    user_email = session.get('user_email')
    g.user = None
    g.doctor_details = None
    if user_email:
        if dynamodb:
            user_table = get_user_table()
            try:
                response = user_table.get_item(Key={'email': user_email})
                if 'Item' in response:
                    g.user = response['Item']
                    if g.user['role'] == 'doctor':
                        doctor_table = get_doctor_details_table()
                        doc_resp = doctor_table.get_item(Key={'email': user_email})
                        g.doctor_details = doc_resp.get('Item')
            except Exception as e:
                print(f"Error loading user from DynamoDB: {e}")

```

Description:

Loads the current logged-in user into Flask's g object before each request, so the user session can be accessed throughout routes.

Routes:

- **Index Route (/):**

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

Description:

Renders the landing page index.html.

- **Signup Route (/signup):**

Description:

Handles GET to show the signup page, and POST to validate user inputs, hash passwords, save new users to DynamoDB, then redirect them to the relevant detail page based on role.

```

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        role = request.form.get('role')
        name = request.form.get('name')
        email = request.form.get('email')
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')
        age = request.form.get('age')
        gender = request.form.get('gender')
        specialization = request.form.get('specialization') if role == 'doctor' else ''
        hashed_password = generate_password_hash(password)

        user_data = {
            'email': email,
            'name': name,
            'password': hashed_password,
            'role': role,
            'age': age,
            'gender': gender,
            'specialization': specialization,
            'created_at': datetime.now().isoformat()
        }

        if dynamodb:
            user_table = get_user_table()
            try:
                user_table.put_item(Item=user_data)
                publish_to sns(
                    f"Welcome {name}! Your {role} account has been created successfully.",
                    "New User Registration"
                )
            except Exception as e:

```



```

        else:
            # local_db['users'][email] = user_data
            pass

            session.clear()
            session['user_email'] = email

            return redirect(url_for('doctor_details' if role == 'doctor' else 'patient_details'))

```

- **Login Route (/login):**

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = None

        if dynamodb:
            user_table = get_user_table()
            try:
                response = user_table.get_item(Key={'email': email})
                if 'Item' in response:
                    user = response['Item']
            except Exception as e:
                print(f"Error fetching user: {e}")
        else:
            # user = local_db['users'].get(email)
            pass

        if user and check_password_hash(user['password'], password):
            session.clear()
            session['user_email'] = email
            return redirect(url_for('doctor_dashboard' if user['role'] == 'doctor' else 'patient_dashboard'))
        else:
            flash("Invalid credentials")

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

Description:

Handles GET to render the login page, POST to verify credentials from DynamoDB, check password, and redirect to the role-based dashboard.

- **Logout Route (/logout):**

```
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('index'))
```

Description:

Clears the user session and redirects to the home page.

- **Patient Dashboard Route (/patient_dashboard):**

Description:

Displays patient dashboard with appointments, personal details, and medical history pulled from DynamoDB.

```

@app.route('/patient_dashboard')
@login_required
def patient_dashboard():
    if g.user['role'] != 'patient':
        return redirect(url_for('login'))

    appointments = []
    history = []
    patient_details = None
    if dynamodb:
        appointments_table = get_appointments_table()
        details_table = get_patient_details_table()
        history_table = get_medical_history_table()
        try:
            a_resp = appointments_table.scan()
            appointments = [a for a in a_resp['Items'] if a['email'] == g.user['email']]
            d_resp = details_table.get_item(Key={'email': g.user['email']})
            patient_details = d_resp.get('Item')
            h_resp = history_table.scan()
            history = [h for h in h_resp['Items'] if h['email'] == g.user['email']]
        except Exception as e:
            print(f"Error loading patient dashboard: {e}")

```

- **Appointment Booking Route (/appointment_dashboard):**

```

@app.route('/appointment_dashboard')
@login_required
def appointment_dashboard():
    if g.user['role'] != 'patient':
        return redirect(url_for('login'))

    doctors = []
    if dynamodb:
        user_table = get_user_table()
        doctor_details_table = get_doctor_details_table()
        try:
            response = user_table.scan()
            doctors = [u for u in response['Items'] if u['role'] == 'doctor']
            for doc in doctors:
                details_resp = doctor_details_table.get_item(Key={'email': doc['email']})
                details = details_resp.get('Item')
                doc['availability'] = details.get('availability', 'Not provided') if details else 'Not provided'
        except Exception as e:
            print(f"Error fetching doctors: {e}")

```

Description:

Lets patients view available doctors and schedule appointments.

- **Doctor Dashboard Route(/doctor_dashboard):**

Description:

Define /doctor_dashboard route to render the doctor's personal dashboard, showing their scheduled appointments (filtered by doctor name), along with their own stored profile details if available. This helps doctors manage appointments and see their patients.

```

@app.route('/doctor_dashboard')
@login_required
def doctor_dashboard():
    if g.user['role'] != 'doctor':
        return redirect(url_for('login'))

    appointments = []
    if dynamodb:
        table = get_appointments_table()
        try:
            response = table.scan()
            appointments = [a for a in response['Items'] if a['doctor'] == g.user['name']]
        except Exception as e:
            print(f"Error fetching appointments: {e}")
    else:
        # appointments = [
        #     a for patient_appts in local_db['appointments'].values()
        #     for a in patient_appts if a['doctor'] == g.user['name']
        # ]
        pass

    for a in appointments:
        if 'status' not in a:
            a['status'] = 'Scheduled'

    return render_template('doctor_dashboard.html', user=g.user, appointments=appointments, doctor_details=g.doctor_details)

```

- **Patient Details Route(/patient_details):**

```

@app.route('/patient_details')
@login_required
def patient_details():
    if g.user['role'] != 'patient':
        return redirect(url_for('login'))
    return render_template('patient_details.html', user=g.user)

```

Description:

Define /patient_details route to render the patient_details.html page, allowing a patient to complete their profile with personal medical details, like contact, address, allergies, and past conditions.

- **Doctor Details Route(/doctor_details):**

```

@app.route('/doctor_details')
@login_required
def doctor_details():
    if g.user['role'] != 'doctor':
        return redirect(url_for('login'))
    return render_template('doctor_details.html')

```

Description:

Define /doctor_details route to render the doctor_details.html page, where a doctor can add or edit their own professional profile details (experience, contact, availability, etc.) right after signing up or while updating.

- **Exit & Static Info Routes:**

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

Tabnine | Edit | Test | Explain | Document
@app.route('/contactus')
def contactus():
    return render_template('contactus.html')
```

Description:

Simple routes for About Us and Contact Us pages.

- **Deployment Code:**

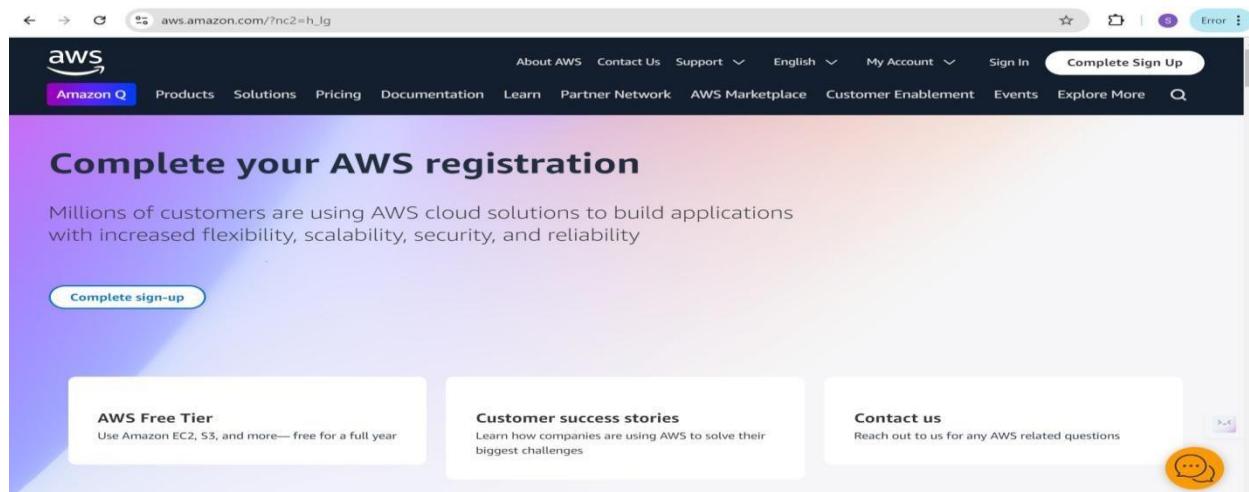
```
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000, debug=True)
```

Description:

Starts the Flask server listening on 0.0.0.0 port 5000 with debug=True.

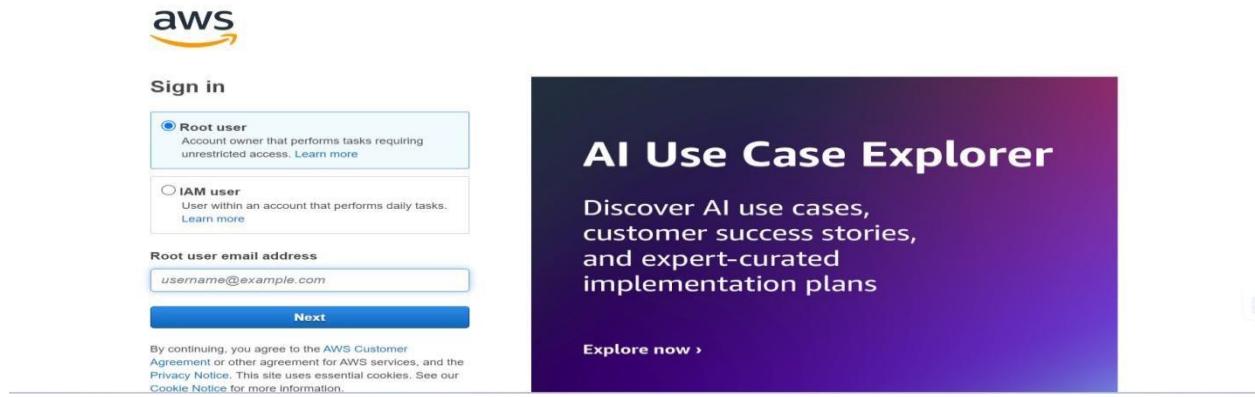
Milestone 2: AWS Account Setup and Login

- **Activity 2.1: Set up an AWS account if not already done.**
 - Sign up for an AWS account and configure billing settings.



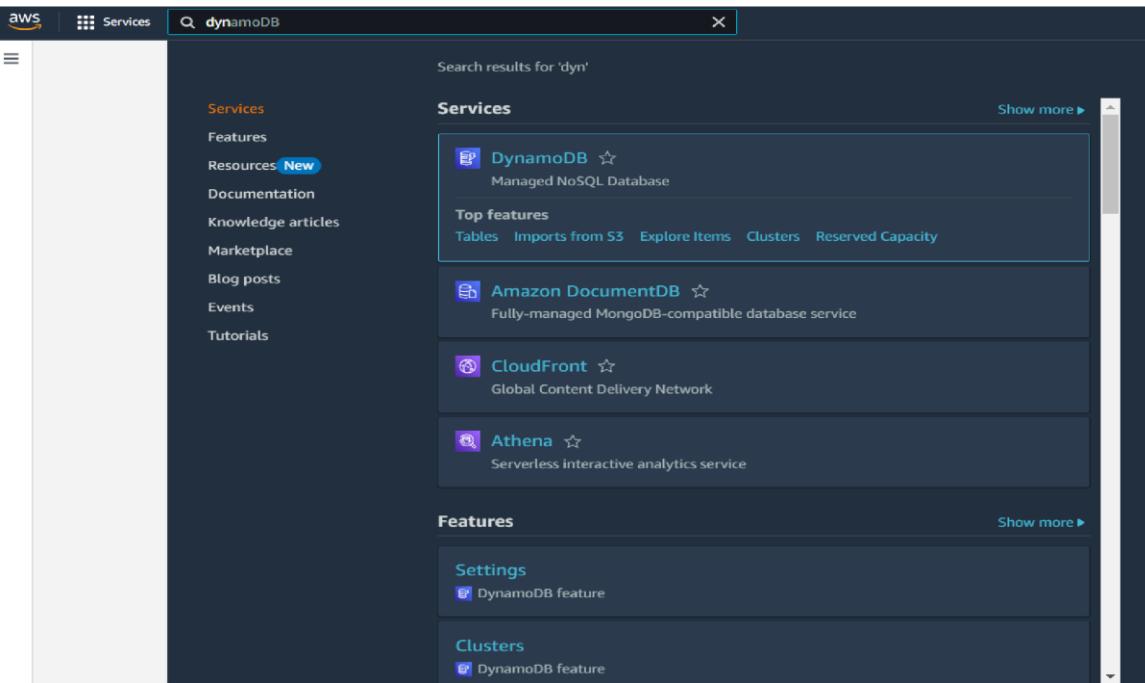
- **Activity 2.2: Log in to the AWS Management Console**

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



Milestone 3: DynamoDB Database Creation and Setup

- **Activity 3.1: Navigate to the DynamoDB**
 - In the AWS Console, navigate to DynamoDB and click on create tables.



The image shows the AWS Services search interface. The search bar at the top contains 'dynamoDB'. The sidebar on the left lists various navigation options like Services, Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main search results area shows a list of services under 'Services' and 'Features'. Under 'Services', 'DynamoDB' is listed as a Managed NoSQL Database. Under 'Features', 'Settings' and 'Clusters' both mention 'DynamoDB feature'. Other services listed include Amazon DocumentDB, CloudFront, and Athena.

DynamoDB

Dashboard

- Tables
- Explore items
- PartiQL editor
- Backups
- Exports to S3
- Imports from S3
- Integrations [New](#)
- Reserved capacity
- Settings

DAX

- Clusters
- Subnet groups
- Parameter groups
- Events

Alarms (0) [Info](#)

DAX clusters (0) [Info](#)

Create resources

Create an Amazon DynamoDB table for fast and predictable database performance at any scale. [Learn more](#)

Create table

Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. [Learn more](#)

Create DAX cluster

What's new

SEP 19 AWS Cost Management now provides purchase recommendations for Amazon DynamoDB...

DynamoDB

Tables

- Dashboard
- Tables
- Explore items
- PartiQL editor
- Backups
- Exports to S3
- Imports from S3
- Integrations [New](#)

Tables (0) [Info](#)

Create table

You have no tables in this account in this AWS Region.

Create table

- **Activity 3.2:Create a DynamoDB table for storing registration details.**

- Create MedTrackUsers table with partition key ‘email’ with type String and click on create tables for storing user data.

us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#create-table

DynamoDB > Tables > Create table

Share feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.

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.

Name
This will be used to identify your table.

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

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 [String](#)
1 to 255 characters and case sensitive.

Table settings

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

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

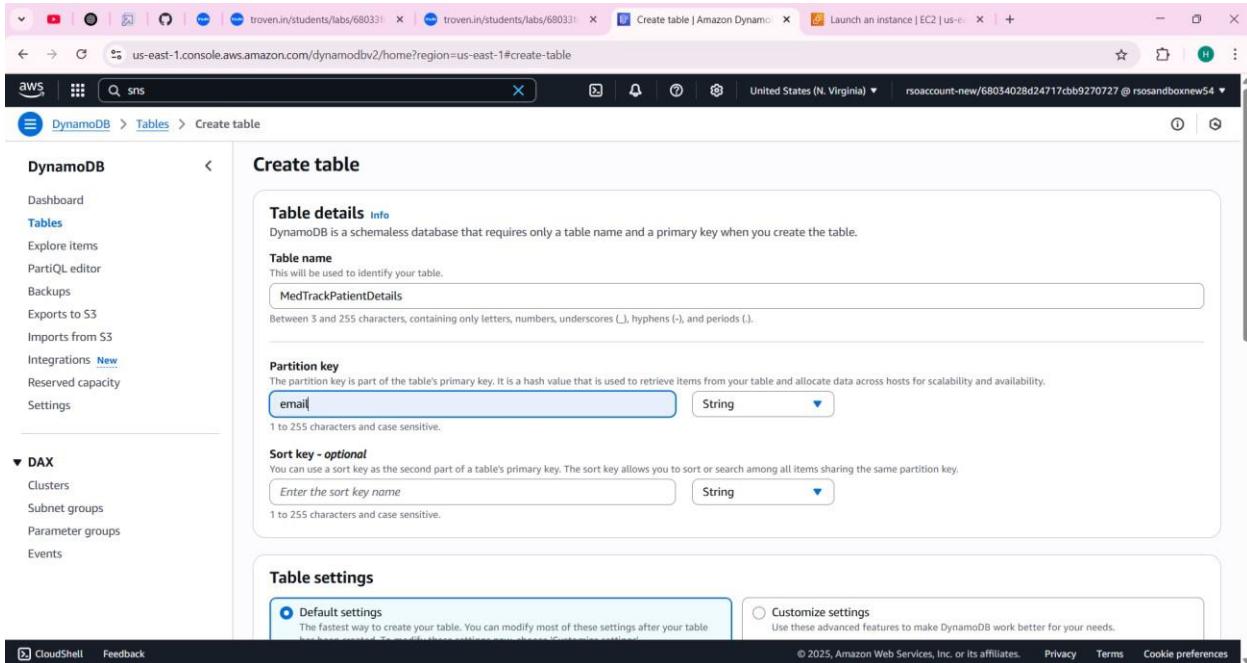
Tags
 Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag
 You can add 50 more tags.

Cancel **Create table**

- Follow the same steps to create a MedTrackPatientDetails table with 'email' as the partition key with type string to store patient details.



