

## 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.

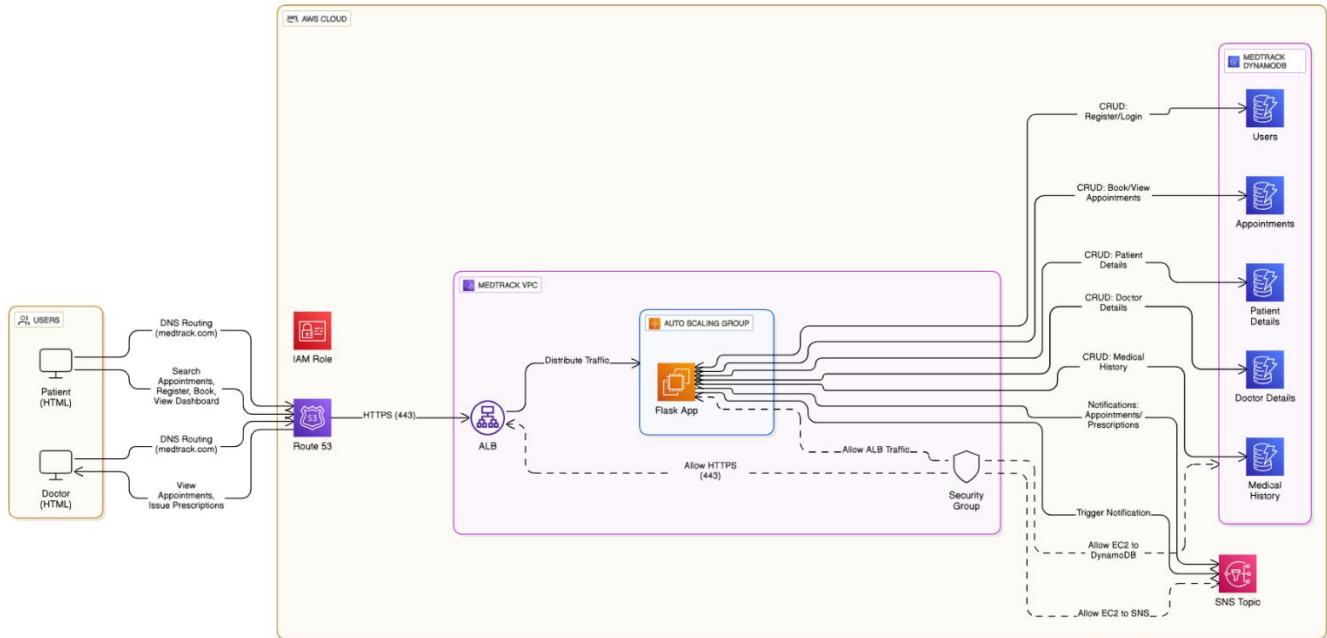
### Scenario 2: Seamless Book Request Notifications for Library Staff

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.

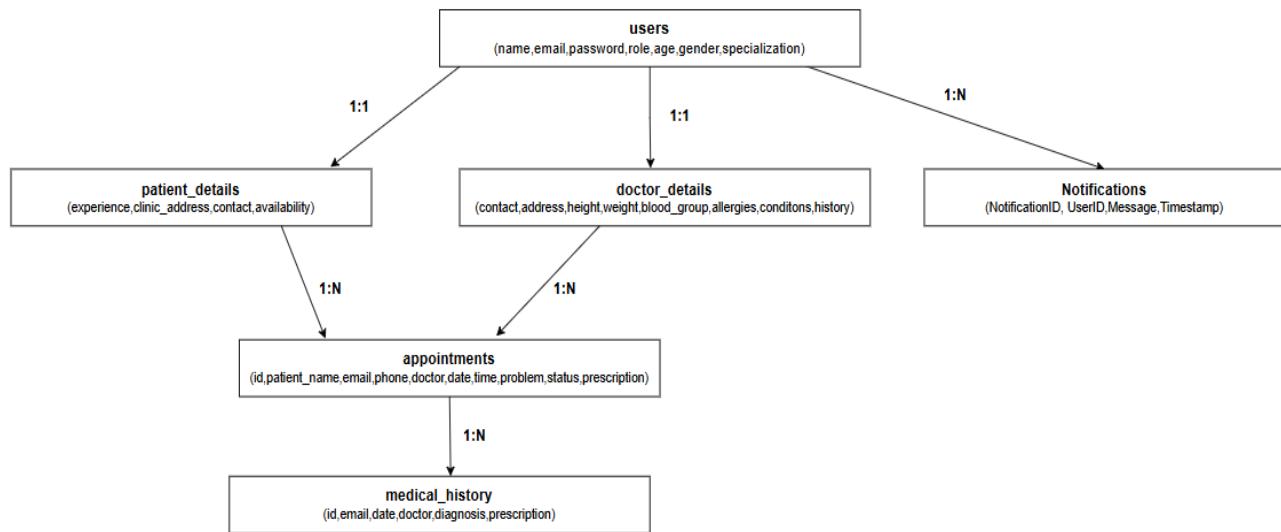
### Scenario 3: Easy Access to Library Resources

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

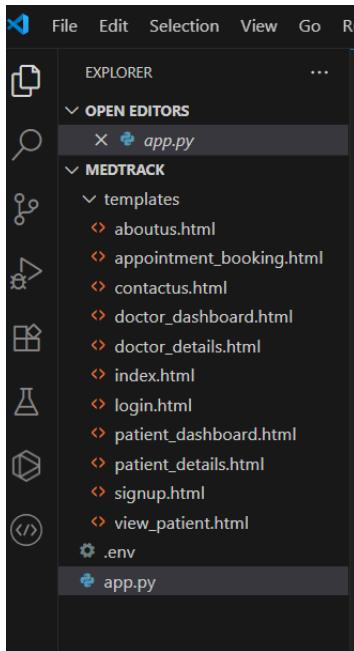
### 8. Testing and Deployment

**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(USER_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 ''
        if password != confirm_password:
            return render_template('signup.html', password_error="Passwords do not match")
        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:
                print(f"Error creating user: {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'))
    return render_template('signup.html')

```

- **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:**

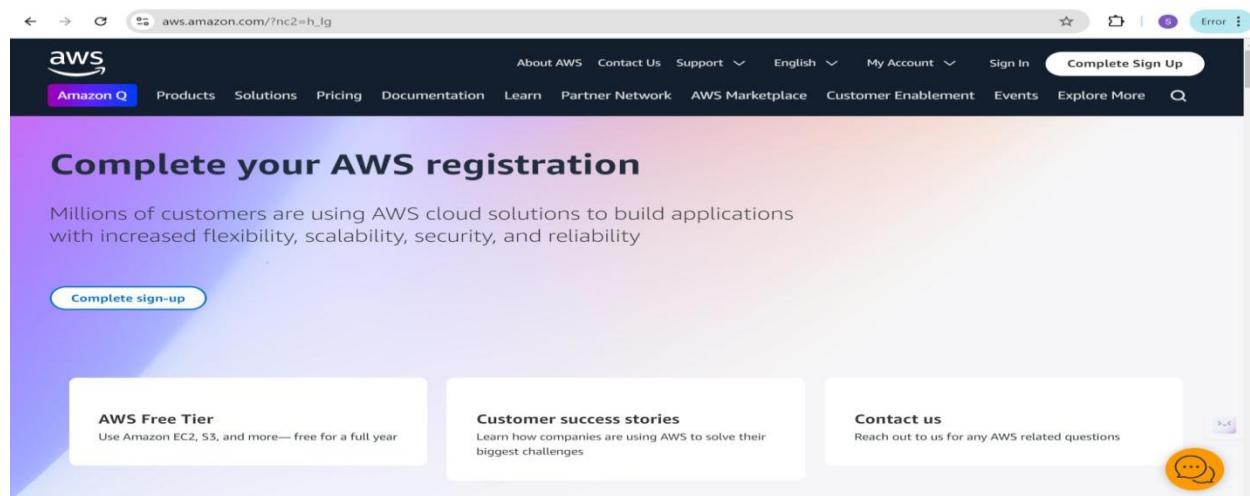
```
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000, debug=True)
```

**Description:**

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

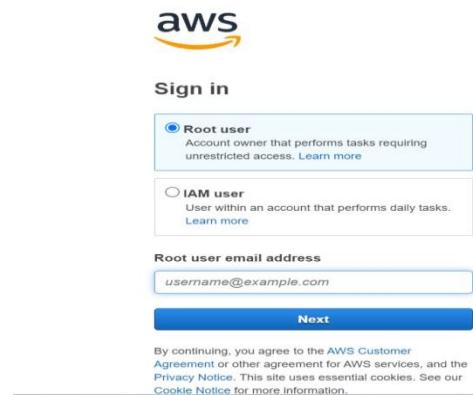
## Milestone 2: AWS Account Setup and Login

- **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**

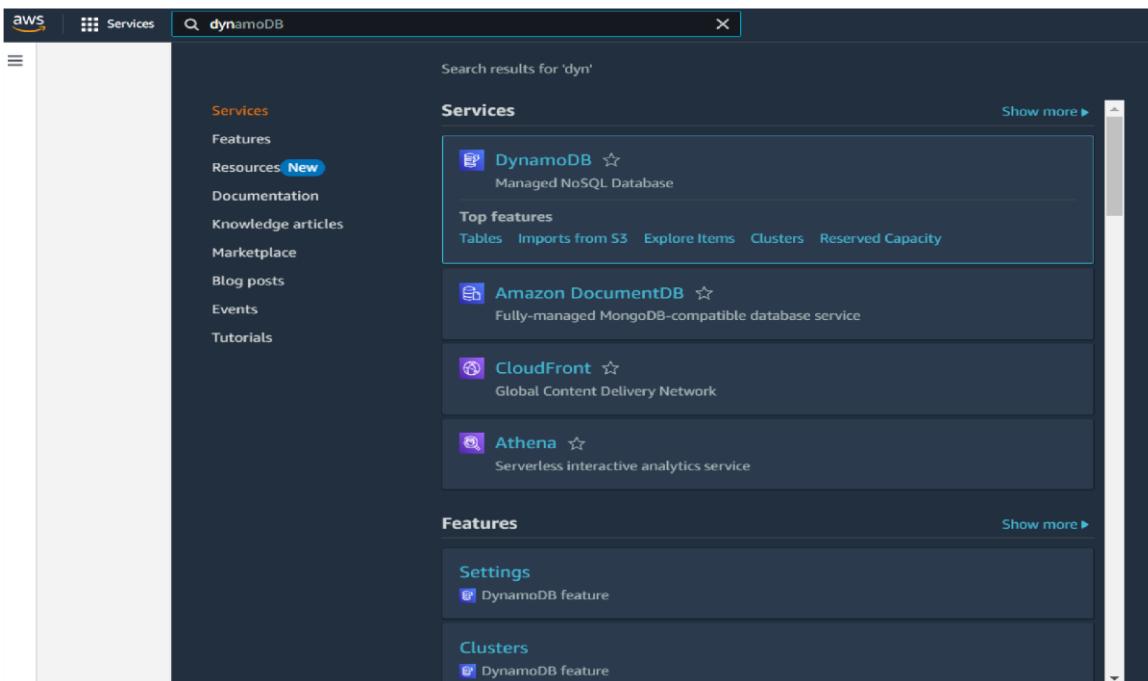
- After setting up your account, log in to the [AWS Management Console](#).



## 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.



DynamoDB X

[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](#)

**Dashboard**

**Alarms (0) Info** [Manage in CloudWatch](#)

< 1 > ⚙

Alarm name	Status
No custom alarms	

**DAX clusters (0) Info** [View details](#)

< 1 > ⚙

Cluster name	Status
No clusters	
No clusters to display	
<a href="#">Create cluster</a>	

**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 X

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

**Tables (0) Info** [Create table](#)

Any tag key Any tag value < 1 > ⚙

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size
You have no tables in this account in this AWS Region.								
<a href="#">Create table</a>								

## ● 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.

troven.in/students/labs/68033... troven.in/students/labs/68033... Create table | Amazon DynamoDB Launch an instance | EC2 us-east-1 #create-table

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

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

DynamoDB > Tables > Create table

**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.  
 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

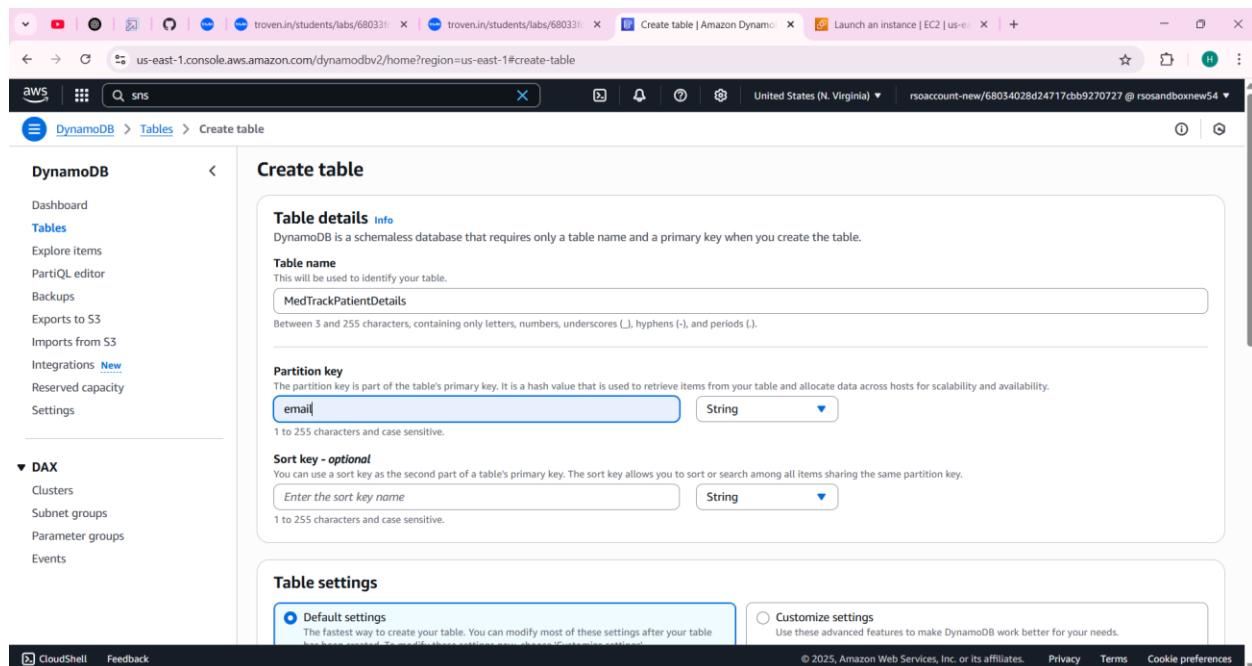
**Tags**  
 Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

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 Amazon DynamoDB console. The left sidebar shows the 'DynamoDB' navigation menu with 'Tables' selected. The main area is titled 'Create table' and contains the following fields:

- Table details**: The table name is set to 'MedTrackPatientDetails'. A note states: "DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table."
- Partition key**: The primary key is named 'email' and is of type 'String'. A note states: "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."
- Sort key - optional**: A note states: "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." An empty input field is shown for 'Enter the sort key name'.
- Table settings**: Two options are available: 'Default settings' (selected) and 'Customize settings'. A note for 'Default settings' says: "The fastest way to create your table. You can modify most of these settings after your table has been created." A note for 'Customize settings' says: "Use these advanced features to make DynamoDB work better for your needs."

At the bottom, there are links for 'CloudShell' and 'Feedback', and a footer with copyright information: "© 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

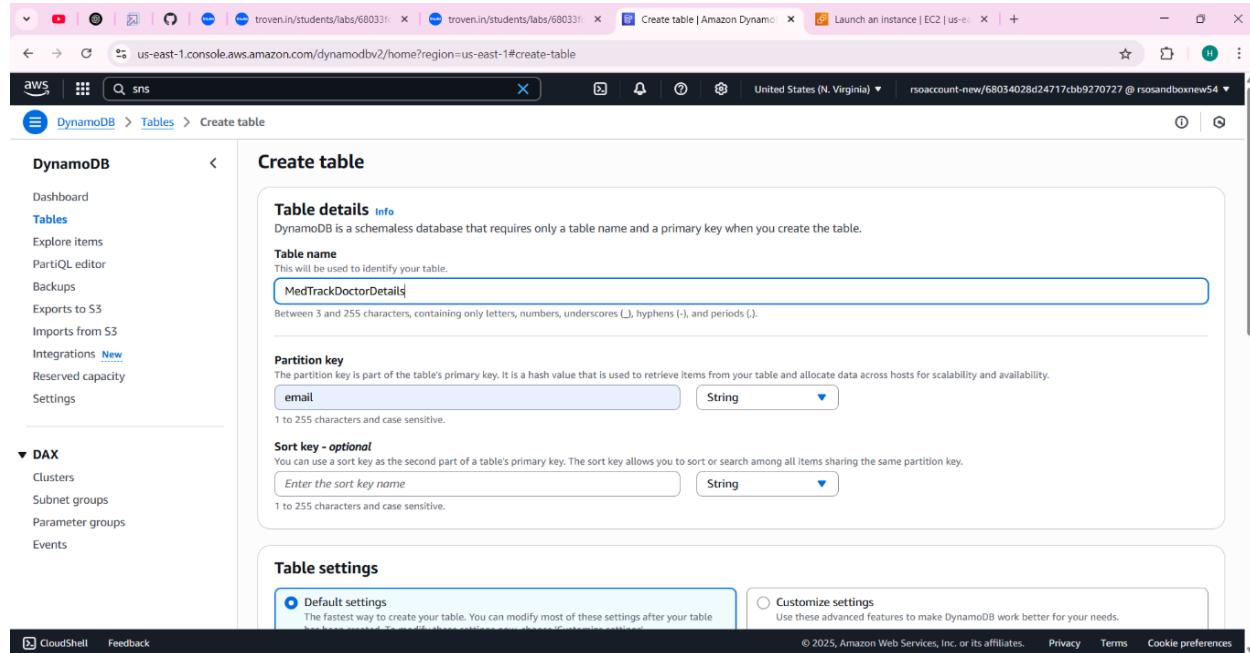
**Tags**  
 Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

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 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 for Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main area is titled '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.

String (1 to 255 characters and case sensitive.)

**Sort key - optional**

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

String (1 to 255 characters and case sensitive.)

