

MedTrack: AWS Cloud-Enabled Healthcare Management System

Project Description:

In today's fast-evolving healthcare landscape, efficient communication and coordination between doctors and patients are crucial. MedTrack is a cloud-based healthcare management system that streamlines patient doctor interactions by providing a centralized platform for booking appointments, managing medical histories, and enabling diagnosis submissions. To address these challenges, the project utilizes Flask for backend development, AWS EC2 for hosting, and DynamoDB for managing data. MedTrack allows patients to register, log in, book appointments, and submit diagnosis reports online. The system ensures real-time notifications, enhancing communication between doctors and patients regarding appointments and medical submissions. Additionally, AWS Identity and Access Management (IAM) is employed to ensure secure access control to AWS resources, allowing only authorized users to access sensitive data. This cloud-based solution improves accessibility and efficiency in healthcare services for all users.

Scenario 1: Efficient Appointment Booking System for Patients

In the MedTrack system, AWS EC2 provides a reliable infrastructure to manage multiple patients accessing the platform simultaneously. For example, a patient can log in, navigate to the appointment booking page, and easily submit a request for an appointment. Flask handles backend operations, efficiently retrieving and processing user data in real-time. The cloud-based architecture allows the platform to handle a high volume of appointment requests during peak periods, ensuring smooth operation without delays.

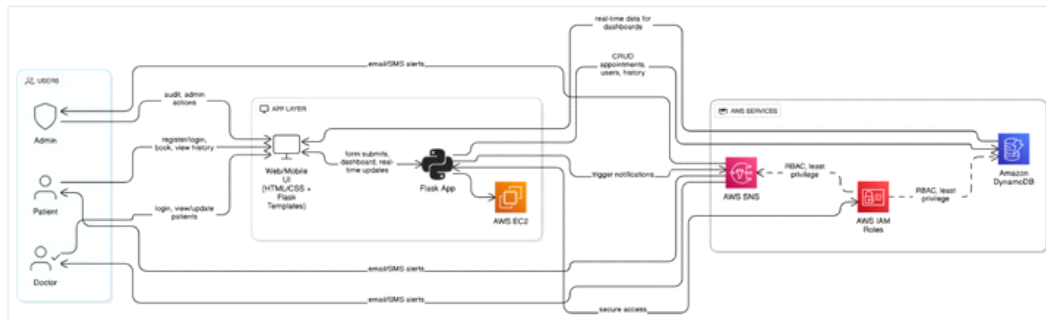
Scenario 2: Secure User Management with IAM

MedTrack utilizes AWS IAM to manage user permissions and ensure secure access to the system. For instance, when a new patient registers, an IAM user is created with specific roles and permissions to access only the features relevant to them. Doctors have their own IAM configurations, allowing them access to patient records and appointment details while maintaining strict security protocols. This setup ensures that sensitive data is accessible only to authorized users.

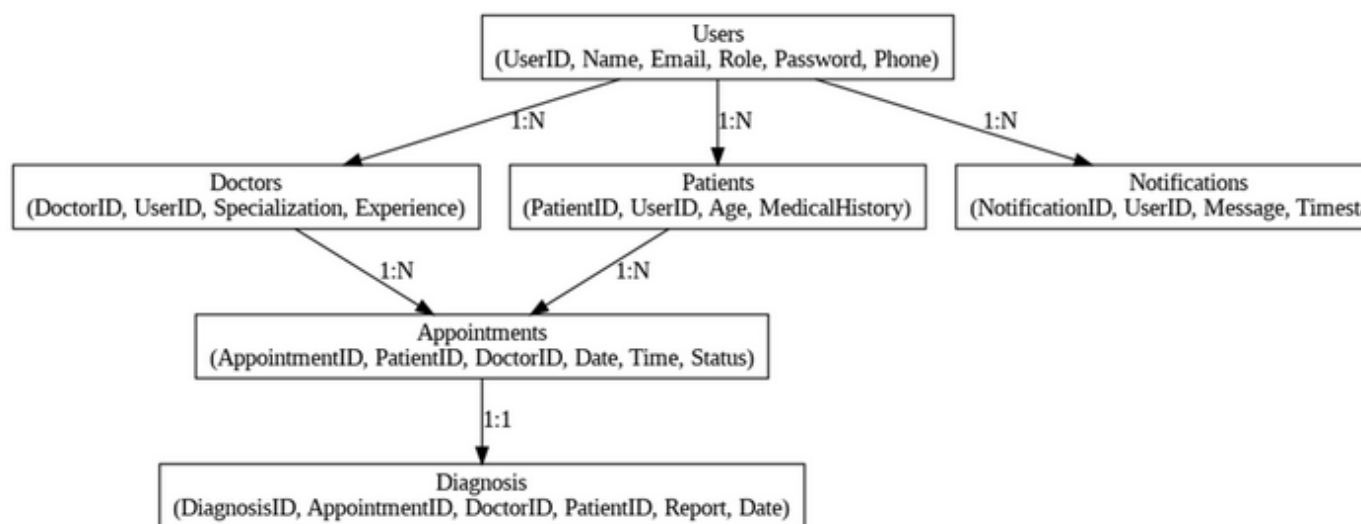
Scenario 3: Easy Access to Medical History and Resources

The MedTrack system provides doctors and patients with easy access to medical histories and relevant resources. For example, a doctor logs in to view a patient's medical history and upcoming appointments. They can quickly access, and update records as needed. Flask manages real-time data fetching from DynamoDB, while EC2 hosting ensures the platform performs seamlessly even when multiple users access it simultaneously, offering a smooth and uninterrupted user experience.

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Pre-requisites:

- AWS Account Setup:
<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
- AWS IAM (Identity and Access Management):
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
- AWS EC2 (Elastic Compute Cloud):
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
- AWS DynamoDB:
<https://docs.aws.amazon.com/amazondynamodb/Introduction.html>
- Amazon SNS:
<https://docs.aws.amazon.com/sns/latest/dg/welcome.htm> HYPERLINK
"https://docs.aws.amazon.com/sns/latest/dg/welcome.html" HYPERLINK

["https://docs.aws.amazon.com/sns/latest/dg/welcome.html"](https://docs.aws.amazon.com/sns/latest/dg/welcome.html) HYPERLINK

["https://docs.aws.amazon.com/sns/latest/dg/welcome.html"](https://docs.aws.amazon.com/sns/latest/dg/welcome.html)

- Git Documentation:
<https://git-scm.com/doc>
- VS Code Installation: (download the VS Code using the below link or you can get that in Microsoft store)
<https://code.visualstudio.com/download>

Project WorkFlow:

Milestone 1. Web Application Development and Setup

Activity 1.1: Develop the Backend Using Flask.

Activity 1.2: Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login

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

Activity 2.2: Login to AWS Management Console.

Milestone 3. DynamoDB Database Creation and Setup

Activity 3.1: Create a DynamoDB Table.

Activity 3.2: Configure Attributes for User Data and Book Requests.

Milestone 4. SNS Notification Setup

Activity 4.1: Create SNS topics for book request notifications.

Activity 4.2: Subscribe users and library staff to SNS email notifications.

Milestone 5. IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

Milestone 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.

Milestone 7. Deployment on EC2

Activity 7.1: Upload Flask Files

Activity 7.2: Run the Flask App

Milestone 8. Testing and Deployment

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

Milestone 1: Web Application Development and Setup

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

Please refer to this sample as a guide for local deployment :

<https://docs.google.com/document/d/1sFF7-tj6lgWtRbawWoA4W3PkkxEFrSJZhKzULgLsjxo/edit?usp=sharing>

Important Instructions:

- Start by creating the necessary HTML pages and Flask routes (app.py) to build the core functionality of your application.
- During the initial development phase, store and retrieve data using Python dictionaries or lists locally. This will allow you to design, test, and validate your application logic without external database dependencies.
- Ensure your app runs smoothly with local data structures before integrating any cloud services.

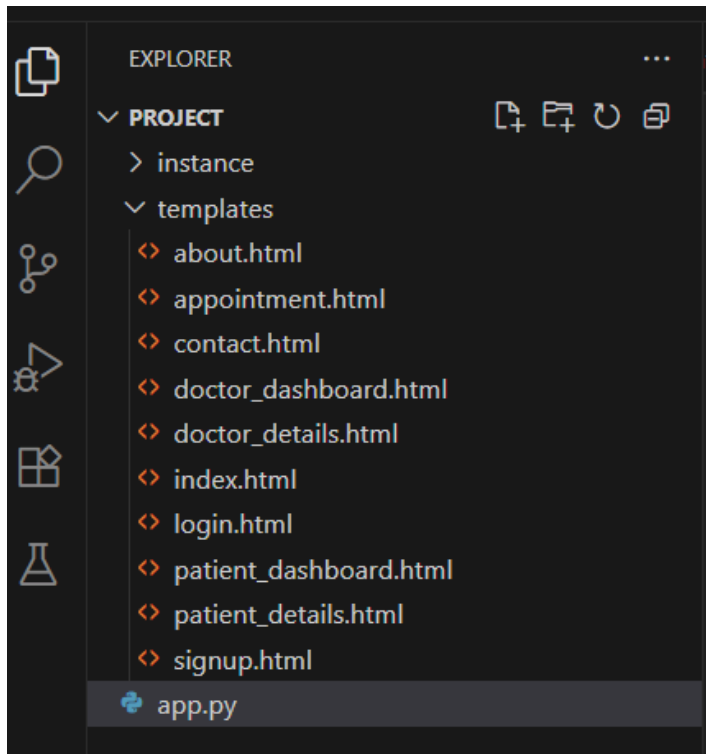
Post Troven Access Activation:

- Once Troven Labs access is provided (valid for 3 hours), you must immediately proceed with Milestone 1 of your Guided Project instructions.
- At this point, modify your app.py and replace local dictionary/list operations with AWS services (such as DynamoDB, RDS, or others as per project requirements).
- Using the temporary credentials provided by Troven Labs, securely connect your application to AWS resources.

- Since the AWS configuration is lightweight and already instructed in the milestones, you should be able to complete the cloud integration efficiently within the allotted time.