The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The left sidebar shows navigation links like Dashboard, Tables, Explore items, PartQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main area is titled 'Create table' and contains the following fields:

- Table details**: A note states "DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table." The 'Table name' field is set to "MedTrackPatientDetails".
- Partition key**: The note says "The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability." The 'email' field is selected as the partition key type "String".
- Sort key - optional**: The note says "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." The 'Enter the sort key name' field is empty.
- Table settings**: Two options are available:
 - Default settings**: "The fastest way to create your table. You can modify most of these settings after your table is created." This option is selected.
 - Customize settings**: "Use these advanced features to make DynamoDB work better for your needs."

At the bottom, there are links for CloudShell, Feedback, © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

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

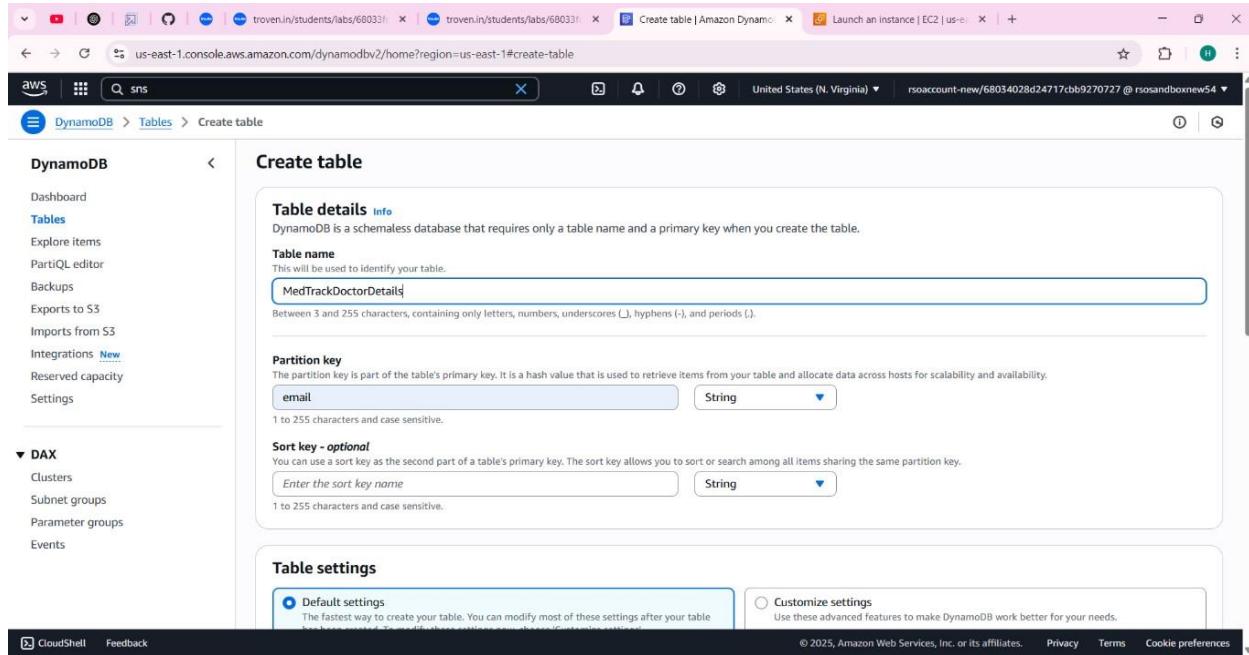
Tags
 Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

[Add new tag](#)
 You can add 50 more tags.

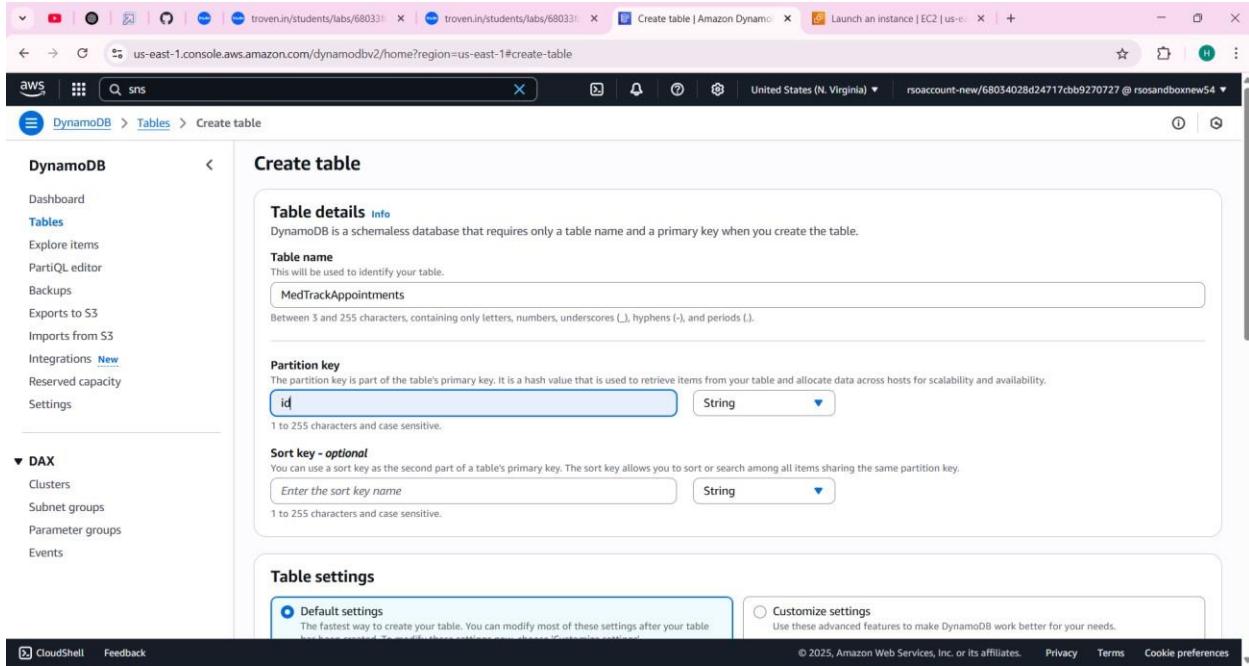
[Cancel](#) [Create table](#)

- Follow the same steps to create a MedTrackDoctorDetails table with 'email' as the partition key with type string to store patient details.



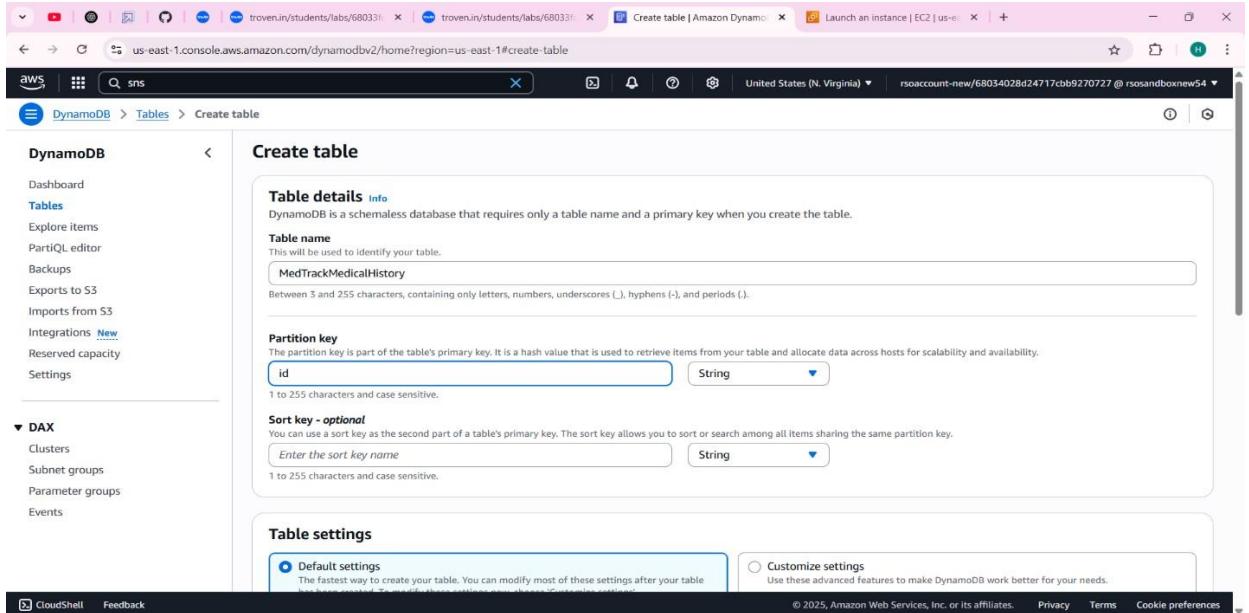
The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The left sidebar shows navigation links like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main area has two tabs: 'Table details' (selected) and 'Info'. Under 'Table details', the 'Table name' is set to 'MedTrackDoctorDetails'. Under 'Partition key', the 'email' field is selected as a String type. Under 'Sort key - optional', there is a placeholder 'Enter the sort key name'. In the 'Table settings' section, the 'Default settings' radio button is selected. At the bottom, there are links for CloudShell, Feedback, and a footer with copyright information.

- Follow the same steps to create a MedTrackAppointments table with 'id' as the partition key with type string to store appointments details.



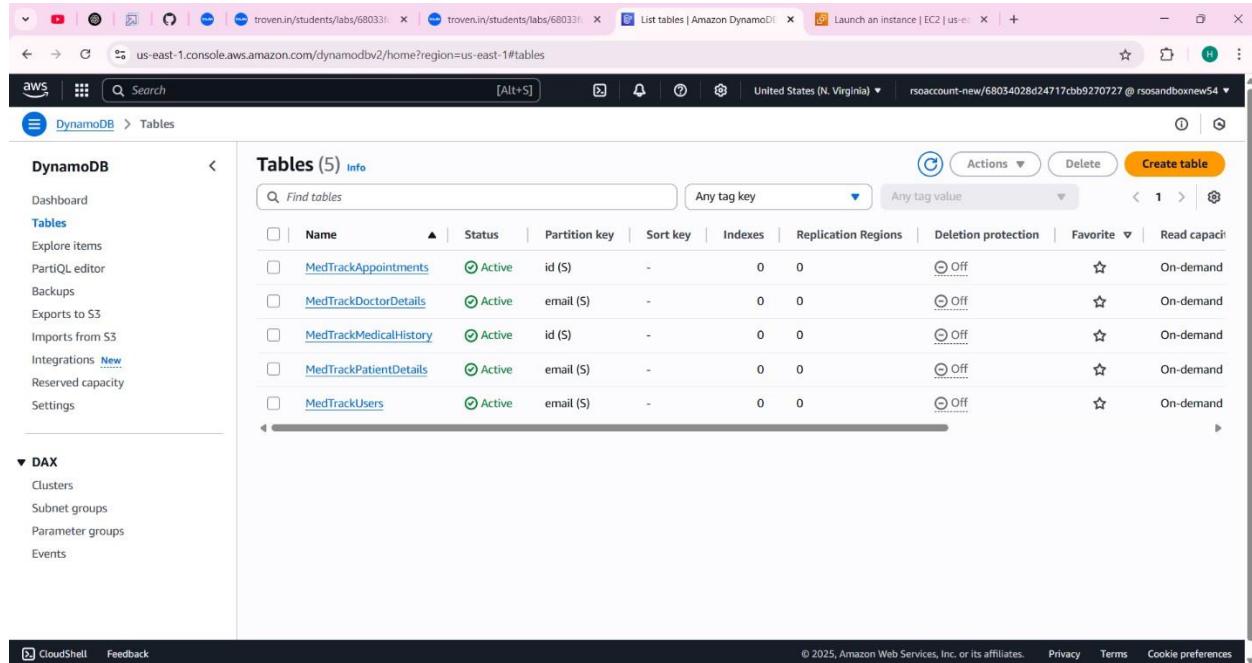
The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The table name is set to 'MedTrackAppointments'. The primary key is defined as 'id' of type String. There is no sort key specified. Under 'Table settings', the 'Default settings' option is selected. The page includes standard AWS navigation and footer links.

- Follow the same steps to create a MedTrackMedicalHistory table with 'id' as the partition key with type string to store details of medical history.



The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The table name is set to 'MedTrackMedicalHistory'. The primary key is defined as 'id' of type String. There is no sort key specified. Under 'Table settings', the 'Default settings' option is selected. The page includes standard AWS navigation and footer links.

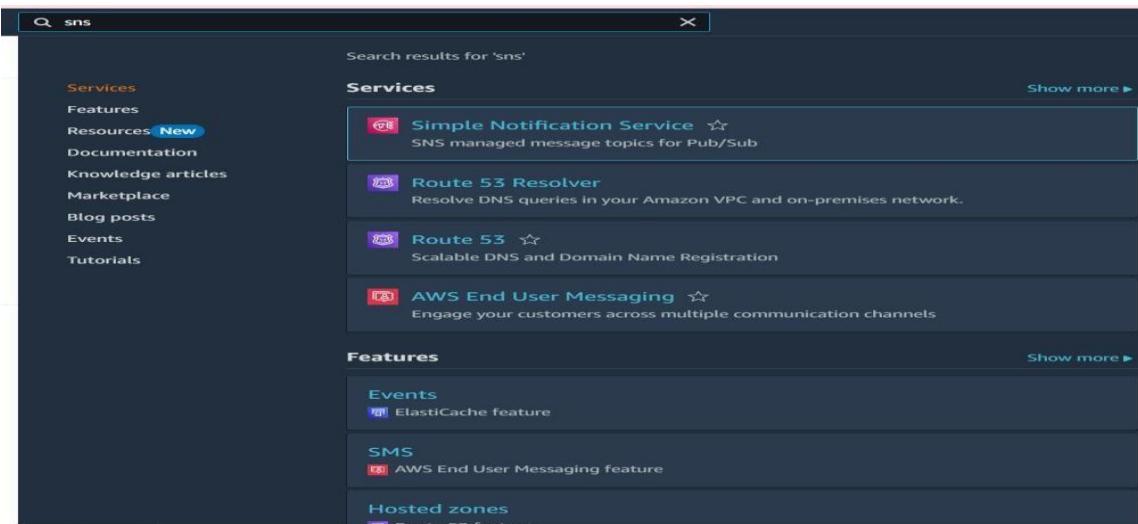
After creating all the tables as mentioned above the tables must be displayed like this.

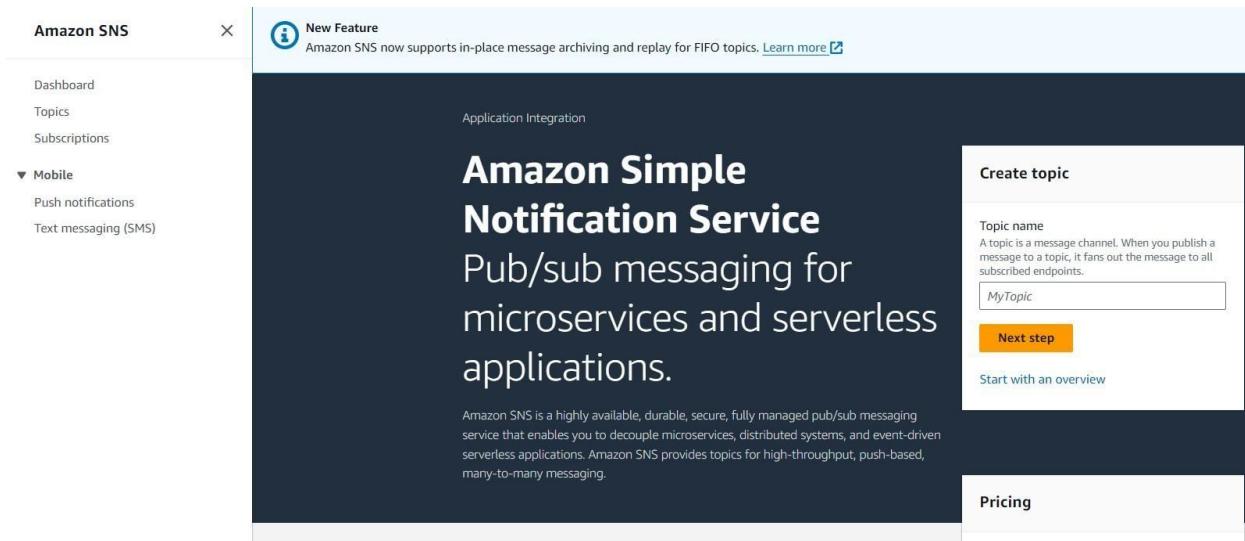


Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity
MedTrackAppointments	Active	id (\$)	-	0	0	Off	☆	On-demand
MedTrackDoctorDetails	Active	email (\$)	-	0	0	Off	☆	On-demand
MedTrackMedicalHistory	Active	id (\$)	-	0	0	Off	☆	On-demand
MedTrackPatientDetails	Active	email (\$)	-	0	0	Off	☆	On-demand
MedTrackUsers	Active	email (\$)	-	0	0	Off	☆	On-demand

Milestone 4: SNS Notification Setup

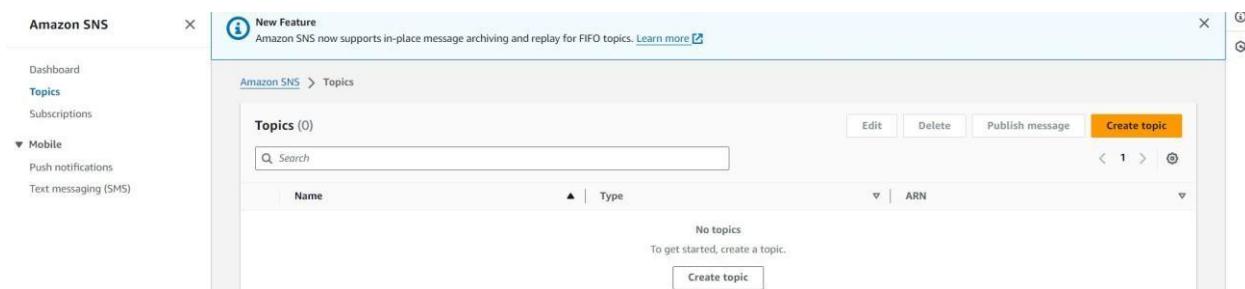
- **Activity 4.1: Create SNS topics for sending notifications When a book request is made to the subscribed emails.**
 - In the AWS Console, search for SNS and navigate to the SNS Dashboard.





