

# **MedTrack: AWS Cloud-Enabled Healthcare Management System**

## **Project Description:**

MedTrack is a secure, cloud-based healthcare management platform designed to streamline patient care and administrative workflows. Built on Amazon Web Services (AWS), it allows healthcare providers to manage electronic health records, schedule appointments, conduct telemedicine consultations, and issue e-prescriptions from a single system.

The platform uses AWS EC2 for hosting applications, Amazon RDS for reliable database storage, S3 for storing medical documents, and Cognito for secure user authentication and access control. Automated notifications and reminders are sent via AWS SNS to keep patients informed.

By leveraging AWS cloud infrastructure, MedTrack delivers high availability, scalability, and robust data protection, helping clinics and hospitals improve efficiency while maintaining compliance with healthcare data regulations like HIPAA.

### **Scenario 1 – Appointment Booking and Notifications**

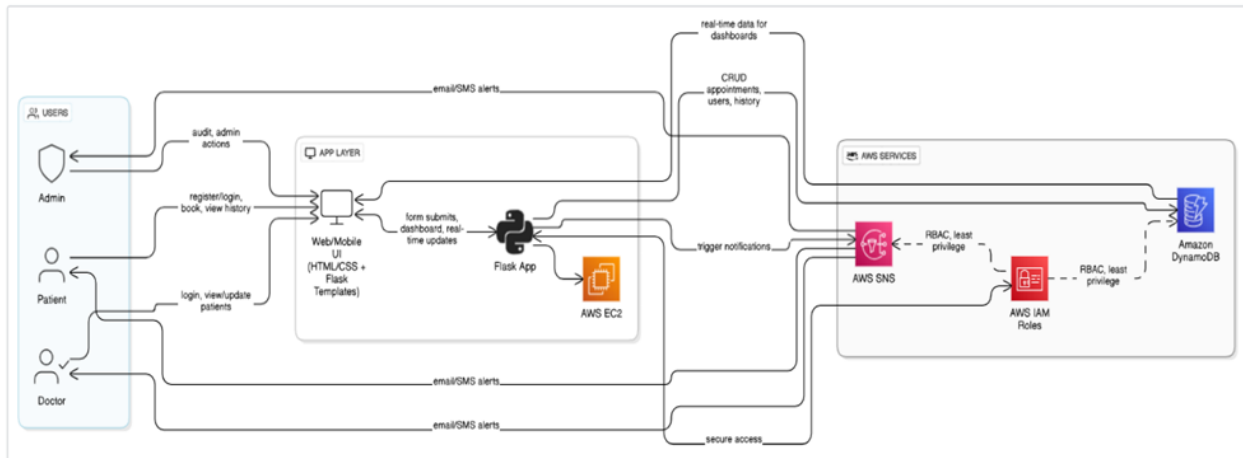
A patient login into the MedTrack web app, which is hosted on Amazon EC2 instances. They schedule an appointment, and the booking details are saved in Amazon DynamoDB, enabling fast retrieval and updates. Immediately, AWS SNS sends a confirmation SMS and email to the patient with appointment details.

### **Scenario 2 – Medical Record Update with Secure Access**

A doctor logs in through the MedTrack portal running on EC2. Using permissions managed by AWS IAM, the doctor can securely access and update patient records stored in DynamoDB. Any changes are recorded in real time so all authorized staff see up-to-date information.

