

Title Page

Project Title: MedTrack – AWS Cloud Enabled Healthcare Management System

Category: AWS Cloud Foundation

Submitted By: Kamdula Veena Madhuri

College/University: Kallam HaranadhaReddy Institute of Technology

Course: CSE -data science

Year: 2022-2026

Demo Video :

https://drive.google.com/drive/folders/1AnCiSpWfIK1cH_B8idcWHOAggHdowGSF?usp=drive_link

Project Description

MedTrack is a full-stack, cloud-enabled healthcare management system using Flask for backend APIs, hosted on AWS EC2, with DynamoDB as its database. It provides a centralized platform for patients and doctors to register, book appointments, view medical history, and receive real-time notifications via AWS SNS. AWS IAM ensures secure, role-based access to system resources.

Hardware Required:

Processor: Intel i5 or equivalent (minimum). RAM: 4 GB (8 GB recommended for Full Stack MERN). Storage: 128 GB SSD or 128 GB HDD. Internet Connectivity: High-speed internet (minimum 10 Mbps per system). Additional: Audio-visual setup for interactive sessions (microphone, speakers, etc.).

Software Required:

Updated web browser (Google Chrome, Firefox, or Microsoft Edge). Visual Studio Code (or any preferred IDE). Git (latest version).

System Required:

Projector and Audio System for presentations in all labs/classrooms Classrooms/Labs are equipped with systems or provisions for students to join sessions with their own laptops.

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.

Scenarios:**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.

Architecture

This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for hosting the backend, with a lightweight framework like Flask handling core logic. Application data is stored in Amazon DynamoDB, ensuring fast, reliable access, while user access is managed through AWS IAM for secure authentication and control. Real-time alerts and system notifications are enabled via Amazon SNS, enhancing communication and user engagement.

Entity Relationship (ER) Diagram

An ER (Entity-Relationship) diagram visually represents the logical structure of a database by defining entities, their attributes, and the relationships between them. It helps organize data efficiently by illustrating how different components of the system interact and relate. This structured approach supports effective database normalization, data integrity, and simplified query design.

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.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 WorkFlowMilestone

Milestone 1. Web Application Development and Setup

- Develop the Backend Using Flask.
- Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login

- Set up an AWS account if not already done.
- Login to AWS Management Console.

Milestone 3. DynamoDB Database Creation and Setup

- Create a DynamoDB Table.
- Configure Attributes for User Data and Book Requests.

Milestone 4. SNS Notification Setup

- Create SNS topics for book request notifications.
- Subscribe users and library staff to SNS email notifications.

Milestone 5. IAM Role Setup

- Create IAM Role
- Attach Policies

Milestone 6. EC2 Instance Setup

- Launch an EC2 instance to host the Flask application.

- Configure security groups for HTTP, and SSH access.

Milestone 7. Deployment using EC2

- Upload Flask Files
- Run the Flask App

Milestone 8. Testing and Deployment

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

Milestone 1: Web Application Development and Setup

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

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.

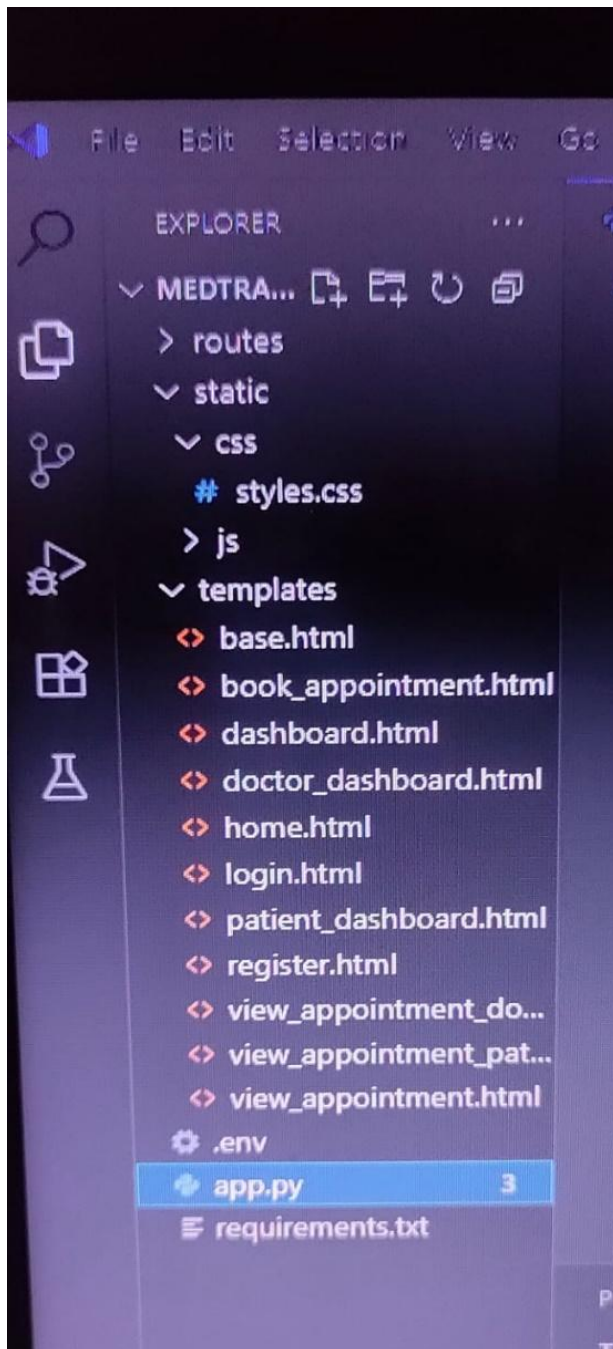
Post Troven Access Activation:

- 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.

```
app.py
1  from flask import Flask, render_template, request, redirect, session
2  from flask_mail import Mail, Message
3  import boto3
4  import uuid
5  from datetime import datetime
6
7  app = Flask(__name__)
8  app.secret_key = 'your_secret_key_here'
9
10 # ----- AWS DynamoDB Configuration -----
11 dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
12 users_table = dynamodb.Table('users')
13 appointments_table = dynamodb.Table('appointments')
14
15 # ----- AWS SNS Configuration -----
16 sns = boto3.client('sns', region_name='us-east-1')
17
18 # ----- Email Configuration -----
19 app.config['MAIL_SERVER'] = 'smtp.gmail.com'
20 app.config['MAIL_PORT'] = 587
21 app.config['MAIL_USE_TLS'] = True
22 app.config['MAIL_USERNAME'] = 'your_email@gmail.com' # ✅ Your Gmail
23 app.config['MAIL_PASSWORD'] = 'your_app_password'   # ✅ App password
24
```

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

- 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 > ...
10 # ----- AWS DynamoDB Configuration -----
11 dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
12 users_table = dynamodb.Table('users')
13 appointments_table = dynamodb.Table('appointments')
14
15 # ----- AWS SNS Configuration -----
16 sns = boto3.client('sns', region_name='us-east-1')
17
18 # ----- Email Configuration -----
19 app.config['MAIL_SERVER'] = 'smtp.gmail.com'
20 app.config['MAIL_PORT'] = 587
21 app.config['MAIL_USE_TLS'] = True
22 app.config['MAIL_USERNAME'] = 'your_email@gmail.com' # ✓ Your Gmail
23 app.config['MAIL_PASSWORD'] = 'your_app_password' # ✓ App password
24
25 mail = Mail(app)
26
27 # ----- Routes -----
28
29 @app.route('/')
30 def home():
31     return render_template('home.html')
32
33
34 @app.route('/register', methods=['GET', 'POST'])
35 def register():
36     if request.method == 'POST':
37         role = request.form['role']

```

- **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.

```

    )
    mail.send(msg)
except Exception as e:
    print(f"Email error: {e}")

try:
    sns.publish(
        PhoneNumber='+15555555555', # Replace with verified phone number
        Message=f"New appointment: {patient_email} -> {doctor_email} on {date}"
    )
except Exception as e:
    print(f"SNS error: {e}")

return "Appointment booked successfully!"

return render_template('book_appointment.html')

```

- **Routes for Web Pages:**

- Register Page**

- The login route handles user input for authentication, verifying credentials against stored data in DynamoDB. On successful login, it increments the login count and redirects the user to the appropriate dashboard. This process ensures secure and efficient access to the platform.

- *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.*

```

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        response = users_table.get_item(Key={'username': username})
        user = response.get('Item')

        if user and user['password'] == password:
            session['username'] = username
            session['role'] = user['role']
            return redirect(f"/{user['role']}")
        return "Invalid credentials!"

    return render_template('login.html')

@app.route('/doctor')
def doctor_dashboard():
    if 'role' in session and session['role'] == 'doctor':
        response = appointments_table.scan()
        appointments = [a for a in response.get('Items', []) if a['doctor_email'] == session['username']]
        return render_template('doctor_dashboard.html', username=session['username'], appointments=appointments)
    return redirect('/login')

```

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.