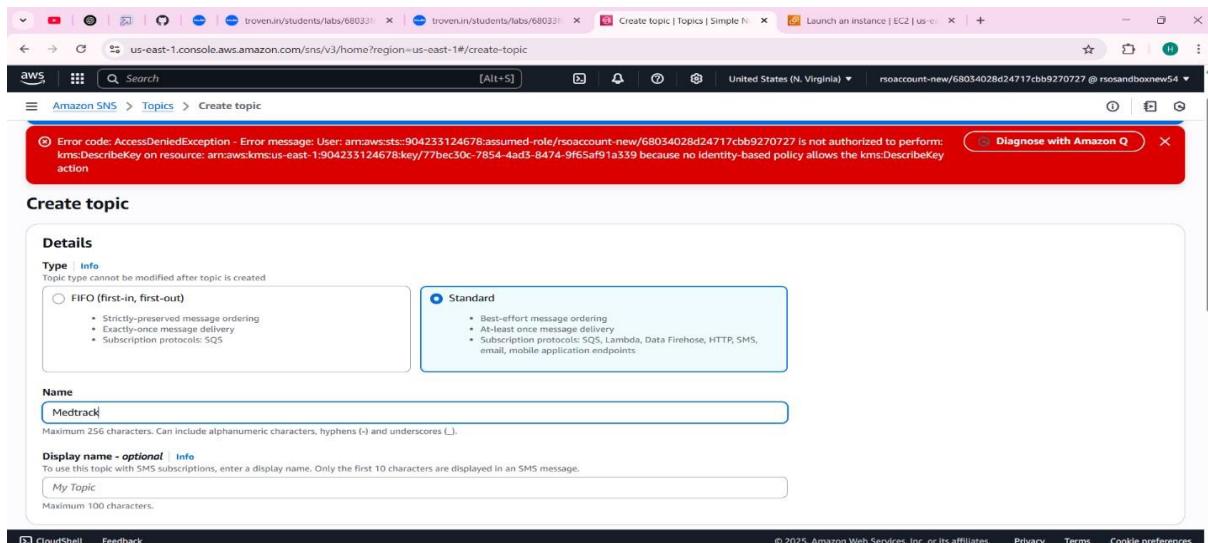
The screenshot shows the Amazon Simple Notification Service (SNS) home page. On the left, there is a sidebar with navigation links: Dashboard, Topics, Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and a New Feature announcement about in-place message archiving and replay for FIFO topics. The main content area features a large title "Amazon Simple Notification Service" with the subtitle "Pub/sub messaging for microservices and serverless applications." Below the title is a brief description of the service and a "Create topic" button.

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



The screenshot shows the "Topics" page within the Amazon SNS service. The sidebar remains the same. The main area displays a table titled "Topics (0)" with a search bar and a "Create topic" button. A message at the bottom encourages users to "To get started, create a topic."

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



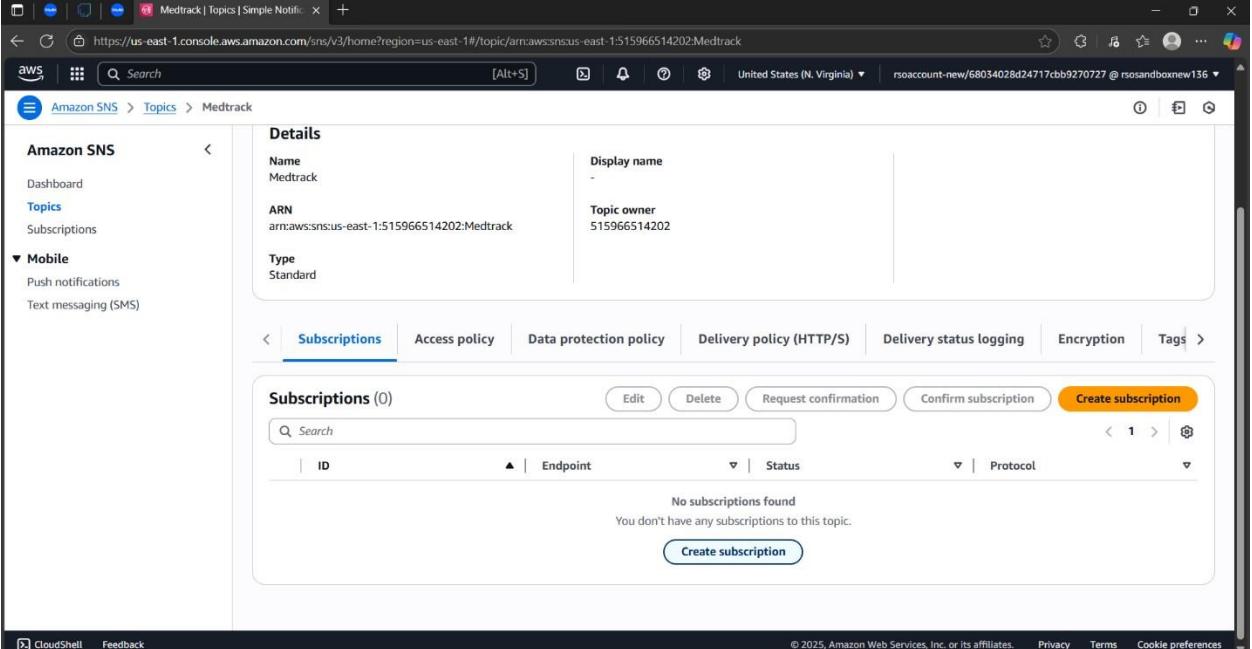
The screenshot shows the "Create topic" page. At the top, there is an error message: "Error code: AccessDeniedException - Error message: User: arnawssts:904233124678:assumed-role/rsoaccount-new/68034028d24717cbb9270727 is not authorized to perform: kms:DescribeKey on resource: arnaws:kms:us-east-1:904233124678:key/77bec30c-7854-4ad3-8474-9f65a91a539 because no identity-based policy allows the kms:DescribeKey action". Below this, the "Create topic" form is shown. It has two options for "Type": "FIFO (first-in, first-out)" and "Standard". The "Standard" option is selected and includes a bulleted list: "Best-effort message ordering", "At-least once message delivery", and "Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints". The "Name" field is filled with "Medtrack". The "Display name - optional" field contains "My Topic". At the bottom, there are links for CloudShell, Feedback, and a footer with copyright information.

=

- ▶ **Access policy - optional** Info
 This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.
- ▶ **Data protection policy - optional** Info
 This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.
- ▶ **Delivery policy (HTTP/S) - optional** Info
 The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.
- ▶ **Delivery status logging - optional** Info
 These settings configure the logging of message delivery status to CloudWatch Logs.
- ▶ **Tags - optional**
 A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#) 
- ▶ **Active tracing - optional** Info
 Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

[Cancel](#) [Create topic](#)

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



The screenshot shows the AWS SNS Topics page for the 'Medtrack' topic. The topic details are as follows:

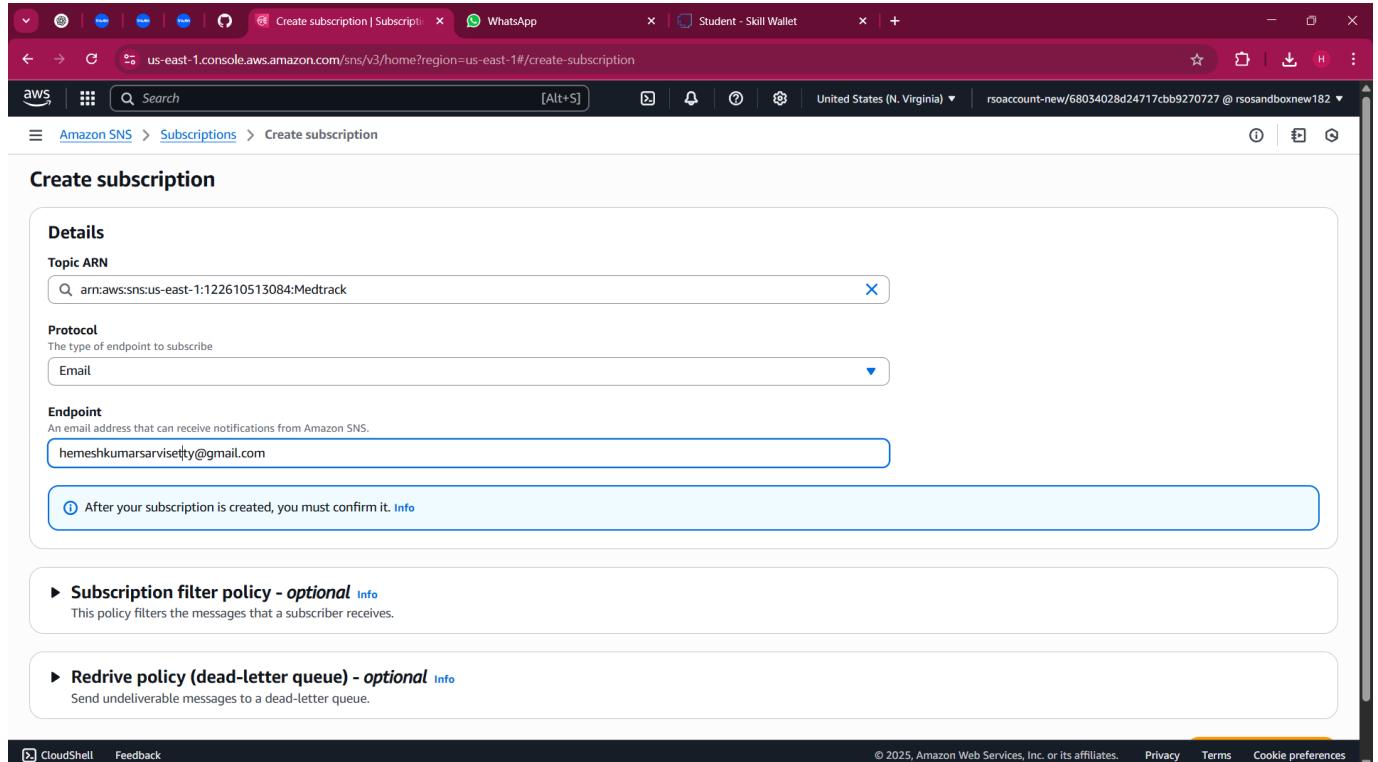
- Name:** Medtrack
- Display name:** (empty)
- ARN:** arn:aws:sns:us-east-1:515966514202:Medtrack
- Topic owner:** 515966514202
- Type:** Standard

The 'Subscriptions' tab is selected, showing a table with the following columns: ID, Endpoint, Status, and Protocol. The table displays the message: "No subscriptions found".

At the bottom of the page, there are links for CloudShell, Feedback, Copyright notice (© 2025, Amazon Web Services, Inc. or its affiliates.), Privacy, Terms, and Cookie preferences.

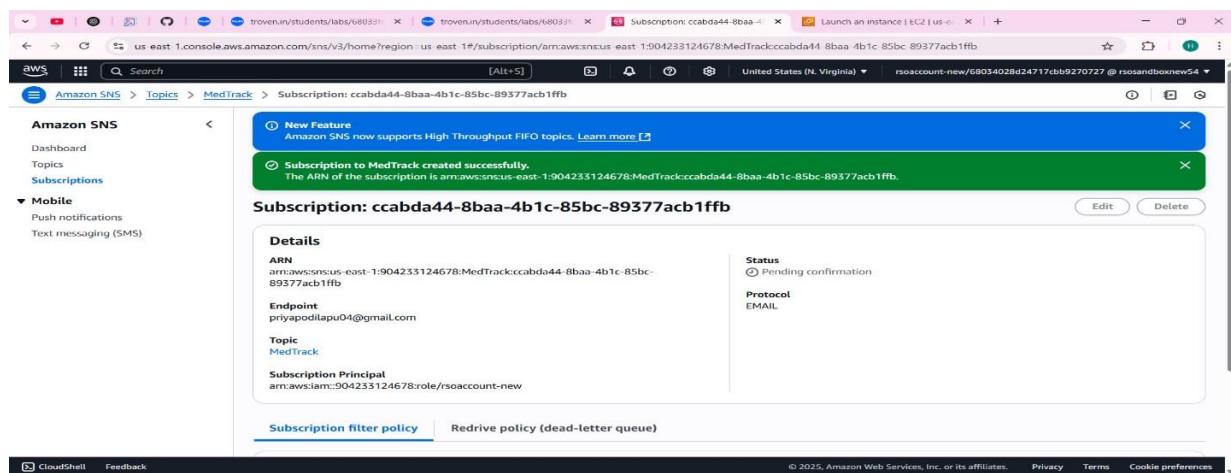
- **Activity 4.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

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



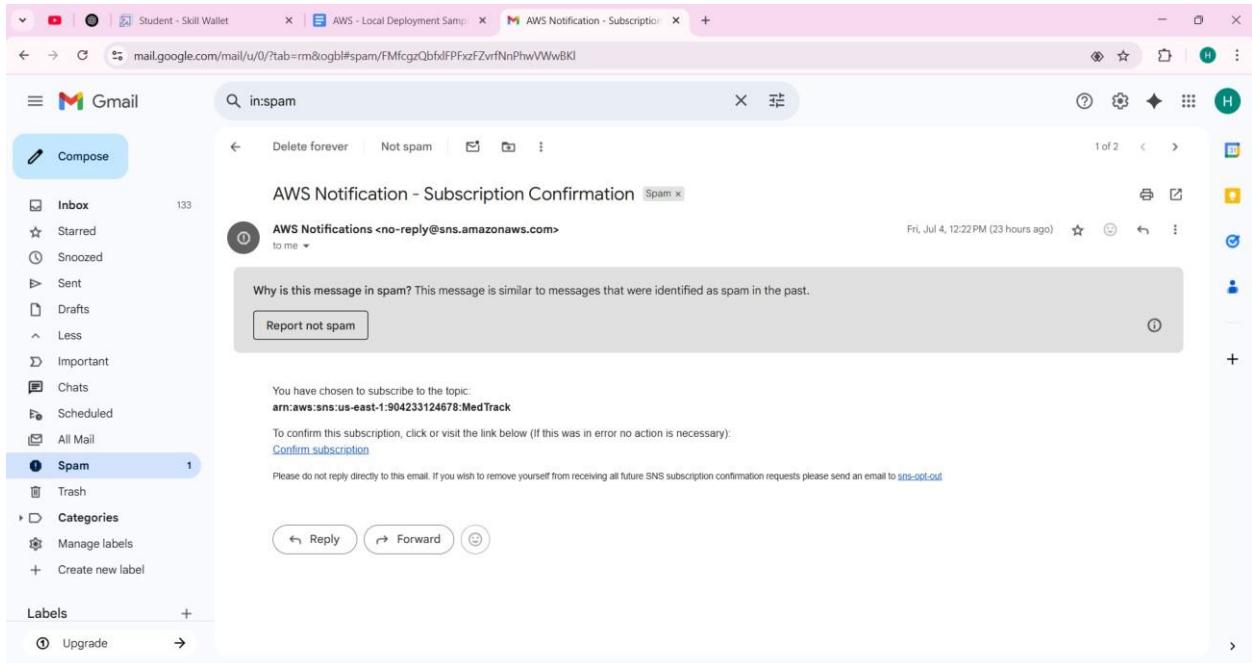
The screenshot shows the 'Create subscription' page in the AWS SNS console. The 'Topic ARN' field contains 'arn:aws:sns:us-east-1:122610513084:Medtrack'. The 'Protocol' dropdown is set to 'Email'. The 'Endpoint' field contains 'hemeshkumarsarvisetty@gmail.com'. A note at the bottom says 'After your subscription is created, you must confirm it.' Below this, there are sections for 'Subscription filter policy - optional' and 'Redrive policy (dead-letter queue) - optional', both of which are currently empty.

- After subscription request for the mail confirmation

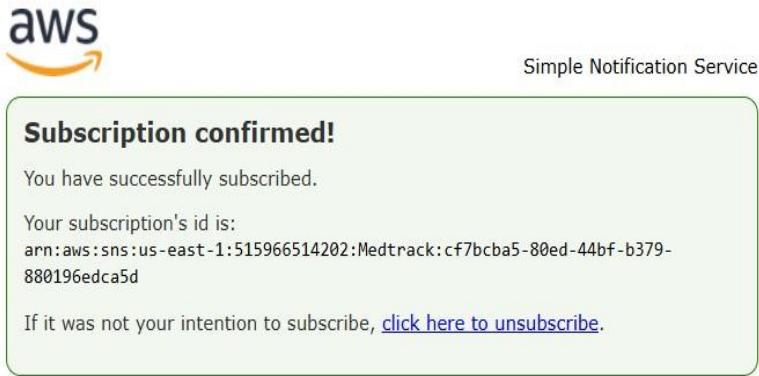


The screenshot shows the 'Topics' page in the AWS SNS console. It displays a successful subscription creation message: 'Subscription: ccabda44-8baa-4b1c-85bc-89377acb1ffb' was created successfully. The ARN is listed as 'arn:aws:sns:us-east-1:1904233124678:MedTrack:ccabda44-8baa-4b1c-85bc-89377acb1ffb'. The details panel shows the ARN, endpoint ('priyapodilapu04@gmail.com'), topic ('MedTrack'), and subscription principal ('arn:aws:iam::1904233124678:role/rsoaccount-new'). The status is 'Pending confirmation' and the protocol is 'EMAIL'.