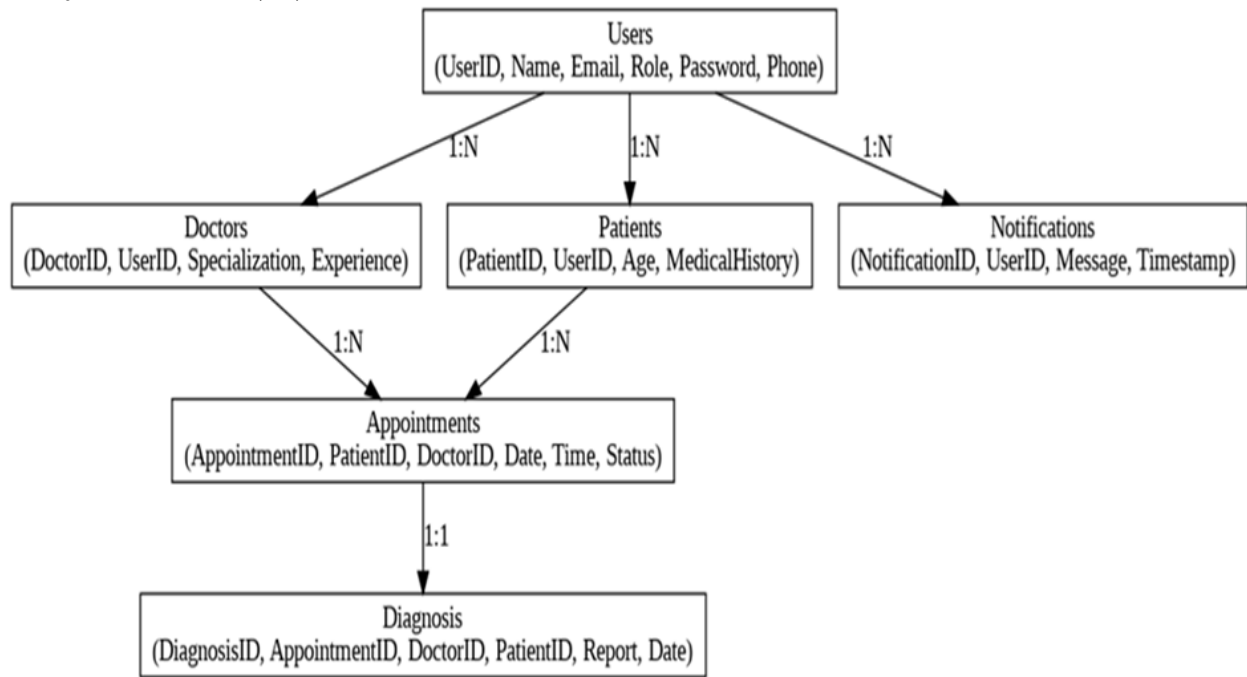
### **Scenario 3 – Emergency Alerts and Notifications**

When critical lab results are entered into the system, MedTrack triggers an alert. A Lambda function scans the DynamoDB table for urgent flags and uses AWS SNS to send high-priority notifications to the assigned doctor's mobile device. IAM policies ensure only authorized medical staff can receive and act on these alerts.



## AWS ARCHITECTURE

Entity Relationship (ER)Diagram:



Pre-requisites:

1. AWS Account Setup:

<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>

2. AWS IAM (Identity and Access Management):  
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
3. AWS EC2 (Elastic Compute Cloud):  
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
4. AWS DynamoDB:  
<https://docs.aws.amazon.com/amazondynamodb/Introduction.html>
5. Amazon SNS:  
<https://docs.aws.amazon.com/sns/latest/dg/welcome.htm>
6. Git Documentation:  
<https://git-scm.com/doc>

7. 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:**

1. AWS Account Setup and Login  
Activity 1.1: Set up an AWS account if not already done.  
Activity 1.2: Log in to the AWS Management Console
2. DynamoDB Database Creation and Setup  
Activity 2.1: Create a DynamoDB Table.  
Activity 2.2: Configure Attributes for User Data and Book Requests.
3. SNS Notification Setup  
Activity 3.1: Create SNS topics for book request notifications.  
Activity 3.2: Subscribe users and library staff to SNS email notifications.
4. Backend Development and Application Setup  
Activity 4.1: Develop the Backend using JavaScript.  
Activity 4.2: Integrate AWS Services using boto3.
5. IAM Role Setup  
Activity 5.1: Create IAM Role.  
Activity 5.2: Attach Policies.
6. EC2 Instance Setup  
Activity 6.1: Launch an EC2 instance to host the JavaScript application.  
Activity 6.2: Configure security groups for HTTP, and SSH access.
7. Deployment on EC2  
Activity 7.1: Upload JavaScript files.

Activity 7.2: Run the JavaScript App.