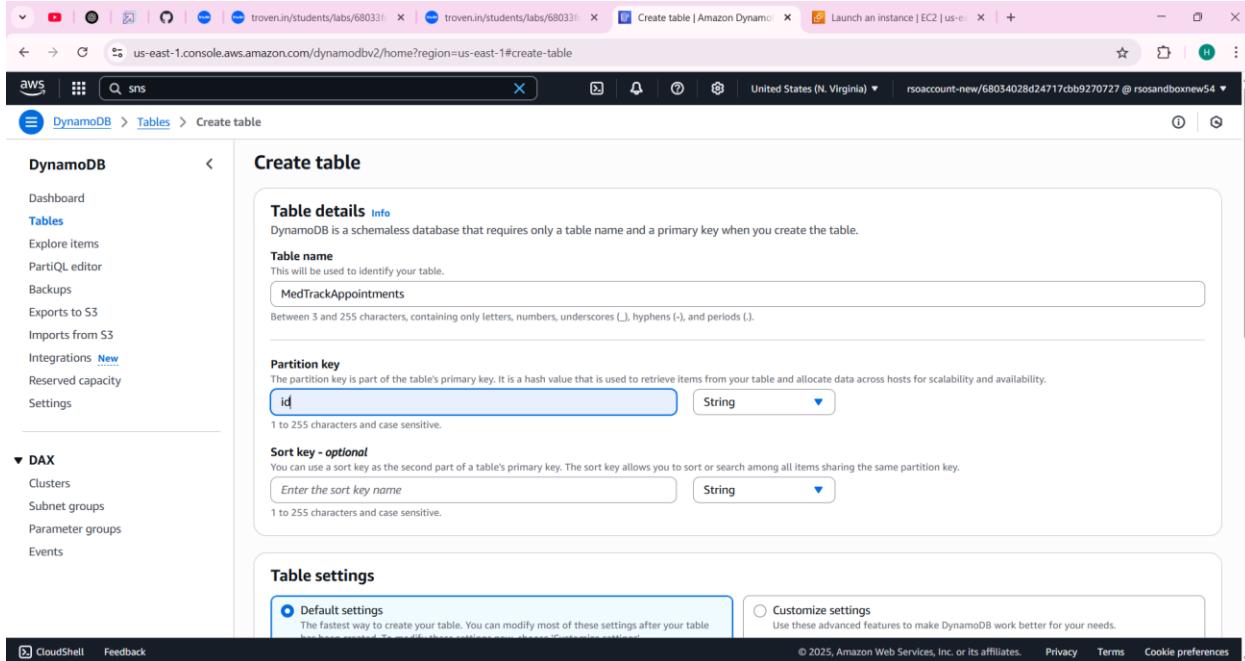
**Table settings**

Default settings (The fastest way to create your table. You can modify most of these settings after your table has been created.)

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

© 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

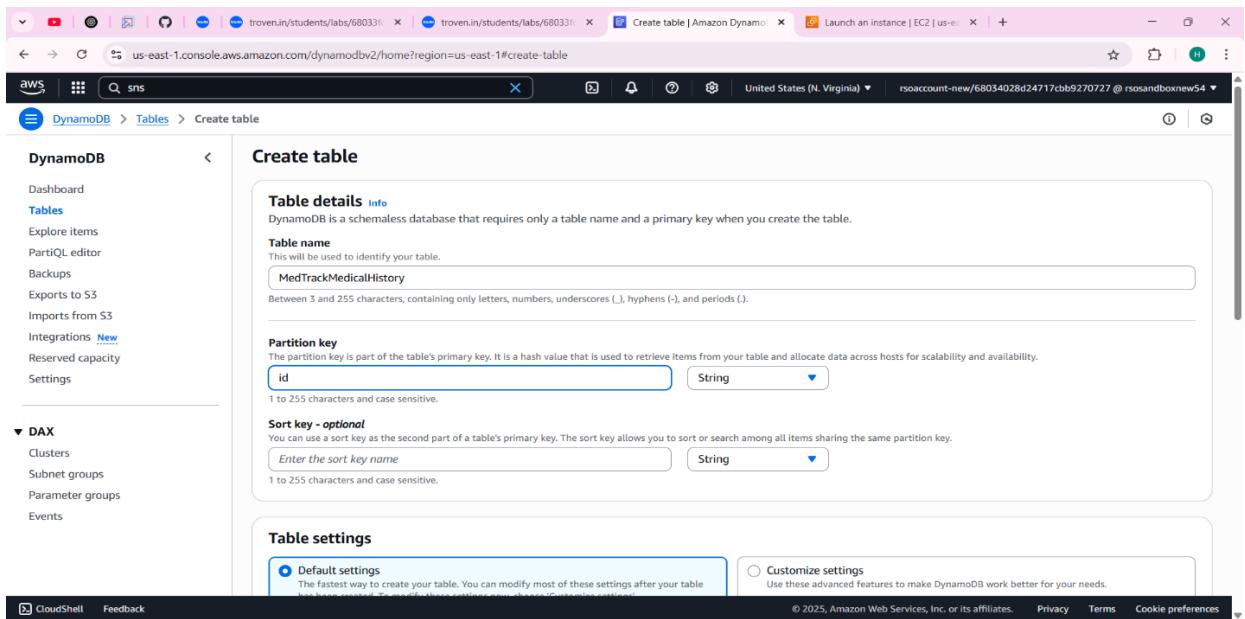
- Follow the same steps to create a MedTrackAppointments table with 'id' as the partition key with type string to store appointments details.



The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The left sidebar shows the 'Tables' section selected under 'DynamoDB'. The main area is titled 'Create table' and contains the following fields:

- Table details**: Table name is set to 'MedTrackAppointments'.
- Partition key**: Key name is 'id', type is 'String'.
- Sort key - optional**: An empty input field for sort key name and a dropdown for type 'String'.
- Table settings**: A radio button for 'Default settings' is selected, with a note that it's the fastest way to create the table. There is also an option for 'Customize settings'.

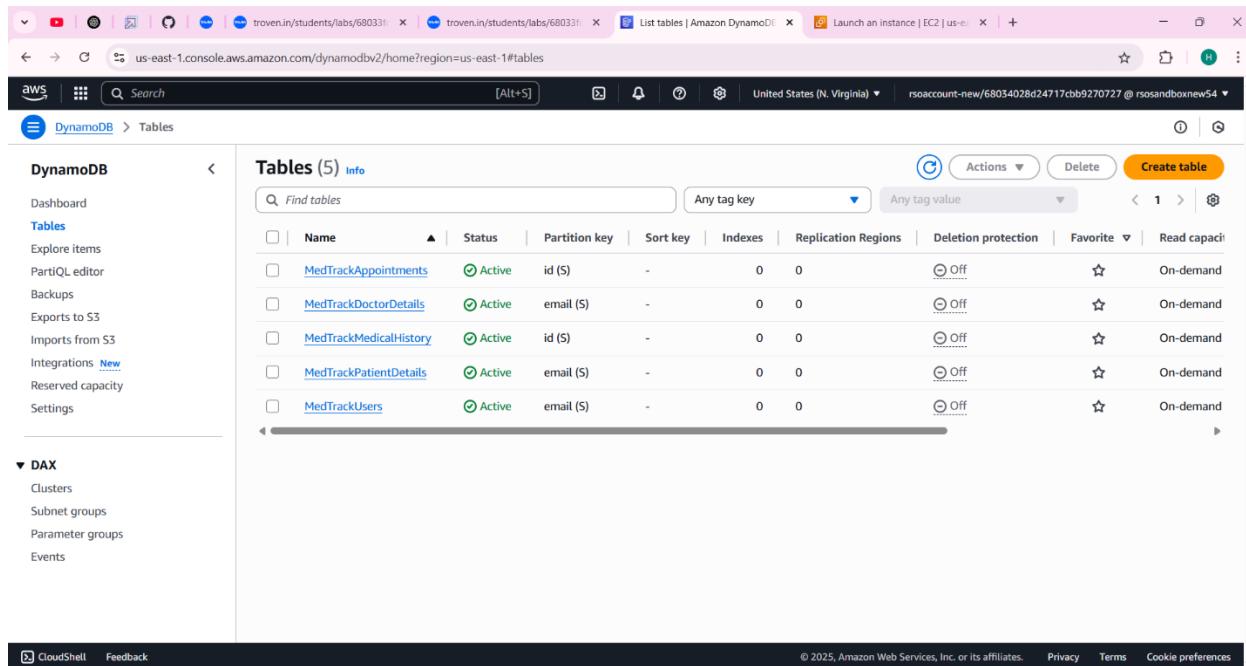
- Follow the same steps to create a MedTrackMedicalHistory table with 'id' as the partition key with type string to store details of medical history.



The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The left sidebar shows the 'Tables' section selected under 'DynamoDB'. The main area is titled 'Create table' and contains the following fields:

- Table details**: Table name is set to 'MedTrackMedicalHistory'.
- Partition key**: Key name is 'id', type is 'String'.
- Sort key - optional**: An empty input field for sort key name and a dropdown for type 'String'.
- Table settings**: A radio button for 'Default settings' is selected, with a note that it's the fastest way to create the table. There is also an option for 'Customize settings'.

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

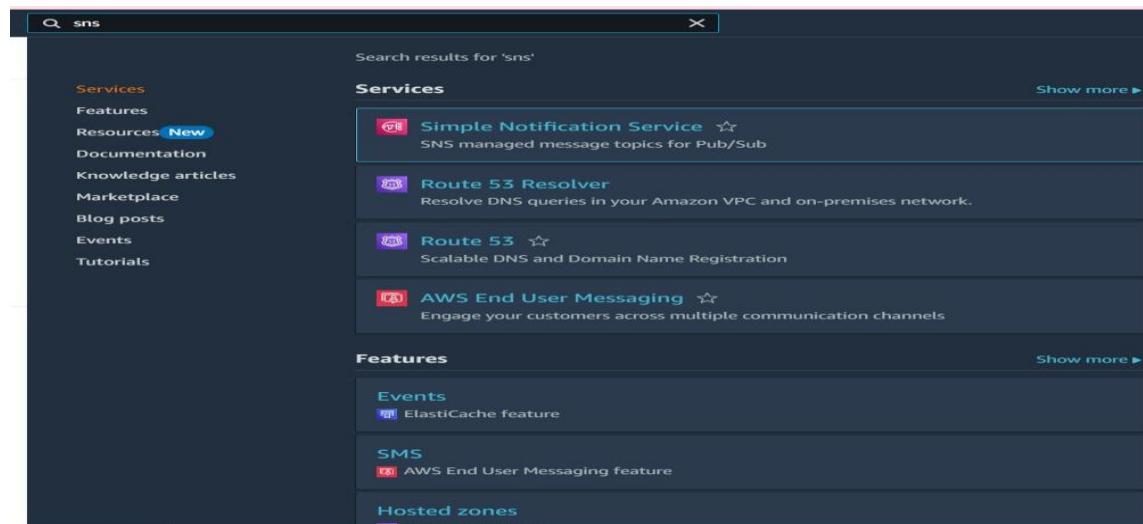


The screenshot shows the AWS DynamoDB Tables page. The left sidebar has links for Dashboard, Tables (Explore items, PartQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, Settings), and DAX (Clusters, Subnet groups, Parameter groups, Events). The main area shows a table titled 'Tables (5) Info' with columns: Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read capacity. The five tables listed are:

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.



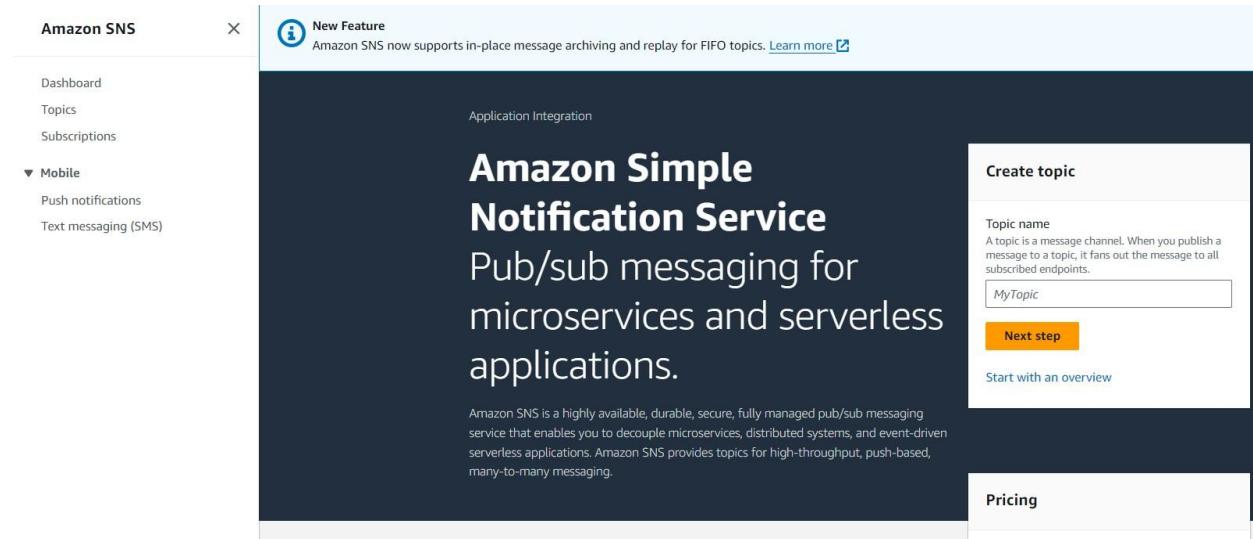
The screenshot shows the AWS search results for 'sns'. The left sidebar has links for Services (Features, Resources New, Documentation, Knowledge articles, Marketplace, Blog posts, Events, Tutorials), and a search bar. The main area shows a list of services under 'Services' and 'Features'.

**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)

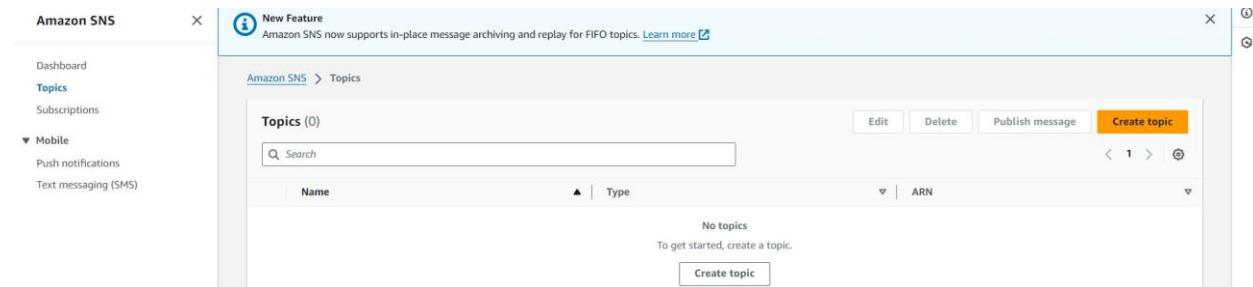
**Features**

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



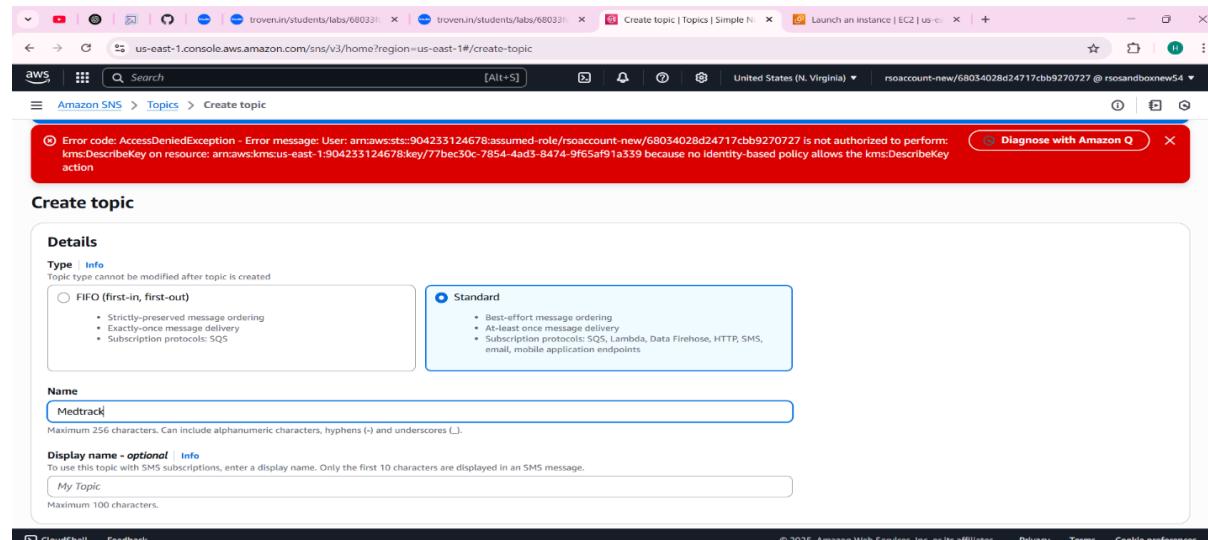
The screenshot shows the Amazon Simple Notification Service (SNS) home page. On the left, there is a sidebar with navigation links: Dashboard, Topics, Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and a New Feature announcement about in-place message archiving and replay for FIFO topics. The main content area features a dark header "Application Integration" and a large title "Amazon Simple Notification Service" followed by a subtitle "Pub/sub messaging for microservices and serverless applications." Below the title, a paragraph describes Amazon SNS as a highly available, durable, secure, fully managed pub/sub messaging service. To the right, there is a "Create topic" form with a "Topic name" input field containing "MyTopic", a "Next step" button, and a link to "Start with an overview". At the bottom right, there is a "Pricing" section.

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



The screenshot shows the "Topics" page within the Amazon SNS service. The left sidebar includes links for Dashboard, Topics (which is selected), Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and a New Feature announcement. The main content area shows a table titled "Topics (0)" with columns for Name, Type, and ARN. A search bar and a "Create topic" button are at the top of the table. A message below the table says "To get started, create a topic." and features a "Create topic" button.