FLASK DEPLOYMENT

- File Explorer Structure



Description of the code :

Flask App Initialization:

In the MedTrack project, the Flask app is initialized to establish the backend infrastructure, enabling it to handle multiple user interactions such as patient registration, appointment

booking, and submission of medical reports. The Flask framework processes incoming requests, communicates with the DynamoDB database for storing user data, and integrates seamlessly with AWS services. Additionally, the routes and APIs are defined to manage different functionalities like secure

login, appointment scheduling, and medical history retrieval. This initialization sets up the foundation for smooth, real-time communication between patients and doctors while ensuring the app is scalable and secure.

```
tml appointment.html patient_details.html patient_dashboard.html about.html
app.py > ...
1  from flask import Flask, render_template, request, redirect, url_for, session, flash
2  import boto3
3  import uuid
4  from botocore.exceptions import ClientError, NoCredentialsError
5
6  app = Flask(__name__)
7  app.secret_key = 'medtrack_secret'
8
9  # AWS Setup
10 try:
11     dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')
12     sns_client = boto3.client('sns', region_name='ap-south-1')
13     users_table = dynamodb.Table('Users')
14     appointments_table = dynamodb.Table('Appointments')
15     AWS_READY = True
16 except Exception as e:
17     print("AWS Setup Error:", str(e))
18     AWS_READY = False
19
20 TOPIC_ARN = 'arn:aws:sns:ap-south-1:123456789012:YourTopicName' # Replace later
21
22 def send_sns_notification(message, subject='MedTrack Alert'):
23     if not AWS_READY:
```

```
app = Flask(__name__)
app.secret_key = 'your_secret_key'
```

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

SNS and Dynamodb initialization:

- In the MedTrack project, AWS SNS sends real-time notifications to patients and doctors about appointments and updates. DynamoDB stores user data, medical records, and appointments securely, offering fast, scalable access. Both services

are integrated with Flask to ensure smooth communication and efficient data management.

```
app.py > signup
10  try:
11      dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
12      sns_client = boto3.client('sns', region_name='us-east-1')
13      users_table = dynamodb.Table('Users')
14      appointments_table = dynamodb.Table('Appointments')
15      AWS_READY = True
16  except Exception as e:
17      print("AWS Setup Error:", str(e))
18      AWS_READY = False
19
20  TOPIC_ARN = 'arn:aws:sns:ap-south-1:123456789012:YourTopicName' # Replace later
21
22  def send_sns_notification(message, subject='MedTrack Alert'):
23      if not AWS_READY:
24          print("Skipping SNS (No AWS credentials).")
25          return
26      try:
27          sns_client.publish(
28              TopicArn=TOPIC_ARN,
29              Message=message,
30              Subject=subject
31          )
32      except ClientError as e:
```

- **SNS Connection**

Configure SNS to send notifications when a book request is submitted. Paste your stored ARN link in the `sns_topic_arn` space, along with the `region_name` where the SNS topic is created. Also, specify the chosen email service in `SMTP_SERVER` (e.g., Gmail, Yahoo, etc.) and enter the subscribed email in the `SENDER_EMAIL` section. Create an 'App password' for the email ID and store it in the `SENDER_PASSWORD` section.

```
app.py > _signup
21
22 def send_sns_notification(message, subject='MedTrack Alert'):
23     if not AWS_READY:
24         print("Skipping SNS (No AWS credentials).")
25         return
26     try:
27         sns_client.publish(
28             TopicArn=TOPIC_ARN,
29             Message=message,
30             Subject=subject
31         )
32     except ClientError as e:
33         print("SNS Error:", e.response['Error']['Message'])
34
35
36 @app.route('/')
37 def index():
38     return render_template('index.html')
39
40
41 @app.route('/about')
42 def about():
43     return render_template('about.html')
```

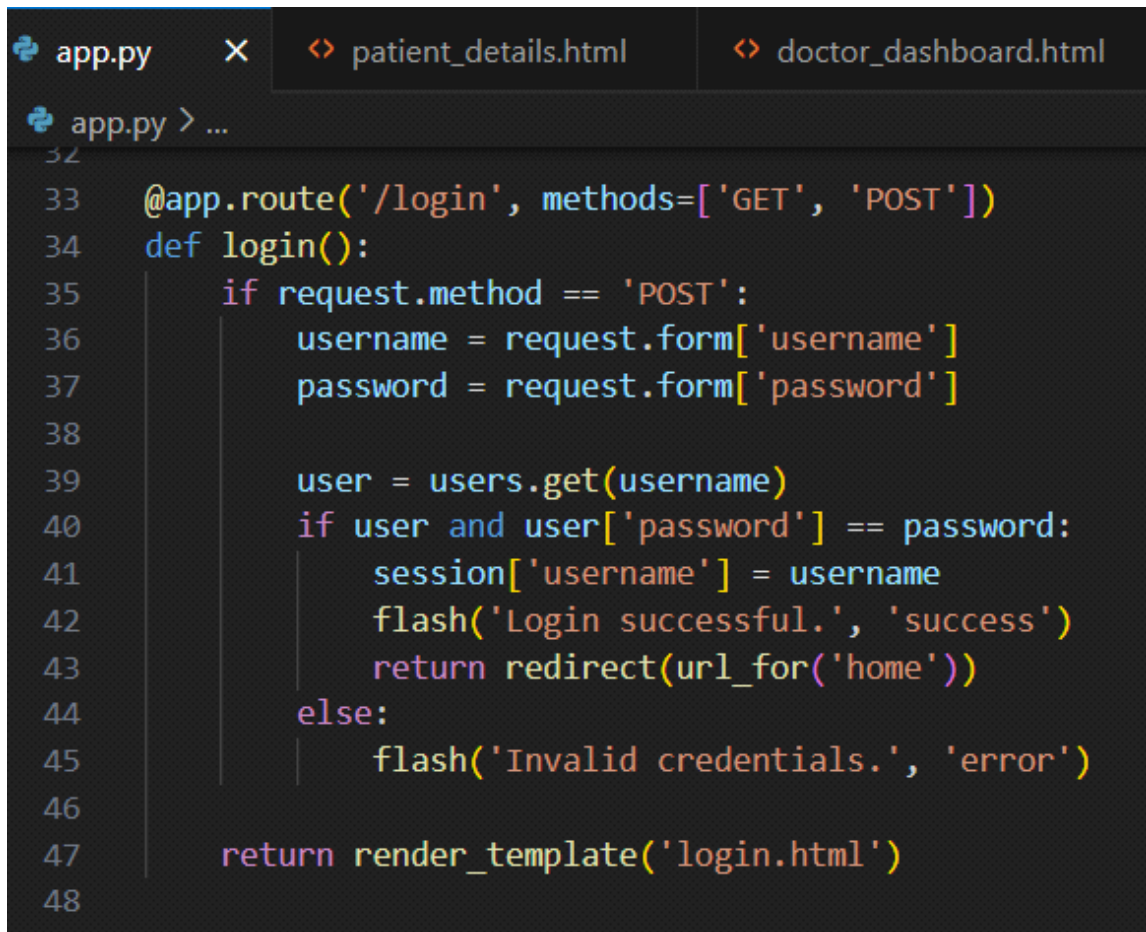
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- **Routes for Web Pages:**
Register Page


```
app.py  X  <> patient_details.html  <> doctor_dashboard.html  <> patient_dashboard.html
app.py > ...
10  @app.route('/')
11  def index():
12      return render_template('index.html')
13
14  @app.route('/signup', methods=['GET', 'POST'])
15  def signup():
16      if request.method == 'POST':
17          username = request.form['username']
18          email = request.form['email']
19          password = request.form['password']
20          confirm_password = request.form['confirm_password']
21
22          if username in users:
23              flash('Username already exists.', 'error')
24          elif password != confirm_password:
25              flash('Passwords do not match.', 'error')
26          else:
27              users[username] = {'email': email, 'password': password}
28              flash('Signup successful! Please log in.', 'success')
29              return redirect(url_for('login'))
30
31      return render_template('signup.html')
32
33  @app.route('/login', methods=['GET', 'POST'])
34  def login():
35      if request.method == 'POST':
36          username = request.form['username']
37          password = request.form['password']
38
39          user = users.get(username)
40          if user and user['password'] == password:
41              session['username'] = username
42              flash('Login successful.', 'success')
43              return redirect(url_for('home'))
44          else:
45              flash('Invalid credentials.', 'error')
46
```

- The **login route** handles user authentication by verifying credentials stored in **DynamoDB**. Upon successful login, it increments the **login**

count and redirects the user to their dashboard. This ensures secure access to the platform while maintaining user activity logs.

A screenshot of a code editor with a dark theme. At the top, there are three tabs: 'app.py' (active), 'patient_details.html', and 'doctor_dashboard.html'. Below the tabs, the code for the login route is visible. The code is as follows:

```
32
33 @app.route('/login', methods=['GET', 'POST'])
34 def login():
35     if request.method == 'POST':
36         username = request.form['username']
37         password = request.form['password']
38
39         user = users.get(username)
40         if user and user['password'] == password:
41             session['username'] = username
42             flash('Login successful.', 'success')
43             return redirect(url_for('home'))
44         else:
45             flash('Invalid credentials.', 'error')
46
47     return render_template('login.html')
48
```

Logout Route:

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

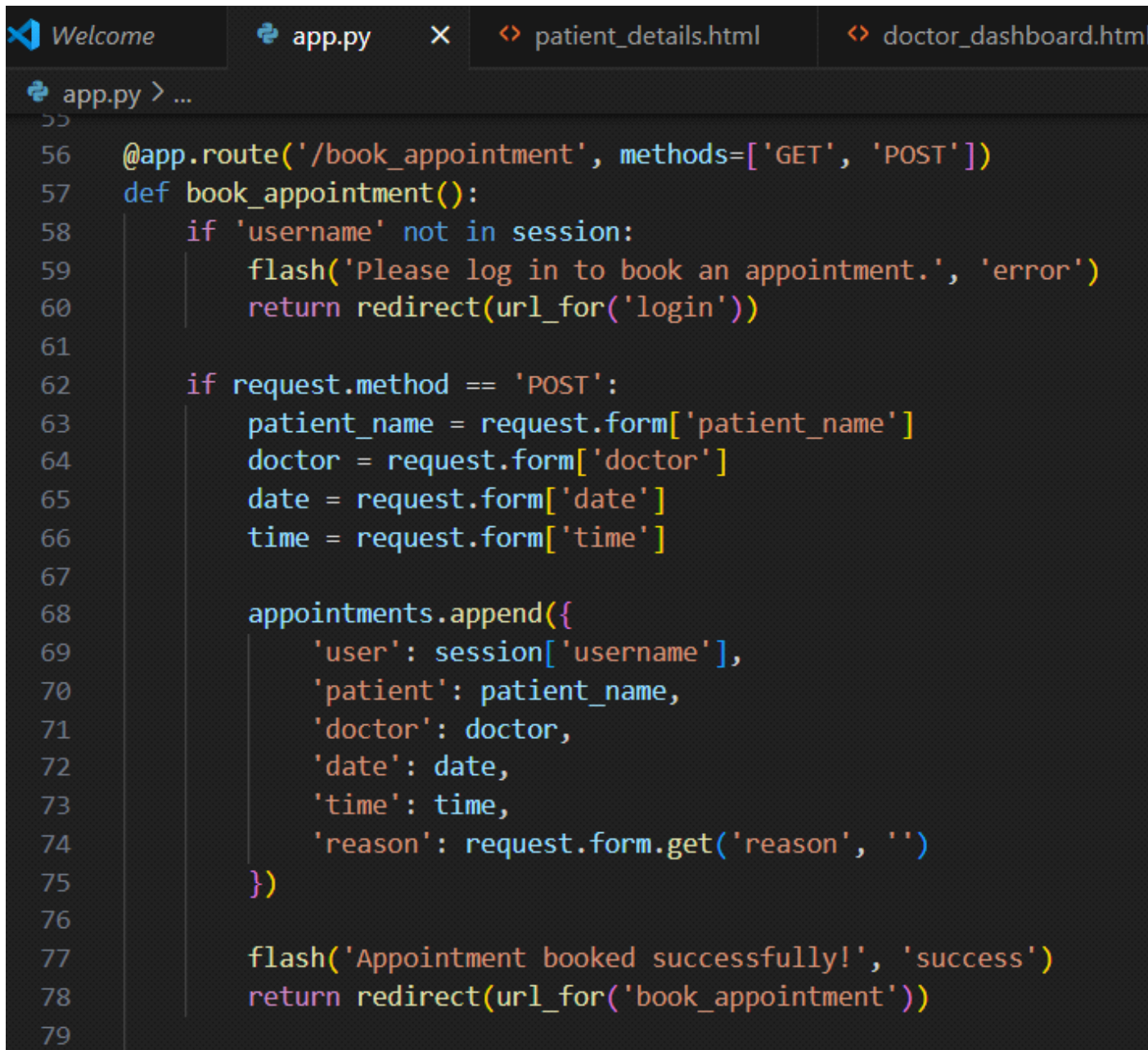
```

welcome  app.py  patient_details.html  doctor_dashboard.html
app.py > ...
53 def patient_details():
54     if not user:
55         flash('User not found.', 'error')
56         return redirect(url_for('login'))
57
58     return render_template('patient_details.html', username=username, email=email)
59
60 @app.route('/logout')
61 def logout():
62     session.pop('username', None)
63     flash('Logged out successfully.', 'info')
64     return redirect(url_for('login'))
65
66

```

Book Appointment Route:

The book appointment route allows users to select a date, time, and doctor for their appointment. Upon submission, the system stores the appointment details in DynamoDB and sends a confirmation notification via SNS. This ensures smooth scheduling and timely updates for both patients and doctors.



```
55
56 @app.route('/book_appointment', methods=['GET', 'POST'])
57 def book_appointment():
58     if 'username' not in session:
59         flash('Please log in to book an appointment.', 'error')
60         return redirect(url_for('login'))
61
62     if request.method == 'POST':
63         patient_name = request.form['patient_name']
64         doctor = request.form['doctor']
65         date = request.form['date']
66         time = request.form['time']
67
68         appointments.append({
69             'user': session['username'],
70             'patient': patient_name,
71             'doctor': doctor,
72             'date': date,
73             'time': time,
74             'reason': request.form.get('reason', '')
75         })
76
77         flash('Appointment booked successfully!', 'success')
78         return redirect(url_for('book_appointment'))
79
```

Deployment Code:

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



Welcome

app.py



patient_details.html

doctor_dashboard.html



app.py > ...

```
144 def patient_appointments():
145     flash('Please log in.', 'error')
146     return redirect(url_for('login'))
147
148
149     user_appts = [a for a in appointments if a['user'] == session['username']]
150     return render_template('patient_appointments.html', appointments=user_appts)
151
152 @app.route('/patient_details')
153 def patient_details():
154     if 'username' not in session:
155         flash('Please log in to view your details.', 'error')
156         return redirect(url_for('login'))
157
158     username = session['username']
159     user = users.get(username)
160
161     if not user:
162         flash('User not found.', 'error')
163         return redirect(url_for('login'))
164
165     return render_template('patient_details.html', username=username, email=user.email)
166
167
168
169 @app.route('/logout')
170 def logout():
171     session.pop('username', None)
172     flash('Logged out successfully.', 'info')
173     return redirect(url_for('login'))
174
175 if __name__ == '__main__':
176     app.run(debug=True)
177
```



AWS - Local Deployment Sample Code
Google Docs..

No description..

<https://docs.google.com/document/d/1sFF7-tJ6lWoA4W3PkxEFrSJZhKzULgLsxo/edit?usp=sharing>

Milestone 2: AWS Account Setup

Important Notice: Use Troven Labs for AWS Access

Students are strictly advised not to create their own AWS accounts, as doing so may incur charges. Instead, we have set up a dedicated section called “Labs” on the Troven platform, which provides temporary and cost-free access to AWS services.

Once your website is locally deployed and fully functional, you must proceed with integrating AWS services only through the Troven Labs environment. This ensures secure, controlled access to AWS resources without any risk of personal billing.

All steps involving AWS (such as deploying to EC2, connecting to DynamoDB, or using SNS) must be carried out within the Troven Labs platform, as we've configured temporary credentials for each student.

Reminder: You must complete the Web Development task before gaining access to Troven. Once accessed, the AWS Console via Troven is available for only 3 hours—please plan your work accordingly.

Please follow the provided guidelines and access AWS exclusively through Troven to avoid unnecessary issues.

Please refer the below link -

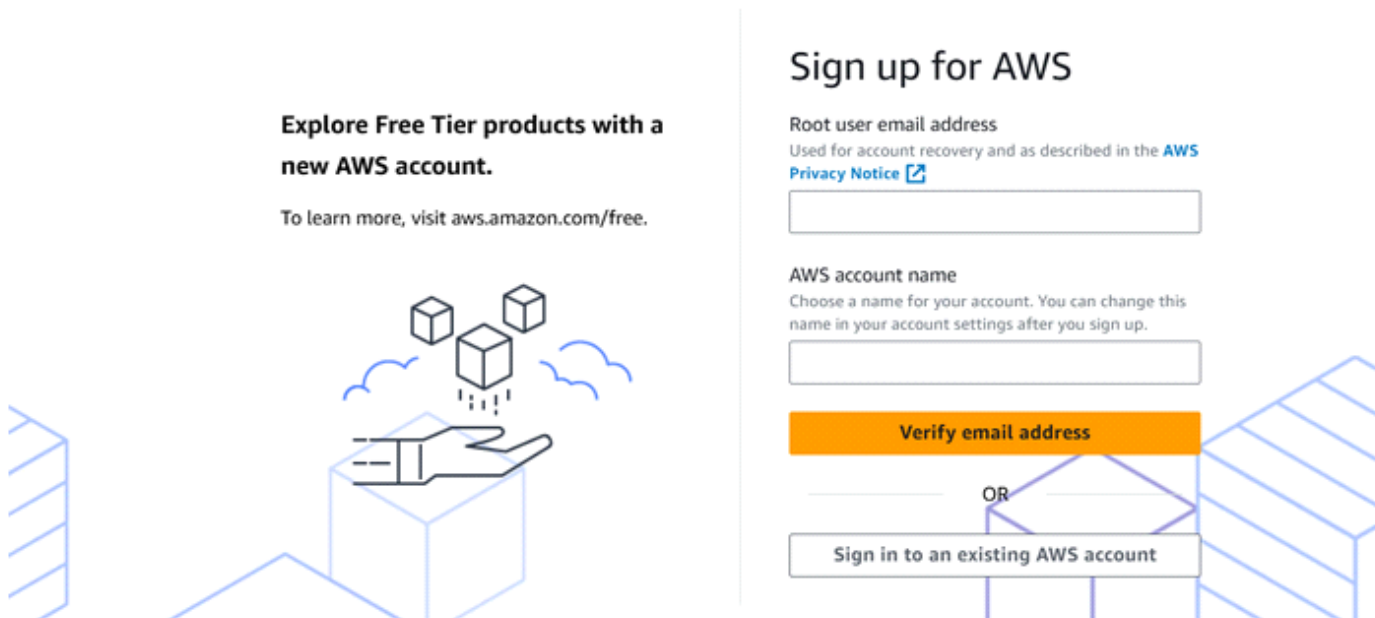
<https://drive.google.com/file/d/1HzWc7AMJ2BrxhV-uaw5s0vWtcd-28qgl/view?usp=sharing>

AWS Account Setup and Login

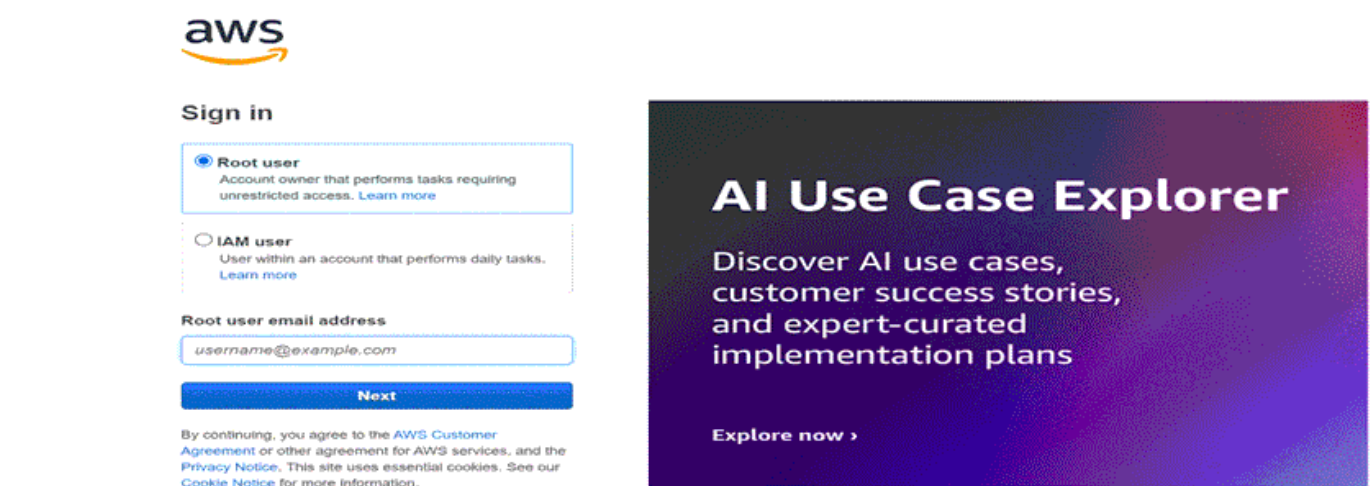
This is for your understanding only, please refrain from creating an AWS account. A temporary account will be provided via Troven.