```

pp.py > ...
def book_appointment():

    try:
        sns.publish(
            PhoneNumber='+15555555555', # Replace with verified phone number
            Message=f"New appointment: {patient_email} -> {doctor_email} on {date}"
        )
    except Exception as e:
        print(f"SNS error: {e}")

    return "Appointment booked successfully!"

return render_template('book_appointment.html')

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

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

```

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.

```

pp.py > ...
def book_appointment():

    try:
        sns.publish(
            PhoneNumber='+15555555555', # Replace with verified phone number
            Message=f"New appointment: {patient_email} -> {doctor_email} on {date}"
        )
    except Exception as e:
        print(f"SNS error: {e}")

    return "Appointment booked successfully!"

return render_template('book_appointment.html')

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

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

```

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.

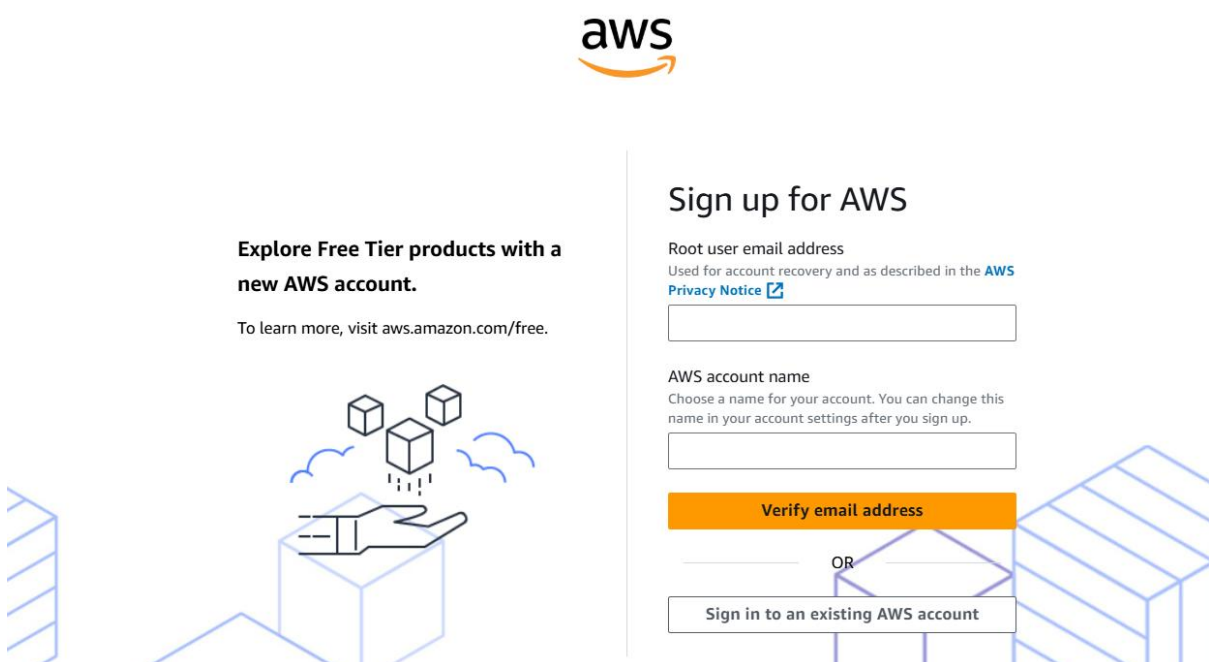
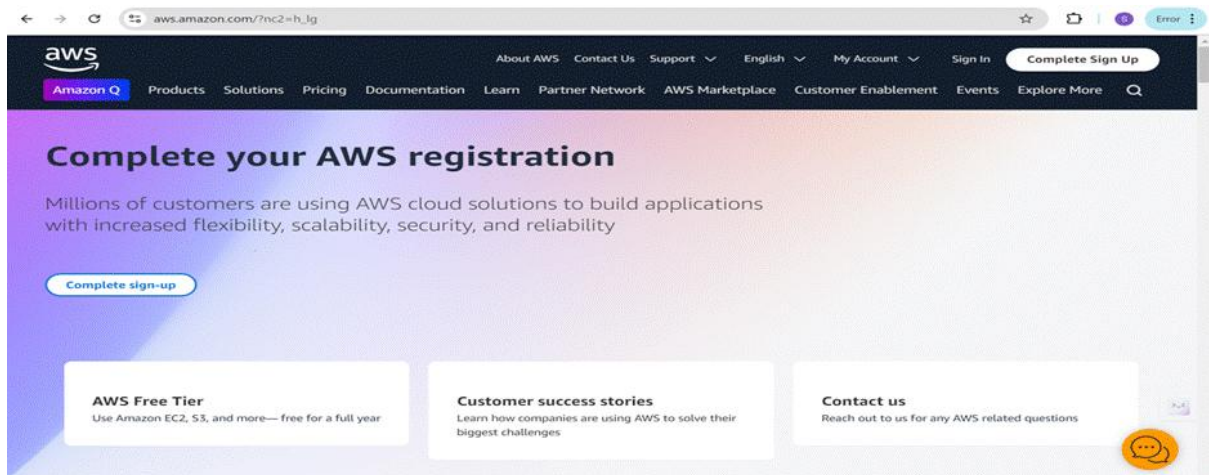
Once your website is locally deployed and fully functional, you must proceed with integrating AWS services only through the Troven Labs environment. This ensures secure, controlled access to AWS resources without any risk of personal billing.

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.

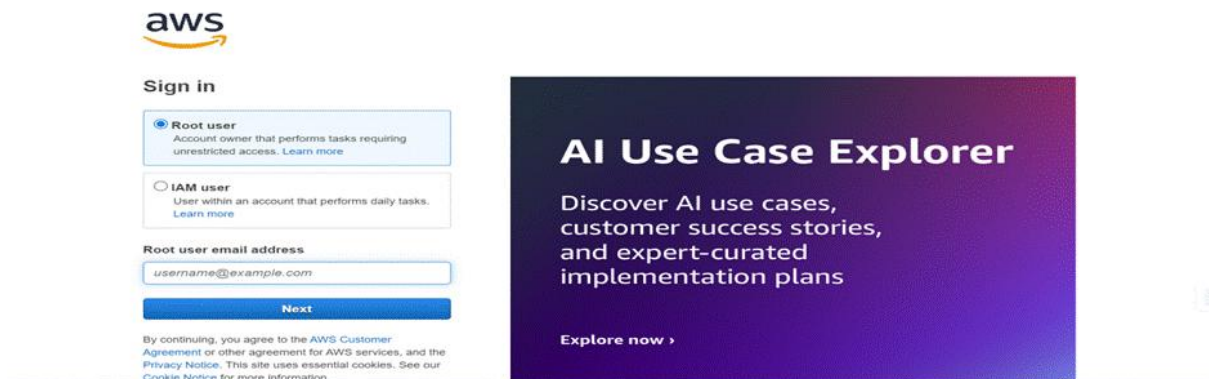
AWS Account Setup and Login

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.



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

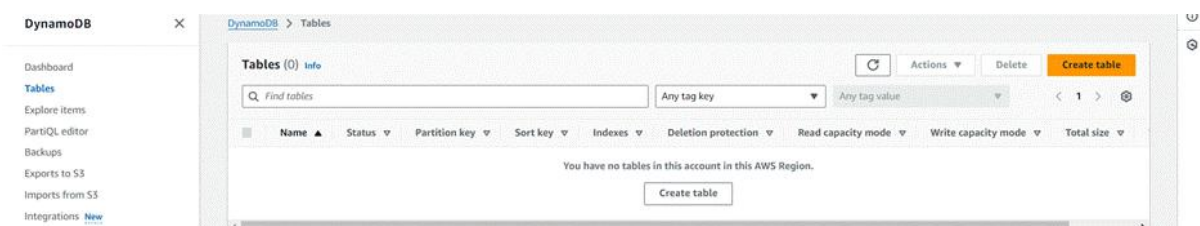
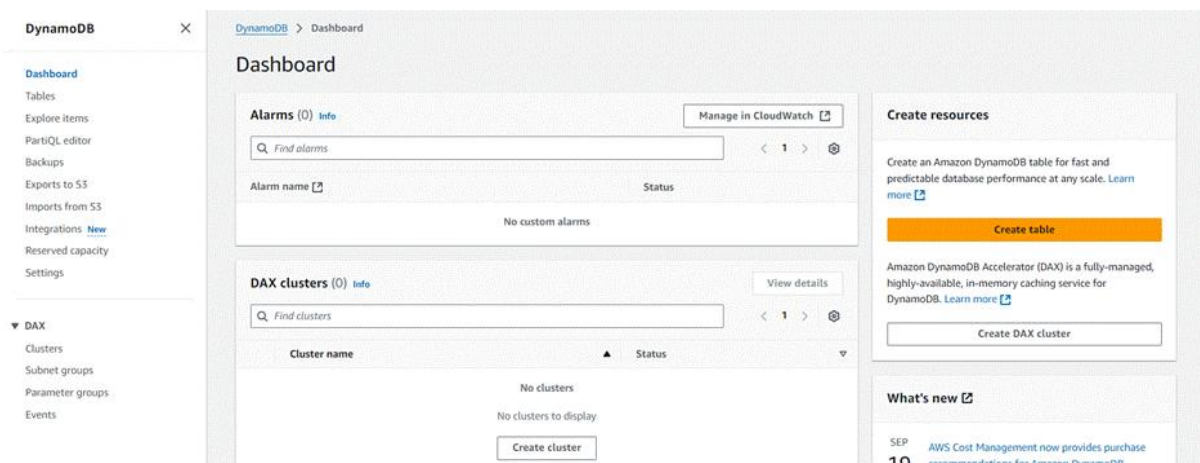
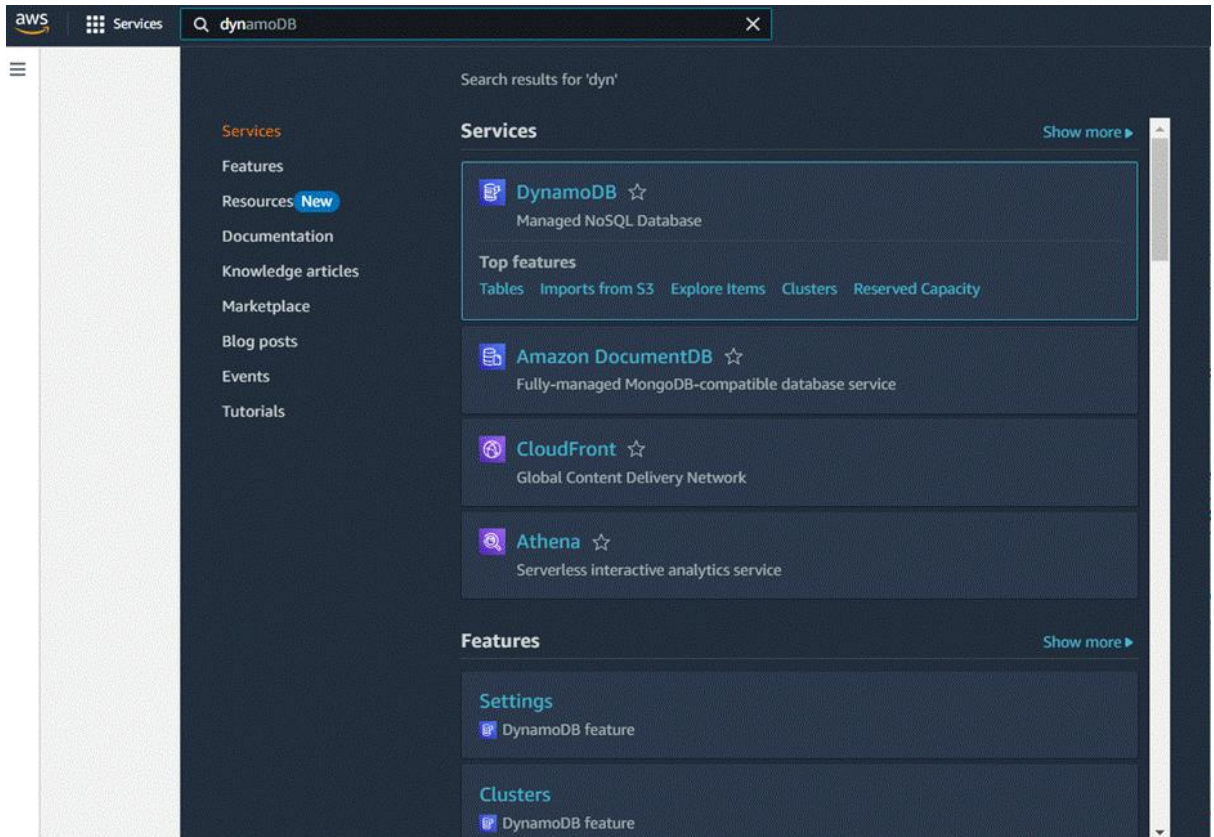


Milestone 3: DynamoDB Database Creation and Setup

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

Navigate to the DynamoDB

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



Create a DynamoDB table for storing data

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

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 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](#)

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