## 8. Testing and Deployment

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

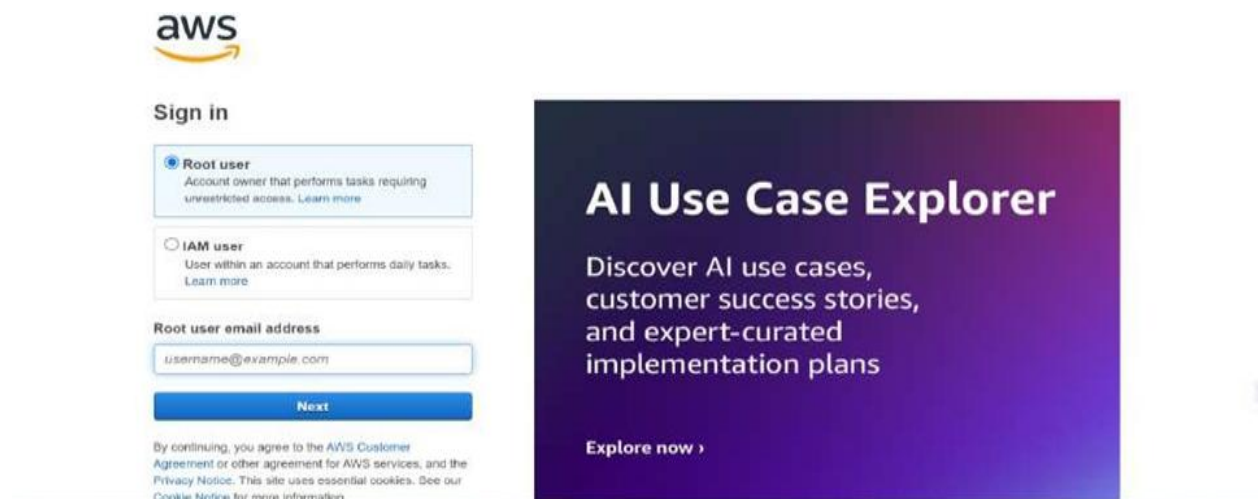
### Milestone 1: AWS Account Setup and Login

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

Sign up for an AWS account and configure billing settings.

Activity 1.2: Log in to the AWS Management Console

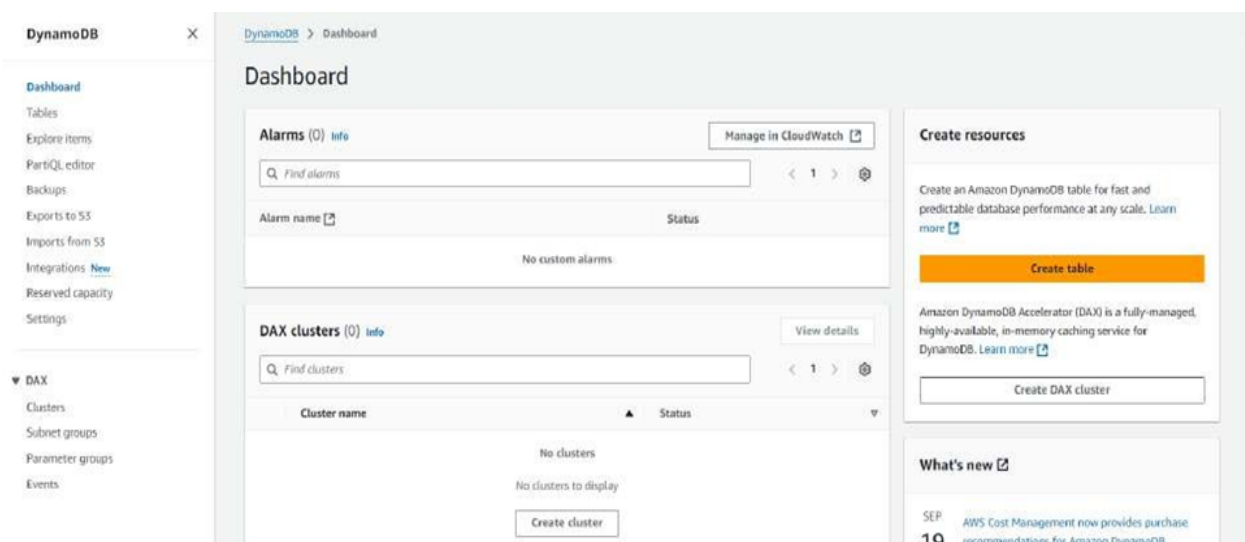
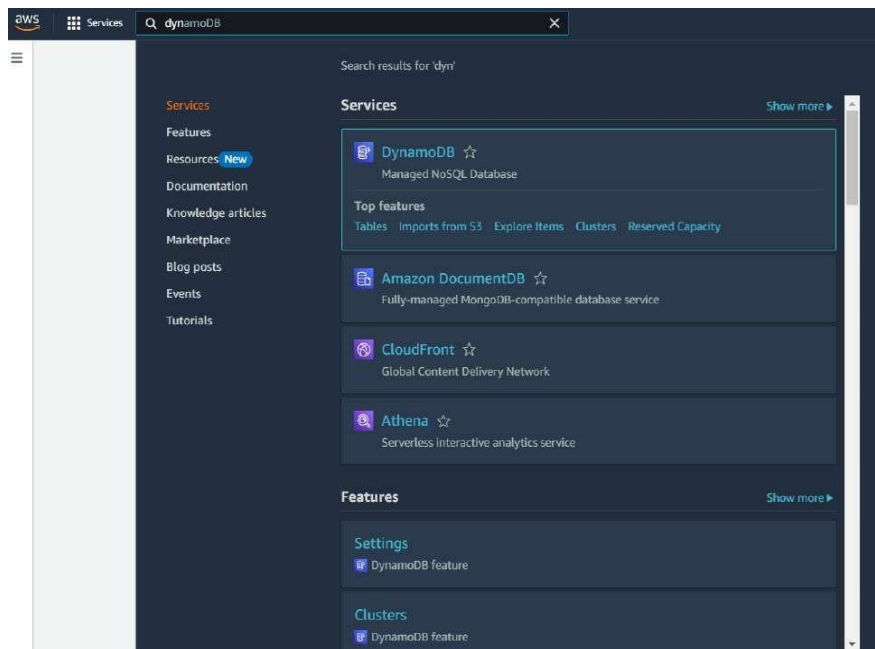
After setting up your account, log in to the [AWS Management Console](#).