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





- [illegible]



Milestone 3: DynamoDB Database Creation and Setup

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

Navigate to the DynamoDB

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



Services

Q dynamoDB



Services

Features

Resources **New**

Documentation

Knowledge articles

Marketplace

Blog posts

Events

Tutorials

Search results for 'dyn'

Services



DynamoDB ☆

Managed NoSQL Database

Top features

[Tables](#) [Imports from S3](#) [Explore Items](#) [Clusters](#) [Reserved Capacity](#)



Amazon DocumentDB ☆

Fully-managed MongoDB-compatible database service



CloudFront ☆

Global Content Delivery Network



Athena ☆

Serverless interactive analytics service

Features

Settings



DynamoDB feature

Clusters



DynamoDB feature

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

DynamoDB > Dashboard

Dashboard

Alarms (0) Info

Manage in CloudWatch

Find alarms

Alarm name

Status

No custom alarms

DAX clusters (0) Info

View details

Find clusters

Cluster name

Status

No clusters

No clusters to display

Create cluster

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 Amazon DynamoDB. [Learn more](#)

Create DAX cluster

What's new

SEP 19 AWS Cost Management now provides recommendations for Amazon DynamoDB. [Learn more](#)

DynamoDB

Dashboard

Tables

Explore items

PartiQL editor

Backups

Exports to S3

Imports from S3

Integrations New

DynamoDB > Tables

Tables (0) Info

Find tables

Any tag key

Any tag value

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.

Create table

Create a DynamoDB table for storing data

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

Create table

Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

UsersTable

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.

email

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.

Enter the sort key name

String

1 to 255 characters and case sensitive.

Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags

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

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel

Create

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

aws

Search

[Alt+S]

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.

AppointmentsTable

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.

appointment_id

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.

Enter the sort key name

String

1 to 255 characters and case sensitive.

Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags

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

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel

Create

Tables (2/10) Info

↺

Actions

Delete

Create

Q Status

✕

Any tag key

▼

Any tag value

▼

<

1

[-]

Name

▲

Status

▼

Partition key

▼

Sort key

▼

Indexes

▼

Replication Regions

▼

Deletion protection

▼

✓

AppointmentsTable

✓

Active

appointment_id (S)

-

0

0

⊖

Off

☐

NextGenHospital_Appointments

✓

Active

appointment_id (S)

-

0

0

⊖

Off

☐

NextGenHospital_ContactMessages

✓

Active

message_id (S)

-

0

0

⊖

Off

☐

NextGenHospital_Doctors

✓

Active

doctor_id (S)

-

0

0

⊖

Off

☐

NextGenHospital_PatientRecords

✓

Active

patient_id (S)

-

0

0

⊖

Off

☐

NextGenHospital_Users

✓

Active

email (S)

-

0

0

⊖

Off

☐

SalonAppointments

✓

Active

appointment_id (S)

user_email (S)

0

0

⊖

Off

☐

SalonStylists

✓

Active

stylist_id (S)

-

0

0

⊖

Off

☐

SalonUsers

✓

Active

email (S)

-

0

0

⊖

Off

✓

UsersTable

✓

Active

email (S)

-

0

0

⊖

Off

Milestone 4 : SNS Notification Setup

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

SNS topics for email notifications

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

Q sns

X

Search results for 'sns'

Services

Features

Resources **New**

Documentation

Knowledge articles


Marketplace


Blog posts


Events


Tutorials

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

Amazon SNS

DashboardTopicsSubscriptions▼ MobilePush notificationsText messaging (SMS)

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

Application Integration

Amazon Simple Notification Service

Pub/sub messaging for microservices and serverless applications.

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

Create topic

Topic name
A topic is a message channel. When a message is published to a topic, it fans out to the subscribed endpoints.

Next step

Start with an overview

Pricing

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

Amazon SNS

DashboardTopicsSubscriptions▼ MobilePush notificationsText messaging (SMS)

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

Amazon SNS > Topics

Topics (0)

Name	Type	ARN
No topics To get started, create a topic. <div>Create topic</div>		

EditDeletePublish messageCreate

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

New Feature

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

Create topic

Details

Type | [Info](#)

Topic type cannot be modified after topic is created

☐ FIFO (first-in, first-out)

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

☒ Standard

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

Name

Medtrack

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

Display name - optional | [Info](#)

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

My Topic

Maximum 100 characters.



► **Access policy - optional** [Info](#)

This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

► **Data protection policy - optional** [Info](#)

This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

► **Delivery policy (HTTP/S) - optional** [Info](#)

The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

► **Delivery status logging - optional** [Info](#)

These settings configure the logging of message delivery status to CloudWatch Logs.

► **Tags - optional**

A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track 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

Cr

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

Amazon SNS > Topics > Medtrack

Amazon SNS

- Dashboard
- Topics
- Subscriptions
- ▼ Mobile
 - Push notifications
 - Text messaging (SMS)

Medtrack Edit Delete Publish

Details

Name Medtrack	Display name -
ARN arn:aws:sns:ap-south-1:940482422578:Medtrack	Topic owner 940482422578
Type Standard	

Subscriptions (1) Edit Delete Request confirmation Confirm subscription Create subscription

Search

Subscribe users and Admin

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

Amazon SNS > Subscriptions > Create subscription

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

Protocol
The type of endpoint to subscribe
Email

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

After your subscription is created, you must confirm it. Info

► **Subscription filter policy - optional** Info
This policy filters the messages that a subscriber receives.

► **Redrive policy (dead-letter queue) - optional** Info
Send undeliverable messages to a dead-letter queue.

Cancel Create

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

- After subscription request for the mail confirmation

<

Subscriptions

Access policy

Data protection policy

Delivery policy (HTTP/S)

Delivery status logging

Encryption

Subscriptions (1)

EditDeleteRequest confirmationConfirm subscriptionCreate subscri

Q Search

< 1

ID	Endpoint	Status	Protocol
<div><div></div><div>2c78944f-bb5d-4fb1-9982-74334...</div></div>	sairaviteja478@gmail.com	<div><div>✔ Confirmed</div></div>	EMAIL

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

AWS Notification - Subscription Confirmation Inbox x

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

to me ▼

You have chosen to subscribe to the topic:

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

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

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to

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

to me ▼

...

You have chosen to subscribe to the topic:

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

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

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to



Simple Notification Ser

Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

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

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

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

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

Medtrack

EditDeletePublish

Details

Name

Medtrack

ARN

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

Type

Standard

Display name

-

Topic owner

940482422578

<Subscriptions

Access policy

Data protection policy

Delivery policy (HTTP/S)

Delivery status logging

Encryption

Subscriptions (1)

EditDeleteRequest confirmationConfirm subscriptionCreate subscri

Q Search

< 1

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

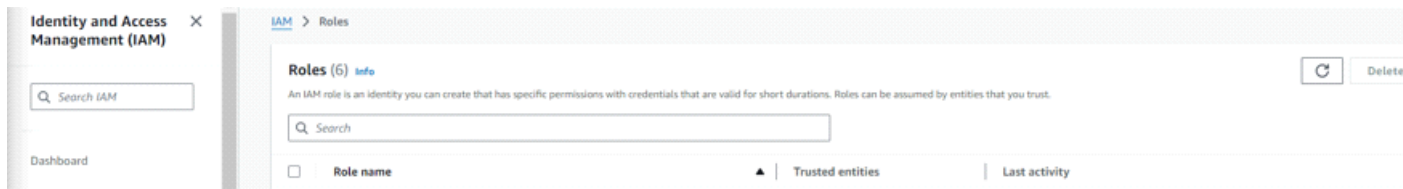
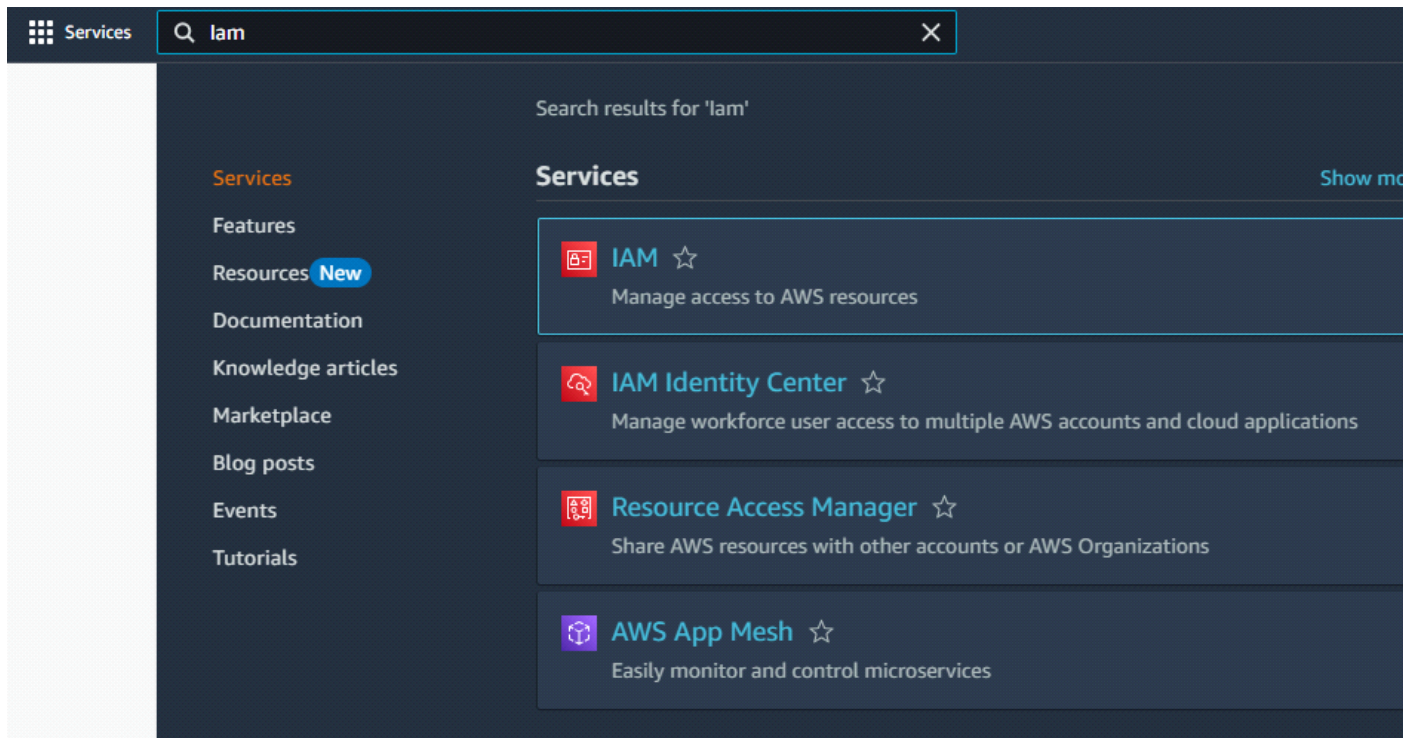
© 2025, Amazon Web Services, Inc. or its affiliates. PrivacyTermsCoc

Milestone 5: IAM Role Setup

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

Create IAM Role.