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

This will be used to identify your table.

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

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

1 to 255 characters and case sensitive.

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

Tags 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.

You can add 50 more tags.

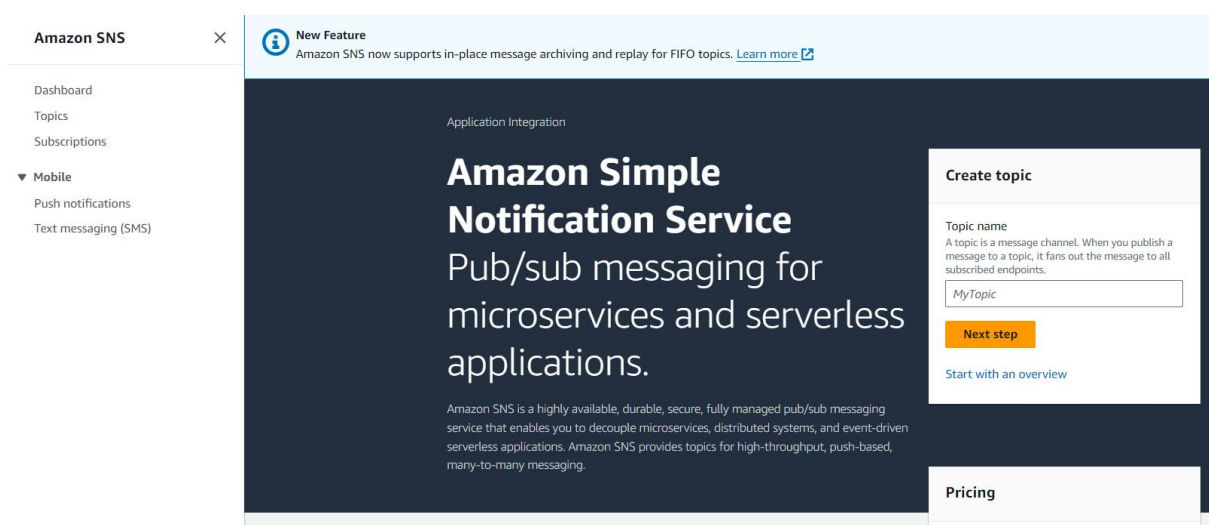
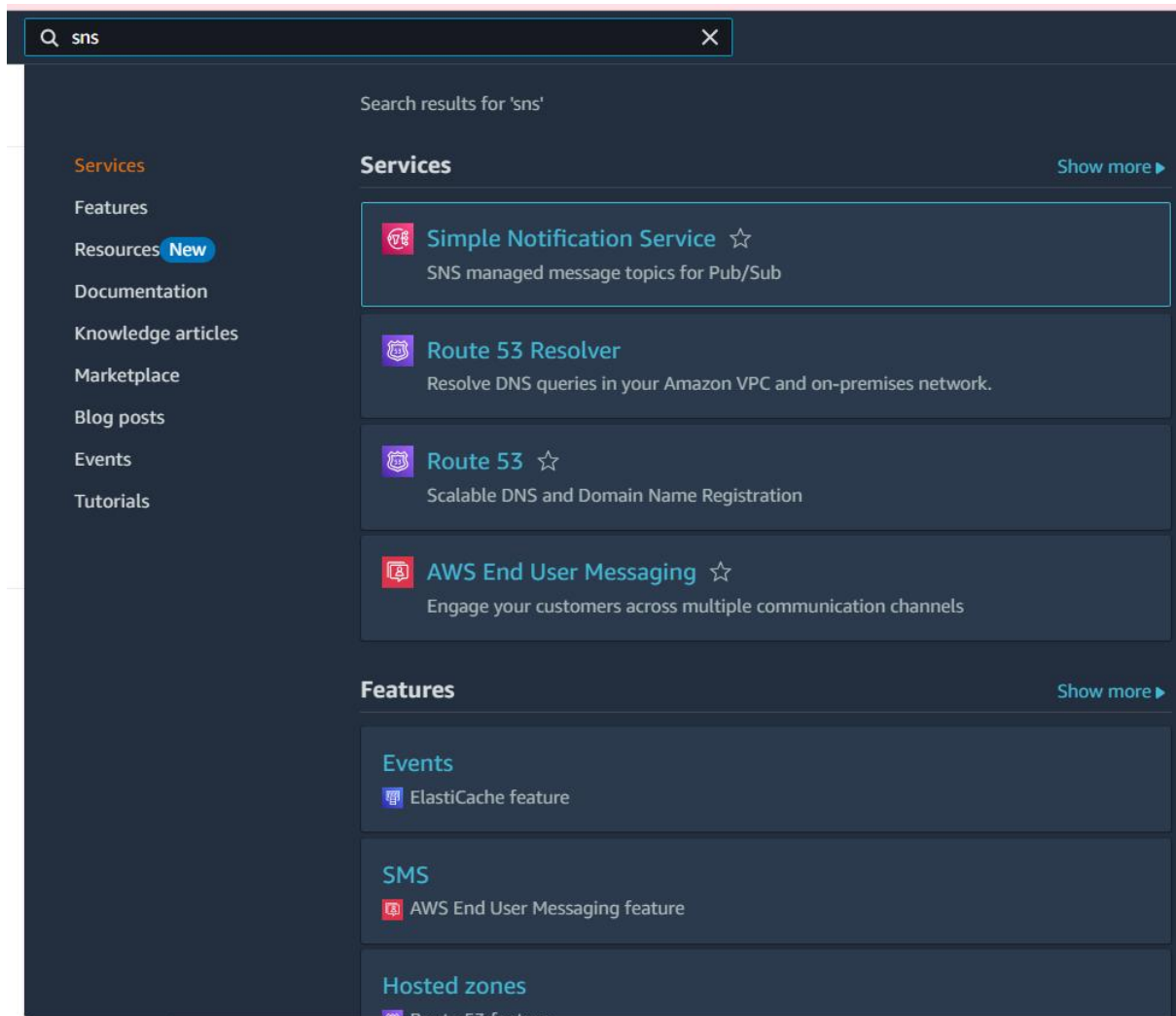
Tables (2/10) Info		Actions Delete Create table					
Q Status X		Any tag key ▼		Any tag value ▼		< 1 > ⚙️	
<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection
<input checked="" type="checkbox"/>	AppointmentsTable	Active	appointment_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_Appointments	Active	appointment_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_ContactMessages	Active	message_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_Doctors	Active	doctor_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_PatientRecords	Active	patient_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_Users	Active	email (S)	-	0 0		Off
<input type="checkbox"/>	SalonAppointments	Active	appointment_id (S)	user_email (S)	0 0		Off
<input type="checkbox"/>	SalonStylists	Active	stylist_id (S)	-	0 0		Off
<input type="checkbox"/>	SalonUsers	Active	email (S)	-	0 0		Off
<input checked="" type="checkbox"/>	UsersTable	Active	email (S)	-	0 0		Off

Milestone 4 : SNS Notification Setup

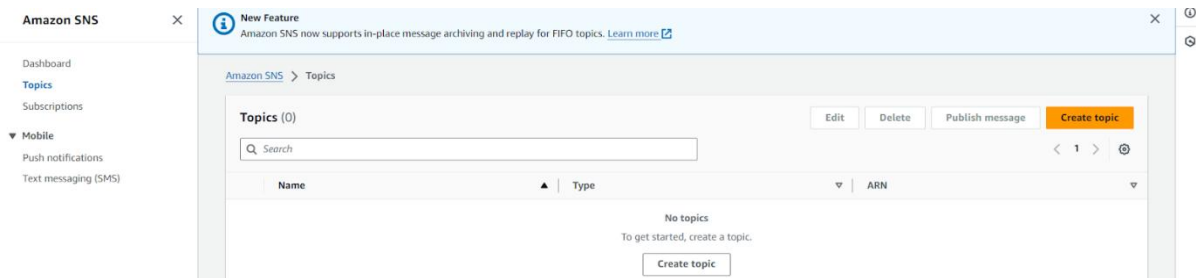
Amazon SNS is a fully managed messaging service that enables real-time notifications through channels like SMS, email, or app endpoints. You create topics, configure subscriptions, and integrate SNS into your app to send notifications based on specific events.

SNS topics for email notifications

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



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



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

Amazon SNS > Topics > Create topic

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

Create topic

Details

Type | [Info](#)
Topic type cannot be modified after topic is created

☐ FIFO (first-in, first-out)

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