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

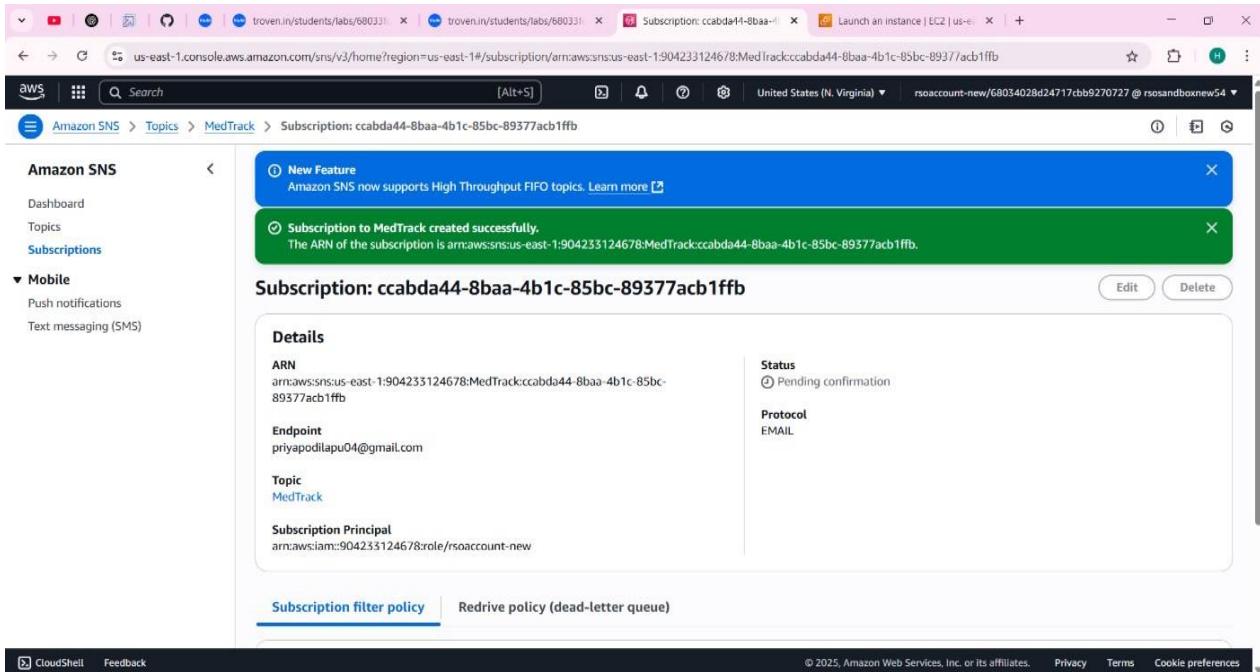


The screenshot shows a Gmail inbox with the search term "in:spam". There is one unread message from "AWS Notifications <no-reply@sns.amazonaws.com> to me" dated Fri, Jul 4, 12:22 PM (23 hours ago). The subject is "AWS Notification - Subscription Confirmation". The message is marked as spam and has a "Report not spam" button. The message content includes a topic ID "arn:aws:sns:us-east-1:904233124678:MedTrack" and a "Confirm subscription" link. The Gmail interface shows various navigation and search tools.

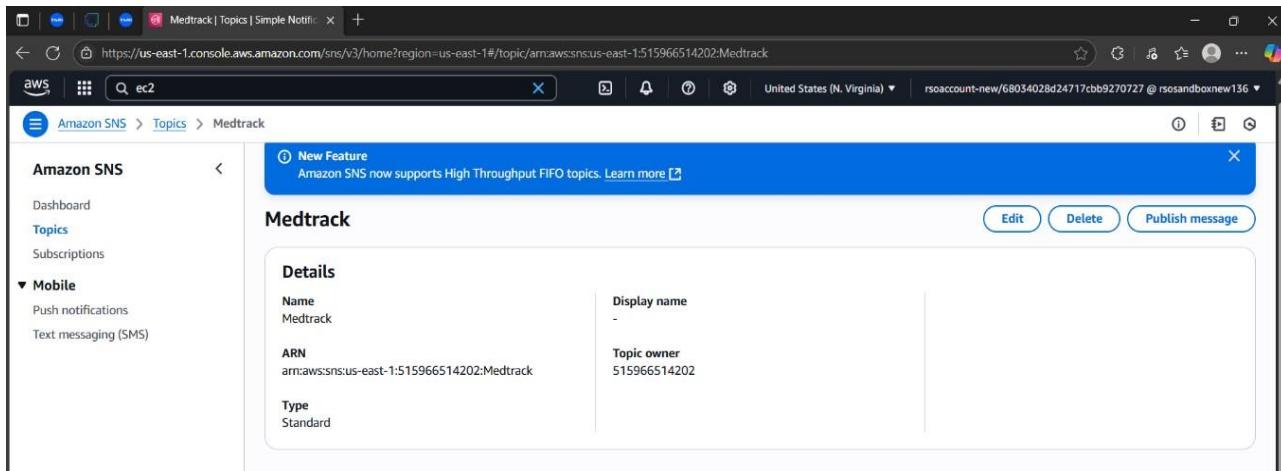


The screenshot shows an AWS Simple Notification Service (SNS) confirmation email. The subject is "Subscription confirmed!". The body of the email states: "You have successfully subscribed. Your subscription's id is: arn:aws:sns:us-east-1:515966514202:Medtrack:cf7bcba5-80ed-44bf-b379-880196edca5d. If 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 under the 'Topics' section. A new subscription has been created for the topic 'MedTrack'. The ARN of the subscription is listed as `arn:aws:sns:us-east-1:904233124678:MedTrack:ccabda44-8baa-4b1c-85bc-89377acb1ffb`. The status is shown as 'Pending confirmation'. The endpoint is set to `priyapodilapu04@gmail.com`. The protocol is set to 'EMAIL'.

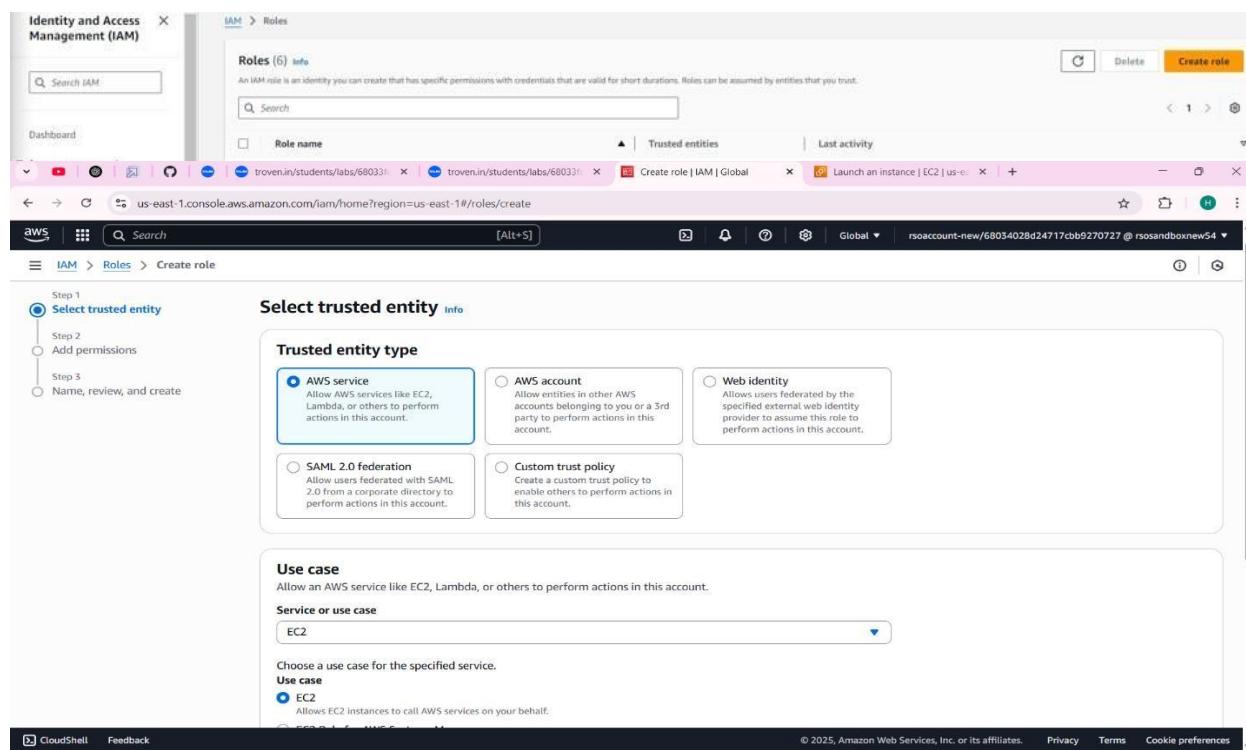
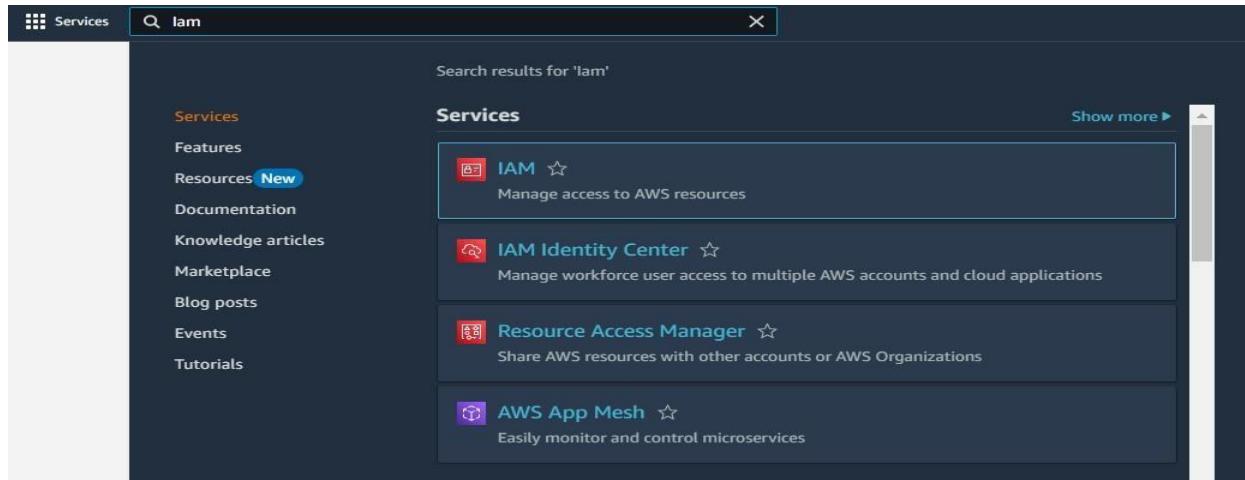


The screenshot shows the AWS SNS console under the 'Topics' section. The topic 'Medtrack' is selected. The topic details are displayed, including the name 'Medtrack', ARN `arn:aws sns:us-east-1:515966514202:Medtrack`, and type 'Standard'. The topic owner is listed as `515966514202`.

Milestone 5: IAM Role Setup

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

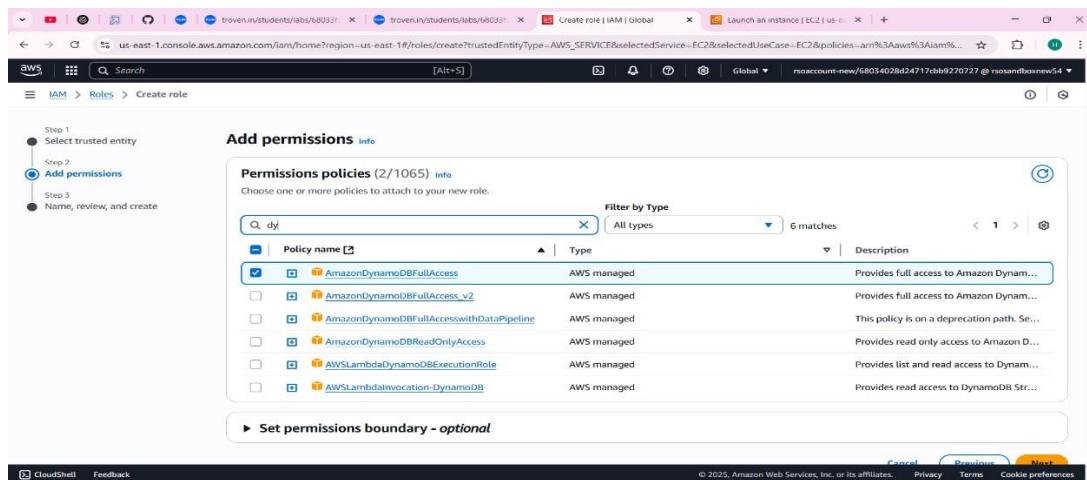
- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with Dynamo db and SNS.



- **Activity 5.2: Attach Policies.**

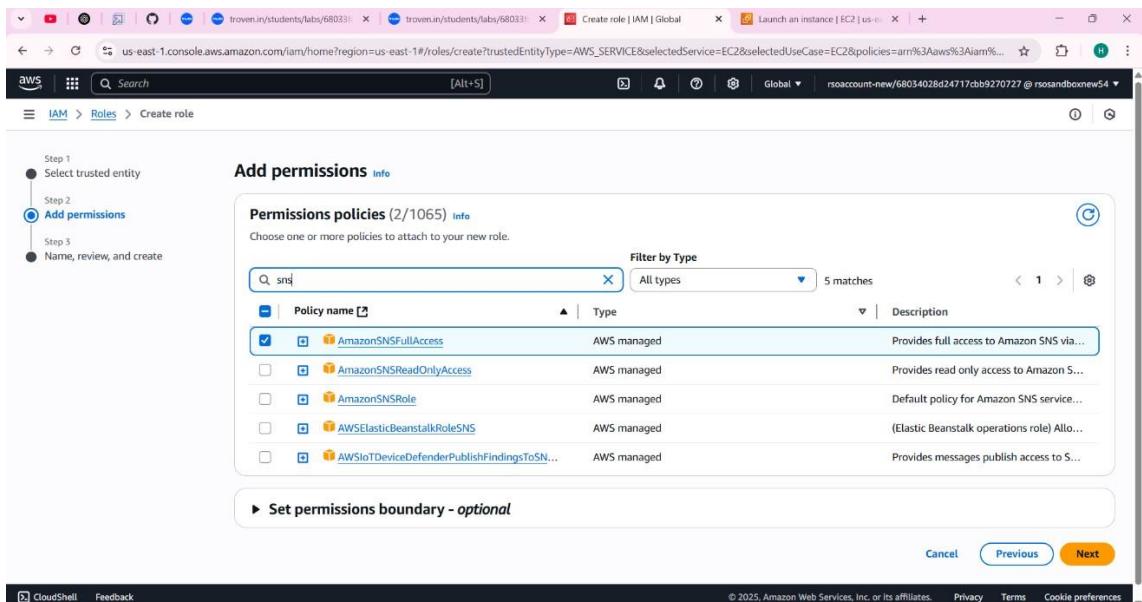
Attach the following policies to the role:

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



The screenshot shows the 'Add permissions' step of the IAM role creation wizard. The search bar at the top has 'dj' typed into it. Below the search bar, there is a table of permissions policies. One row is highlighted with a blue border, indicating it is selected. The table includes columns for Policy name, Type, and Description.

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...



The screenshot shows the 'Add permissions' step of the IAM role creation wizard. The search bar at the top has 'sns' typed into it. Below the search bar, there is a table of permissions policies. One row is highlighted with a blue border, indicating it is selected. The table includes columns for Policy name, Type, and Description.

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"/>  AWSLambdaBeanstalkRoleSN...	AWS managed	(Elastic Beanstalk operations role) Allo...
<input type="checkbox"/>  AWSIoTDeviceDefenderPublishFindingsToSN...	AWS managed	Provides messages publish access to S...

us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/create?trustedEntityType=AWS_SERVICE&selectedService=EC2&policies=arn%3Aaws%3Aiam%... rsoaccount-new/68034028d24717cbb9270727 @ rsosandboxnew54

IAM > Roles > Create role

Name, review, and create

Step 1 Select trusted entity

Step 2 Add permissions

Step 3 Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
EC2_MedTrack_Role

Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy

```

1 - [
2 -   "Version": "2012-10-17",
3 -   "Statement": [
4 -     {
5 -       "Effect": "Allow",
6 -       "Action": [
7 -         "sts:AssumeRole"

```

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

us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/create?trustedEntityType=AWS_SERVICE&selectedService=EC2&policies=arn%3Aaws%3Aiam%... rsoaccount-new/68034028d24717cbb9270727 @ rsosandboxnew54

IAM > Roles > Create role

Step 2: Add permissions

Permissions policy summary

Policy name	Type	Attached as
AmazonDynamoDBFullAccess	AWS managed	Permissions policy
AmazonSNSFullAccess	AWS managed	Permissions policy

Step 3: Add tags

Add tags - optional Info

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

Add new tag

You can add up to 50 more tags.

Cancel Previous Create role

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

Milestone 6: EC2 Instance Setup

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

 S-hemeshkumar	Add files via upload	b42589c · now	 7 Commits
 templates	Add files via upload	5 days ago	
 .env	Update .env	5 days ago	
 Demo vedio.txt	Add files via upload	now	
 app.py	Update app.py	5 days ago	

Local Codespaces

 **Clone** 

HTTPS SSH GitHub CLI

<https://github.com/S-hemeshkumar/Medtrack.git> 

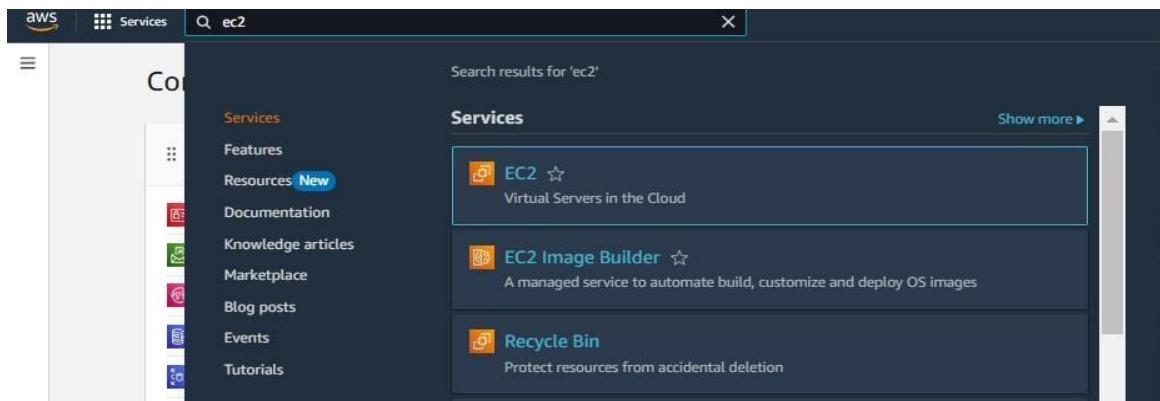
Clone using the web URL.