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



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

The screenshot shows the 'Select trusted entity' step in the AWS IAM console. The left sidebar indicates the current step is 'Step 1: Select trusted entity'. The main content area is titled 'Select trusted entity' and contains two sections: 'Trusted entity type' and 'Use case'.

Trusted entity type

- ☒ **AWS service**
Allow AWS service like EC2, Lambda, or others to perform actions in this account.
- ☐ **AWS account**
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- ☐ **Web identity**
Allow users federated by the specified external web identity provider to assume this role to perform actions in this account.
- ☐ **SAML 2.0 Federation**
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- ☐ **Custom trust policy**
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case:

Choose a use case for the specified service.

Use case:

- ☒ **EC2**
Allow EC2 instances to act AWS on your behalf.
- ☐ **EC2 Role for AWS Systems Manager**
Allow EC2 instances to use AWS Systems Manager CloudWatch and Systems Manager on your behalf.
- ☐ **EC2 Spot Fleet Rule**
Allow EC2 Spot Fleet to request and terminate Spot instances on your behalf.
- ☐ **EC2 - Spot Fleet Auto Scaling**
Allow Auto Scaling to create and update EC2 spot fleets on your behalf.
- ☐ **EC2 - Spot Fleet Tagging**
Allow EC2 to search spot instances and attach tags to the searched instances on your behalf.
- ☐ **EC2 - Spot Instances**
Allow EC2 Spot instances to search and manage spot instances on your behalf.
- ☐ **EC2 - Spot Fleet**
Allow EC2 Spot Fleet to search and manage spot fleet instances on your behalf.
- ☐ **EC2 - Scheduled instances**
Allow EC2 Scheduled instances to manage instances on your behalf.

The screenshot shows the 'Add permissions' step in the AWS IAM console. The left sidebar indicates the current step is 'Step 2: Add permissions'. The main content area is titled 'Add permissions' and contains a search bar, a filter dropdown, and a list of permissions policies.

Permissions policies (1/955)

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

Filter by Type: 2 matches

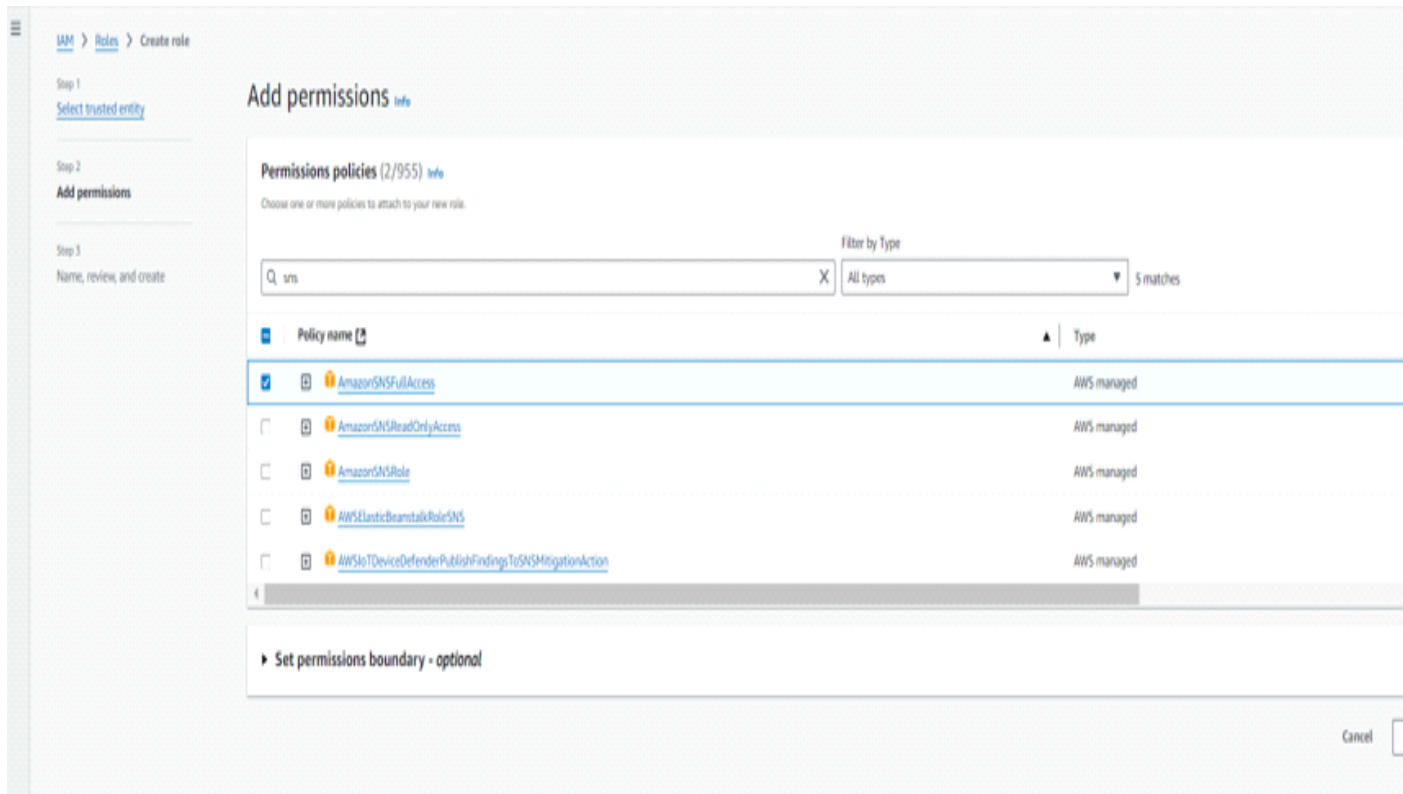
<input checked="" type="checkbox"/>	Policy name	Type
<input checked="" type="checkbox"/>	AmazonDynamoDBFullAccess	AWS managed
<input type="checkbox"/>	AmazonDynamoDBReadOnlyAccess	AWS managed

Set permissions boundary - optional

Attach Policies.

Attach the following policies to the role:

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



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

FlaskDynamoSNSRole [Info](#)

Allows EC2 instances to call AWS services on your behalf.

Summary


Creation date

April 16, 2025, 22:10 (UTC+05:30)


Last activity

5 hours ago

ARN

 arn:aws:iam::940482422578:role/FlaskDynamoSNSRole

Instance profile ARN

 arn:aws:iam::940482422578:instance-profile/FlaskDynamoSNSRole

Maximum session duration



1 hour

Permissions[Trust relationships](#)[Tags](#)[Last Accessed](#)[Revoke sessions](#)

Permissions policies (2) [Info](#)

You can attach up to 10 managed policies.

[Simulate](#)[Remove](#)[Add permission](#)

Search		Filter by Type		
<input type="text" value="Search"/>		All types		< 1
<input type="checkbox"/>	Policy name ↗	Type	Attached entities	
<input type="checkbox"/>	 AmazonDynamoDBFullAccess	AWS managed	3	
<input type="checkbox"/>	 AmazonSNSFullAccess	AWS managed	3	

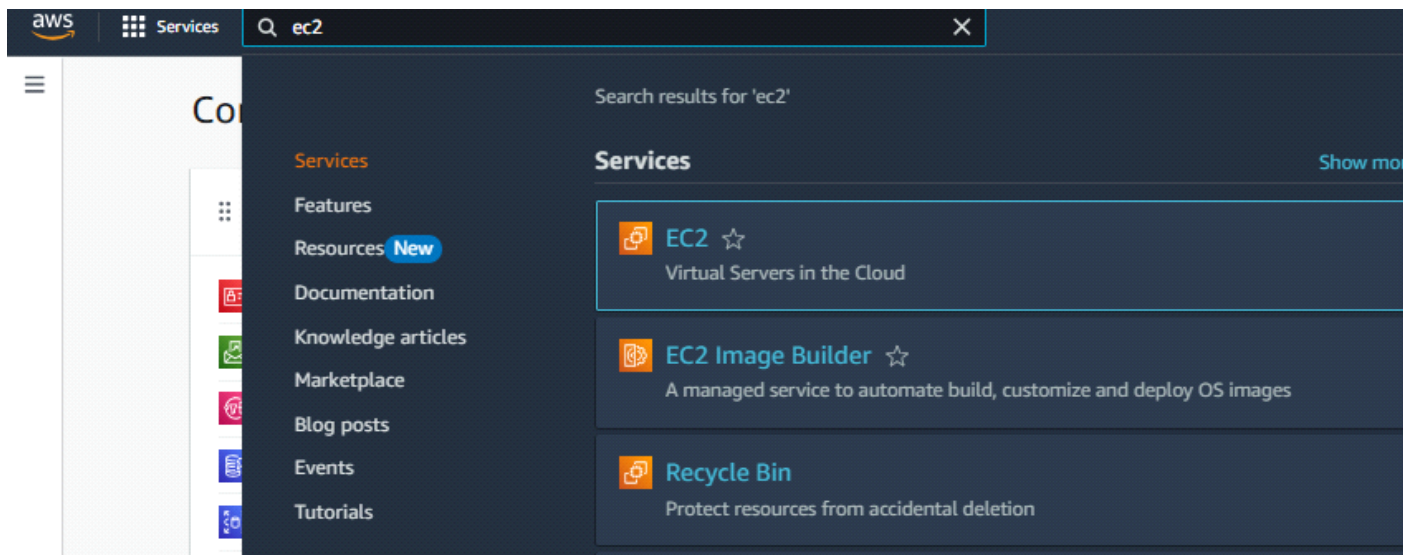
Milestone 6: EC2 Instance setup

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

Launch an EC2 instance to host the Flask application.