☒ 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 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.

Amazon SNS

Topics

Medtrack

Dashboard

Topics

Subscriptions

Mobile

Push notifications

Text messaging (SMS)

New Feature

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

Topic Medtrack created successfully.

You can create subscriptions and send messages to them from this topic.

Publish message

Medtrack

Edit

Delete

Publish message

Details

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

Subscriptions

Access policy

Data protection policy

Delivery policy (HTTP/S)

Delivery status logging

Encryption

Tags

Subscriptions (1)

Edit

Delete

Request confirmation

Confirm subscription

Create subscription

Search

1

- 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 subscription](#)

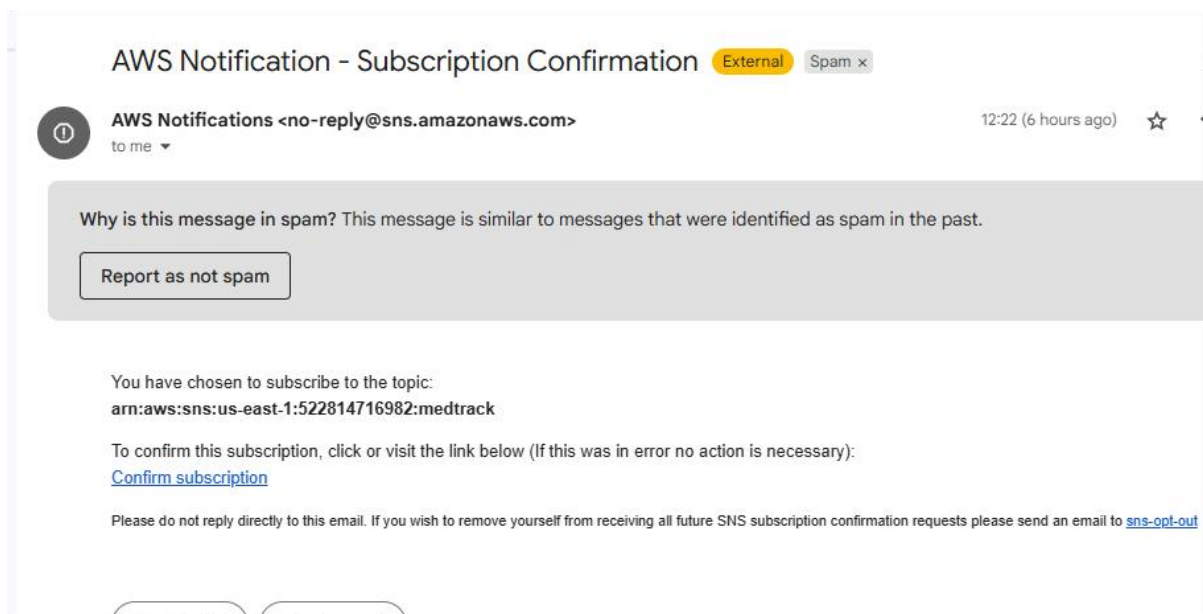
- After subscription request for the mail confirmation

< **Subscriptions** | Access policy | Data protection policy | Delivery policy (HTTP/S) | Delivery status logging | Encryption | Tags >

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

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

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





Simple Notification Service

Subscription confirmed!

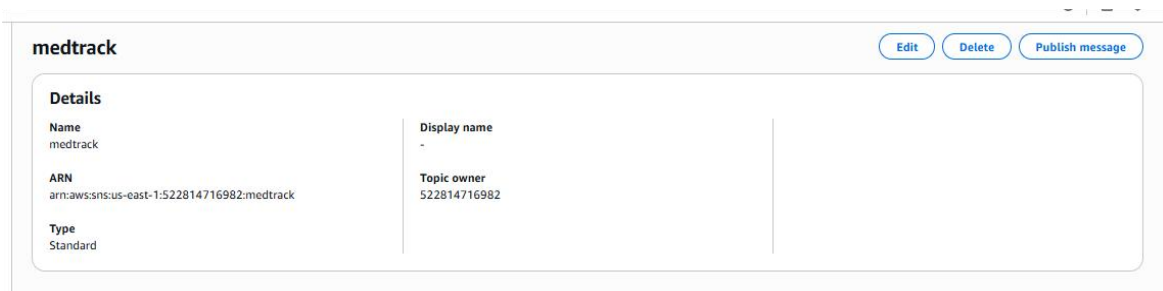
You have successfully subscribed.

Your subscription's id is:

`arn:aws:sns:us-east-1:522814716982:medtrack:71557bf5-9796-4a75-8e48-b2553b0a6ae4`

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.

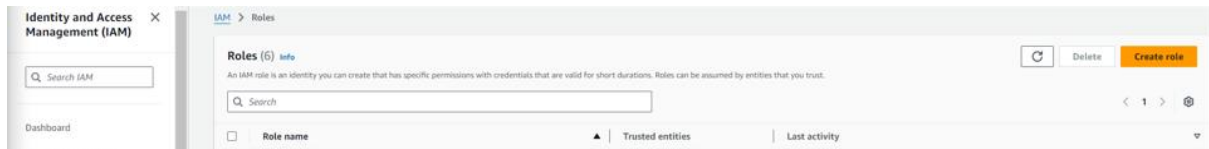
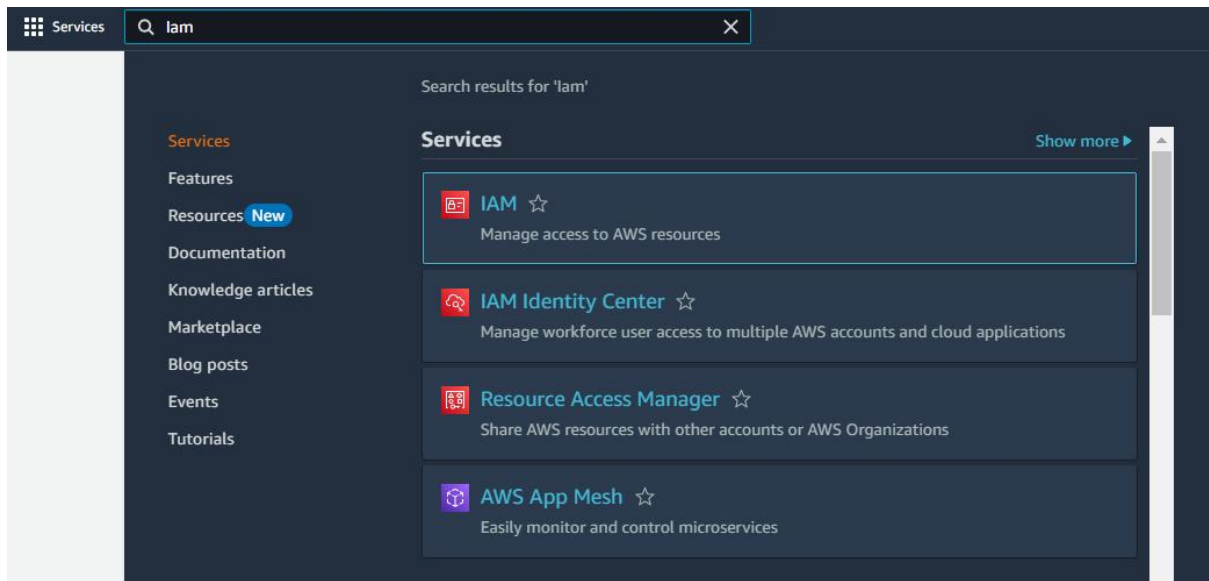


Milestone 5: IAM Role Setup

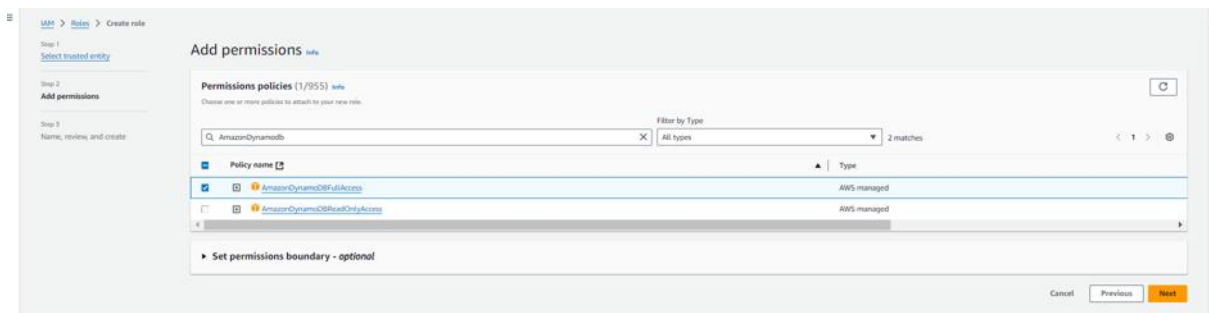
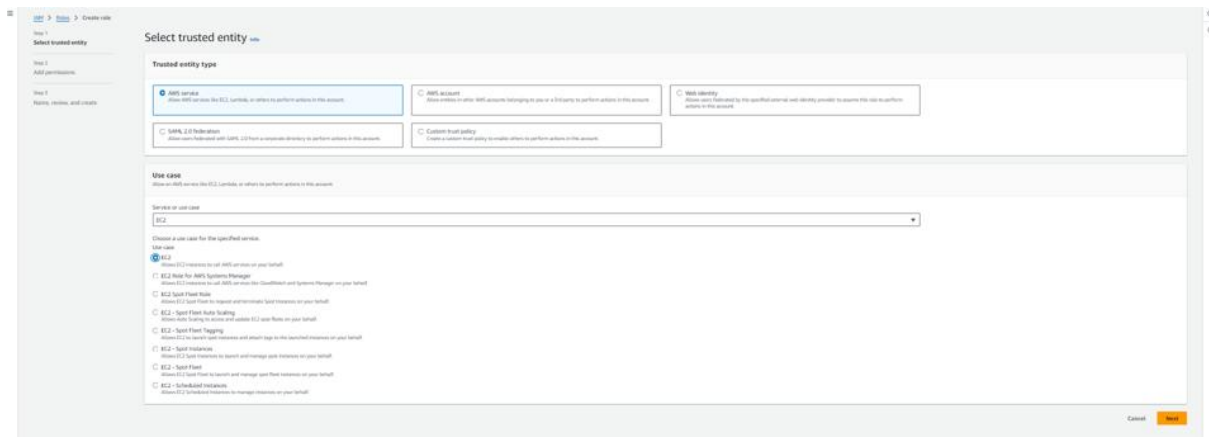
IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows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



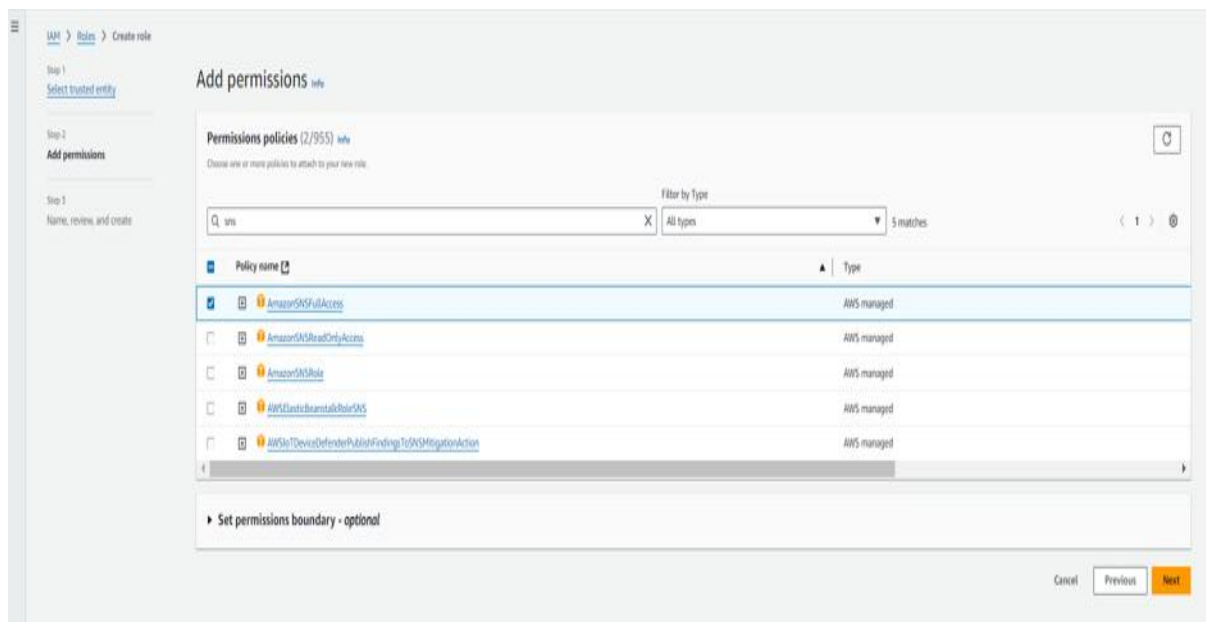
To create and select DynamoDBFullAccess and SNSFullAccess, go to the AWS IAM console, create a new role, and assign the respective policies. DynamoDBFullAccess allows full access to DynamoDB resources, while SNSFullAccess enables sending notifications via SNS. Attach the role to the relevant services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



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