- 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 a red error message: "Error code: AccessDeniedException - Error message: User: arnaws:sts::904233124678:assumed-role/rsoaccount-new/68034028d24717cbb9270727 is not authorized to perform: kms:DescribeKey on resource: arnaws:kms:us-east-1:904233124678:key/77bec30c-7854-4ad3-8474-9f65af91a339 because no identity-based policy allows the kms:DescribeKey action". Below this, there is a "Create topic" button. The main form has sections for "Details", "Name", and "Display name - optional". Under "Details", there are two radio button options: "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 contains "Medtrack" and the "Display name - optional" field contains "My Topic".

► **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.

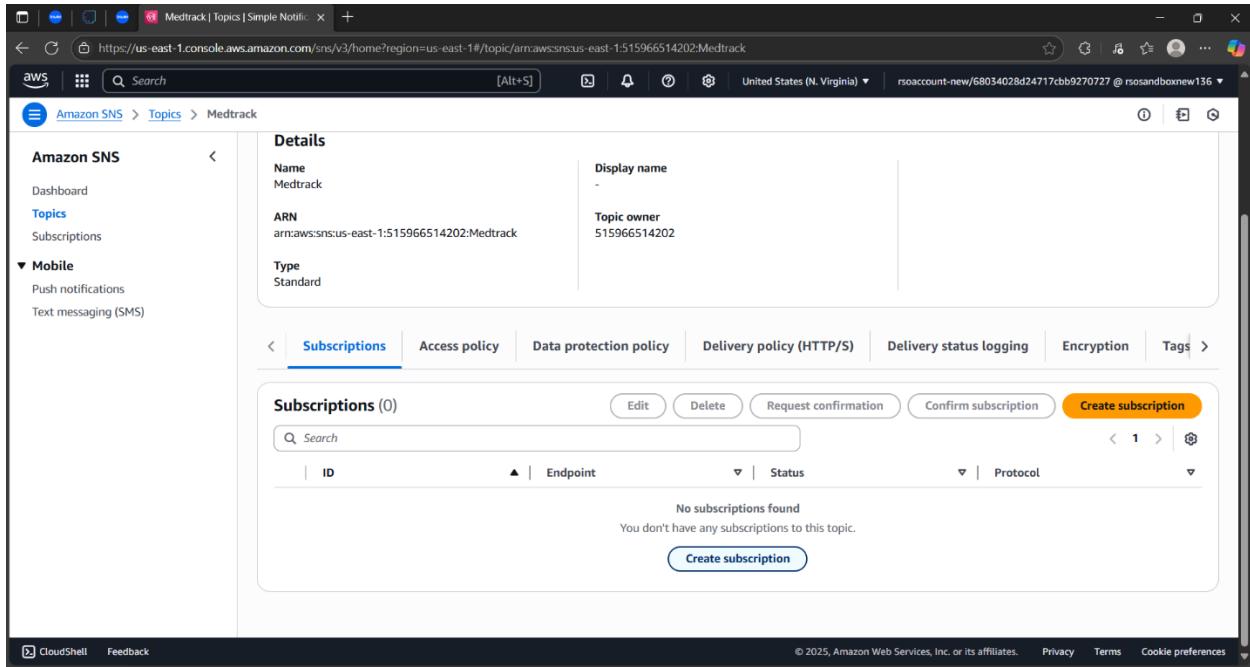
► **Delivery status logging - optional** [Info](#)  
 These settings configure the logging of message delivery status to CloudWatch Logs.

► **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](#)

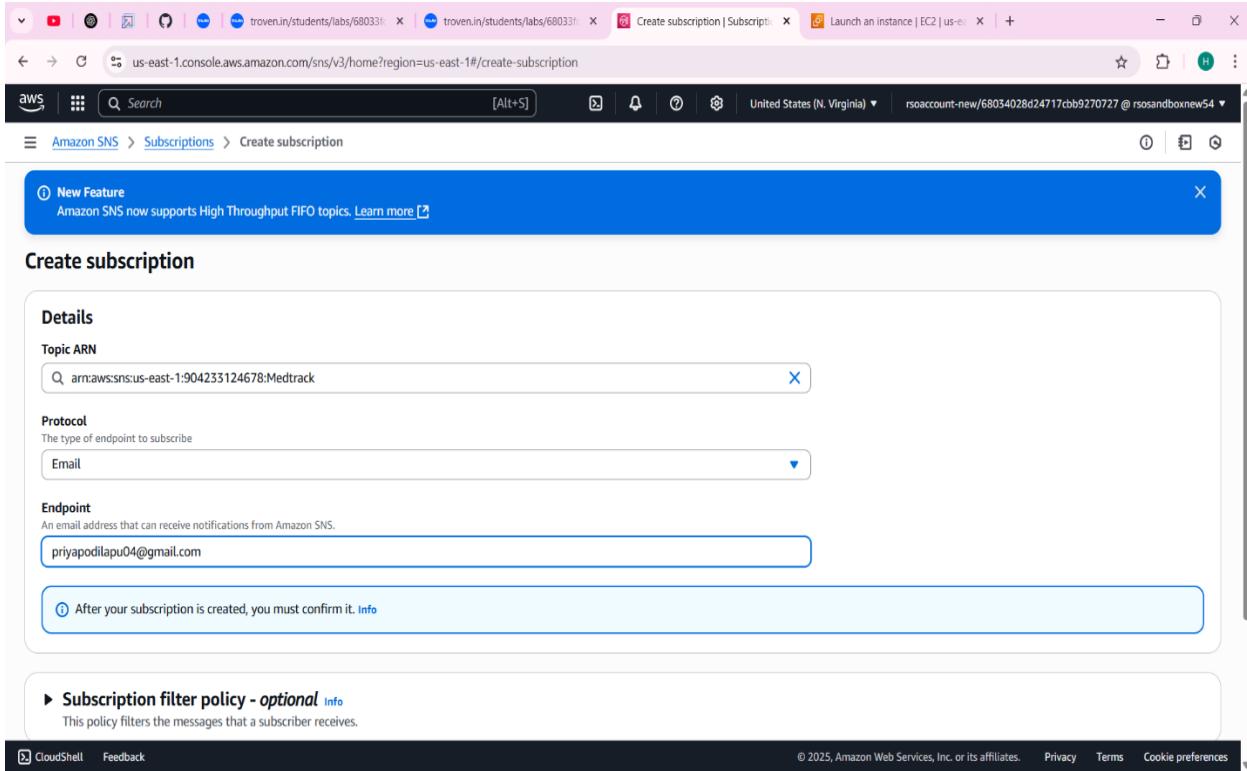
- Configure the SNS topic and note down the **Topic ARN**.



The screenshot shows the AWS SNS Topics page for the 'Medtrack' topic. The left sidebar shows navigation options like Dashboard, Topics, Subscriptions, and Mobile (Push notifications, Text messaging (SMS)). The main area displays the 'Details' for the 'Medtrack' topic, including Name (Medtrack), ARN (arn:aws:sns:us-east-1:515966514202:Medtrack), Type (Standard), and Topic owner (515966514202). Below the details, there are tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags. The Subscriptions tab is selected, showing a table with columns ID, Endpoint, Status, and Protocol. A message indicates 'No subscriptions found'. At the bottom of the table, there is a 'Create subscription' button. The URL in the browser is https://us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/topic/arm:aws:sns:us-east-1:515966514202:Medtrack.

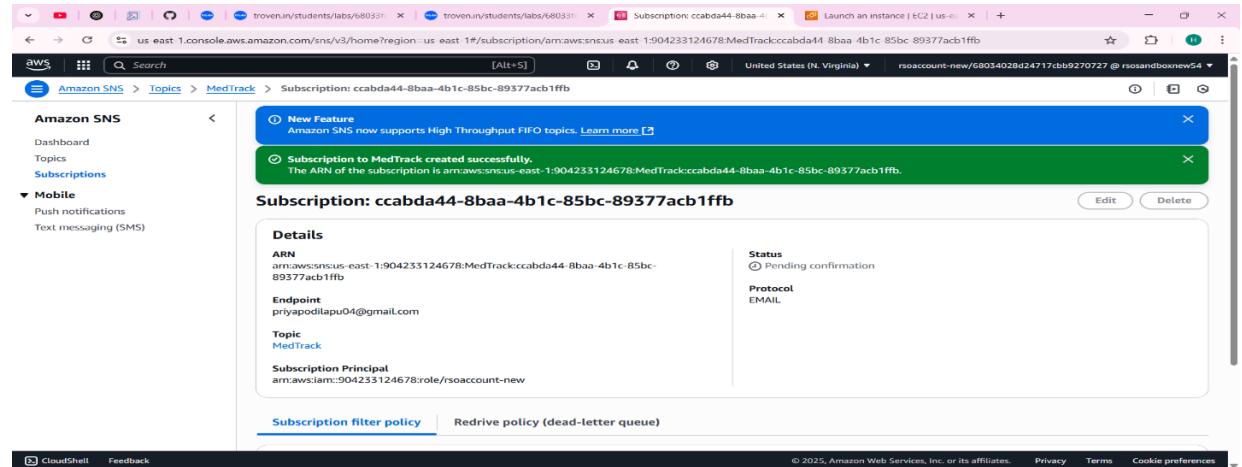
- **Activity 4.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

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



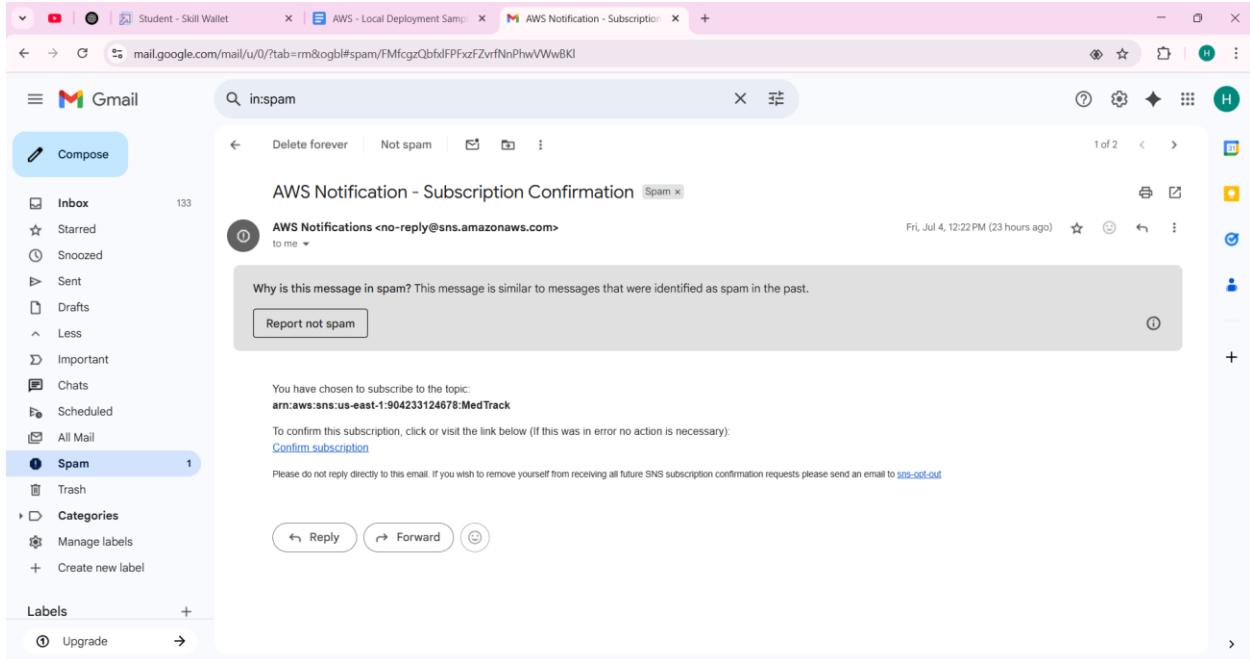
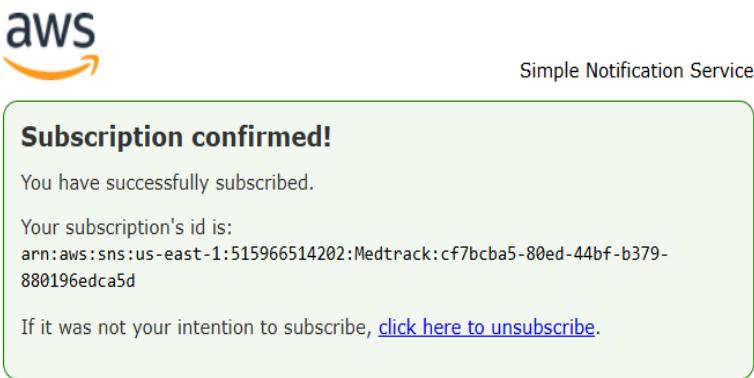
The screenshot shows the 'Create subscription' page in the AWS SNS console. The 'Topic ARN' field contains 'arn:aws:sns:us-east-1:904233124678:Medtrack'. The 'Protocol' dropdown is set to 'Email', and the 'Endpoint' field contains 'priyapodilapu04@gmail.com'. A note at the bottom says, 'After your subscription is created, you must confirm it.' Below this, a section titled 'Subscription filter policy - optional' is shown with the note 'This policy filters the messages that a subscriber receives.'

- 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 to MedTrack created successfully. The ARN of the subscription is arn:aws:sns:us-east-1:904233124678:MedTrack:ccabda44-8baa-4b1c-85bc-89377acb1ffb.' Below this, the 'Subscription: ccabda44-8baa-4b1c-85bc-89377acb1ffb' details are shown, including the ARN, endpoint, topic, and status 'Pending confirmation'.

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

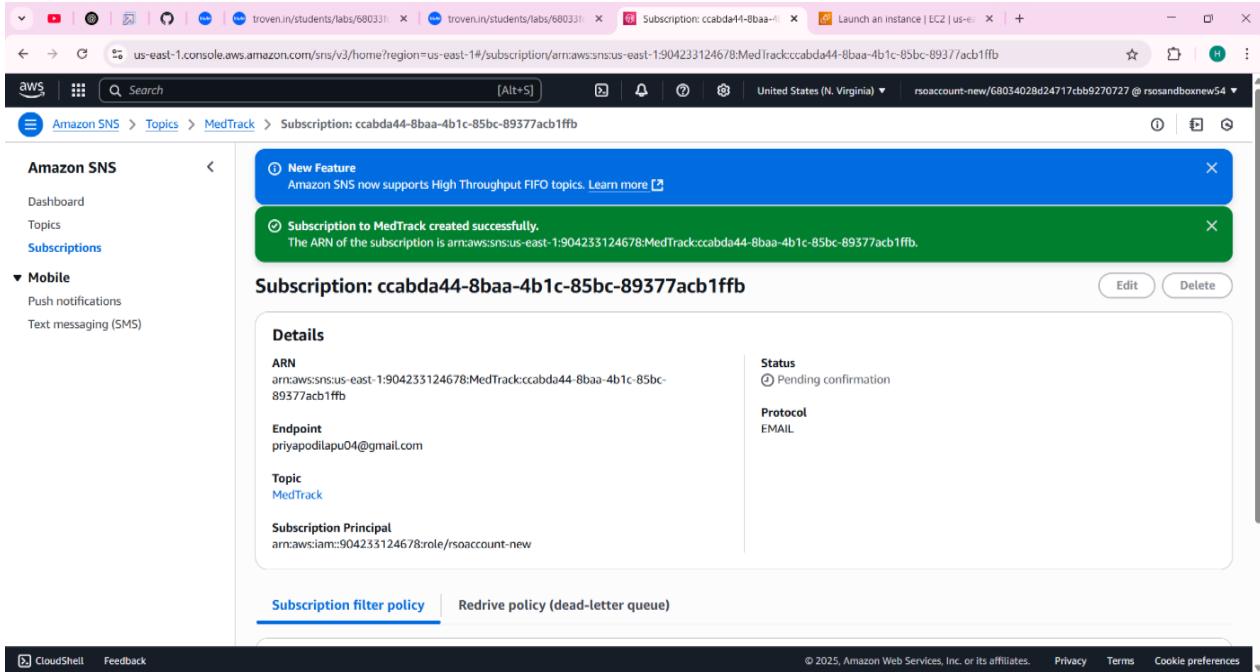
**Subscription confirmed!**

You have successfully subscribed.

Your subscription's id is:  
 arn:aws:sns:us-east-1:515966514202:Medtrack:c7bcba5-80ed-44bf-b379-880196edca5d

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

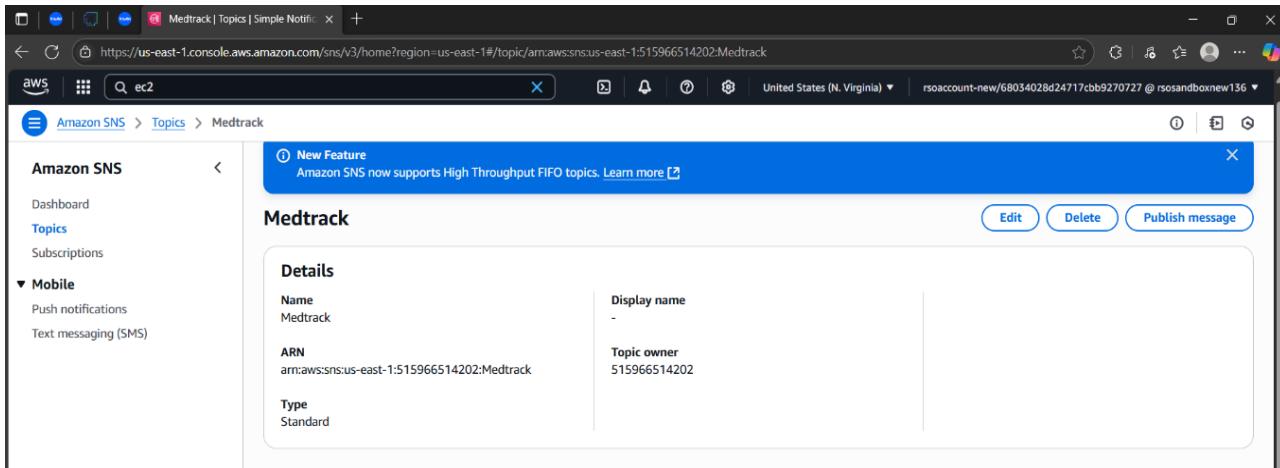
- Successfully done with the SNS mail subscription and setup, now store the ARN link.