 Open with GitHub Desktop

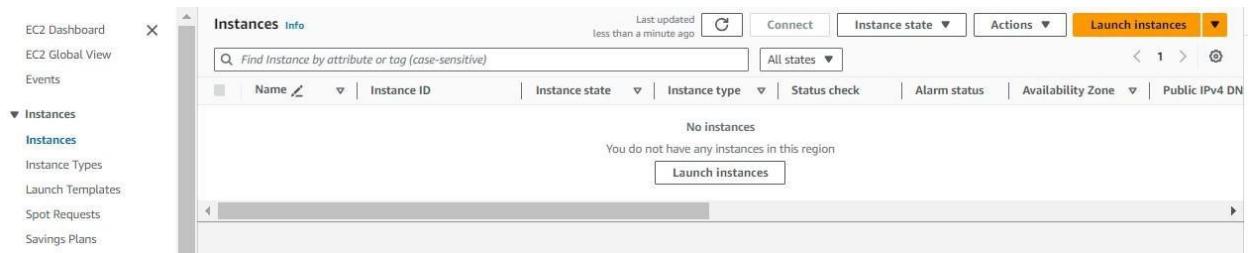
 Download ZIP

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

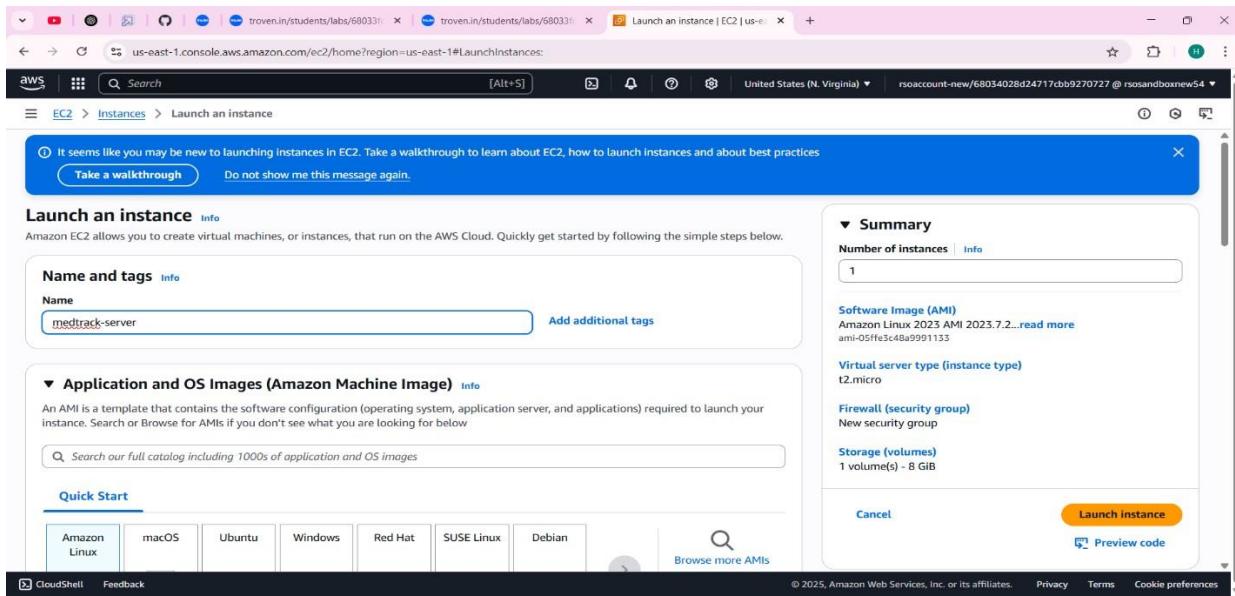
Launch EC2 Instance ○ In the AWS Console, navigate to EC2 and launch a new instance.



○ Click on Launch instance to launch EC2 instance

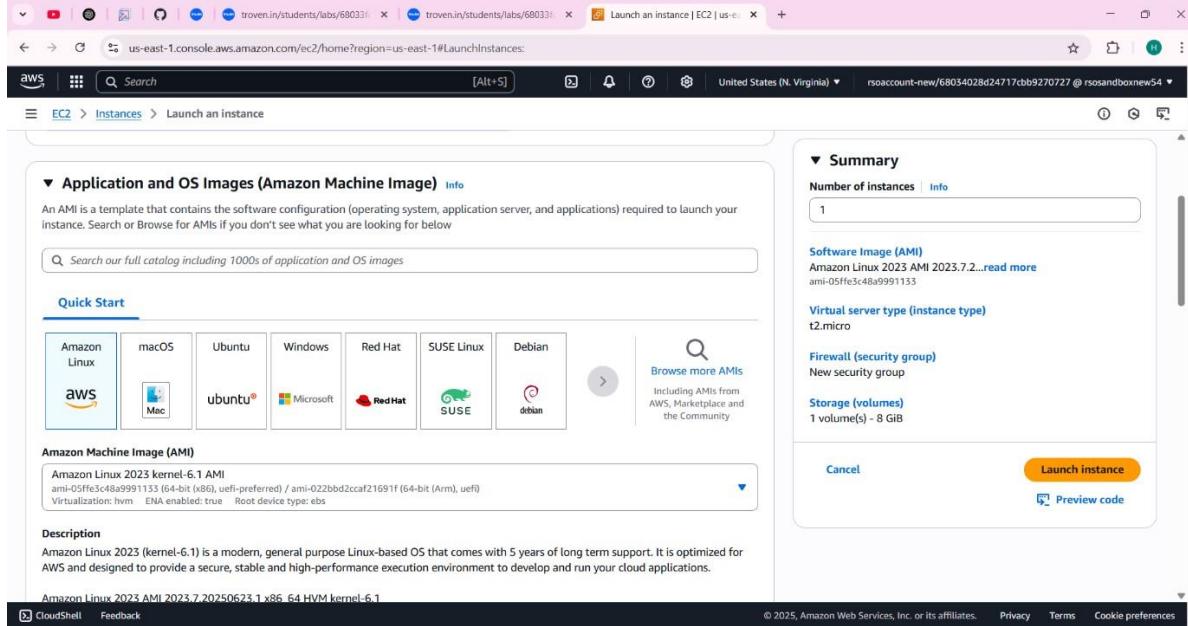


○ Create an instance with the name medtrack-server



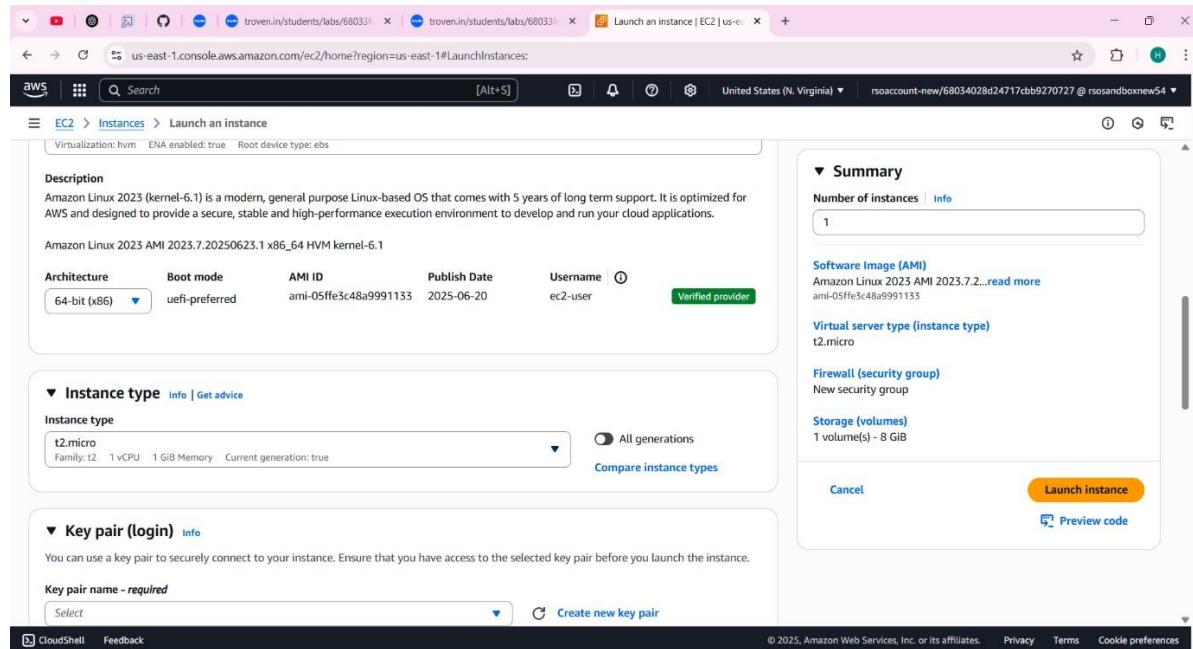
The screenshot shows the 'Launch an instance' wizard on the AWS EC2 console. The current step is 'Name and tags'. A message at the top says: 'It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices' with 'Take a walkthrough' and 'Do not show me this message again.' buttons. Below this, the 'Launch an instance' section has a sub-section 'Name and tags' where the name 'medtrack-server' is entered. To the right, the 'Summary' section shows: Number of instances (1), Software Image (AMI) as Amazon Linux 2023 AMI 2023.7.2..., Virtual server type (instance type) as t2.micro, Firewall (security group) as New security group, Storage (volumes) as 1 volume(s) - 8 GiB, and a 'Launch instance' button.

- Choose Amazon Linux 2 and choose Amazon Linux 2023 as the AMI and t2.micro as the instance type (free-tier eligible).



The screenshot shows the 'Launch an instance' wizard on the AWS EC2 console. The current step is 'Application and OS Images (Amazon Machine Image)'. It lists various OS options: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian. Below this, the 'Amazon Machine Image (AMI)' section shows 'Amazon Linux 2023 kernel-6.1 AMI' (ami-05ffe5c48a9991133) selected. The 'Description' section provides details: 'Amazon Linux 2023 (kernel-6.1) 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.' At the bottom, it shows 'Amazon Linux 2023 AMI 2023.7.20250623.1 x86_64 HVM kernel-6.1'.

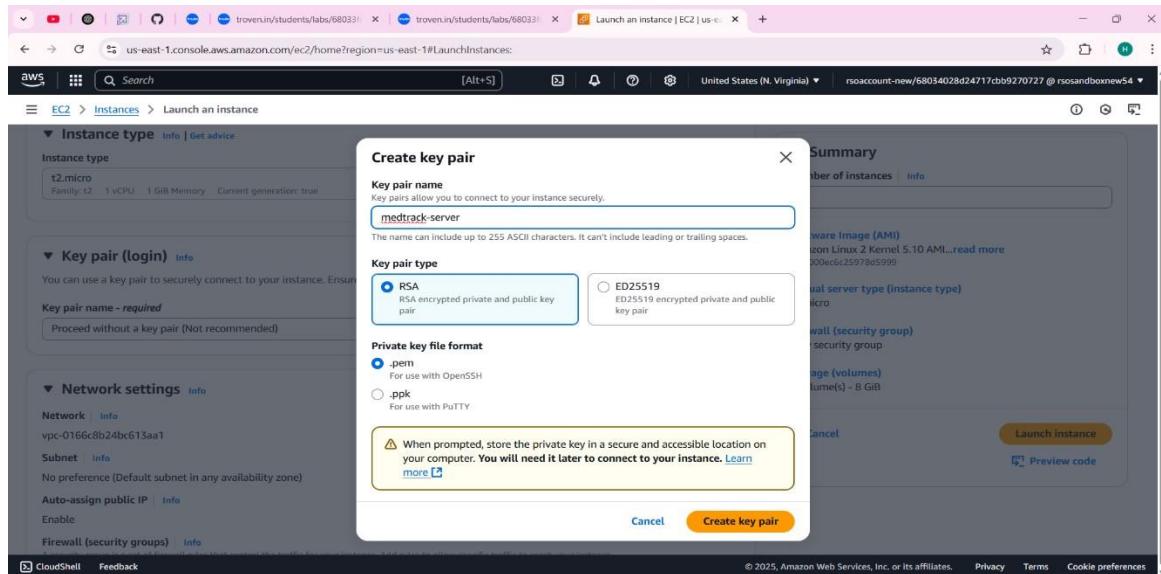
- Create and download the key pair for Server access.



The screenshot shows the AWS EC2 'Launch an instance' wizard. The 'Summary' step is displayed, showing the following details:

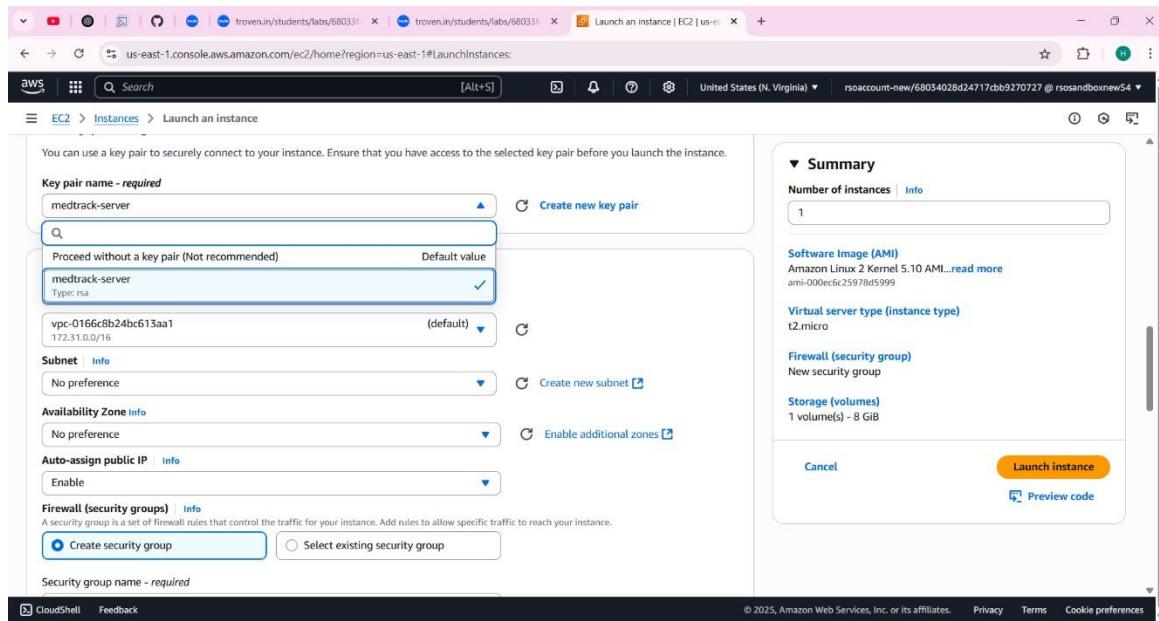
- Number of instances:** 1
- Software Image (AMI):** Amazon Linux 2023 AMI 2023.7.20250623.1 x86_64 HVM kernel-6.1
- Virtual server type (instance type):** t2.micro
- Storage (volumes):** 1 volume(s) - 8 GiB
- Launch instance** button

- Create a Key pair as RSA and only and select that Key pair.



The screenshot shows the AWS EC2 'Launch an instance' wizard. The 'Create key pair' step is displayed, showing the following details:

- Key pair name:** medtrack-server
- Key pair type:** RSA (selected)
- Private key file format:** pem (selected)
- When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance.**
- Summary** section shows the same instance configuration as the previous screenshot.