Delete

Allows EC2 instances to call AWS services on your behalf.

Summary

Edit

Creation date

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

ARN

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

Instance profile ARN

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

Last activity

5 hours ago

Maximum session duration

1 hour

Permissions

Trust relationships

Tags

Last Accessed

Revoke sessions

Permissions policies (2)

Info

Simulate

Remove

Add permissions

You can attach up to 10 managed policies.

Search

Filter by Type

All types

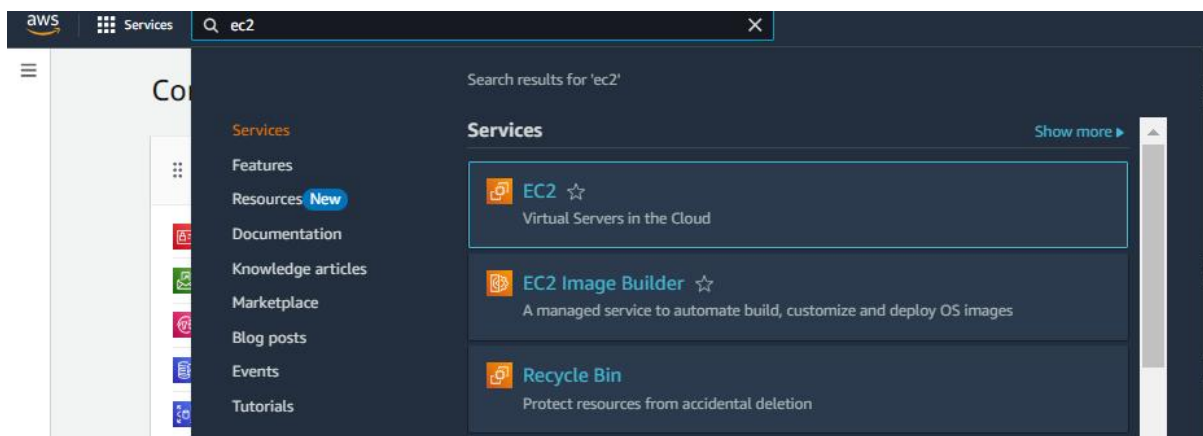
Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	3
AmazonSNSFullAccess	AWS managed	3

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.

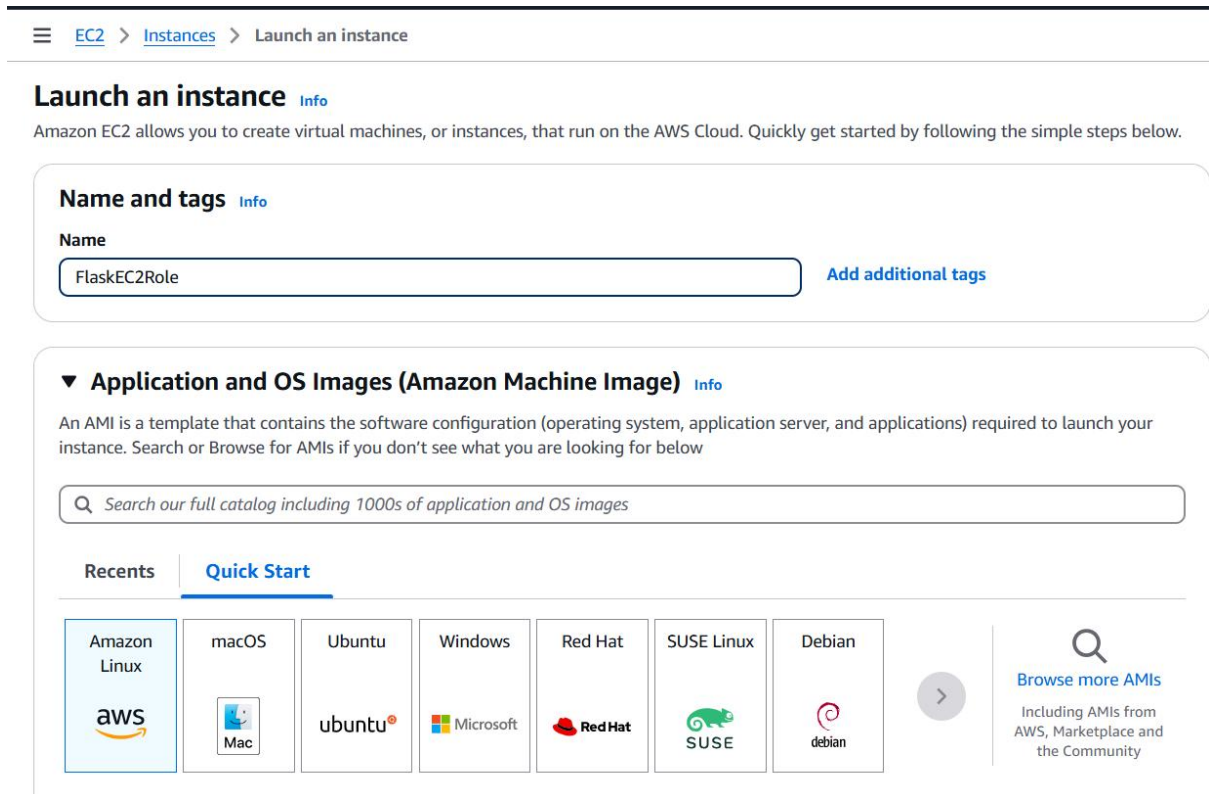
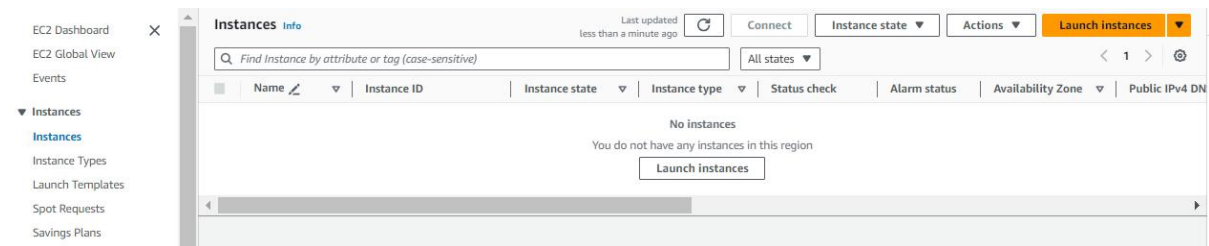
- **Launch EC2 Instance**
 - In the AWS Console, navigate to EC2 and launch a new instance.



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

1. Go to the **AWS EC2 Dashboard** and click on **Launch Instance**.

2. Select your desired AMI, instance type, configure instance details, and under **IAM role**, choose the role **flaskec2role**. Finally, launch the instance.



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

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

-

Amazon Linux
aws

macOS
Mac

Ubuntu
ubuntu

Windows
Microsoft

Red Hat
Red Hat

Browse more AMIs
Including AMIs from AWS, Marketplace and the Community

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:

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

▼ Instance type Info | Get advice

Instance type

t2.micro

Free tier eligible

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

On-Demand Linux base pricing: 0.0124 USD per Hour

On-Demand Windows base pricing: 0.017 USD per Hour

On-Demand RHEL base pricing: 0.0268 USD per Hour

On-Demand SUSE base pricing: 0.0124 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) Info

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

Key pair name - required

Select

Create new key pair

▼ Key pair (login) [Info](#)

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

Key pair name - *required*

linuxkeypair



[Create new key pair](#)

Configure security groups for HTTP, and SSH access.