The screenshot shows the AWS SNS console. On the left, there's a sidebar with 'Amazon SNS' and sections for 'Dashboard', 'Topics', 'Subscriptions', and 'Mobile' (Push notifications, Text messaging (SMS)). The main area shows a success message: 'Subscription to MedTrack created successfully.' Below it, the subscription details are listed:

Subscription: ccabda44-8baa-4b1c-85bc-89377acb1ffb	
<b>Details</b>	<b>Status</b>
ARN arn:aws:sns:us-east-1:904233124678:MedTrack:ccabda44-8baa-4b1c-85bc-89377acb1ffb	Pending confirmation
Endpoint priyapodilapu04@gmail.com	<b>Protocol</b> EMAIL
Topic MedTrack	
Subscription Principal arn:aws:iam::904233124678:role/rsoaccount-new	

At the bottom, there are tabs for 'Subscription filter policy' (which is selected) and 'Redrive policy (dead-letter queue)'. The status bar at the bottom right includes links for CloudShell, Feedback, and cookie preferences.



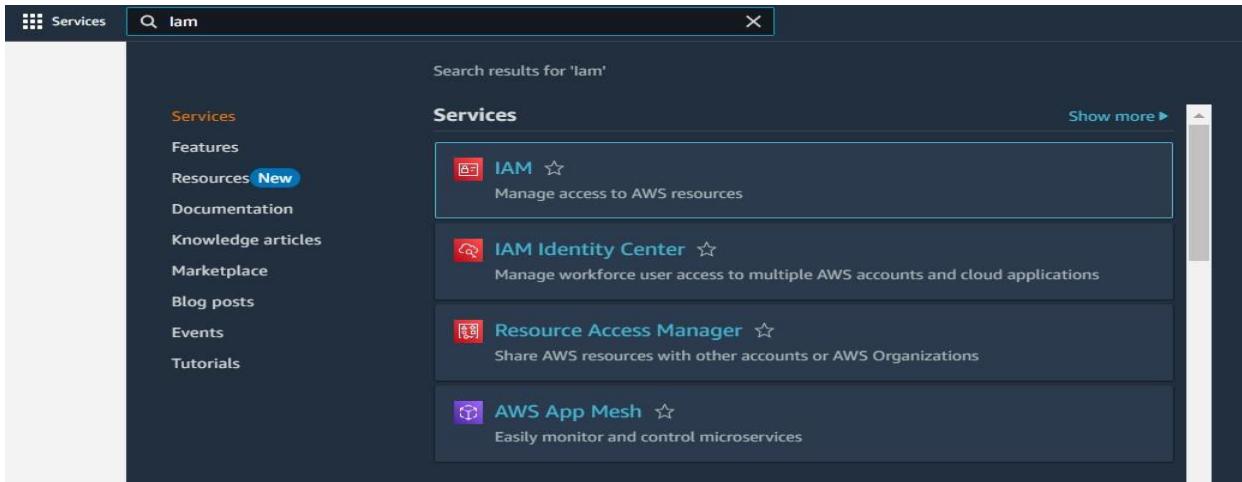
The screenshot shows the AWS SNS 'Topics' page. The left sidebar is identical to the previous screenshot. The main area shows a 'Medtrack' topic with the following details:

Medtrack	
<b>Details</b>	<b>Edit</b> <b>Delete</b> <b>Publish message</b>
Name Medtrack	Display name -
ARN arn:aws:sns:us-east-1:515966514202:Medtrack	Topic owner 515966514202
Type Standard	

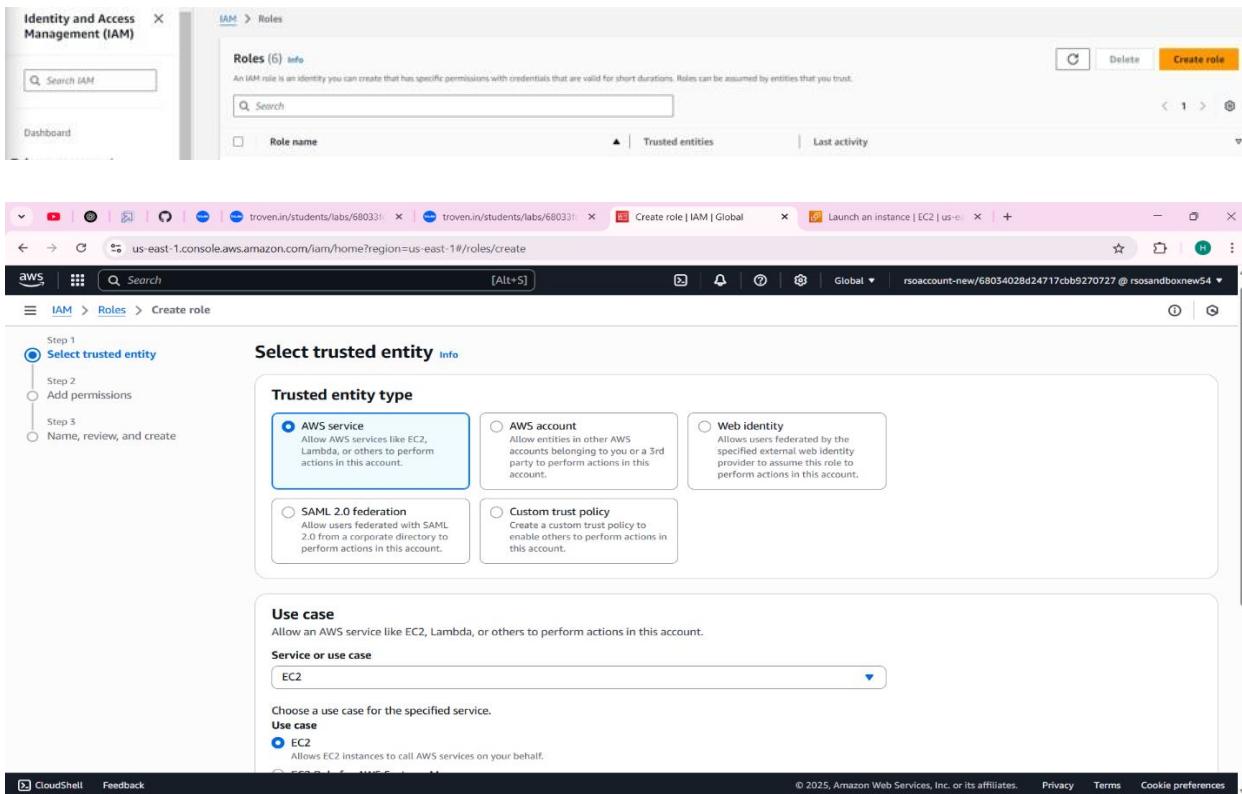
## Milestone 5: IAM Role Setup

- **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.



The screenshot shows the AWS Services search interface. A search bar at the top contains the text 'iam'. Below it, a sidebar on the left lists various services and features under 'Services' and 'Features'. The 'Resources' section is highlighted with a red box. The main area displays a list of services: 'IAM' (Manage access to AWS resources), 'IAM Identity Center' (Manage workforce user access to multiple AWS accounts and cloud applications), 'Resource Access Manager' (Share AWS resources with other accounts or AWS Organizations), and 'AWS App Mesh' (Easily monitor and control microservices). The IAM item is also highlighted with a red box.

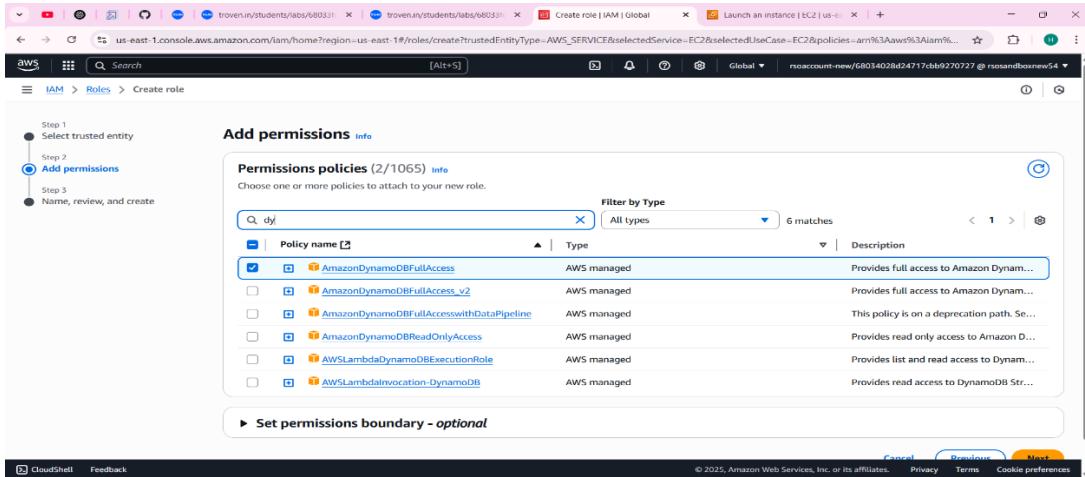


The screenshot shows the 'Create role' wizard in the AWS IAM console. The current step is 'Step 1: Select trusted entity'. On the left, a sidebar shows steps: 'Select trusted entity' (selected), 'Add permissions', and 'Name, review, and create'. The main area is titled 'Select trusted entity' with a 'Trusted entity type' section. It lists five options: 'AWS service' (selected), 'AWS account', 'Web identity', 'SAML 2.0 federation', and 'Custom trust policy'. Below this is a 'Use case' section with a dropdown menu set to 'EC2'. At the bottom, there are sections for 'Choose a use case for the specified service' (with 'EC2' selected) and 'AWS services' (with 'Amazon Simple Queue Service (SQS)' selected). The browser address bar shows the URL: 'us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#roles/create'.

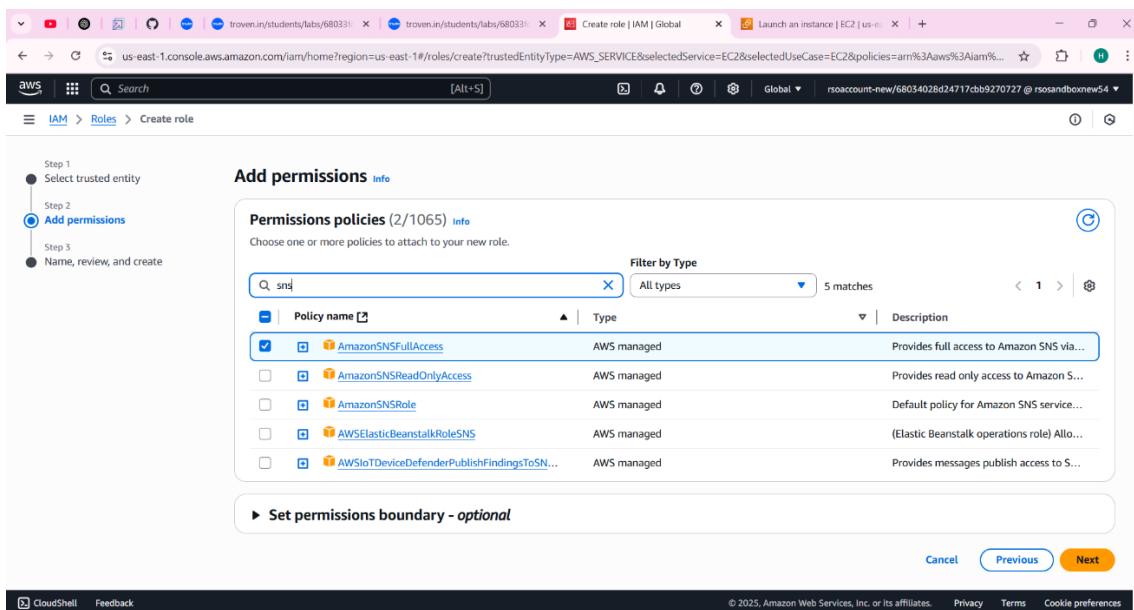
- **Activity 5.2: Attach Policies.**

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 creating a new IAM role. The search bar at the top has 'dy' entered. The results list several AWS managed policies, with 'AmazonDynamoDBFullAccess' selected. Other visible policies include 'AmazonDynamoDBFullAccess\_v2', 'AmazonDynamoDBFullAccesswithDataPipeline', 'AmazonDynamoDBReadOnlyAccess', 'AWSLambdaDynamoDBExecutionRole', and 'AWSLambdaInvocation-DynamoDB'. A 'Set permissions boundary - optional' link is below the policy list. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons.



The screenshot shows the 'Add permissions' step of creating a new IAM role. The search bar at the top has 'sns' entered. The results list several AWS managed policies, with 'AmazonSNSFullAccess' selected. Other visible policies include 'AmazonSNSReadOnlyAccess', 'AmazonSNSRole', 'AWS>ElasticBeanstalkRoleSNS', and 'AWSIoTDeviceDefenderPublishFindingsToSN...'. A 'Set permissions boundary - optional' link is below the policy list. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons.

troven.in/students/labs/68033/ troven.in/students/labs/68033/ Create role | IAM | Global Launch an instance | EC2 | us-east-1 - +

aws Search [Alt+S] Global ⓘ rsosaccount-new/68034028d24717cbb9270727 @ rsosandboxnew54

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.  
**EC2\_MedTrack\_Role**

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

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** Edit

**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

troven.in/students/labs/68033/ troven.in/students/labs/68033/ Create role | IAM | Global Launch an instance | EC2 | us-east-1 - +

aws Search [Alt+S] Global ⓘ rsosaccount-new/68034028d24717cbb9270727 @ rsosandboxnew54

IAM > Roles > Create role

14 ]  
15 ]  
16 ]

### Step 2: Add permissions

Edit

**Permissions policy summary**

Policy name	Type	Attached as
<a href="#">AmazonDynamoDBFullAccess</a>	AWS managed	Permissions policy
<a href="#">AmazonSNSFullAccess</a>	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.

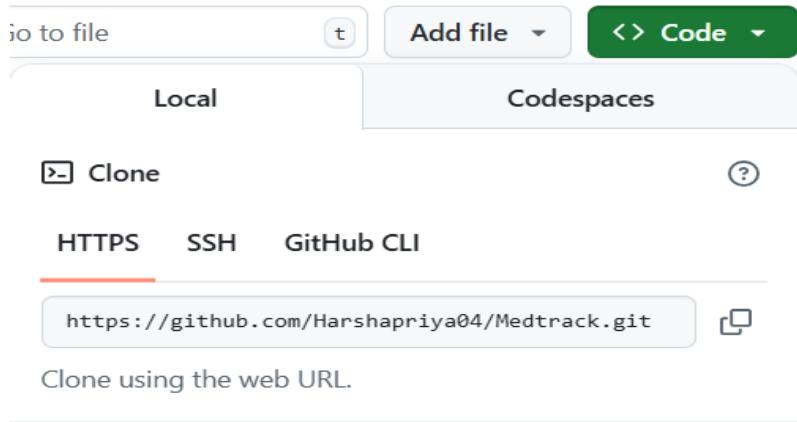
Cancel Previous Create role

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.**

 Harshapriya04	Update app.py	b8a95c3 · 2 days ago	 23 Commits
 templates	Add files via upload	3 days ago	
 .env	Update .env	2 days ago	
 Demo vedio	Create Demo vedio	2 days ago	
 app.py	Update app.py	2 days ago	

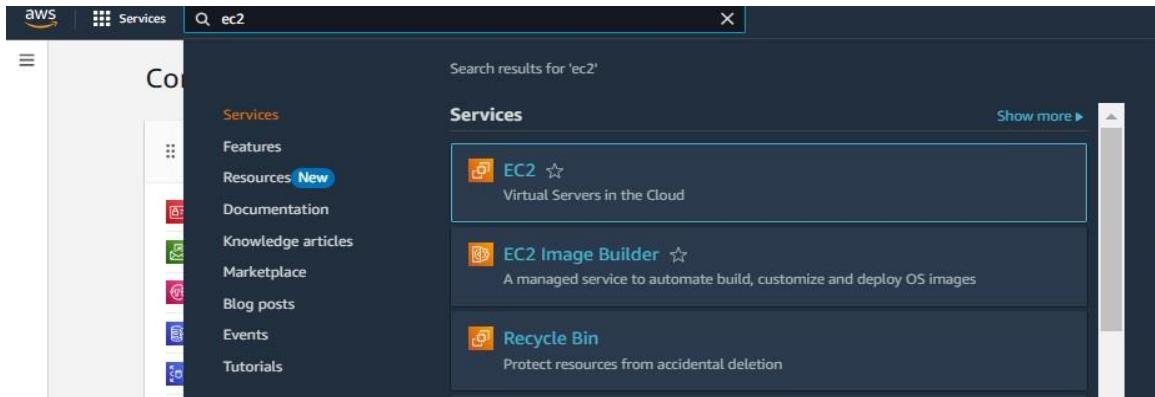


The image shows the GitHub 'Clone' interface. At the top, there are buttons for 'Add file' and 'Code'. Below that, there are tabs for 'Local' and 'Codespaces'. Under the 'Local' tab, there is a 'Clone' button with a 'Clone' icon and a question mark icon. Below the tabs, there are three cloning options: 'HTTPS', 'SSH', and 'GitHub CLI'. The 'HTTPS' option is selected and highlighted with a red underline. Below these options is a text input field containing the URL <https://github.com/Harshapriya04/Medtrack.git>. To the right of the URL is a copy icon. Below the URL, there is a note: 'Clone using the web URL.'