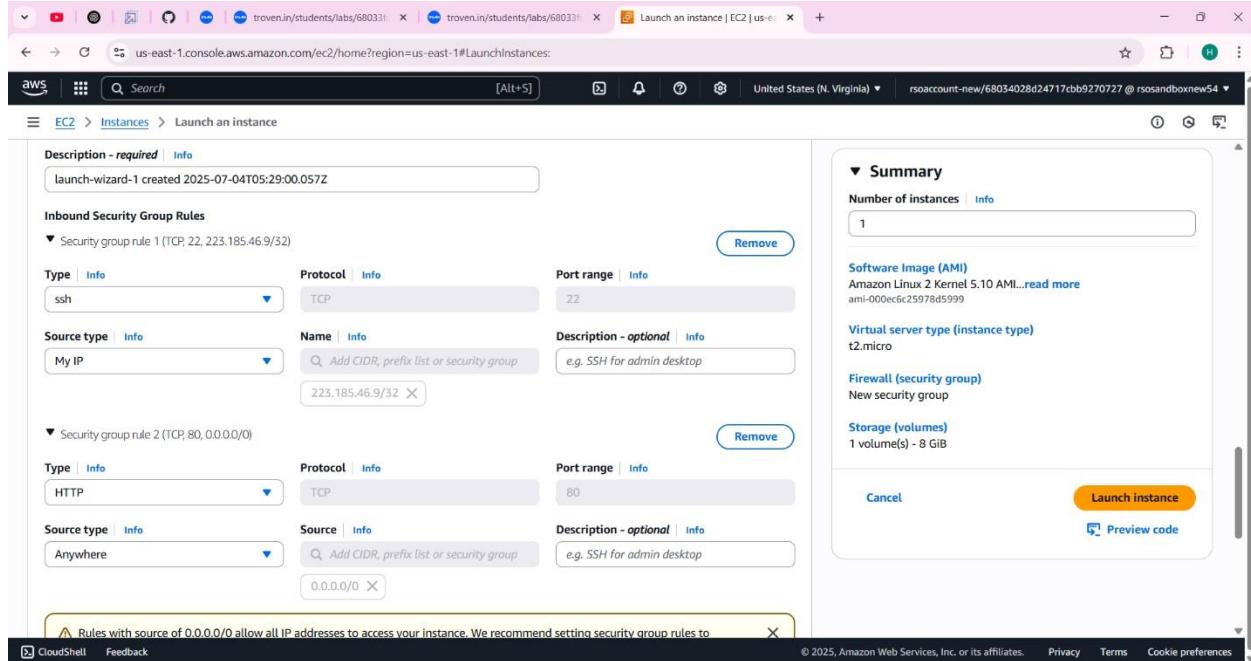
The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The configuration includes:

- Key pair name - required:** medtrack-server
- Proceed without a key pair (Not recommended):** Default value (medtrack-server)
- Subnet:** vpc-0166c8b24bc613aa1 (default)
- Availability Zone Info:** No preference
- Auto-assign public IP:** Enabled
- Firewall (security groups):** Create security group (selected)
- Summary:** Number of instances: 1, Software Image (AMI): Amazon Linux 2 Kernel 5.10 AMI..., Virtual server type (instance type): t2.micro, Firewall (security group): New security group, Storage (volumes): 1 volume(s) - 8 GiB

At the bottom right are 'Launch instance' and 'Preview code' buttons.

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

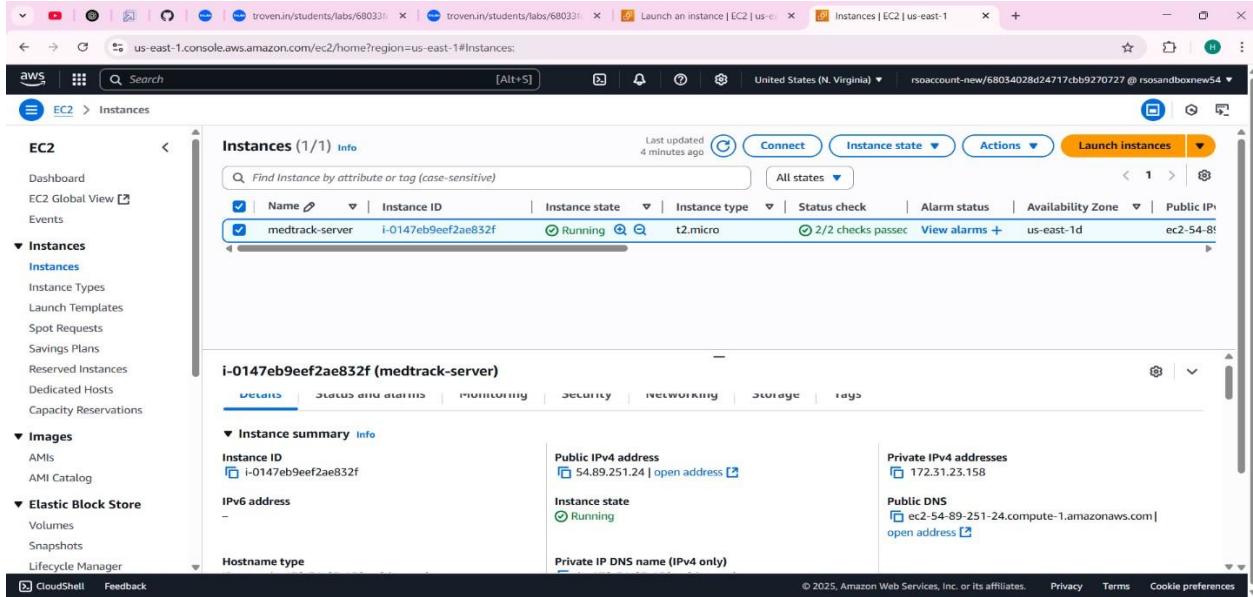
In network settings configure security groups for http – port range 80(type Anywhere) and ssh – port range 22 (type MY IP) and click on launch instance.



The screenshot shows the 'Launch an instance' wizard with two inbound security group rules added:

- Inbound Security Group Rules:**
 - Security group rule 1 (TCP, 22, 22.185.46.9/32):** Type: ssh, Protocol: TCP, Port range: 22, Source type: My IP, Description: e.g. SSH for admin desktop
 - Security group rule 2 (TCP, 80, 0.0.0.0/0):** Type: HTTP, Protocol: TCP, Port range: 80, Source type: Anywhere, Description: e.g. SSH for admin desktop

A warning message at the bottom states: "⚠️ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to..."



The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar for EC2, including links for Dashboard, EC2 Global View, Events, Instances (which is selected), Images, and Elastic Block Store. The main content area shows a table of instances with one entry:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
medtrack-server	i-0147eb9eef2ae832f	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1d	ec2-54-81-

Below the table, there's a detailed view for the selected instance (i-0147eb9eef2ae832f). It shows the following details:

- Details:** Instance ID: i-0147eb9eef2ae832f, Public IPv4 address: 54.89.251.24, Instance state: Running, Hostname type: -.
- Public IPv4 address:** 54.89.251.24 | [open address](#)
- Private IPv4 addresses:** 172.31.23.158
- Public DNS:** ec2-54-89-251-24.compute-1.amazonaws.com | [open address](#)

- 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.

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#instancesinstanceState=running

Instances (1) Info

EC2 Instances

Instances (1)

Name	Instance ID	Instance state	Instance type
medtrack-server	i-0147eb9eef2ae832f	Running	t2.micro

Actions ▾ Launch instances ▾

Instance diagnostics
Instance settings
Networking
Security ▾ Public IP
Get Windows password
Modify IAM role
Image and templates ▾ ec2-54-81-13-138
Monitor and troubleshoot

CloudShell Feedback

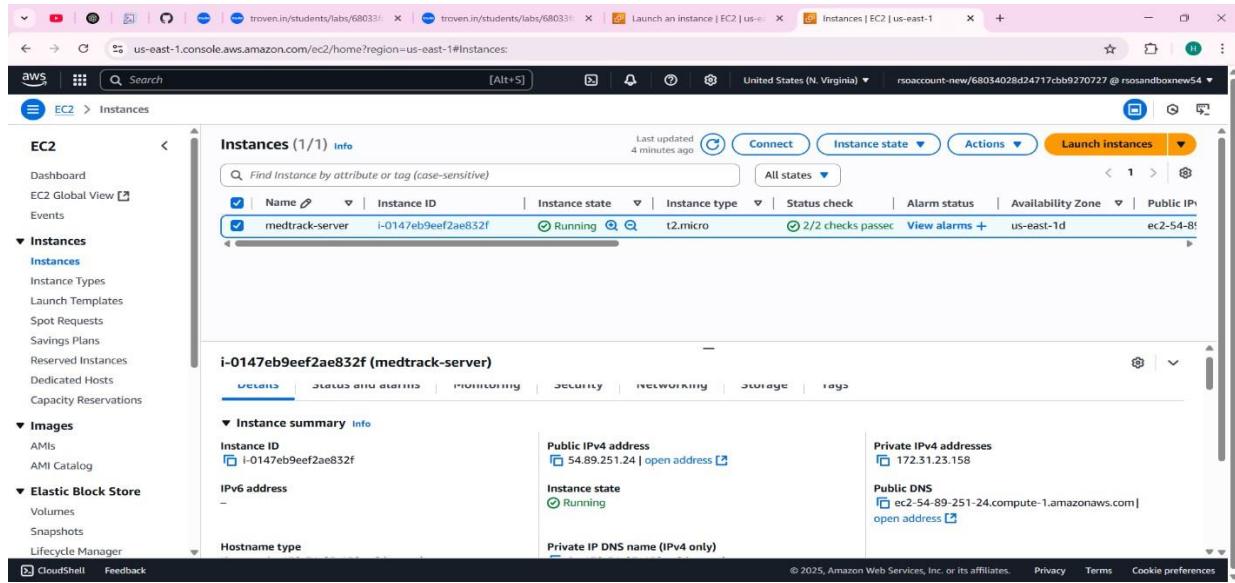
IAM role

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

EC2_MedTrack_Role

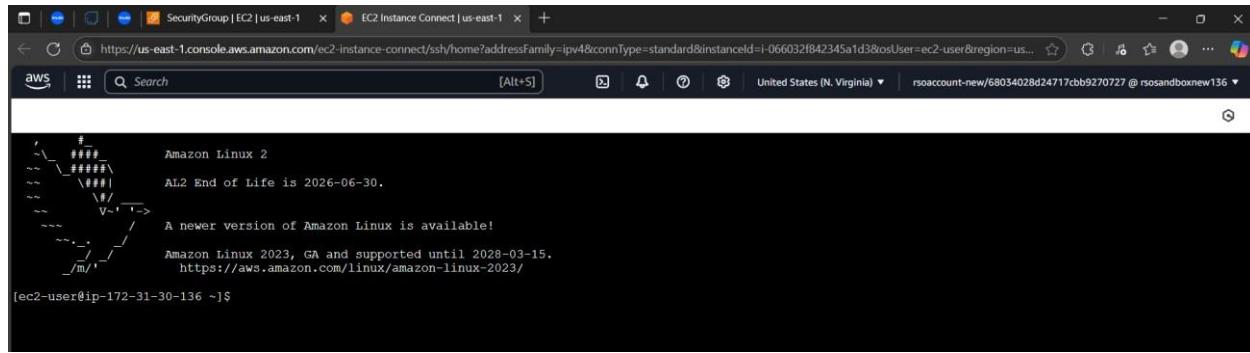
Create new IAM role

Cancel Update IAM role



The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with options like Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, AMI Catalog, and Elastic Block Store. The main area displays a table for 'Instances (1/1) Info'. The table has columns for Name (medtrack-server), Instance ID (i-0147eb9eef2ae832f), Instance state (Running), Instance type (t2.micro), Status check (2/2 checks passed), Availability Zone (us-east-1d), and Public IP (ec2-54-89-251-24). Below the table, there's a detailed view for the instance 'i-0147eb9eef2ae832f (medtrack-server)' with tabs for Details, Status and Metrics, Monitoring, Security, Networking, Storage, and Tags. Under the Details tab, it shows Instance ID (i-0147eb9eef2ae832f), IPv6 address (-), Hostname type (IPv4 only), Public IPv4 address (54.89.251.24), Instance state (Running), and Private IP DNS name (IPV4 only) (ec2-54-89-251-24.compute-1.amazonaws.com).

- Now connect the EC2 with the files



Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux2:

```
sudo yum update -y
```

```
sudo yum install python3 git
```

```
sudo pip3 install flask boto3
```

Verify

Installations: flask –version

git --version

Activity 7.2: Clone Your Flask Project from GitHub

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

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

Note: change your-github-username and your-repository-name with your credentials here: 'git clone https://github.com/Harshapriya04/Medtrack.git'

- This will download your project to the EC2 instance.

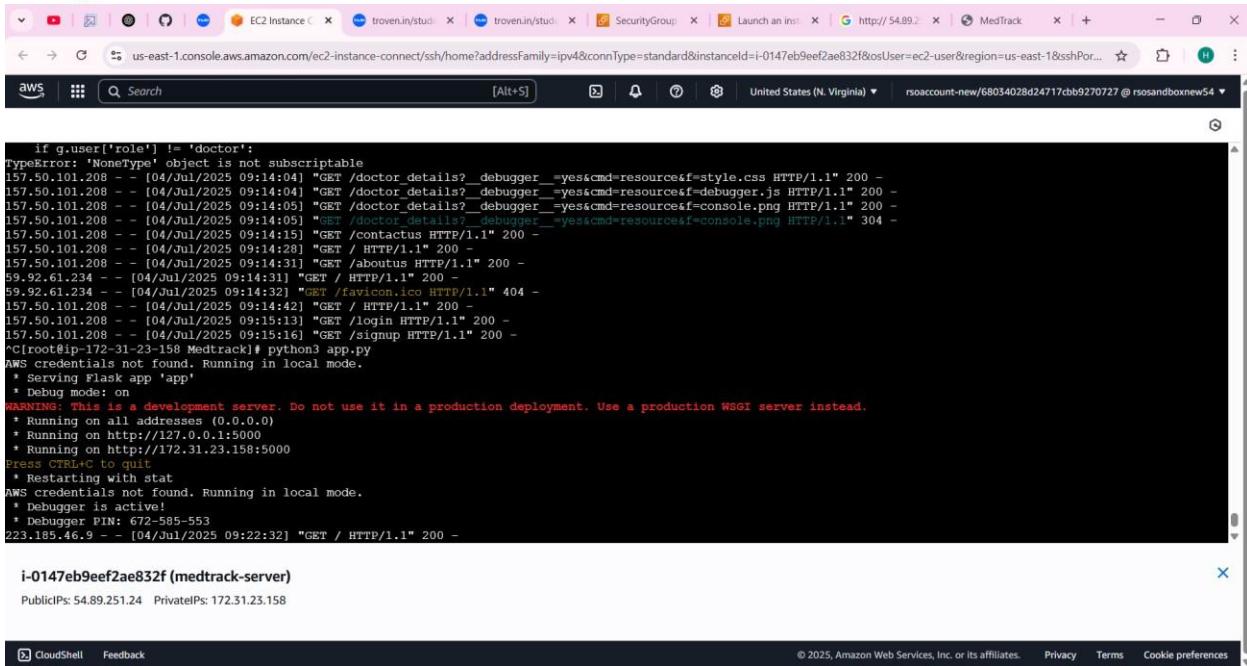
To navigate to the project directory, run the following command:

cd Medtrack

Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application

sudo flask run --host=0.0.0.0 --port=80



```

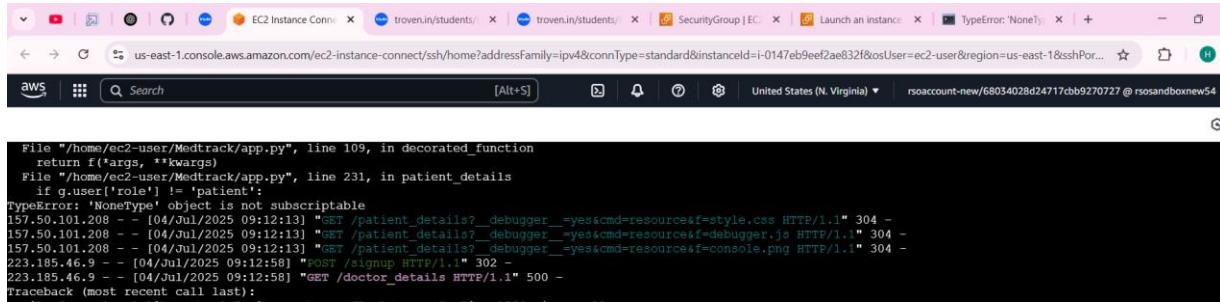
if g.user['role'] != 'doctor':
TypeError: 'NoneType' object is not subscriptable
157.50.101.208 - [04/Jul/2025 09:14:04] "GET /doctor_details? _debugger_=yes&cmd=resource&f=style.css HTTP/1.1" 200 -
157.50.101.208 - [04/Jul/2025 09:14:04] "GET /doctor_details? _debugger_=yes&cmd=resource&f=debugger.js HTTP/1.1" 200 -
157.50.101.208 - [04/Jul/2025 09:14:05] "GET /doctor_details? _debugger_=yes&cmd=resource&f=console.png HTTP/1.1" 200 -
157.50.101.208 - [04/Jul/2025 09:14:05] "GET /doctor_details? _debugger_=yes&cmd=resource&f=console.png HTTP/1.1" 304 -
157.50.101.208 - [04/Jul/2025 09:14:15] "GET /contactus HTTP/1.1" 200 -
157.50.101.208 - [04/Jul/2025 09:14:28] "GET / HTTP/1.1" 200 -
157.50.101.208 - [04/Jul/2025 09:14:31] "GET /aboutus HTTP/1.1" 200 -
59.92.61.234 - [04/Jul/2025 09:14:31] "GET /favicon.ico HTTP/1.1" 404 -
59.92.61.234 - [04/Jul/2025 09:14:32] "GET /favicon.ico HTTP/1.1" 404 -
157.50.101.208 - [04/Jul/2025 09:14:42] "GET / HTTP/1.1" 200 -
157.50.101.208 - [04/Jul/2025 09:15:13] "GET /login HTTP/1.1" 200 -
157.50.101.208 - [04/Jul/2025 09:15:16] "GET /signup HTTP/1.1" 200 -
^Croot@ip-172-31-23-158 Medtrack]# python3 app.py
AWS credentials not found. Running in local mode.
* Serving Flask app 'app'
* Debug mode: on
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://172.31.23.158:5000
Press CTRL+C to quit
* Restarting with stat
AWS credentials not found. Running in local mode.
* Debugger is active!
* Debugger PIN: 672-585-553
223.185.46.9 - [04/Jul/2025 09:22:32] "GET / HTTP/1.1" 200 -
i-0147eb9eff2ae832f (medtrack-server)
PublicIPs: 54.89.251.24 PrivateIPs: 172.31.23.158