For network settings during EC2 instance launch:

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

▼ Network settings [Info](#)

VPC - *required* [Info](#)

vpc-03cdc7b6f19dd7211
172.31.0.0/16

(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 is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and _-:/()#,@!+=&:()!\$*

Description - *required* [Info](#)

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

Inbound Security Group Rules

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

Type Info	Protocol Info	Port range Info
ssh	TCP	22
Source type Info	Source Info	Description - optional Info
Anywhere	<input type="text" value="Add CIDR, prefix list or security"/> 0.0.0.0/0 <input type="button" value="X"/>	<input type="text" value="e.g. SSH for admin desktop"/>

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

Type Info	Protocol Info	Port range Info
HTTP	TCP	80
Source type Info	Source Info	Description - optional Info
Custom	<input type="text" value="Add CIDR, prefix list or security"/> 0.0.0.0/0 <input type="button" value="X"/>	<input type="text" value="e.g. SSH for admin desktop"/>

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

Type Info	Protocol Info	Port range Info
Custom TCP	TCP	5000
Source type Info	Source Info	Description - optional Info
Custom	<input type="text" value="Add CIDR, prefix list or security"/> 0.0.0.0/0 <input type="button" value="X"/>	<input type="text" value="e.g. SSH for admin desktop"/>

EC2 > ... > Launch an instance

Success
Successfully initiated launch of instance (i-001861022fbcac290)

► Launch log

Next Steps

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.

Connect to your instance

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

[Learn more](#)

Connect an RDS database

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

[Create a new RDS database](#)

[Learn more](#)

Create EBS snapshot policy

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

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.

Create Load Balancer

Create an application, network gateway or classic Elastic 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.

Manage CloudWatch alarms

Create or update Amazon CloudWatch alarms for the instance.