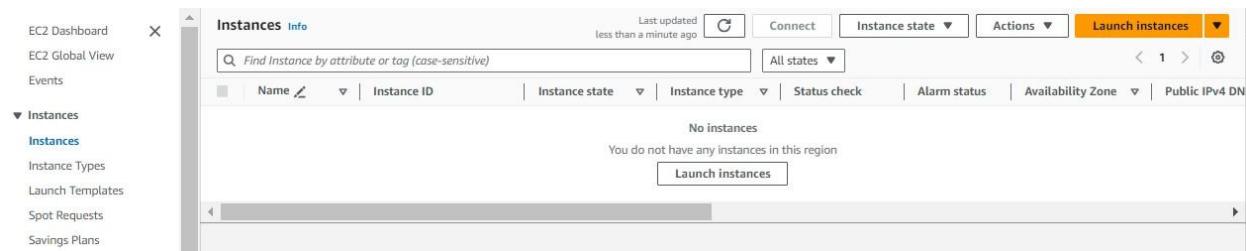
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

### Launch EC2 Instance

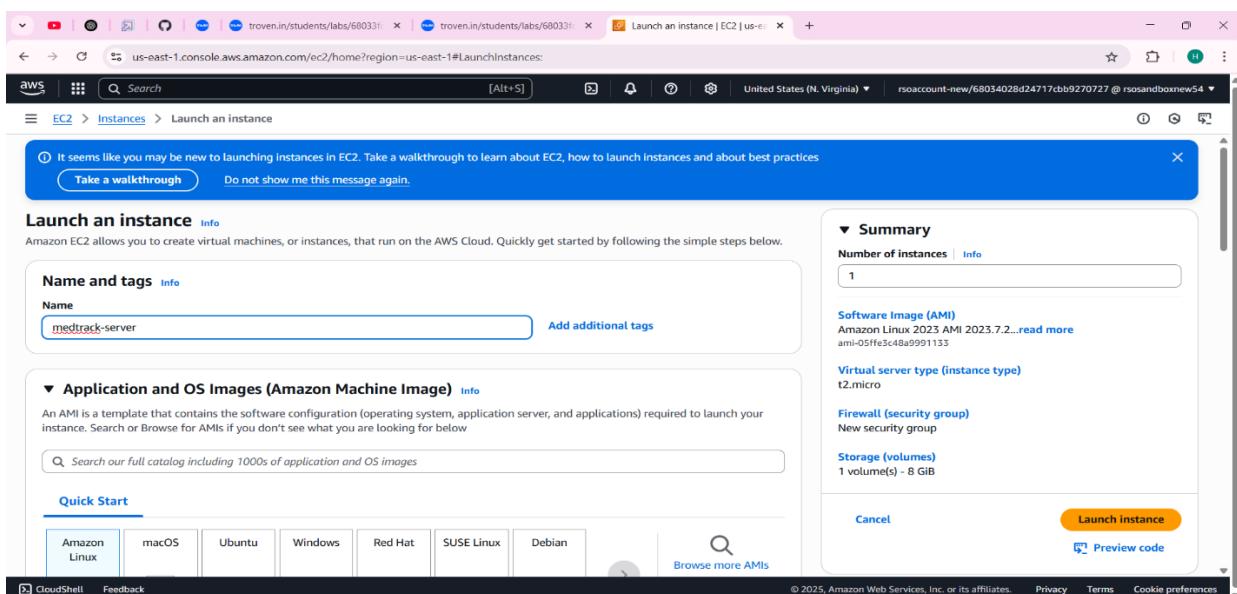
- In the AWS Console, navigate to EC2 and launch a new instance.



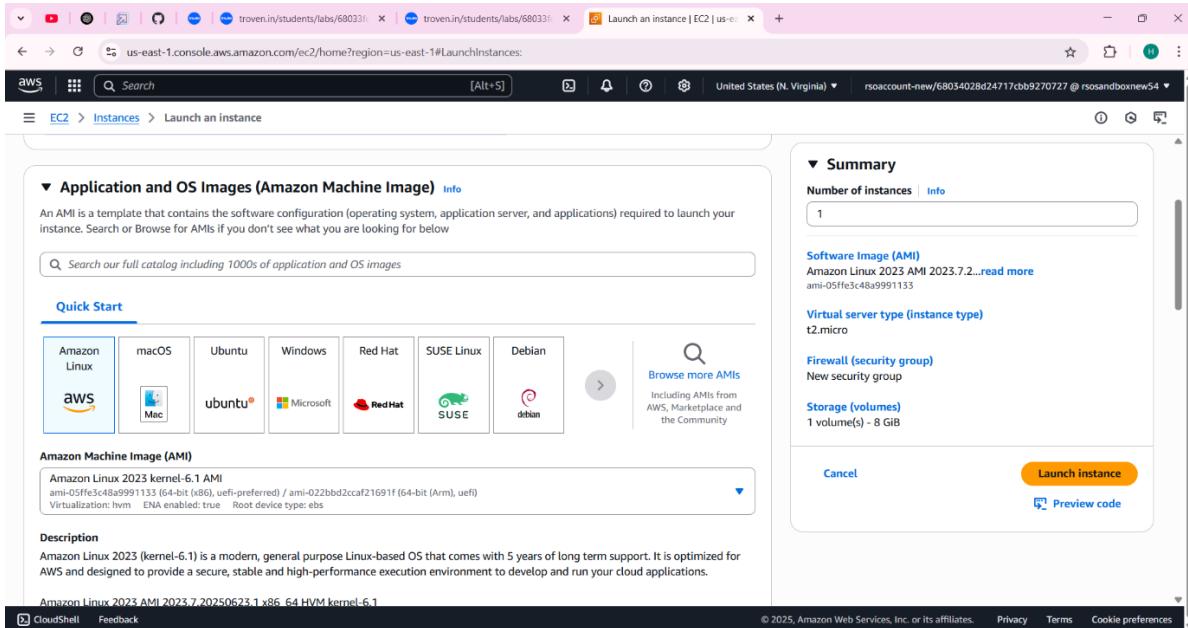
- Click on Launch instance to launch EC2 instance



- Create an instance with the name medtrack-server

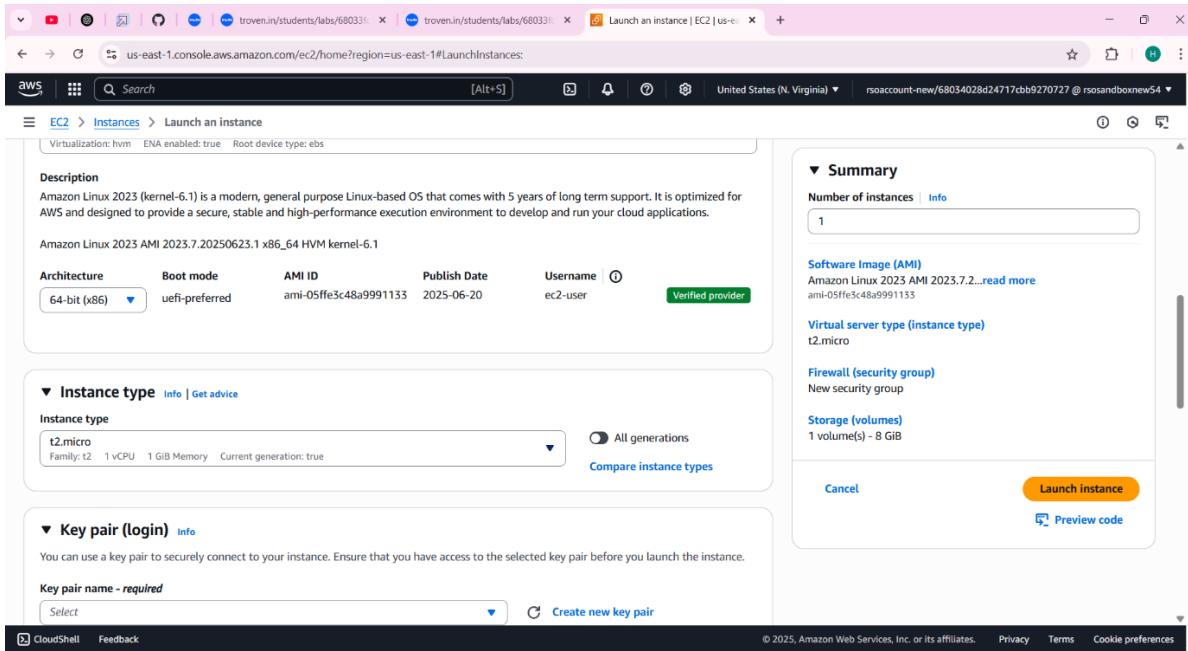


- Choose Amazon Linux 2 and choose Amazon Linux 2023 as the AMI and t2.micro as the instance type (free-tier eligible).



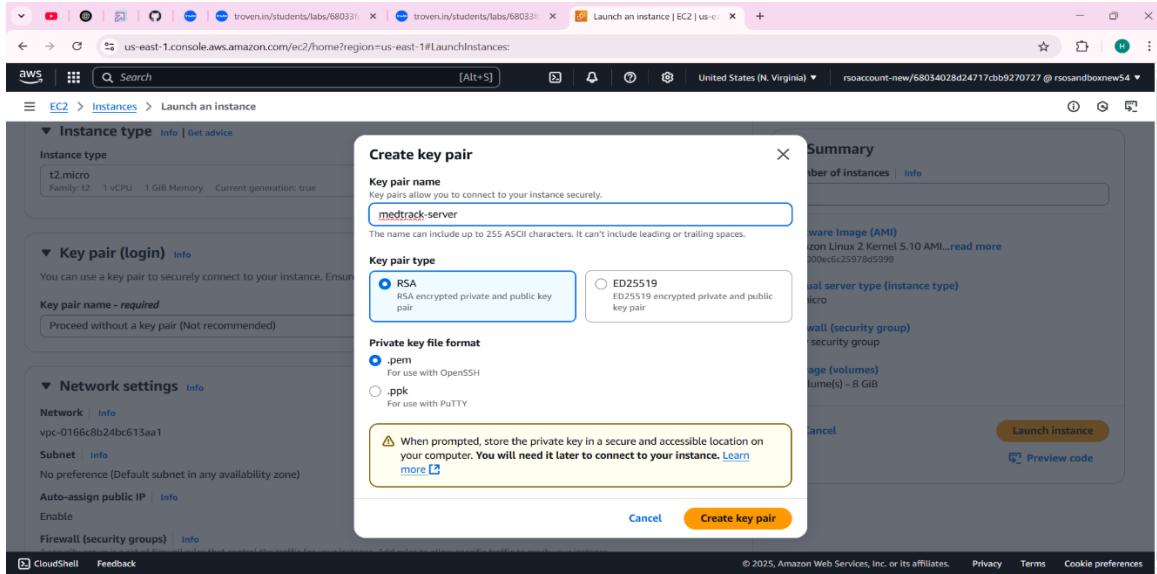
The screenshot shows the AWS EC2 'Launch an instance' wizard. In the 'Application and OS Images (Amazon Machine Image)' step, the 'Amazon Linux' AMI is selected. The instance type is set to 't2.micro'. The 'Virtual server type (instance type)' dropdown also shows 't2.micro'. The 'Firewall (security group)' dropdown shows 'New security group'. Under 'Storage (volumes)', it says '1 volume(s) - 8 GiB'. At the bottom right, there is a large orange 'Launch instance' button.

- Create and download the key pair for Server access.

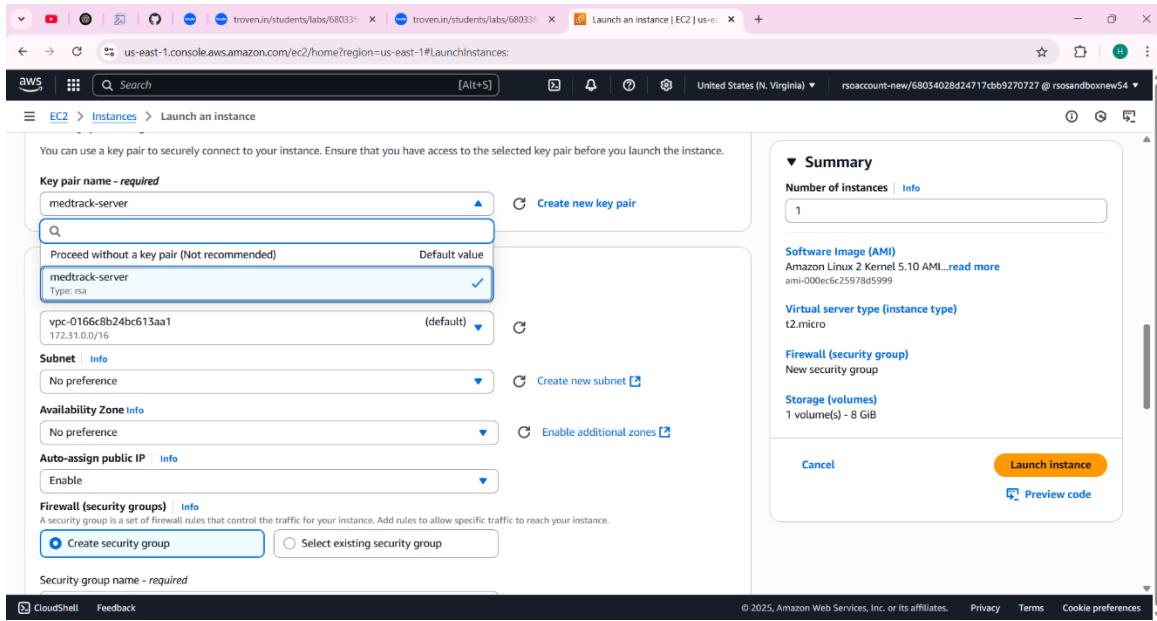


The screenshot shows the AWS EC2 'Launch an instance' wizard. In the 'Instance type' step, the 't2.micro' instance type is selected. In the 'Key pair (login)' step, a new key pair is being created with the name 'Select'. The 'Create new key pair' button is visible. The 'Launch instance' button is highlighted at the bottom right.

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



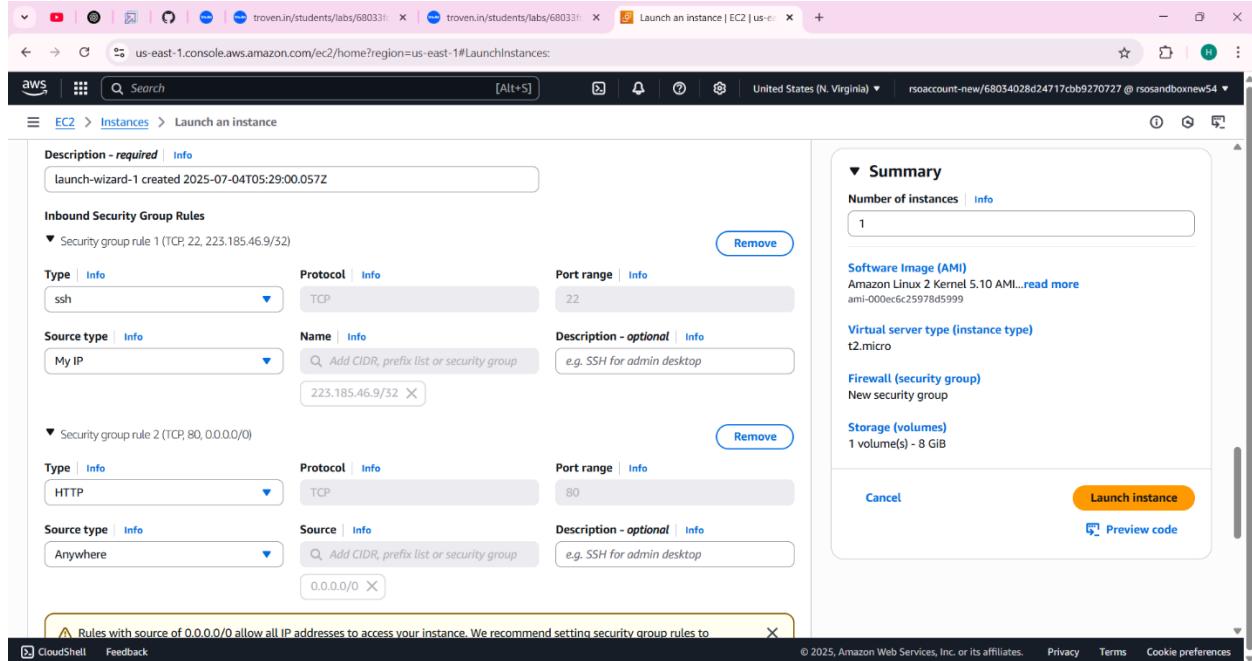
The screenshot shows the AWS EC2 'Launch an instance' wizard. On the left, the main configuration screen includes sections for Instance type (t2.micro), Key pair (medtrack-server), Network settings, and Subnet. A modal window titled 'Create key pair' is open, showing the 'Key pair name' field filled with 'medtrack-server'. Below it, the 'Key pair type' section has 'RSA' selected. The 'Private key file format' section has 'pem' selected. A note at the bottom of the modal says: '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.' At the bottom of the modal are 'Cancel' and 'Create key pair' buttons. To the right, the 'Summary' tab displays the instance configuration: AMI (Amazon Linux 2 Kernel 5.10 AMI), Virtual server type (t2.micro), and Storage (1 volume(s) - 8 GiB). The 'Launch instance' button is highlighted in orange.



This screenshot shows the same AWS EC2 'Launch an instance' wizard as the previous one, but the 'Create key pair' modal is no longer open. The main configuration screen now shows the selected key pair 'medtrack-server' in the 'Key pair name - required' field. The 'Summary' tab on the right shows the final configuration: Number of instances (1), Software Image (Amazon Linux 2 Kernel 5.10 AMI), Virtual server type (t2.micro), and Storage (1 volume(s) - 8 GiB). The 'Launch instance' button is highlighted in orange.

- **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.

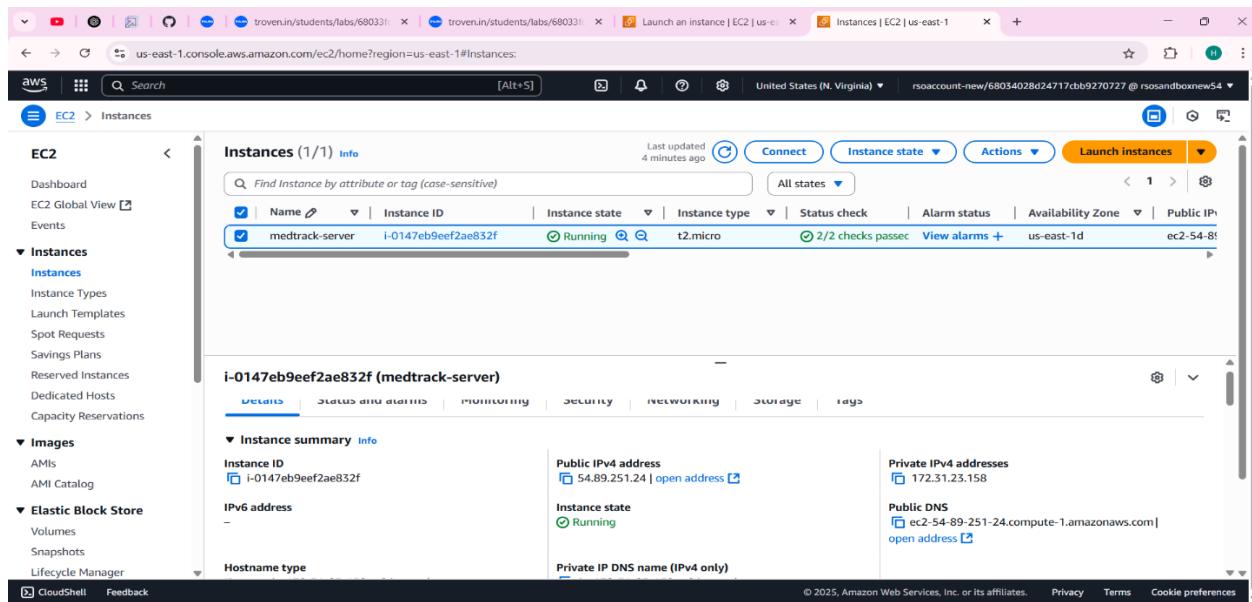


The screenshot shows the AWS EC2 'Launch an instance' wizard. In the 'Inbound Security Group Rules' section, there are two rules defined:

- Security group rule 1 (TCP; 22, 223.185.46.9/32):** Type: ssh, Protocol: TCP, Port range: 22, Source type: My IP, Description: e.g. SSH for admin desktop. The source is set to 223.185.46.9/32.