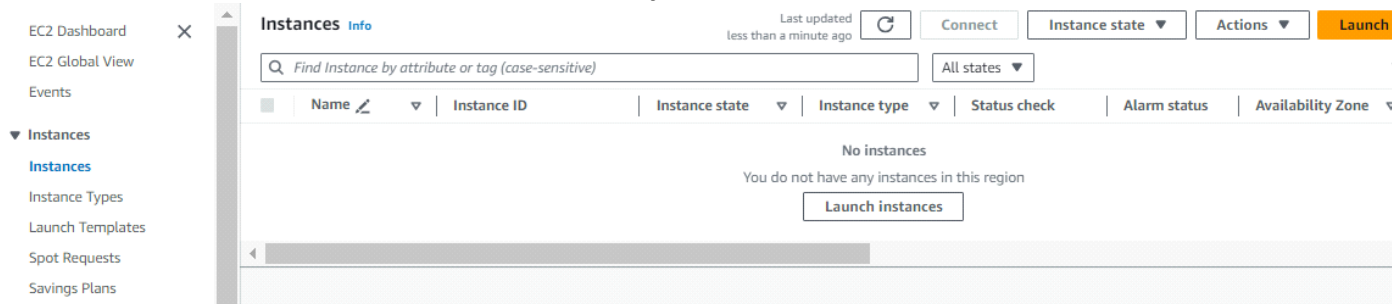
- **Launch EC2 Instance**

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



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

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



Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags [Info](#)

Name

[Add additional tags](#)

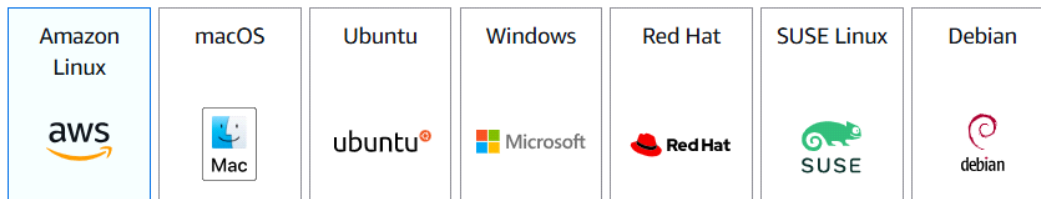
▼ Application and OS Images (Amazon Machine Image) [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch an instance. Search or Browse for AMIs if you don't see what you are looking for below.

🔍 Search our full catalog including 1000s of application and OS images

Recents

Quick Start



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

- In the **Launch Instance** wizard, choose **Amazon Linux 2** or **Ubuntu** from the available AMIs.
- Select **t2.micro** as the instance type, which is free-tier eligible, and continue with the configuration and launch steps.



Amazon Machine Image (AMI)

Amazon Linux 2023 AMI

Free tier eligible

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

Description

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

Architecture

64-bit (x86)

Boot mode

uefi-preferred

AMI ID

ami-02b49a24cfb95941c

Verified provider

To create and download the key pair for server access:

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

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

Instance type

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

Free tier eligible

▼

☒ All generations

[Compare instance types](#)

[Additional costs apply for AMIs with pre-installed software](#)

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

Select ▼

[↻ Create new key pair](#)

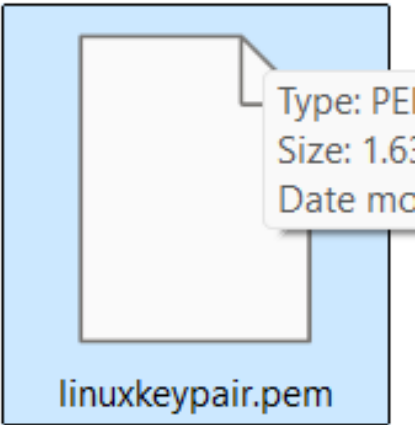
▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

linuxkeypair ▼

[↻ Create new key pair](#)



Configure security groups for HTTP, and SSH access.

For network settings during EC2 instance launch:

- In the **Network Settings** section, select the **VPC** and **Subnet** you wish to use (if unsure, the default VPC and subnet should work).
- Ensure **Auto-assign Public IP** is enabled so your instance can be accessed from the internet.
- In **Security Group**, either select an existing one or create a new one that allows SSH (port 22) access to your EC2 instance for remote login.

The screenshot shows the 'Network settings' section of the AWS Management Console. It includes fields for VPC (vpc-03cdc7b6f19dd7211), Subnet (No preference), and Auto-assign public IP (Enable). Below these, there is a section for 'Firewall (security groups)' with two radio buttons: 'Create security group' (selected) and 'Select existing security group'. The 'Create security group' option is active, showing a text input for the 'Security group name' (launch-wizard) and a text input for the 'Description' (launch-wizard created 2024-10-13T17:49:56.622Z). The 'Select existing security group' option is also visible but not selected.

▼ **Network settings** [Info](#)

VPC - *required* | [Info](#)

vpc-03cdc7b6f19dd7211 (default) ▼ ↻

Subnet | [Info](#)

No preference ▼ ↻ [Create new subnet](#)

Auto-assign public IP | [Info](#)

Enable ▼

[Additional charges apply](#) when outside of [free tier allowance](#)

Firewall (security groups) | [Info](#)

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

☒ Create security group ☐ Select existing security group

Security group name - *required*

launch-wizard

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

Description - *required* | [Info](#)

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

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0)

Type | Info

ssh

Protocol | Info

TCP

Port range | Info

22

Source type | Info

Anywhere

Source | Info

Q Add CIDR, prefix list or security

0.0.0.0/0 X

Description - optional | Info

e.g. SSH for admin desktop

Remove

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

Type | Info

HTTP

Protocol | Info

TCP

Port range | Info

80

Source type | Info

Custom

Source | Info

Q Add CIDR, prefix list or security

0.0.0.0/0 X

Description - optional | Info

e.g. SSH for admin desktop

Remove

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

Type | Info

Custom TCP

Protocol | Info

TCP

Port range | Info

5000

Source type | Info

Custom

Source | Info

Q Add CIDR, prefix list or security

0.0.0.0/0 X

Description - optional | Info

e.g. SSH for admin desktop

Remove

Add security group rule

EC2 > ... > Launch an instance

Success
Successfully initiated launch of instance (i-001861022fbca290)

► Launch log

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

Create billing and free tier usage alerts

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

Create billing alerts

Connect to your instance

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

Connect to instance

Learn more

Connect an RDS database

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

Connect an RDS database

Create a new RDS database

Learn more

Create EBS snapshot policy

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

Create EBS snapshot policy

Manage detailed monitoring

Enable or disable detailed monitoring for the instance. If you enable detailed monitoring, the Amazon EC2 console displays monitoring graphs with a 1-minute period.

Manage detailed monitoring

Create Load Balancing

Create an application, network, or classic Elastic Load Balancing

Create Load Balancing

Create AWS budget

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

Create AWS budget

Manage CloudWatch alarms

Create or update Amazon CloudWatch alarms for the instance.

Manage CloudWatch alarms

Disaster recovery for your instances

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

Disaster recovery for your instances

Monitor for suspicious runtime activities

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

Monitor for suspicious runtime activities

Get instance screenshot

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

Get instance screenshot

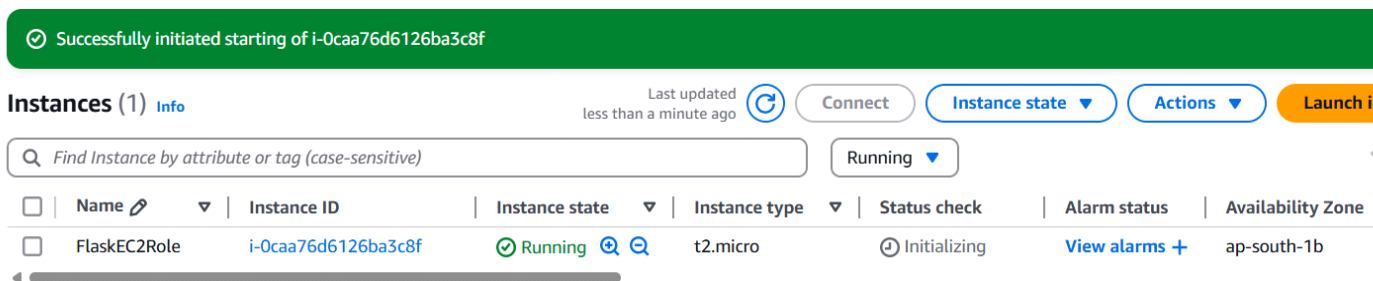
Get system log

View the instance's system log.