```

Verify the Flask app is running: <http://your-ec2-public-ip>

public-ip

- Run the Flask app on the EC2 instance



```

File "/home/ec2-user/Medtrack/app.py", line 109, in decorated_function
    return f(*args, **kwargs)
File "/home/ec2-user/Medtrack/app.py", line 231, in patient_details
    if g.user['role'] != 'patient':
TypeError: 'NoneType' object is not subscriptable
157.50.101.208 - [04/Jul/2025 09:12:13] "GET /patient_details? _debugger_=yes&cmd=resource&f=style.css HTTP/1.1" 304 -
157.50.101.208 - [04/Jul/2025 09:12:13] "GET /patient_details? _debugger_=yes&cmd=resource&f=debugger.js HTTP/1.1" 304 -
157.50.101.208 - [04/Jul/2025 09:12:13] "GET /patient_details? _debugger_=yes&cmd=resource&f=console.png HTTP/1.1" 304 -
223.185.46.9 - [04/Jul/2025 09:12:58] "POST /signup HTTP/1.1" 302 -
223.185.46.9 - [04/Jul/2025 09:12:58] "GET /doctor_details HTTP/1.1" 500 -
Traceback (most recent call last):

```

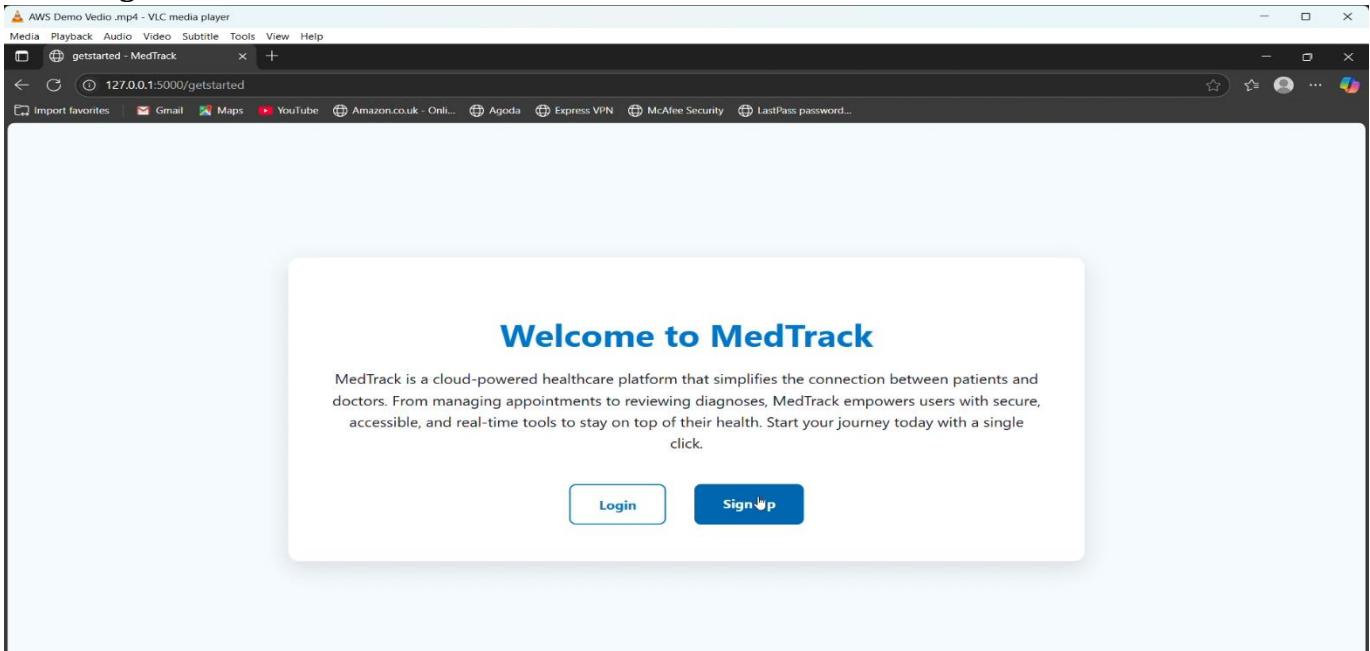
Access the website through:

PublicIPs: <http://54.89.251.24:5000/>

Milestone 8: Testing and Deployment

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

Index Page:



Signup Page:

Sign Up - MedTrack

127.0.0.1:5000/signup

Create an Account

Patient Doctor

Full Name

Email Address

Password

Confirm Password

Age

Gender

--Select--

Sign Up - MedTrack

127.0.0.1:5000/signup

Full Name

Email Address

Password

Confirm Password

Age

Gender

--Select--

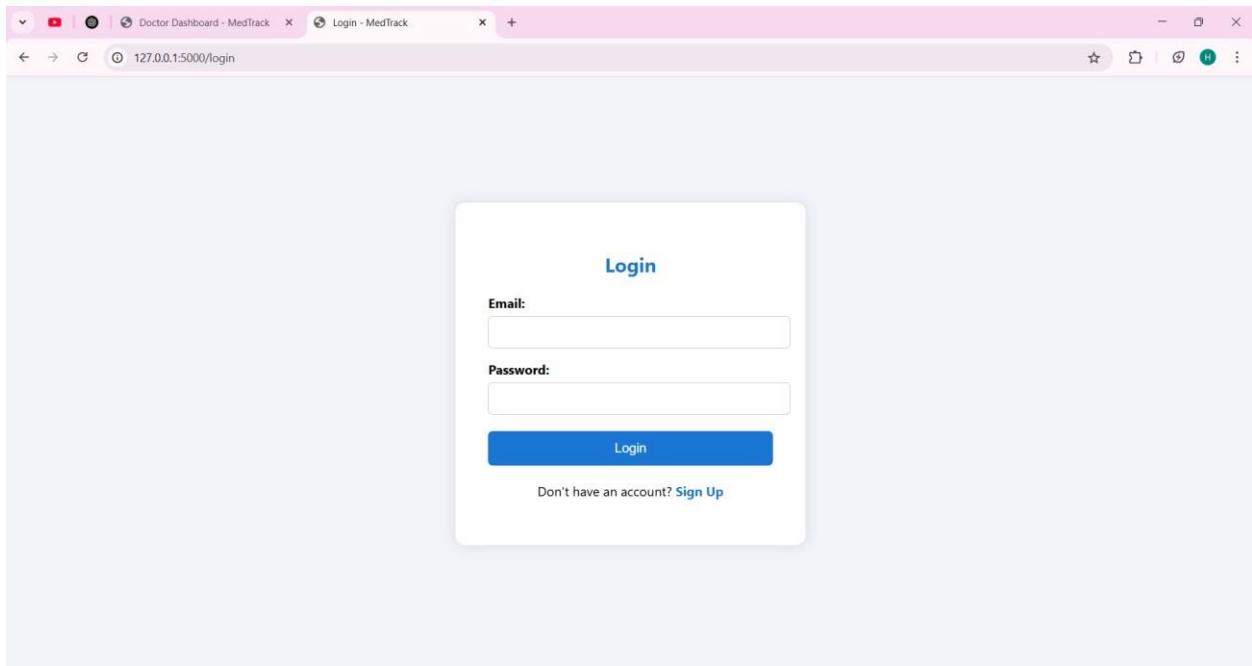
Specialization

--Select--

Only required if you are registering as a Doctor.

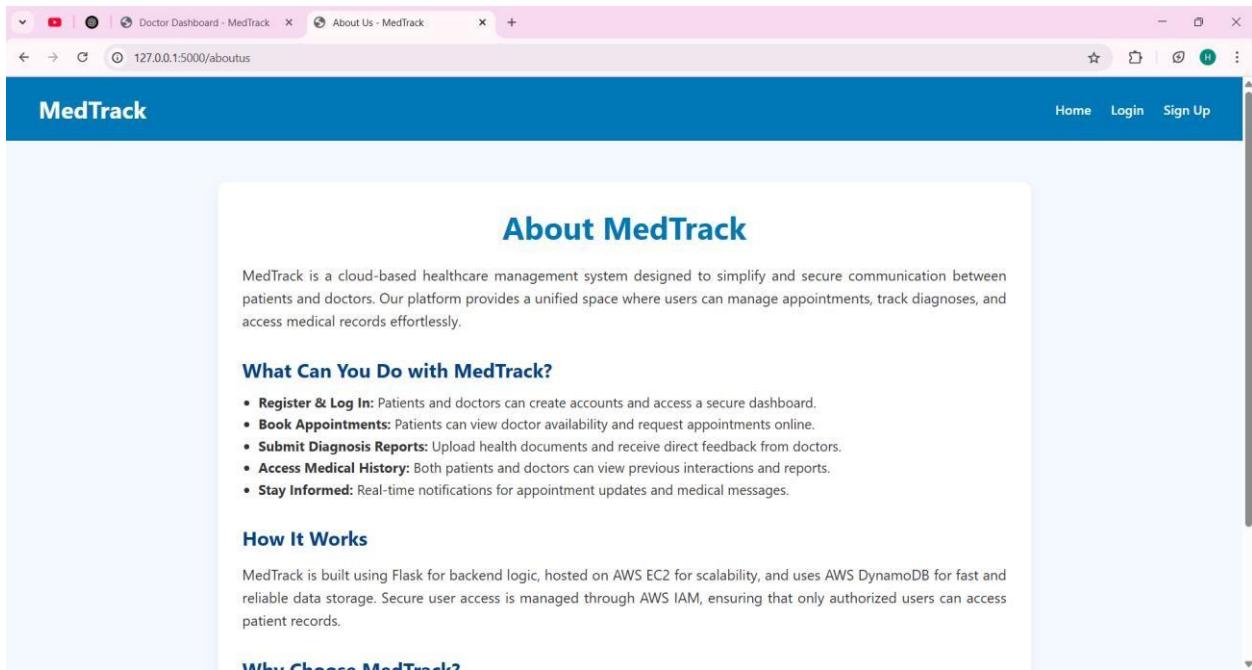
Sign Up

Login page:



The screenshot shows a web browser window with a pink header bar. The address bar displays '127.0.0.1:5000/login'. The main content area contains a light gray 'Login' form. The form has two input fields: 'Email:' and 'Password:', both with placeholder text. Below the fields is a large blue rectangular button with the word 'Login' in white. At the bottom of the form, there is a small link 'Don't have an account? [Sign Up](#)'.

About Us page:



The screenshot shows a web browser window with a pink header bar. The address bar displays '127.0.0.1:5000/aboutus'. The main content area has a dark blue header with the 'MedTrack' logo on the left and 'Home', 'Login', and 'Sign Up' links on the right. Below the header is a white content area. The first section is titled 'About MedTrack' in bold blue text. It contains a paragraph about the system's purpose. Below that is a section titled 'What Can You Do with MedTrack?' with a bulleted list of five features. At the bottom is a section titled 'How It Works' with a detailed paragraph about the system's architecture and data storage.



The screenshot shows a web browser window with two tabs open: 'Doctor Dashboard - MedTrack' and 'About Us - MedTrack'. The current view is the 'About Us' page at the URL 127.0.0.1:5000/aboutus. The page content includes an introduction to MedTrack, a section titled 'What Can You Do with MedTrack?' listing features like account creation, appointment booking, and diagnosis submission, and a 'How It Works' section explaining the technology stack (Flask, AWS EC2, AWS DynamoDB, AWS IAM). There is also a 'Why Choose MedTrack?' section highlighting the goal of removing paperwork and bringing healthcare into the cloud era.

MedTrack is a cloud-based healthcare management system designed to simplify and secure communication between patients and doctors. Our platform provides a unified space where users can manage appointments, track diagnoses, and access medical records effortlessly.

What Can You Do with MedTrack?

- **Register & Log In:** Patients and doctors can create accounts and access a secure dashboard.
- **Book Appointments:** Patients can view doctor availability and request appointments online.
- **Submit Diagnosis Reports:** Upload health documents and receive direct feedback from doctors.
- **Access Medical History:** Both patients and doctors can view previous interactions and reports.
- **Stay Informed:** Real-time notifications for appointment updates and medical messages.

How It Works

MedTrack is built using Flask for backend logic, hosted on AWS EC2 for scalability, and uses AWS DynamoDB for fast and reliable data storage. Secure user access is managed through AWS IAM, ensuring that only authorized users can access patient records.

Why Choose MedTrack?

We aim to remove paperwork, eliminate delays, and bring healthcare into the cloud era. Whether you're managing your own health or helping others, MedTrack provides the tools and access you need—all in one place.

Contactus Page:

The screenshot shows a web browser window with two tabs open: 'Doctor Dashboard - MedTrack' and 'Contact Us - MedTrack'. The current view is the 'Contact Us' page at the URL 127.0.0.1:5000/contactus. The page has a blue header bar with 'Contact Us' on the left and a 'Back to Home' button on the right. The main content area features a message 'We'd love to hear from you!' and instructions for reaching out via official website, email, or phone.

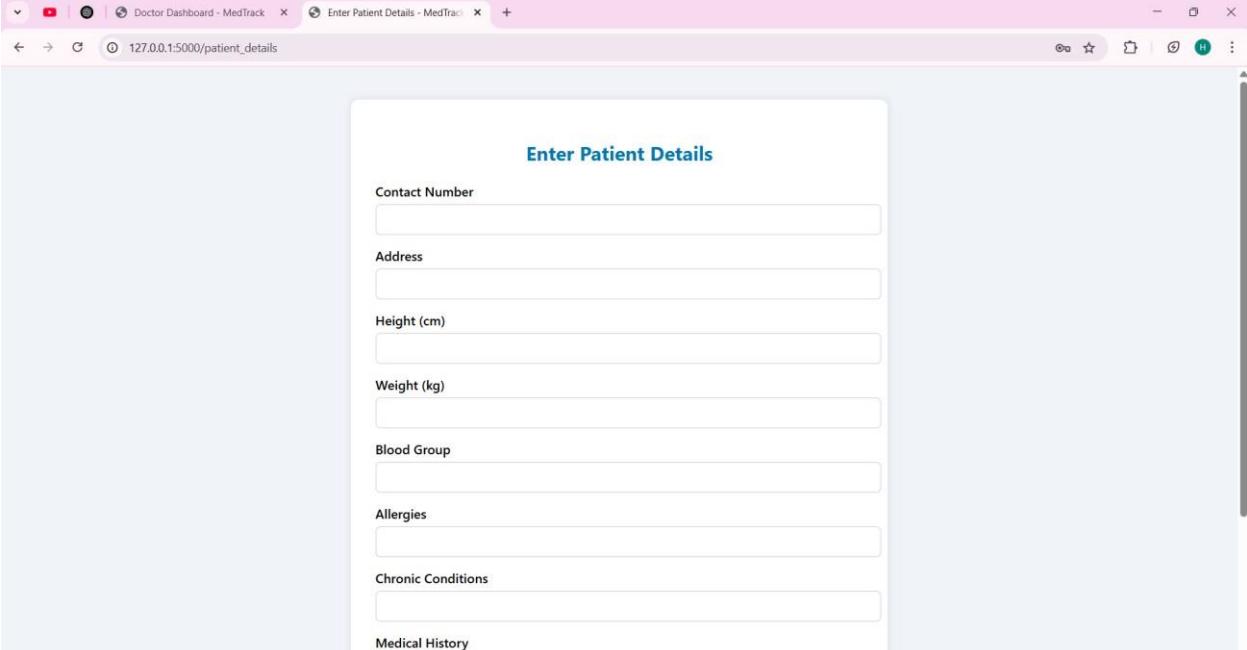
We'd love to hear from you!

If you have questions about MedTrack, feedback, or need technical support, feel free to reach out.