- 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. The source is set to 0.0.0.0/0.

A note at the bottom left says: "⚠️ 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 right side of the screen displays a summary of the instance configuration, including the number of instances (1), software image (Amazon Linux 2 Kernel 5.10 AMI), virtual server type (t2.micro), and storage (1 volume(s) - 8 GiB). At the bottom right are 'Cancel', 'Launch instance', and 'Preview code' buttons.



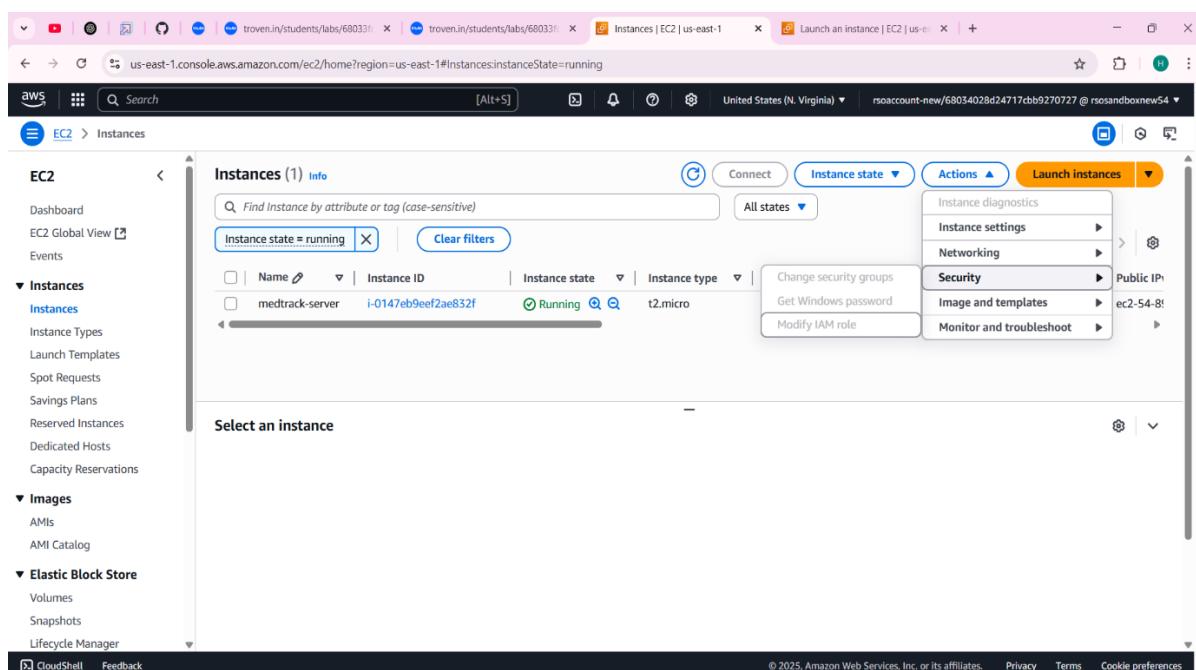
The screenshot shows the AWS EC2 Instances page. The main table lists one instance:

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-

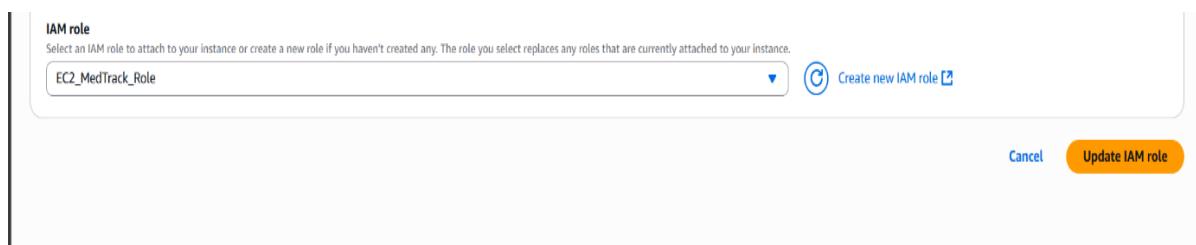
Below the table, the details for the instance 'i-0147eb9eef2ae832f (medtrack-server)' are shown. Key information includes:

- Public IPv4 address:** 54.89.251.24 (with a 'View address' link)
- Instance state:** Running
- Private IP DNS name (IPv4 only):** ec2-54-89-251-24.compute-1.amazonaws.com (with a 'View address' link)
- Private IPv4 address:** 172.31.23.158

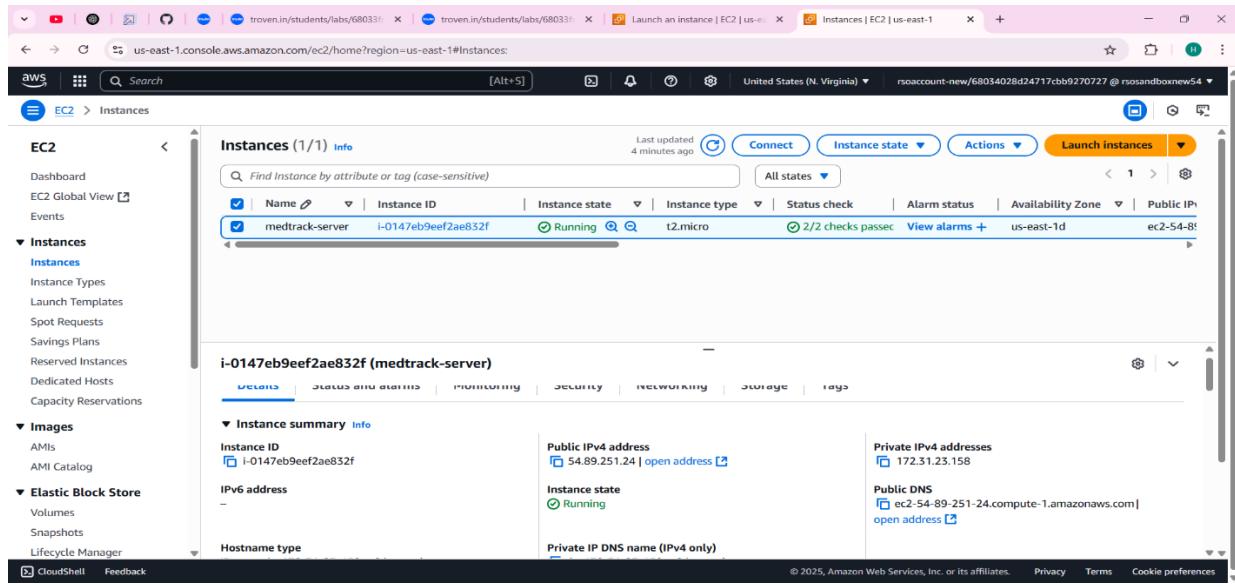
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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



The screenshot shows the AWS Management Console interface for the EC2 service. The left sidebar has a tree view with 'EC2' selected, under which 'Instances' is expanded, showing options like 'Instances', 'Instance Types', 'Launch Templates', etc. The main content area is titled 'Instances (1) info'. It lists one instance: 'medtrack-server' (Instance ID: i-0147eb9ef2ae832f), which is 'Running'. To the right of the instance list is a context menu with options like 'Connect', 'Instance state', 'Actions', and 'Launch instances'. Below the instance list is a 'Select an instance' dropdown. At the bottom of the page, there are links for 'CloudShell', 'Feedback', and copyright information: '© 2025, Amazon Web Services, Inc. or its affiliates.' and 'Privacy Terms Cookie preferences'.

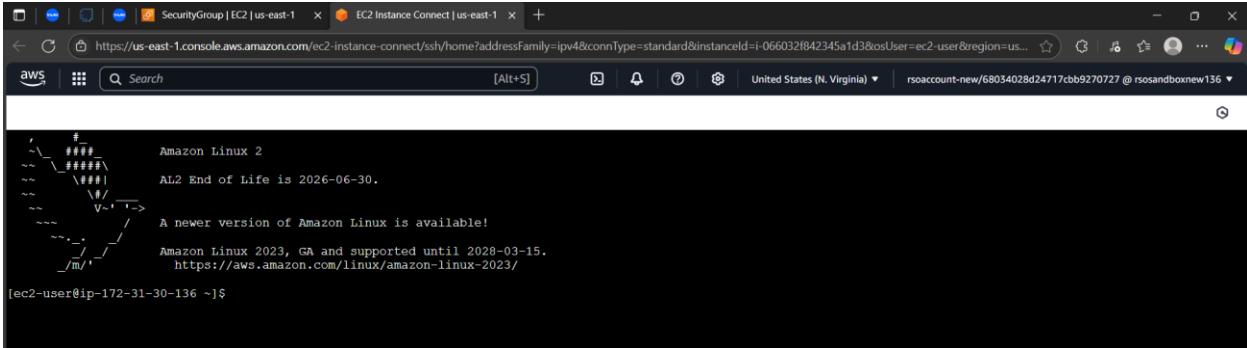


The screenshot shows a modal dialog box titled 'IAM role'. It contains a text input field with the value 'EC2\_MedTrack\_Role' and a 'Create new IAM role' button. At the bottom of the dialog are 'Cancel' and 'Update IAM role' buttons.



The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with sections like Dashboard, EC2 Global View, Events, Instances (with sub-options like Instances Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes, Snapshots, Lifecycle Manager). The main content area displays a table titled 'Instances (1/1) Info'. It shows one instance: 'medtrack-server' (Instance ID: i-0147eb9eef2ae832f), which is 'Running' (t2.micro), has 2/2 checks passed, and is located in us-east-1d with a Public IP of ec2-54-8. There are buttons for Connect, Instance state, Actions, and Launch instances.

- Now connect the EC2 with the files



The screenshot shows an EC2 Instance Connect session. The terminal window displays a message about the end of life for Amazon Linux 2, mentioning that it will end in 2026-06-30 and that a newer version, Amazon Linux 2023, is available until 2028-03-15. The URL <https://aws.amazon.com/linux/amazon-linux-2023/> is provided. The command [ec2-user@ip-172-31-30-136 ~]\$ is visible at the bottom.

## 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
```



```
SMARTBRIDGE
Let's Bridge the Gap

us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-0147eb9eef2ae832f&osUser=ec2-user&region=us-east-1&sshPort=223.185.46.9
aws | Search [Alt+S] | United States (N. Virginia) | rsoaccount-new/68034028d24717ccb9270727 @ rsandboxnew54

if g.user['role'] != 'doctor':
    raise 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 / HTTP/1.1" 200 -
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-0147eb9eef2ae832f (medtrack-server)
PublicIPs: 54.89.251.24 PrivateIPs: 172.31.23.158

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

## Verify the Flask app is running:

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

- Run the Flask app on the EC2 instance

```
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-0147eb9eef2ae832f&osUser=ec2-user&region=us-east-1&sshPort=223.185.46.9
aws | Search [Alt+S] | United States (N. Virginia) | rsoaccount-new/68034028d24717ccb9270727 @ rsandboxnew54

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):
```

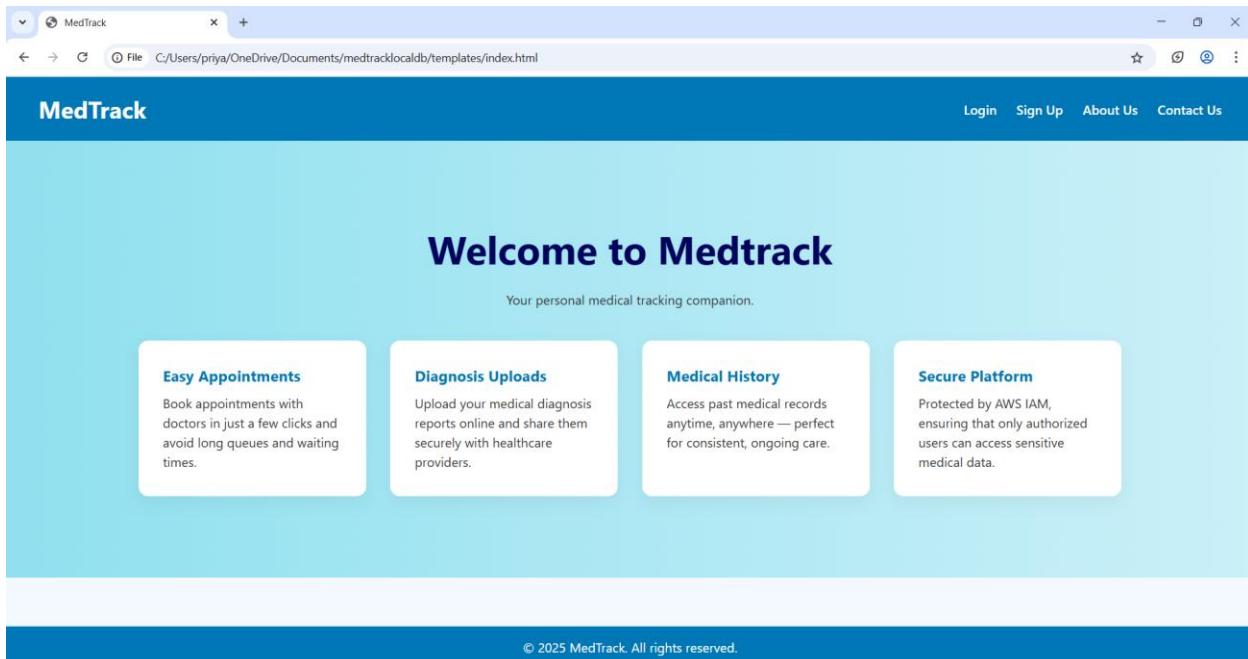
## Access the website through:

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

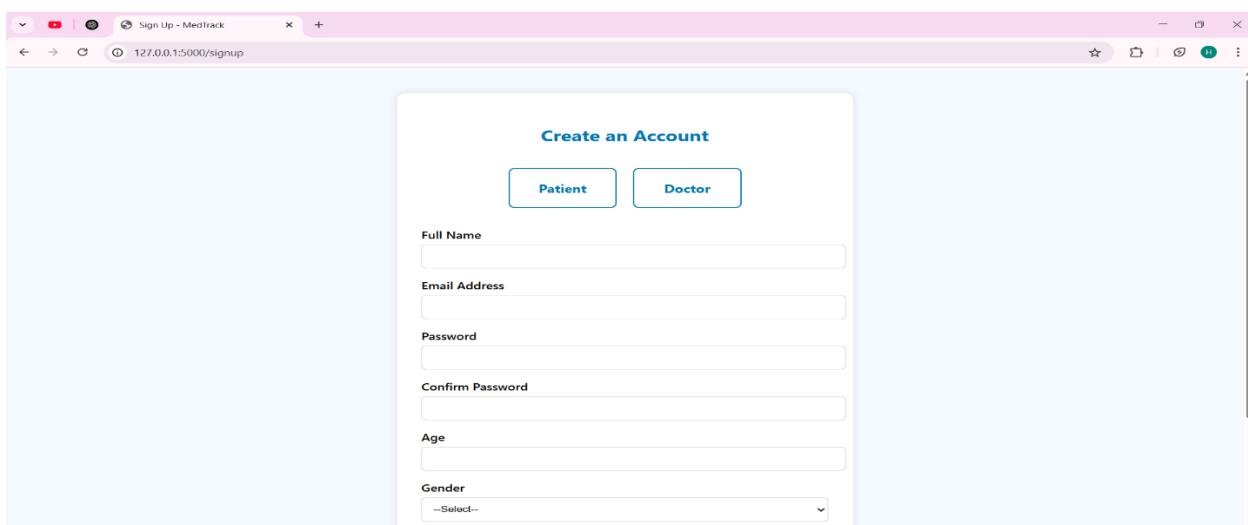
## Milestone 8: Testing and Deployment

- **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

Full Name

Email Address

Password

Confirm Password

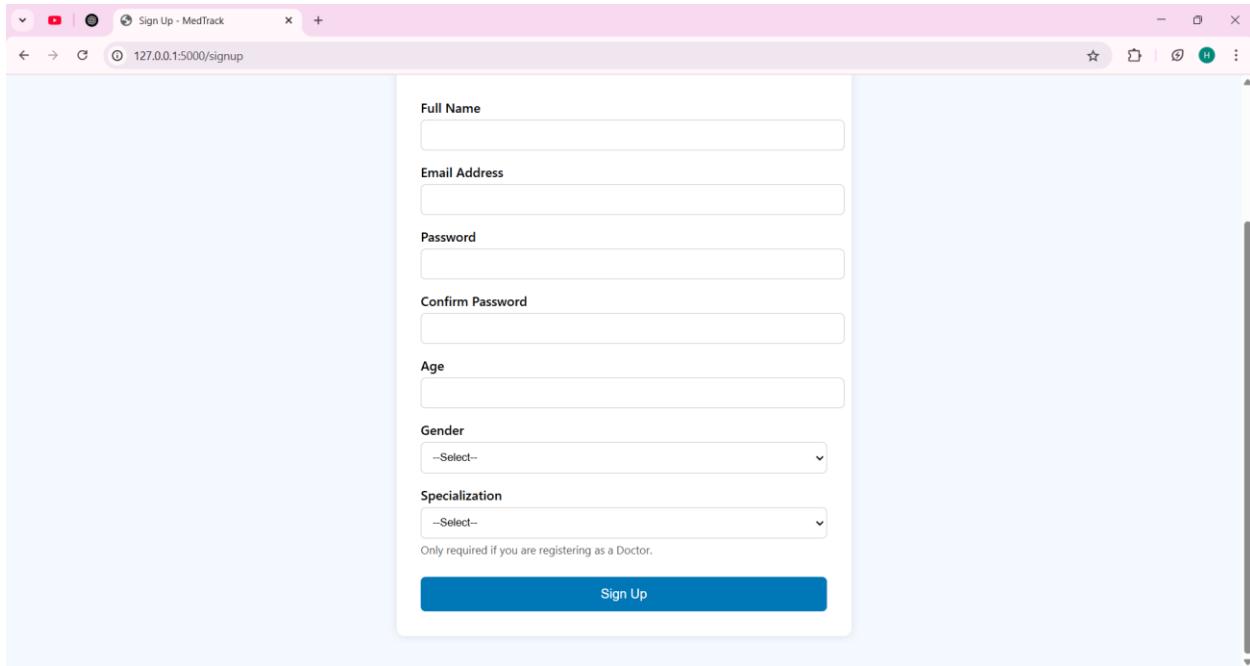
Age

Gender

Specialization

Only required if you are registering as a Doctor.