Disaster recovery for your instances

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

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.

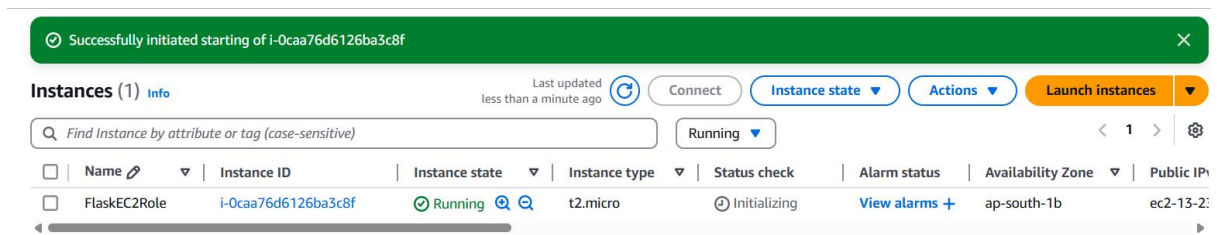
Get instance screenshot

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

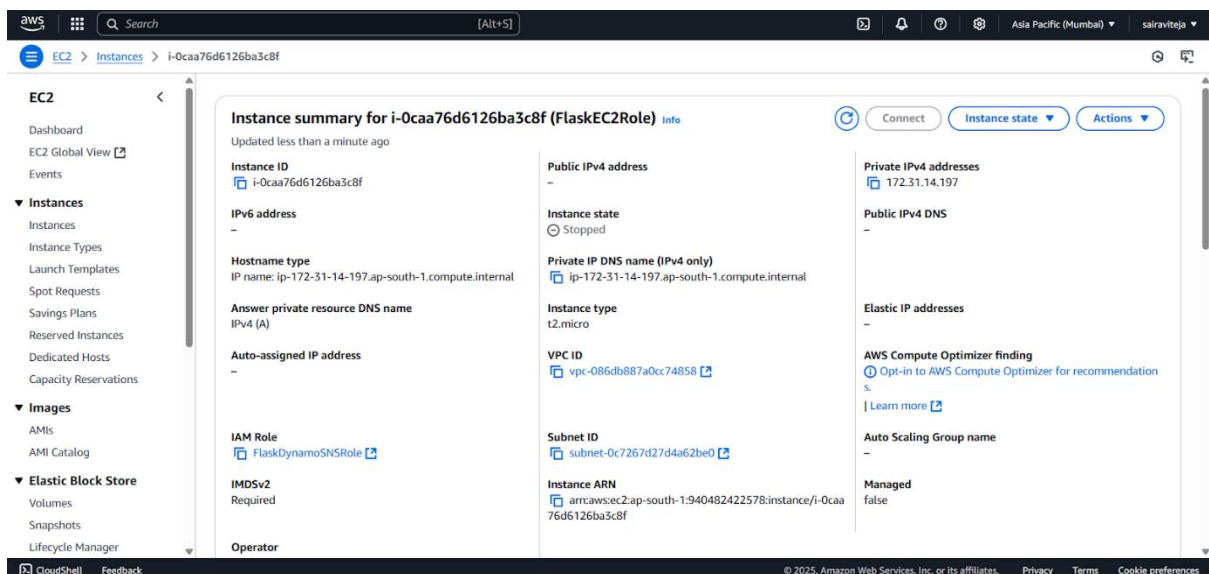
Get system log

View the instance's system log to troubleshoot issues.

- 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:

1. Go to the **AWS IAM Console**, select **Roles**, and find the **flaskec2role**.
2. Click **Attach Policies**, then choose the required policies (e.g., **DynamoDBFullAccess**, **SNSFullAccess**) and click **Attach Policy**.
3. If needed, update the instance to use this modified role by selecting the EC2 instance, clicking **Actions**, then **Security**, and **Modify IAM role** to select the updated role.

Modify IAM role Info

Attach an IAM role to your instance.

Instance ID
i-0caa76d6126ba3c8f (FlaskEC2Role)

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

[Cancel](#) [Update IAM role](#)

To connect to your EC2 instance:

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

Instances (1) Info Last updated less than a minute ago [Connect](#) [Instance state](#) [Actions](#) [Launch instances](#)

[Running](#) < 1 > [Filter table to exclude running instances](#)

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
<input type="checkbox"/>	FlaskEC2Role	i-0caa76d6126ba3c8f	Running	t2.micro	2/2 checks passed	View alarms	ap-south-1b	ec2-13-203

- Now connect the EC2 with the files

Connect to instance Info

Connect to your instance i-0caa76d6126ba3c8f (FlaskEC2Role) using any of these options

[EC2 Instance Connect](#) [Session Manager](#) [SSH client](#) [EC2 serial console](#)

Instance ID
i-0caa76d6126ba3c8f (FlaskEC2Role)

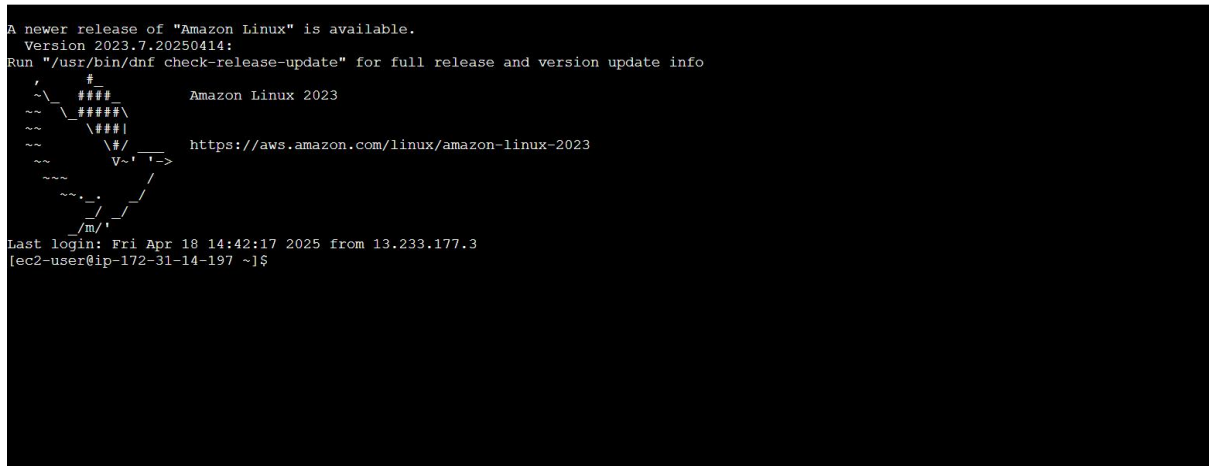
Connection Type
☒ Connect using EC2 Instance Connect
 Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.
 ☐ Connect using EC2 Instance Connect Endpoint
 Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

☒ Public IPv4 address
 13.203.157.118
 ☐ IPv6 address

Username
 Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

[Cancel](#) [Connect](#)



Milestone 7 : Deployment on EC2

Install Software on the EC2 Instance

On Amazon Linux 2:

```
sudo pip3 install flask boto3
```

```
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/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/KVeenaMadhuri/medtrack.git

- This will download your project to the EC2 instance.

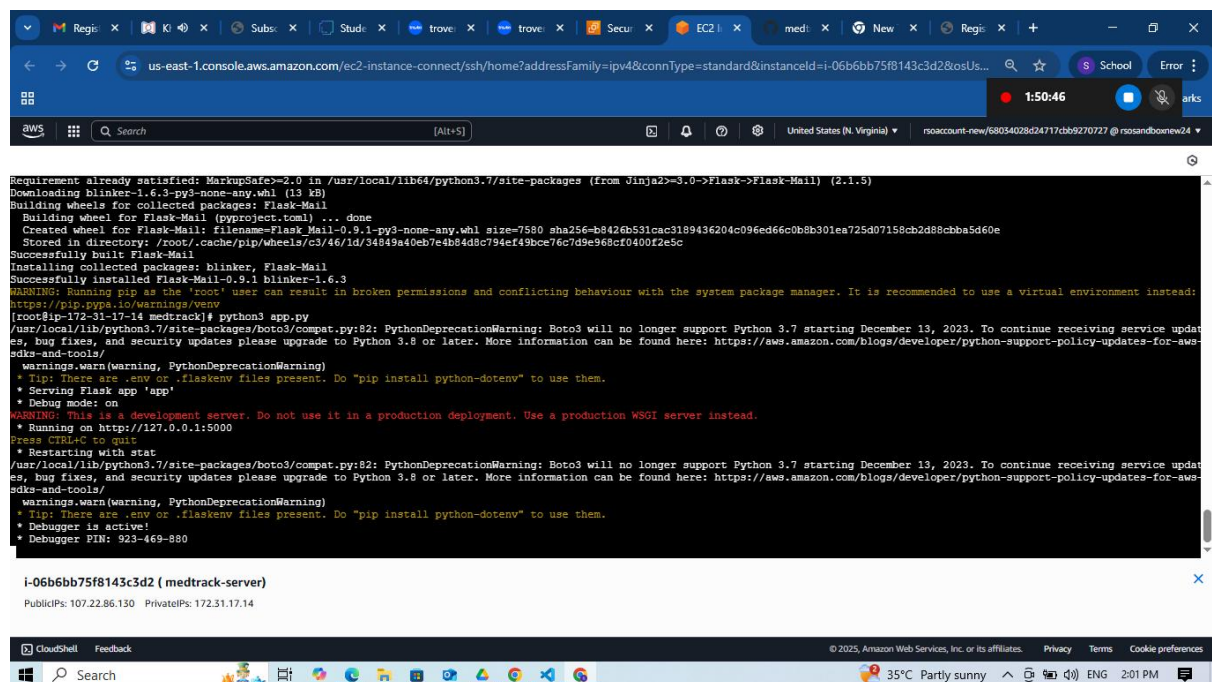
To navigate to the project directory, run the following command:

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=5000



The screenshot shows a terminal window within the AWS Management Console. The terminal output displays the following steps:

- Requirement already satisfied: MarkupSafe<=2.0 in /usr/local/lib64/python3.7/site-packages (from Jinja2>=3.0->Flask->Flask-Mail) (2.1.5)
- Downloading blinker-1.6.3-py3-none-any.whl (13 kB)
- Building wheels for collected packages: Flask-Mail
- Building wheel for Flask-Mail (pyproject.toml) ... done
- Created wheel for Flask-Mail: filename=Flask-Mail-0.9.1-py3-none-any.whl size=7580 sha256=b8426b531cac3189436204c096ed66c0b8b301ea725d07158cb2d88c8ba5d60e
- Stored in directory: /root/.cache/pip/wheels/c3/46/1d/34849a40eb7e4b848c794ef49bce76c7d9e968cf0400f2e5c
- Successfully built Flask-Mail
- Installing collected packages: blinker, Flask-Mail
- Successfully installed Flask-Mail-0.9.1 blinker-1.6.3
- WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
- [root@ip-172-31-17-14 medtrack]# python3 app.py
- /usr/local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
- warnings.warn(warning, PythonDeprecationWarning)
- * Tip: There are .env or .flaskenv files present. Do "pip install python-dotenv" to use them.
- * 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
- /usr/local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
- warnings.warn(warning, PythonDeprecationWarning)
- * Tip: There are .env or .flaskenv files present. Do "pip install python-dotenv" to use them.
- * Debugger is active!
- * Debugger PIN: 923-469-880

Below the terminal output, the instance details are shown:

- Instance: i-06b6bb75f8143c3d2 (medtrack-server)
- Public IPs: 107.22.86.130 Private IPs: 172.31.17.14

The bottom of the screenshot shows the AWS console header with the date and time: 35°C Partly sunny, 2:01 PM.

Verify the Flask app is running:

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

- Run the Flask app on the EC2 instance

```
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:80
* Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```

Access the website through:

PublicIPs: <http://192.168.43.173:5000>

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.

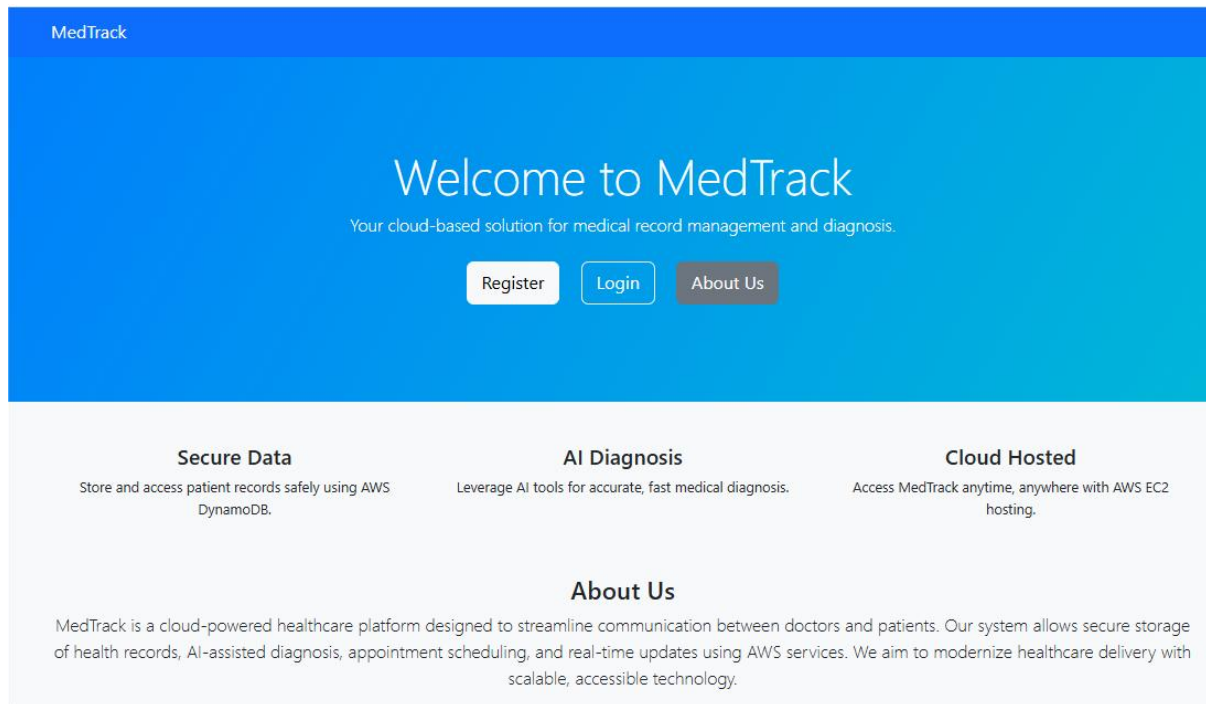
Functional testing to verify the project

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:

1. Input Fields: For users to enter basic information like appointment requests, diagnosis submissions, or service bookings.
2. Navigation: Links to other sections such as the login page, dashboard, or service options.
3. Responsive Design: Ensures the page is accessible across devices with a clean, user-friendly interface.

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



DOCTOR AND PATIENT REGISTRATION PAGE :

The Doctor Registration Page allows doctors to register and create an account on the platform. It typically includes:

1. **Input Fields:** For doctor details such as name, specialty, qualifications, and contact information.
2. **Login Credentials:** Fields for setting a username and password for secure access.
3. **Submit Button:** A button to submit the registration details, which will then be stored in the database after validation.

The screenshot displays a web browser window with a single tab titled 'Register'. The address bar shows the URL '192.168.43.173:5000/register' with a 'Not secure' warning. The main content area features a white 'Register' form with the following fields: 'Name' (text input), 'Email' (text input), 'Password' (text input), 'Role' (dropdown menu with 'Patient' selected), 'Age' (text input), and 'Gender' (dropdown menu with 'Male' selected). A blue 'Register' button is positioned at the bottom of the form. The browser's Windows taskbar is visible at the bottom, showing the search bar, application icons, and system tray information including '32°C Mostly cloudy' and '7:39 PM'.

PATIENT AND DOCTOR LOGIN PAGES:

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

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

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

The image shows a login form with a white background and rounded corners, set against a light gray background. The form is titled "Login" in a bold, black font. Below the title, there are three input fields: "Email" (a text box), "Password" (a text box), and "Role" (a dropdown menu). The "Role" dropdown menu is currently set to "Doctor" and has a small downward arrow icon. Below these fields is a blue button with the text "Login" in white.

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.