Official Website:
<http://127.0.0.1:5000>

Email:
support@medtrack.health (example only)

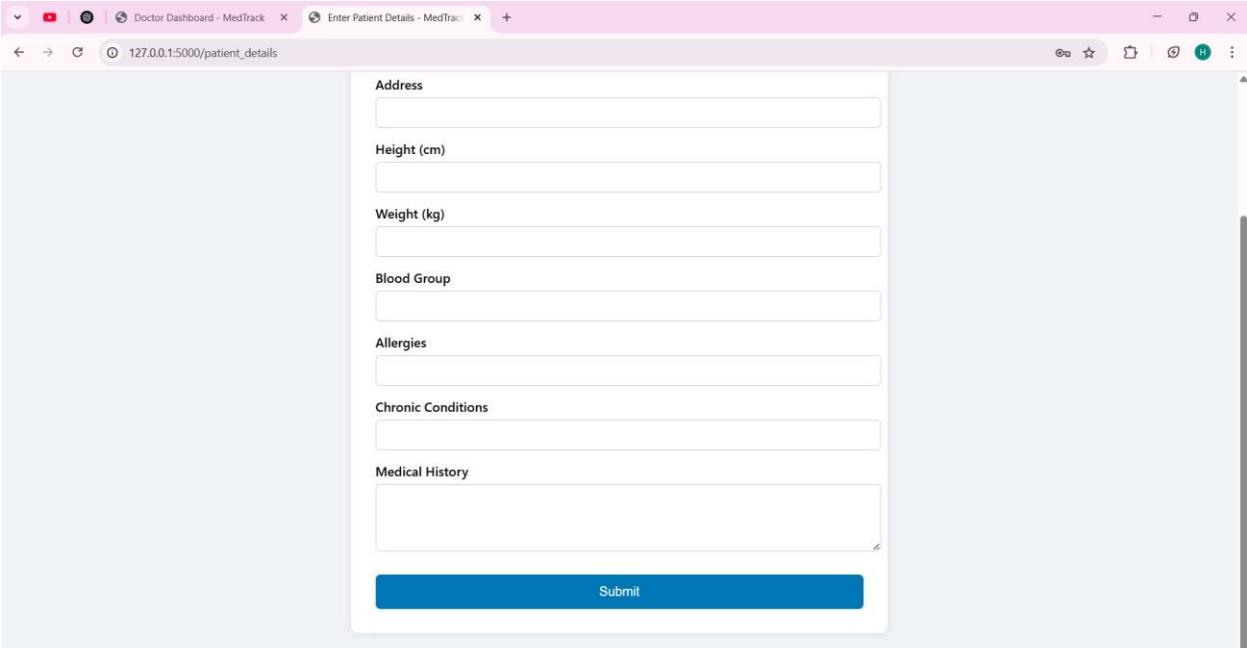
Phone:
+91-123-456-7890 (example only)

Patient Details page:

The screenshot shows a web browser window with two tabs: "Doctor Dashboard - MedTrack" and "Enter Patient Details - MedTrack". The URL in the address bar is 127.0.0.1:5000/patient_details. The main content area is titled "Enter Patient Details" and contains the following form fields:

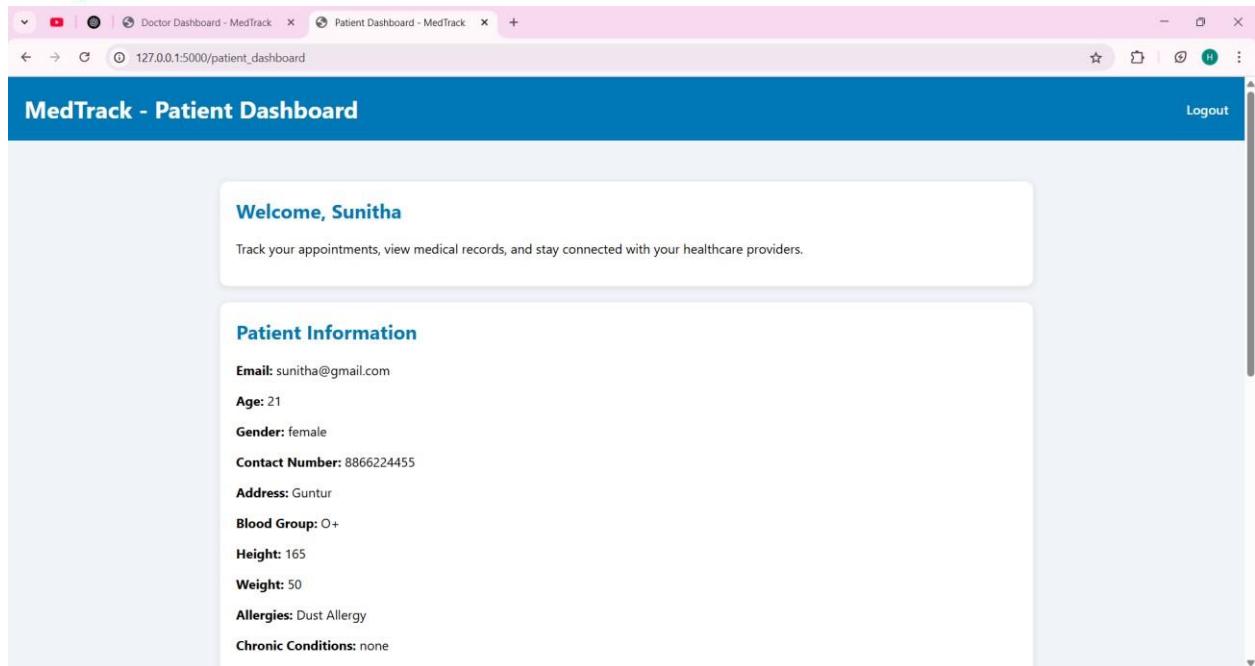
- Contact Number
- Address
- Height (cm)
- Weight (kg)
- Blood Group
- Allergies
- Chronic Conditions
- Medical History

Each field is represented by a text input box.



This screenshot shows the same "Enter Patient Details" page, but the input fields for Address, Height (cm), Weight (kg), Blood Group, Allergies, Chronic Conditions, and Medical History are collapsed into a single large vertical input box. A "Submit" button is visible at the bottom of this collapsed section.

Patient Dashboard:

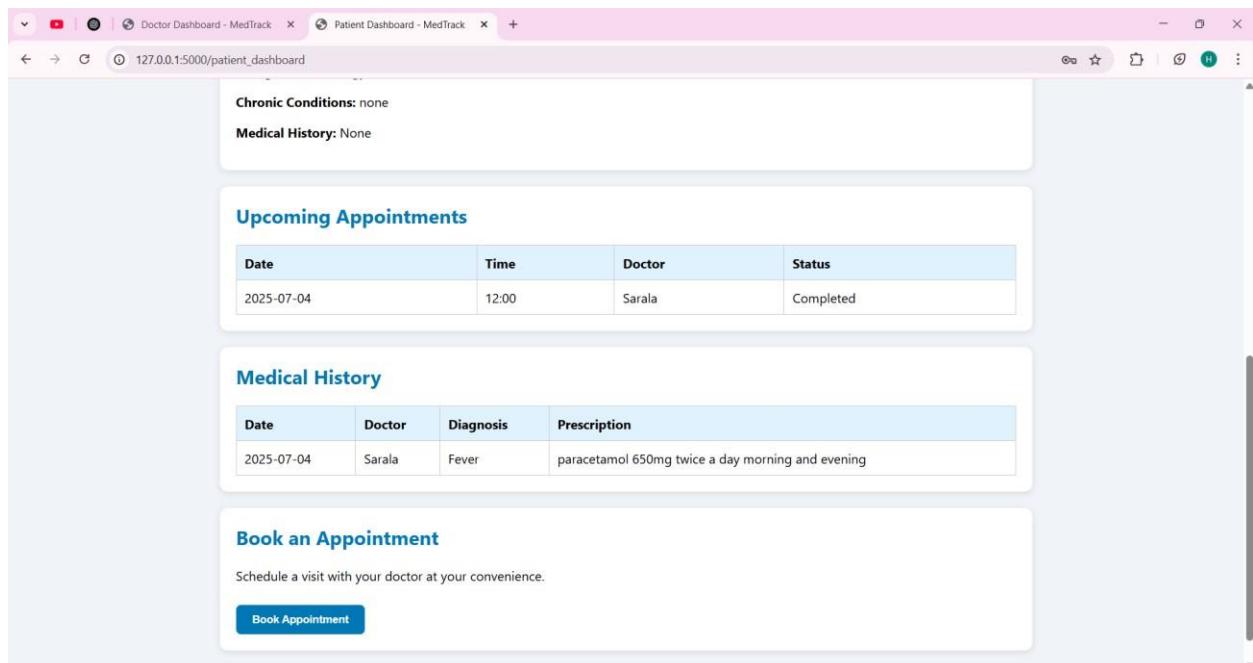


Welcome, Sunitha

Track your appointments, view medical records, and stay connected with your healthcare providers.

Patient Information

Email: sunitha@gmail.com
 Age: 21
 Gender: female
 Contact Number: 8866224455
 Address: Guntur
 Blood Group: O+
 Height: 165
 Weight: 50
 Allergies: Dust Allergy
 Chronic Conditions: none



Chronic Conditions: none
 Medical History: None

Upcoming Appointments

Date	Time	Doctor	Status
2025-07-04	12:00	Sarala	Completed

Medical History

Date	Doctor	Diagnosis	Prescription
2025-07-04	Sarala	Fever	paracetamol 650mg twice a day morning and evening

Book an Appointment

Schedule a visit with your doctor at your convenience.

Book Appointment

Appointment Booking Page:

Doctor Dashboard - MedTrack Book Appointment - MedTrack

127.0.0.1:5000/appointment_dashboard?

MedTrack - Book an Appointment

Book Your Appointment

Patient Name

Email Address

Contact Number

Select Doctor

Preferred Date

Preferred Time

Doctor Dashboard - MedTrack Book Appointment - MedTrack

127.0.0.1:5000/appointment_dashboard?

Patient Name

Email Address

Contact Number

Select Doctor

Preferred Date

Preferred Time

Describe Your Problem

Confirm Appointment

Doctor Details Page:

Doctor Dashboard - MedTrack | Book Appointment - MedTrack | Enter Doctor Details - MedTrack | +

127.0.0.1:5000/doctor_details

MedTrack - Enter Doctor Details

Additional Doctor Information

Experience (in years)

Clinic Address

Contact Info

Availability

[Save Details](#)

Doctor Dashboard Page:

Doctor Dashboard - MedTrack | Book Appointment - MedTrack | Enter Doctor Details - MedTrack | +

127.0.0.1:5000/doctor.dashboard

MedTrack - Doctor Dashboard

Logout

Welcome, Dr. Sarala

My Details

Age: 50

Gender: female

Email: sarala@gmail.com

Specialization: general

Clinic Address: Sampath Nagar, Guntur-522003

Availability: Mon- Fri, from 10:00AM to 12:30 PM and 2:30 PM to 7:00PM

Experience: 10 years

Contact Info: 9955774422

Doctor Dashboard - MedTrack | Book Appointment - MedTrack | Enter Doctor Details - MedTrack | +
 127.0.0.1:5000/doctor_dashboard

Clinic Address: Sampangi Nagar, Guntur 522003

Availability: Mon- Fri, from 10:00AM to 12:30 PM and 2:30 PM to 7:00PM

Experience: 10 years

Contact Info: 9955774422

Appointments Overview

Upcoming Appointments

Patient Name	Date & Time	Problem	Status	Prescription	Action
No upcoming appointments					

Completed Appointments

Patient Name	Date & Time	Problem	Status	Prescription
Sunitha	2025-07-04 12:00	Fever	Completed	paracetamol 650mg twice a day morning and evening

View Patient Details Page:

View Patient Details - MedTrack | Book Appointment - MedTrack | Enter Doctor Details - MedTrack | +
 127.0.0.1:5000/view_patient/sunitha@gmail.com

Patient Details - Sunitha

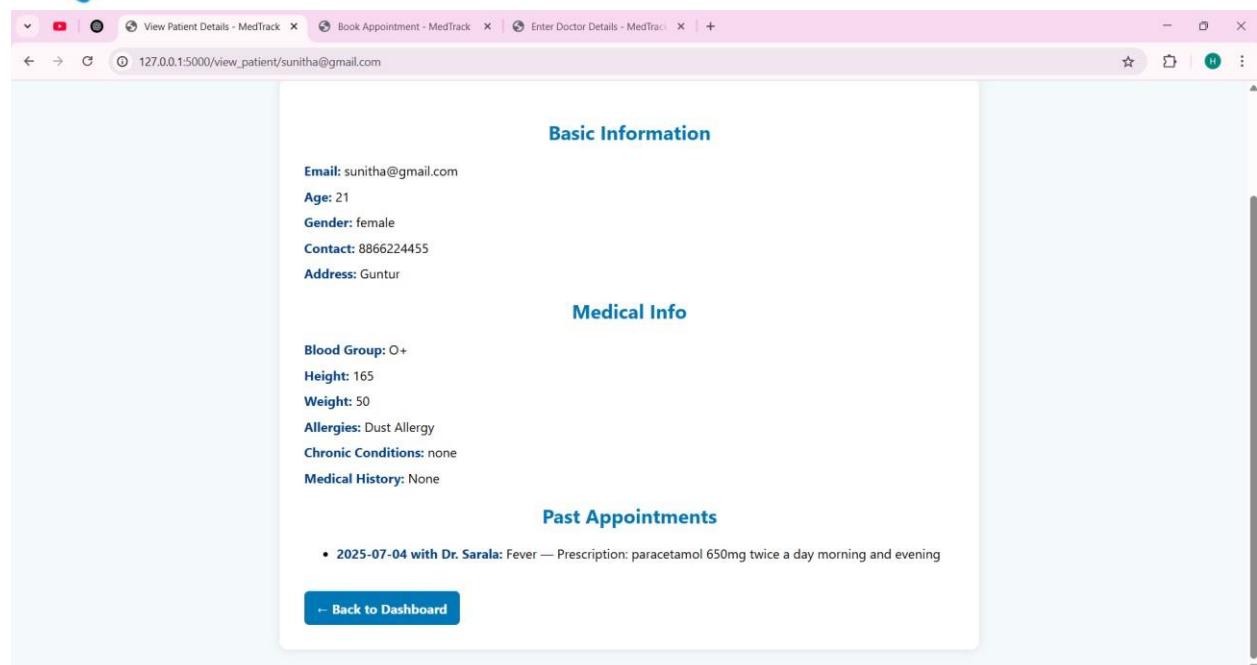
Basic Information

Email: sunitha@gmail.com
 Age: 21
 Gender: female
 Contact: 8866224455
 Address: Guntur

Medical Info

Blood Group: O+
 Height: 165
 Weight: 50
 Allergies: Dust Allergy
 Chronic Conditions: none
 Medical History: None

Past Appointments



Basic Information

Email: sunitha@gmail.com
 Age: 21
 Gender: female
 Contact: 8866224455
 Address: Guntur

Medical Info

Blood Group: O+
 Height: 165
 Weight: 50
 Allergies: Dust Allergy
 Chronic Conditions: none
 Medical History: None

Past Appointments

- 2025-07-04 with Dr. Sarala: Fever — Prescription: paracetamol 650mg twice a day morning and evening

[← Back to Dashboard](#)

Local Database updatons :

1. Users table :

31 • select * from user;

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

	id	role	name	email	password	age	gender	specialization
▶	1	doctor	Sunitha	suni@gmail.com	suni1234	40	female	general
	3	patient	Saketh	saketh@gmail.com	sak1234	15	male	
	4	patient	Praneetha	praneetha@gmail.com	pra1234	21	female	
	5	doctor	Ravindra	ravi@gmail.com	ravi1234	45	male	general
	6	patient	Vamsi	vamsi@gmail.com	vamsi1234	11	male	
	7	doctor	Suma Latha	suma@gmail.com	suma1234	38	female	gynecology
	8	doctor	Ramesh	ramesh@gmail.com	ram1234	48	male	dental
	9	patient	Sowmya	sowmya@gmail.com	sow1234	21	female	
	10	doctor	Sarala	sarala@gmail.com	sarala1234	50	female	ent
	11	patient	Siva	siva@gmail.com	siva1234	18	male	

2. Doctor Details table :

34 • `select * from doctor_detail;`

	Result Grid	Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
▶	id	email	experience	clinic_address	contact
▶	2	suni@gmail.com	8	Laksmipuram, Guntur	9876543210
▶	3	ravi@gmail.com	10	Koritipadu , Guntur	9596442371
▶	4	suma@gmail.com	8	Inner Ring Road, Guntur	9955774422
▶	5	ramesh@gmail.com	9	Nallapadu, Guntur	8844662277
▶	6	sarala@gmail.com	10	Kothapeta, Guntur	8321654972
*	HULL	HULL	HULL	HULL	HULL

3. Patient Details table:

 35 • `select * from patient_detail;`

	Result Grid	Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
▶	id	email	contact	address	height
▶	1	saketh@gmail.com	7755331155	guntur	155
▶	2	praneetha@gmail.com	8529637413	guntur	160
▶	3	vamsi@gmail.com	8321654972	Guntur	145
▶	4	sowmya@gmail.com	8654723791	Guntur	165
▶	5	siva@gmail.com	9652387142	Guntur	175
*	HULL	HULL	HULL	HULL	HULL

4. Appointment table:

32 • select * from appointment;

	id	patient_name	email	phone	doctor	date	time	problem
▶	1	Saketh	saketh@gmail.com	7755331155	Sunitha	2025-06-28	11:00	Fever
	2	Praneetha	praneetha@gmail.com	8529637413	Sunitha	2025-06-27	11:00	Fever
	3	Praneetha	praneetha@gmail.com	852147963	Ravindra	2025-06-27	11:00	Fever
	4	Vamsi	vamsi@gmail.com	8321654972	Sunitha	2025-06-28	11:00	Vomitings and stomach
	5	Vamsi	vamsi@gmail.com	8321654972	Ravindra	2025-06-27	12:00	Fever
	6	Praneetha	praneetha@gmail.com	8529637413	Suma Latha	2025-06-27	12:15	Imbalance Menstrual c
	7	Praneetha	praneetha@gmail.com	8529637413	Ravindra	2025-06-29	12:00	Vomitings and Fever
	8	Sowmya	sowmya@gmail.com	8654723791	Ramesh	2025-06-28	03:00	Severe Tooth Pain
	9	Vamsi	vamsi@gmail.com	8321654972	Ramesh	2025-06-28	03:30	Regular Tooth Checku
	10	Siva	siva@gmail.com	9652387142	Sarala	2025-06-29	11:00	Severe ear pain
*	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

5. Medical History Table:

36 • select * from medical_history;

	id	email	date	doctor	diagnosis	prescription
▶	1	saketh@gmail.com	2025-06-28	Sunitha	Fever	paracetamol 50mg twice a day m
	2	praneetha@gmail.com	2025-06-27	Sunitha	Fever	paracetamol 50mg twice a day m
	3	praneetha@gmail.com	2025-06-27	Ravindra	Fever	paracetamol 50mg twice a day m
	4	vamsi@gmail.com	2025-06-27	Ravindra	Fever	paracetamol 50mg twice a day m
	5	vamsi@gmail.com	2025-06-28	Sunitha	Vomitings and stomach pain	use meftol medicine for per day a
	6	praneetha@gmail.com	2025-06-27	Suma Latha	Imbalance Menstrual cycle	Eat and sleep well, eat papaya a
	7	praneetha@gmail.com	2025-06-29	Ravindra	Vomitings and Fever	paracetamol 50mg twice a day m
	8	sowmya@gmail.com	2025-06-28	Ramesh	Severe Tooth Pain	You are suffering due to cavity b
	9	vamsi@gmail.com	2025-06-28	Ramesh	Regular Tooth Checkup	Your Teeth were all good just bru
	10	siva@gmail.com	2025-06-29	Sarala	Severe ear pain	Use some wet cloth dipped in hot
*	HULL	HULL	HULL	HULL	HULL	HULL

Mail to the admin:

New User Registration Inbox x



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

Welcome Saranya! Your patient account has been created successfully.

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:

<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:122610513084:Medtrack:052bde12-1782-408f-891a-a86b91fd15ac>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>



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



Welcome Saranya! Your patient account has been created successfully.

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:

<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:122610513084:Medtrack:052bde12-1782-408f-891a-a86b91fd15ac>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

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.