Sign Up



### Login page:

Doctor Dashboard - MedTrack

Login - MedTrack

127.0.0.1:5000/login

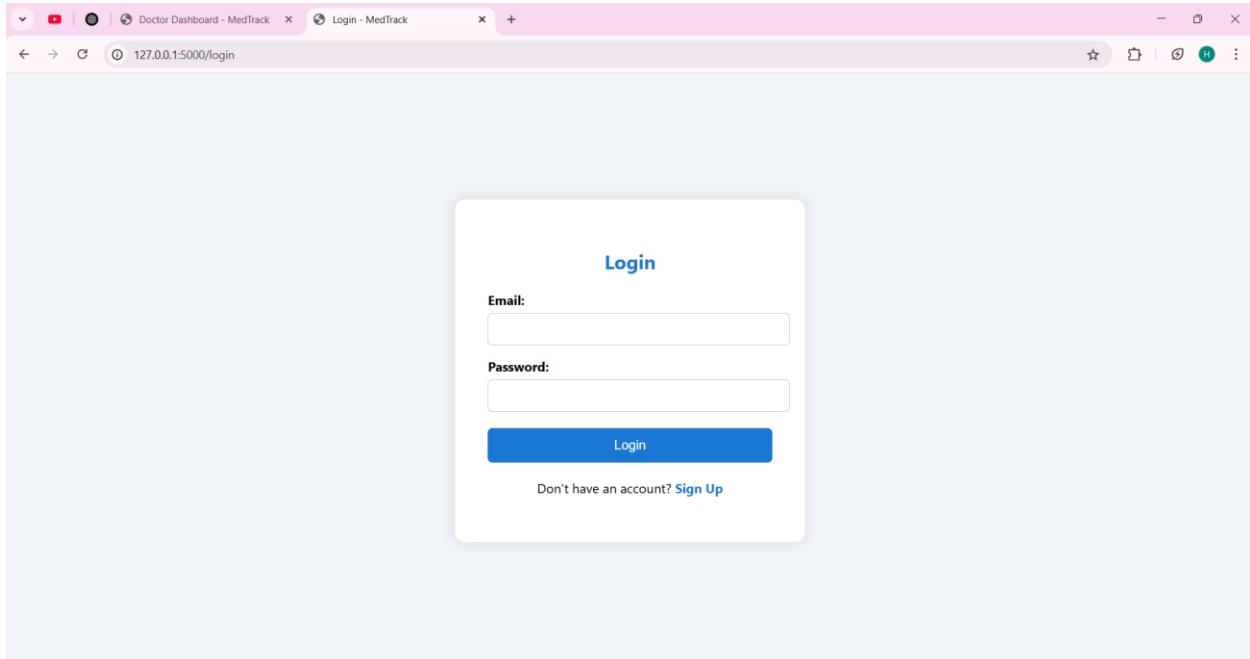
Login

Email:

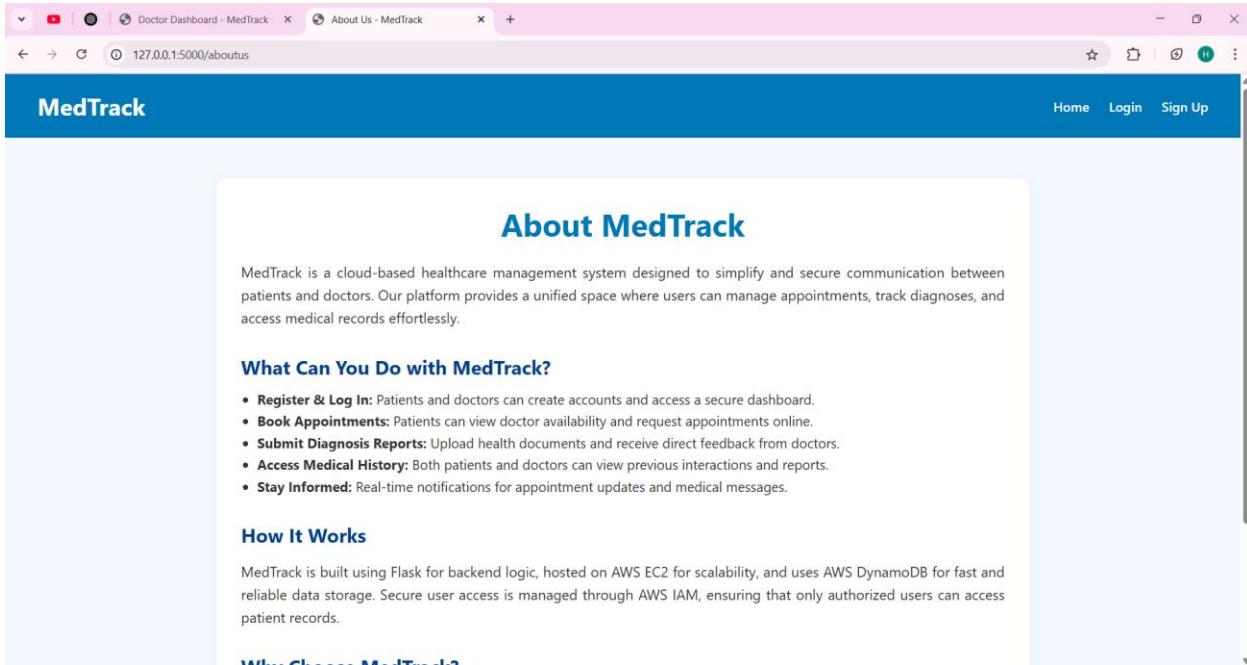
Password:

Login

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



## About Us page:



The screenshot shows a web browser window with a pink header bar. The title bar says "Doctor Dashboard - MedTrack" and "About Us - MedTrack". The address bar shows "127.0.0.1:5000/aboutus". The main content area has a blue header "MedTrack" and a navigation bar with "Home", "Login", and "Sign Up". Below this is a section titled "About MedTrack" which contains a paragraph about the system. Underneath is a section titled "What Can You Do with MedTrack?" with a bulleted list of features. Another section titled "How It Works" follows, with a paragraph explaining the technology stack. At the bottom is a "Why Choose MedTrack?" section.

**About MedTrack**

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.

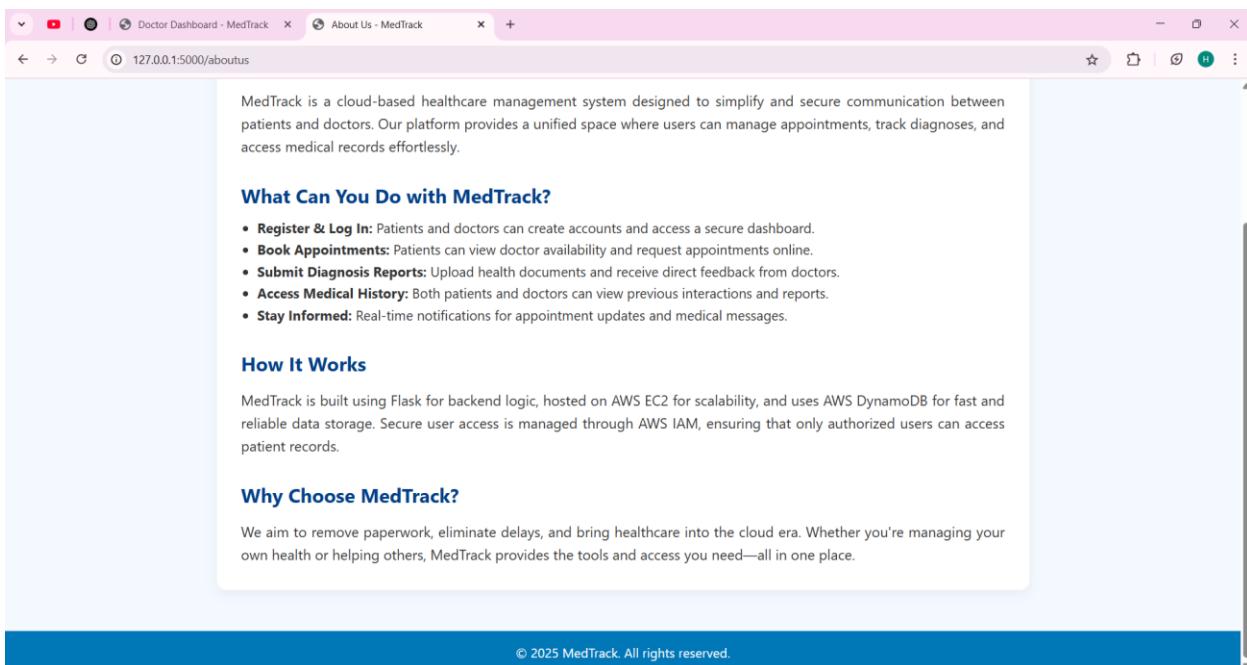
**What Can You Do with MedTrack?**

- **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?**



This screenshot shows the same About Us page but with a different visual style. The overall layout is similar, with the "About Us" section and its sub-sections ("What Can You Do with MedTrack?", "How It Works", and "Why Choose MedTrack?") all contained within a single large white box. The rest of the page (header, navigation, and footer) is visible around this central content area.

**About MedTrack**

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.

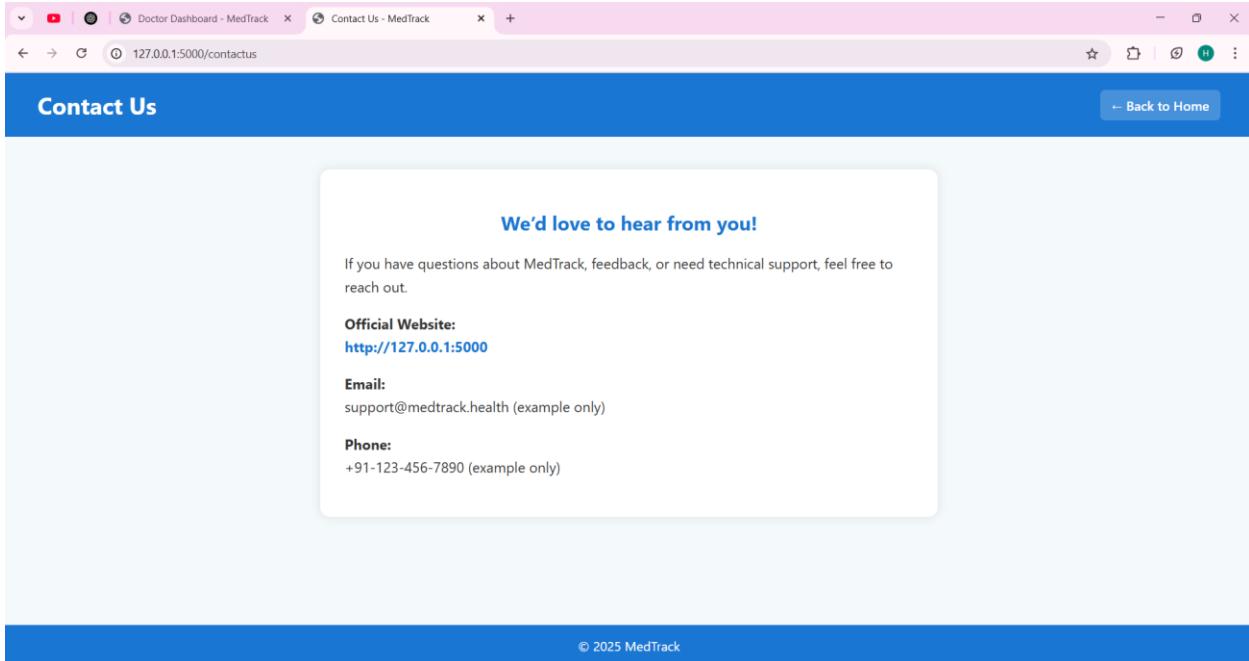
**How It Works**

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?**

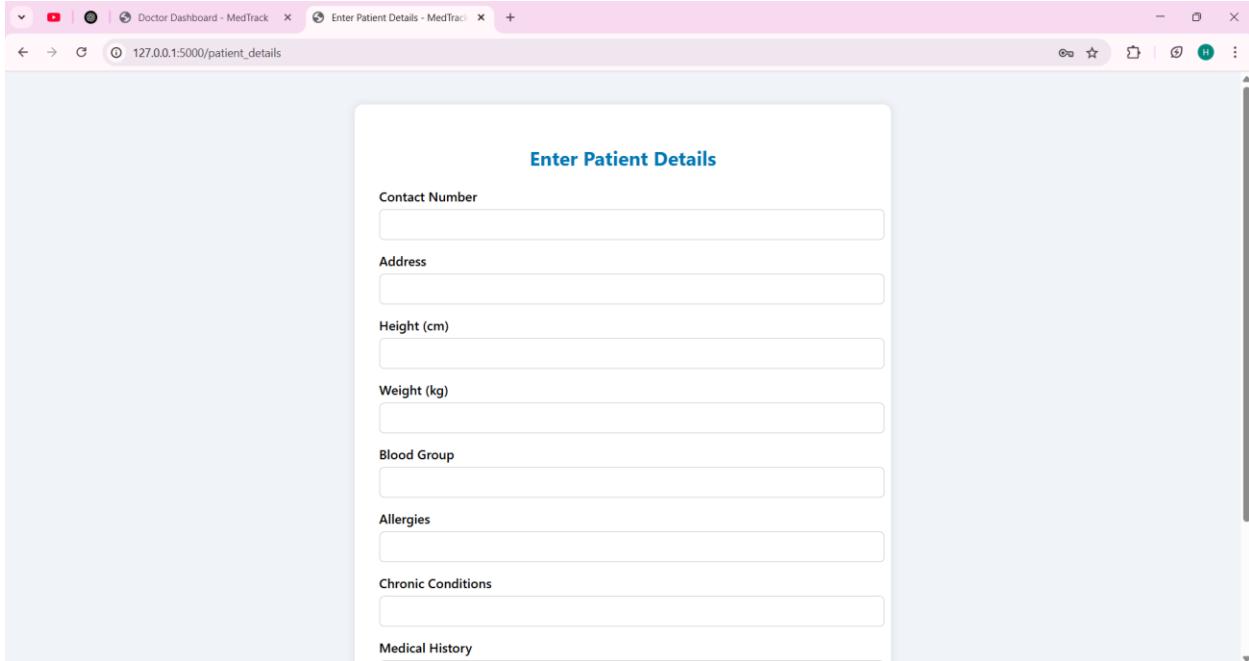
We aim to remove paperwork, eliminate delays, and bring healthcare into the cloud era. Whether you're managing your own health or helping others, MedTrack provides the tools and access you need—all in one place.

## Contactus Page:



The screenshot shows a web browser window with a pink header bar. The address bar shows the URL [127.0.0.1:5000/contactus](http://127.0.0.1:5000/contactus). The main content area has a blue header "Contact Us" and a "Back to Home" button. Below this is a white form box with a title "We'd love to hear from you!". It contains text about reaching out for questions or support, followed by contact information: Official Website (<http://127.0.0.1:5000>), Email (support@medtrack.health), and Phone (+91-123-456-7890). At the bottom of the page is a blue footer bar with the text "© 2025 MedTrack".

## Patient Details page:



The screenshot shows a web browser window with a pink header bar. The address bar shows the URL [127.0.0.1:5000/patient\\_details](http://127.0.0.1:5000/patient_details). The main content area has a white form box titled "Enter Patient Details". It contains fields for entering patient information: Contact Number, Address, Height (cm), Weight (kg), Blood Group, Allergies, Chronic Conditions, and Medical History. Each field is represented by a text input box.

Doctor Dashboard - MedTrack    Enter Patient Details - MedTrack    127.0.0.1:5000/patient\_details

Address

Height (cm)

Weight (kg)

Blood Group

Allergies

Chronic Conditions

Medical History

**Submit**

## Patient Dashboard:

Doctor Dashboard - MedTrack    Patient Dashboard - MedTrack    127.0.0.1:5000/patient\_dashboard

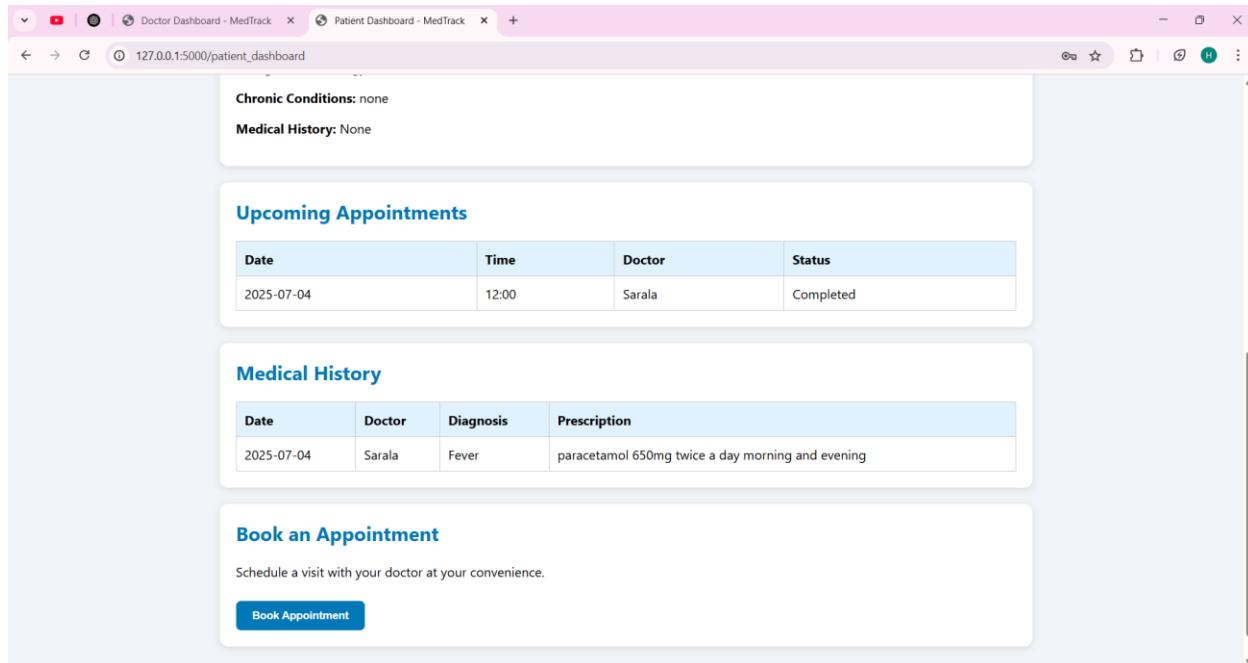
**MedTrack - Patient Dashboard**    [Logout](#)

**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



This screenshot shows the Patient Dashboard interface. At the top, there are two tabs: "Doctor Dashboard - MedTrack" and "Patient Dashboard - MedTrack". The URL in the address bar is 127.0.0.1:5000/patient\_dashboard.