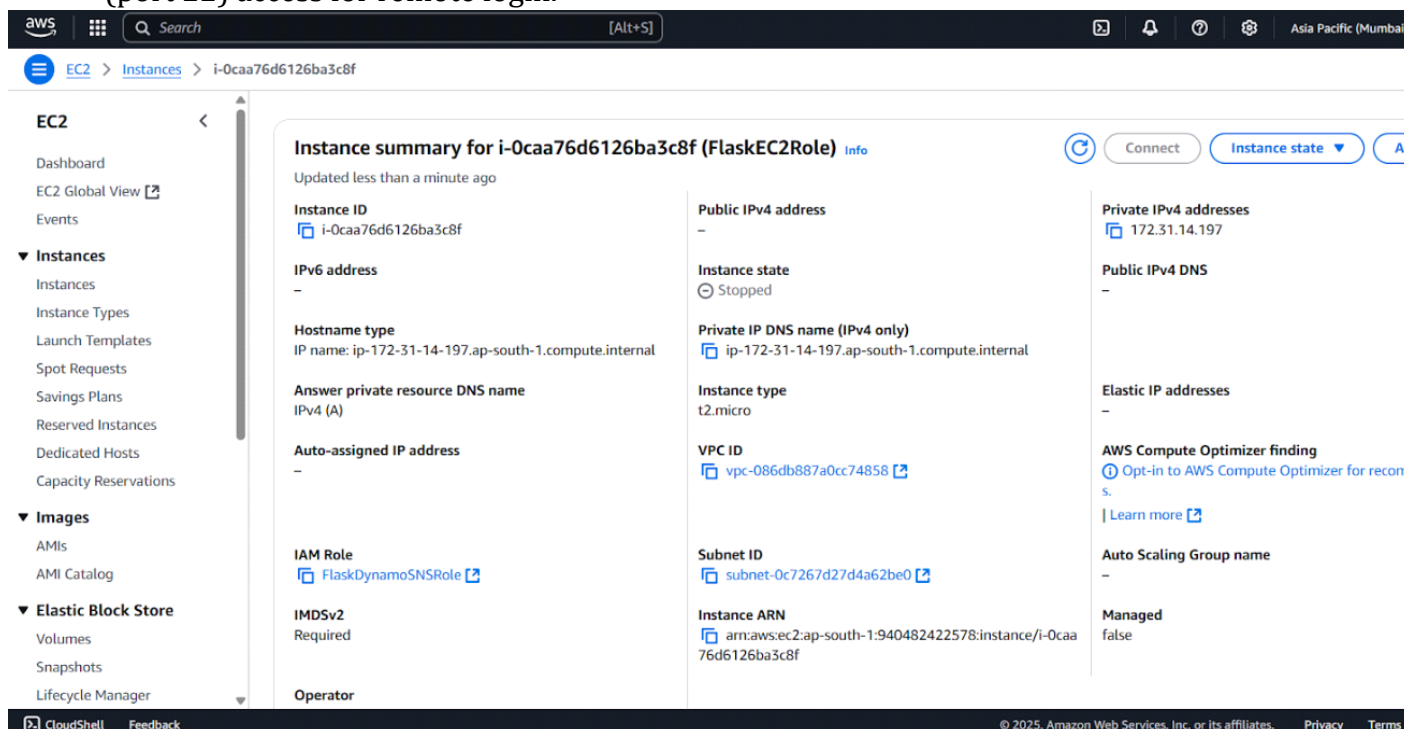
Get system log

- 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.



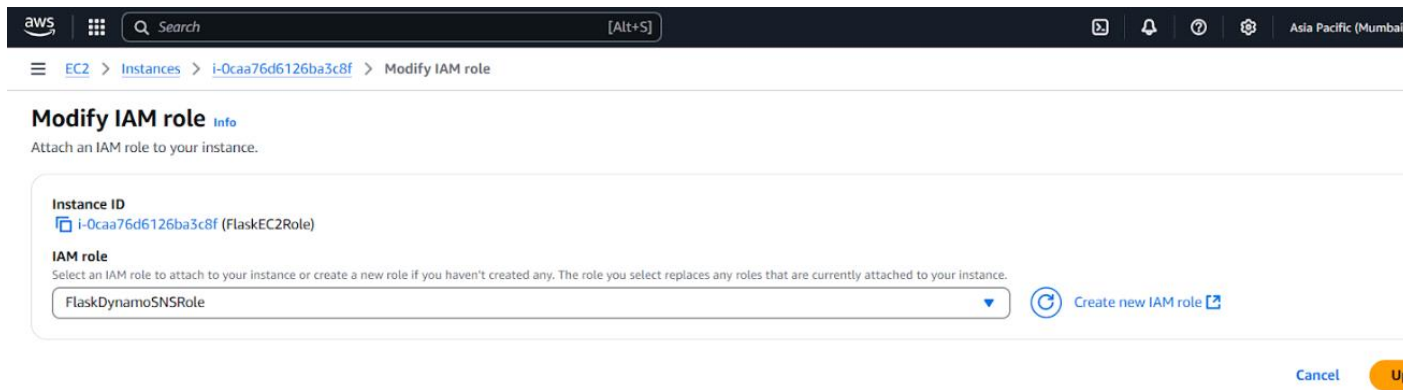
- The EC2 instance you are launching is configured with Amazon Linux 2 or Ubuntu as the AMI, t2.micro as the instance type (free-tier eligible), and flaskec2role IAM role for appropriate permissions. The flaskkeypair key pair is created for secure server access via SSH, and the instance is set to auto-assign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



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

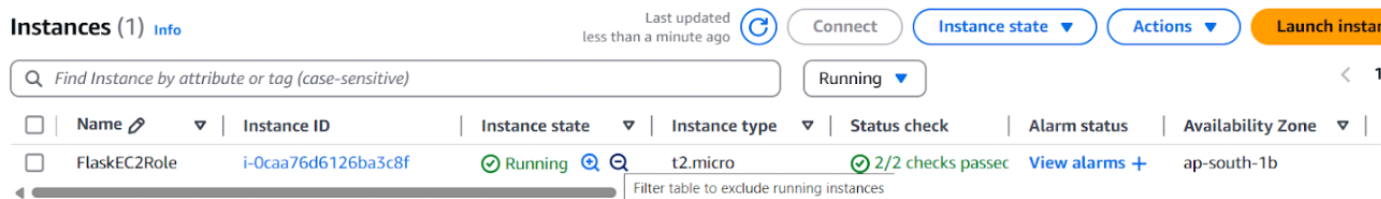
- Go to the **AWS IAM Console**, select **Roles**, and find the **flaskec2role**.

- Click **Attach Policies**, then choose the required policies (e.g., **DynamoDBFullAccess**, **SNSFullAccess**) and click **Attach Policy**.
- If needed, update the instance to use this modified role by selecting the EC2 instance, clicking **Actions**, then **Security**, and **Modify IAM role** to select the updated role.



To connect to your EC2 instance:

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



- Now connect the EC2 with the files

Milestone 7 : Deployment on EC2

Deployment on an EC2 instance involves launching a server, configuring security groups for public access, and uploading your application files. After setting up necessary dependencies and environment variables, start your application and ensure it's running on the correct port. Finally, bind your domain or use the public IP to make the application accessible online.

Install Software on the EC2 Instance

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

```
sudo yum update -y
sudo yum install python3 git
sudo pip3 install flask boto3
```

Verify Installations:

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

Clone Your Flask Project from GitHub

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

- Run: 'git clone <https://github.com/MekalaRaviKiran/Webproject.git>
- This will download your project to the EC2 instance.

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

```
cd Webproject
```

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

sudo flask run --host=0.0.0.0 --port=5000

```
aws [Search] [Alt+S] Asia Pacific (Mumbai)

2025-04-18 08:02:36,630 - __main__ - ERROR - Failed to fetch appointments: An error occurred (ValidationException) when calling the Query operation: The t
have the specified index: DoctorEmailIndex
2025-04-18 08:02:36,637 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:36] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:02:39,258 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:39] "GET /logout HTTP/1.1" 302 -
2025-04-18 08:02:39,289 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:39] "GET / HTTP/1.1" 200 -
2025-04-18 08:02:41,589 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:41] "GET /login HTTP/1.1" 200 -
2025-04-18 08:02:47,468 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:47] "POST /login HTTP/1.1" 302 -
2025-04-18 08:02:47,500 - __main__ - ERROR - Failed to query appointments: An error occurred (ValidationException) when calling the Query operation: The t
have the specified index: PatientEmailIndex
2025-04-18 08:02:47,517 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:47] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:02:49,506 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:49] "GET /book_appointment HTTP/1.1" 200 -
2025-04-18 08:03:25,936 - __main__ - INFO - Email sent to gtharunasri19@gmail.com
2025-04-18 08:03:28,795 - __main__ - INFO - Email sent to sairaviteja478@gmail.com
2025-04-18 08:03:28,796 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:03:28] "POST /book_appointment HTTP/1.1" 302 -
2025-04-18 08:03:28,852 - __main__ - ERROR - Failed to query appointments: An error occurred (ValidationException) when calling the Query operation: The t
have the specified index: PatientEmailIndex
2025-04-18 08:03:28,868 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:03:28] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:03:44,609 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:03:44] "GET /logout HTTP/1.1" 302 -
2025-04-18 08:03:44,654 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:03:44] "GET / HTTP/1.1" 200 -
^C(venv) [ec2-user@ip-172-31-14-197 medtrack_app]$ python app.py
* Serving Flask app 'app'
* Debug mode: off
2025-04-18 08:04:27,745 - werkzeug - INFO - WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.31.14.197:5000
2025-04-18 08:04:27,746 - werkzeug - INFO - Press CTRL+C to quit
```

i-0caa76d6126ba3c8f (FlaskEC2Role)

PublicIPs: 13.203.227.15 PrivateIPs: 172.31.14.197

Verify the Flask app is running:

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

- Run the Flask app on the EC2 instance

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

python app.py
* 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 http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 260-484-395
127.0.0.1 - - [07/Jul/2025 08:50:57] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [07/Jul/2025 08:50:57] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [07/Jul/2025 08:51:01] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [07/Jul/2025 08:51:05] "GET /signup HTTP/1.1" 200 -
127.0.0.1 - - [07/Jul/2025 08:51:20] "POST /signup HTTP/1.1" 302 -
127.0.0.1 - - [07/Jul/2025 08:51:20] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [07/Jul/2025 08:51:37] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [07/Jul/2025 08:51:37] "GET /patient/dashboard HTTP/1.1" 200 -
```

Access the website through:

PublicIPs: <http://54.87.47.9:5000>

Milestone 8: Testing and Deployment

Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment.

Functional testing to verify the project

Home Page:

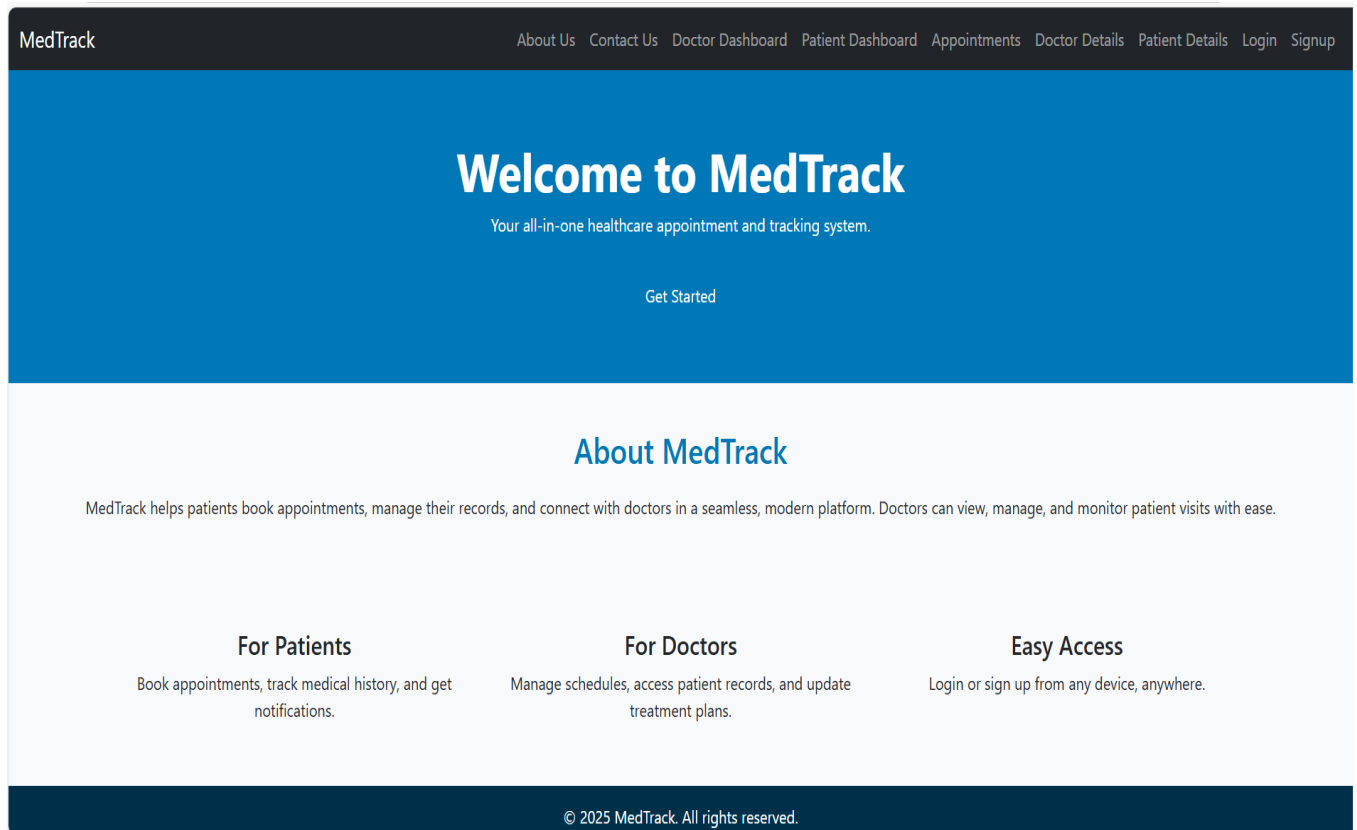
The Home Page of your project is the main entry point for users, where they can interact with the system. It typically includes:

- Input Fields: For users to enter basic information like appointment requests, diagnosis submissions, or service bookings.
- Navigation: Links to other sections such as the login page, dashboard, or service options.

Responsive Design: Ensures the page is accessible across devices with a clean, user-friendly interface.

-

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



LOGIN PAGE:

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

- Select role that leads to login as doctor or patient
- Username and Password Fields: Users enter their credentials (username and password) to authenticate their account.
- Login Button: A button to submit login details and validate user access.

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

Login to MedTrack

I am a:

-- Select Role --



Email

you@example.com

Password

Enter your password

Login

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

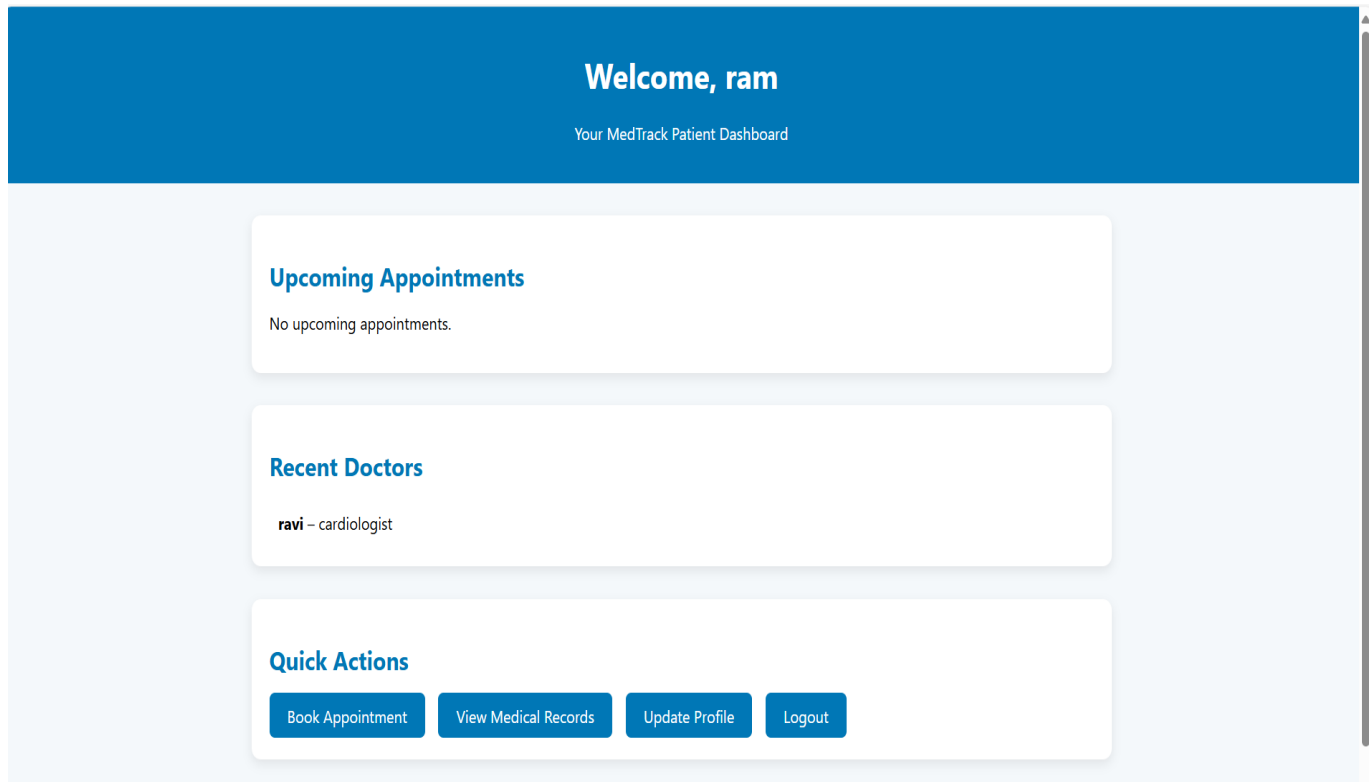
[← Back to Home](#)

User Dashboard:

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

- **Book Appointment Section:** A form for selecting a doctor, choosing an appointment time, and submitting the request.
- **Appointment Status:** A section showing the current status of appointments (e.g., confirmed, pending, or completed) with options to view details or cancel.

- Upcoming Appointments: A list of future appointments with relevant details such as doctor name, date, and time.
- This dashboard helps patients book new appointments and keep track of their healthcare schedules.



Book Your Appointment

Your Full Name

e.g. Riya Sharma

Select Specialist

-- Choose Specialization --

Preferred Date

dd-mm-yyyy

Preferred Time

--:--

Reason / Symptoms

Describe your symptoms...

Book Appointment

Doctor Dashboard:

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

- **Upcoming Appointments List:** A table or list showing patient names, appointment times, and appointment statuses (e.g., confirmed, pending).
- **Patient Details:** Quick access to each patient's medical history, contact information, and previous visit records.

- **Appointment Actions:** Options to view, confirm, reschedule, or cancel appointments, ensuring efficient management.

The dashboard serves as the main interface for doctors to manage their schedules, track patient interactions, and provide timely care.

Welcome, Dr. ravi

Specialist: cardiologist

Today's Appointments

Recent Patients

ravi – Last Visit:

ram – Last Visit:

Quick Actions

View All Appointments

Patient Records

Add Prescription

Logout

© 2025 MedTrack. All rights reserved.

Contact Us

Full Name

Email address

Subject

Message

Write your message here...

Send Message

About MedTrack

MedTrack is a modern and intuitive platform designed to streamline the healthcare appointment system for both patients and doctors.

Our Mission

To revolutionize the healthcare system by providing a seamless and efficient appointment booking experience, ensuring timely care and better patient-doctor interactions.

What We Offer

- Secure and fast appointment scheduling
- Doctor and patient dashboards
- Easy tracking of medical records and visit history
- User-friendly and responsive interface

Our Team

We are a passionate group of developers, designers, and healthcare professionals committed to improving digital healthcare accessibility and transparency for everyone.

DynamoDB Database updates :

- **Users table :**

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

- **Add New Users:** When a new patient or doctor registers, their details such as name, email, role (patient/doctor), contact info, and password hash are added to the table.

- **Update User Info:** If a user updates their profile (e.g., changing contact details), the corresponding record in the table is modified.
- **Status Tracking:** Track the status of user accounts (active, inactive) based on their activity or admin updates.

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

Table: UsersTable - Items returned (4)

Scan started on April 18, 2025, 19:27:34

Actions

Create

<input type="checkbox"/>	email (String)	age	created_at	gender	login_count
<input type="checkbox"/>	sunder@thesmartbrid...	25	2025-04-18...	male	2
<input type="checkbox"/>	gtharunasri19@gmail...	21	2025-04-18...	female	1
<input type="checkbox"/>	sairaviteja478@gmail...	21	2025-04-18...	male	2
<input type="checkbox"/>	siri@thesmartbridge.c...	24	2025-04-18...	female	1

1. Appointment table :

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

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

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

Table: AppointmentTable - Items returned (2)

Actions

Create

Scan started on April 18, 2025, 19:27:09

< 1 >

<input type="checkbox"/>	appointment_id (String)	appointment_date	created_at	diagnosis	doctor
<input type="checkbox"/>	86d52cfe-cce6-46b0-941d-1...	2025-04-30	2025-04-18...		gthar
<input type="checkbox"/>	42e13fef-fbe9-42d1-9905-f...	2025-04-21	2025-04-18...	make sure t...	siri@


Appointment confirmation:

Book Appointment


Appointment booked successfully!

Patient Name:

Select Doctor:

Date:

Time:

Book Appointment

[← Back to Home](#)

My Appointments

Patient	Doctor	Date	Time
rekha prathusha	Dr. Johnson	2025-07-04	15:48

[← Back to Home](#)

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.