The dashboard displays the following sections:

- Chronic Conditions:** none
- Medical History:** None
- Upcoming Appointments:**

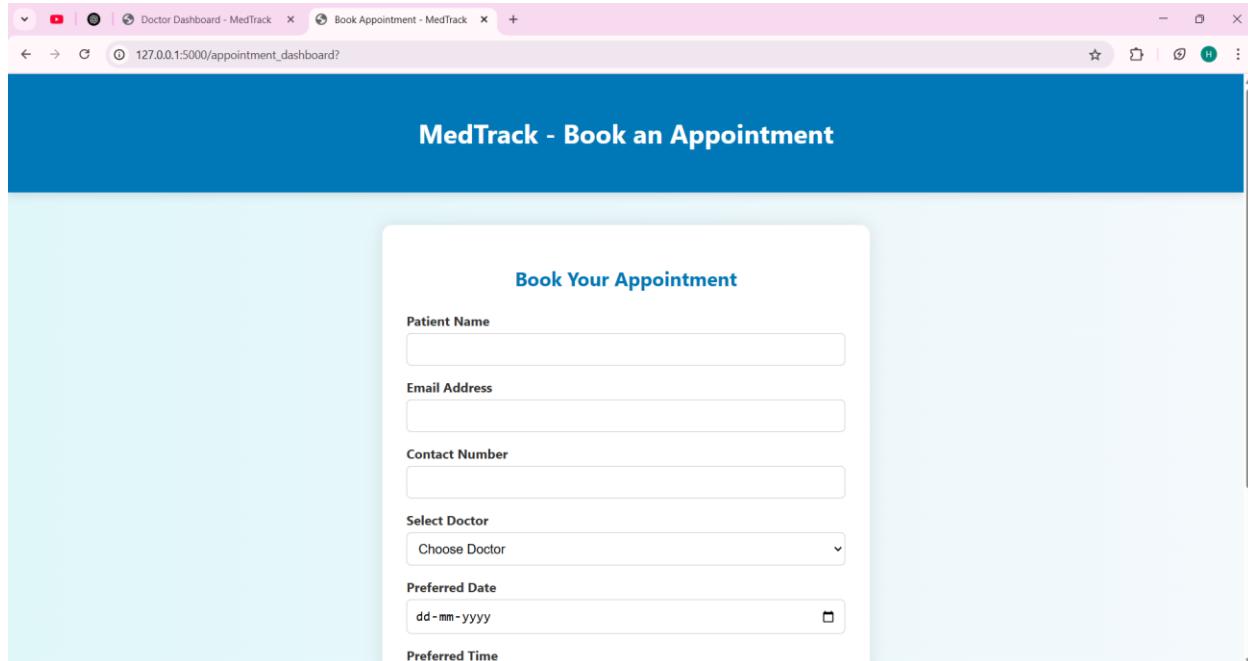
Date	Time	Doctor	Status
2025-07-04	12:00	Sarala	Completed
- Medical History:**

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:



This screenshot shows the Book Appointment page. The title bar reads "MedTrack - Book an Appointment".

The form is titled "Book Your Appointment" and contains the following fields:

- 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

### 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**

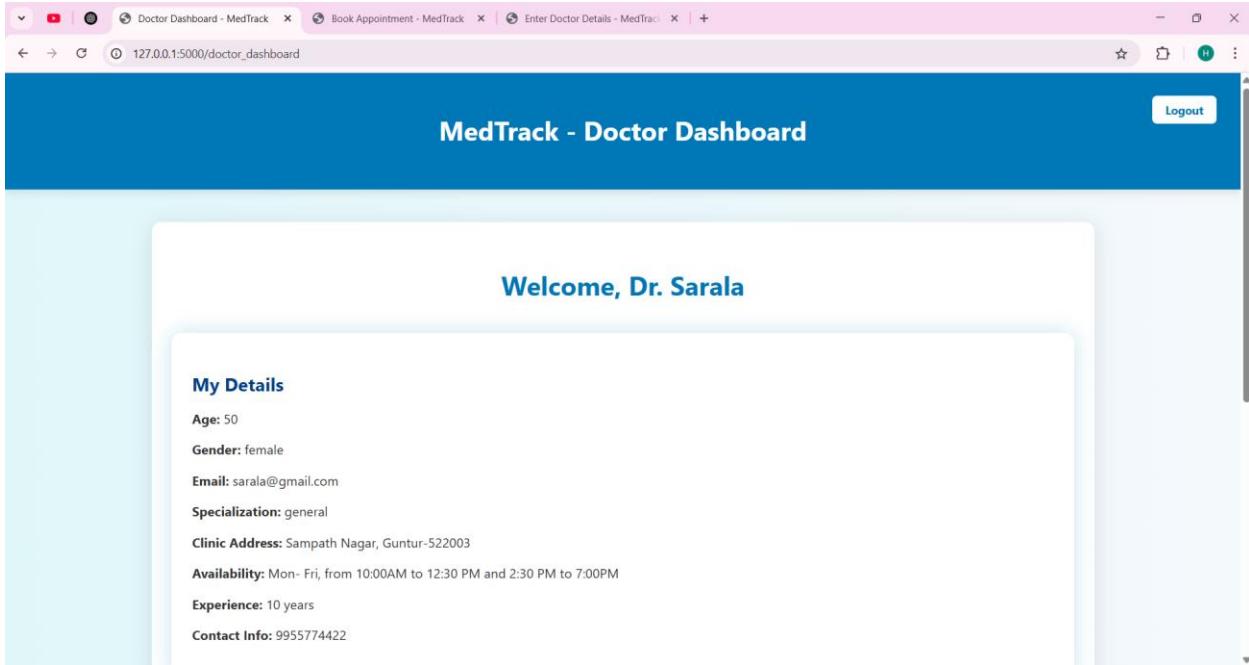
Experience (in years)

Clinic Address

Contact Info

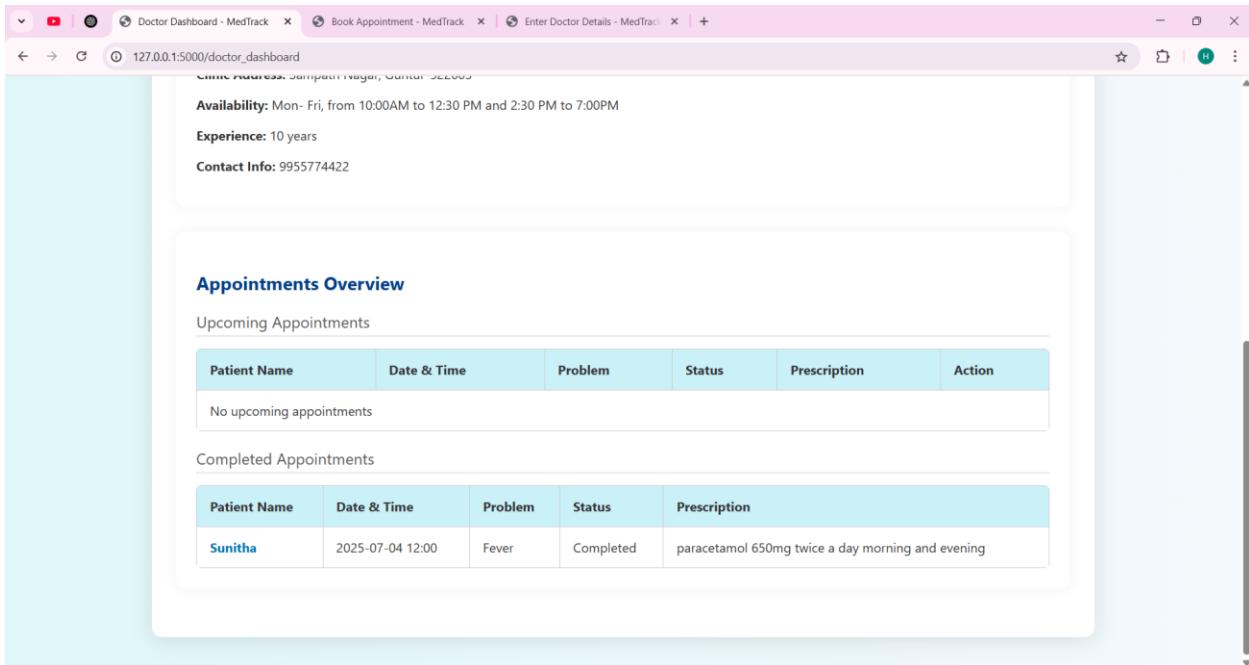
Availability

## Doctor Dashboard Page:



The screenshot shows a web browser window with three tabs open: "Doctor Dashboard - MedTrack", "Book Appointment - MedTrack", and "Enter Doctor Details - MedTrack". The main content area is titled "MedTrack - Doctor Dashboard" and "Welcome, Dr. Sarala". A "Logout" button is in the top right corner. The "My Details" section lists the following information:

- 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



The screenshot shows the same browser window with the "Doctor Dashboard - MedTrack" tab active. The main content area is titled "MedTrack - Doctor Dashboard" and "Welcome, Dr. Sarala". The "Appointments Overview" section contains the following information:

**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

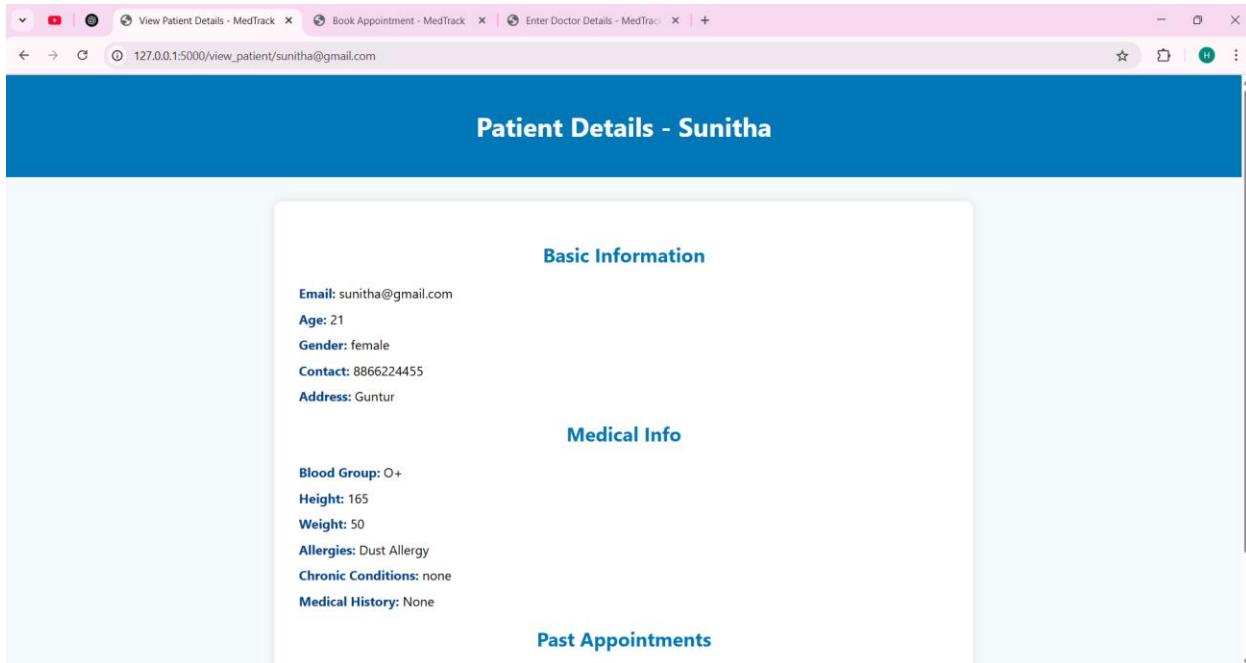
**Upcoming Appointments**

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:



**Patient Details - Sunitha**

**Basic Information**

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

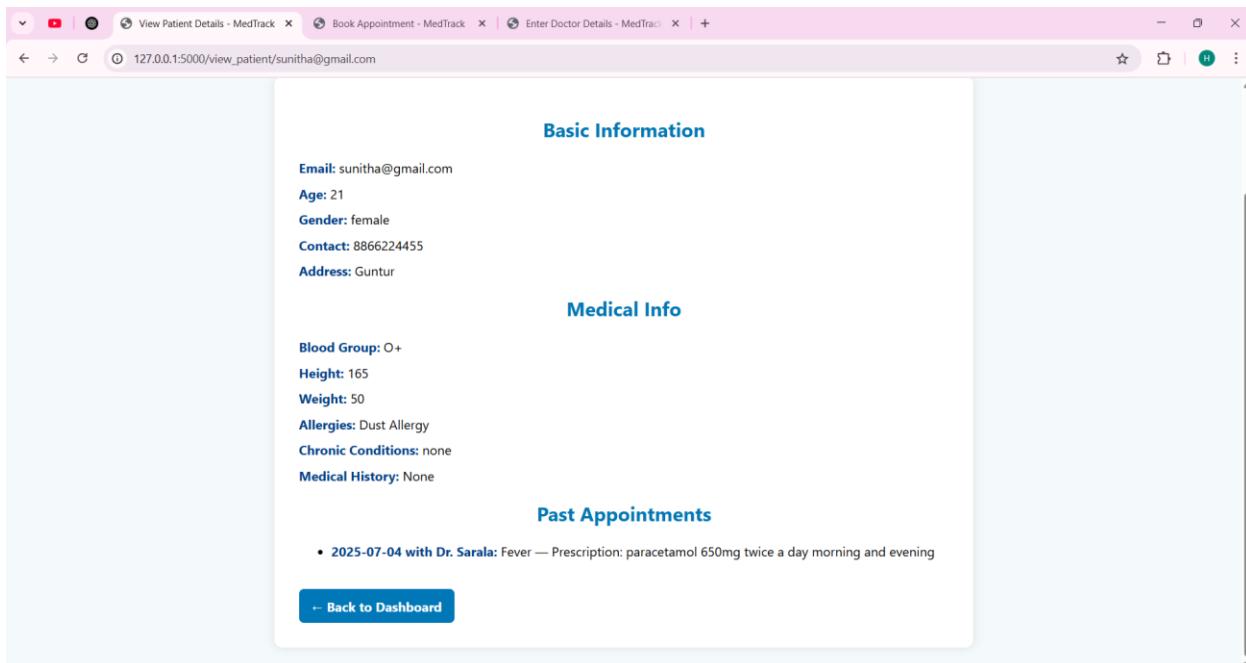
**Medical Info**

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

**Past Appointments**

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

[Back to Dashboard](#)



**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**

- 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									
	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						
	id	email	experience	clinic_address	contact	availability
▶	2	suni@gmail.com	8	Laksmipuram, Guntur	9876543210	Mon-Fri 9am-5pm
	3	ravi@gmail.com	10	Koritipadu , Guntur	9596442371	From 10:00AM to 12:30PM
	4	suma@gmail.com	8	Inner Ring Road, Guntur	9955774422	Mon- Fri, from 10:00AM to 12:30PM
	5	ramesh@gmail.com	9	Nallapadu, Guntur	8844662277	Mon- Fri, from 10:00AM to 12:30 PM an
*	6	sarala@gmail.com	10	Kothapeta, Guntur	8321654972	Mon- Fri, from 10:00AM to 12:30PM
*	NULL	NULL	NULL	NULL	NULL	NULL

### 3. Patient Details table:

35 • `select * from patient_detail;`

Result Grid										
	id	email	contact	address	height	weight	blood_group	allergies	conditions	histor
▶	1	saketh@gmail.com	7755331155	guntur	155	50	o+	none	none	none
	2	praneetha@gmail.com	8529637413	guntur	160	55	o+	none	none	none
	3	vamsi@gmail.com	8321654972	Guntur	145	40	o-	none	none	none
	4	sowmya@gmail.com	8654723791	Guntur	165	50	o+	none	none	None
	5	siva@gmail.com	9652387142	Guntur	175	50	o+	none	none	none
*	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

### 4. Appointment table:

32 • `select * from appointment;`

Result Grid								
	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	Vomiting 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	Vomiting 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;`

Result Grid					
	id	email	date	doctor	diagnosis
▶	1	saketh@gmail.com	2025-06-28	Sunitha	Fever
	2	praneetha@gmail.com	2025-06-27	Sunitha	Fever
	3	praneetha@gmail.com	2025-06-27	Ravindra	Fever
	4	vamsi@gmail.com	2025-06-27	Ravindra	Fever
	5	vamsi@gmail.com	2025-06-28	Sunitha	Vomiting and stomach pain
	6	praneetha@gmail.com	2025-06-27	Suma Latha	Imbalance Menstrual cycle
	7	praneetha@gmail.com	2025-06-29	Ravindra	Vomiting and Fever
	8	sowmya@gmail.com	2025-06-28	Ramesh	Severe Tooth Pain
	9	vamsi@gmail.com	2025-06-28	Ramesh	Regular Tooth Checkup
	10	siva@gmail.com	2025-06-29	Sarala	Severe ear pain
*	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.