### Milestone 2: Dynamo DB Database Creation and Setup

Activity 2.1: Navigate to the DynamoDB

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



Activity 2.2: Create a Dynamo DB table for storing registration details and book requests.

1. Create Users table with partition key "Email" with type String and click on create tables.



[DynamoDB](#) > [Tables](#) > Create table

## Create table

### Table details [Info](#)

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

#### Table name

This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).

#### Partition key

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

String ▼

1 to 255 characters and case sensitive.

#### Sort key - optional

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

String ▼

1 to 255 characters and case sensitive.

2. Follow the same steps to create a requests table with Email as the primary key for book requests data.

DynamoDB

Dashboard

Tables

Explore items

PartiQL editor

Backups

Exports to S3

Imports from S3

Integrations New

The Users table was created successfully.

DynamoDB > Tables

Tables (1) [Info](#)

Any tag key ▼ Any tag value ▼

< 1 > ⚙

	Name ▲	Status ▼	Partition key ▼	Sort key ▼	Indexes ▼	Deletion protection ▼	Read capacity mode ▼	Write capacity mode ▼	Total size ▼
<input type="checkbox"/>	<a href="#">Users</a>	Active	email (S)	-	0	Off	Provisioned (S)	Provisioned (S)	0 bytes

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

DynamoDB \ Tables } Create table  
 retrieve items from your table and allocate data across  
 email

1 to 255 characters and case sensitive.

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:

4 to 255 characters and case sensitive.

[DynamoDB](#) \ [Tables](#) } Create table

## Createtable

### Table details inio

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

Tablename

This will be used to identify your table.

# Smart Internz

Requests

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

email

1 to 255 characters and case sensitive.

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

Enter the sort key name

same partition key.  
4 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.

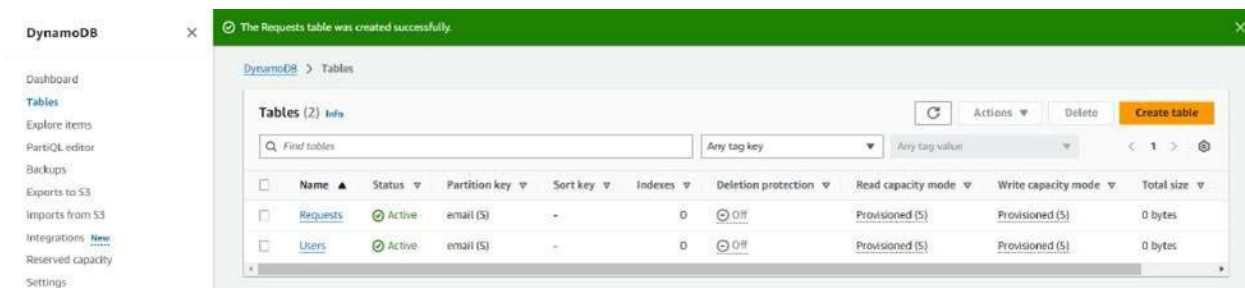
Add new tag

You can add 50 more tags.

Cancel

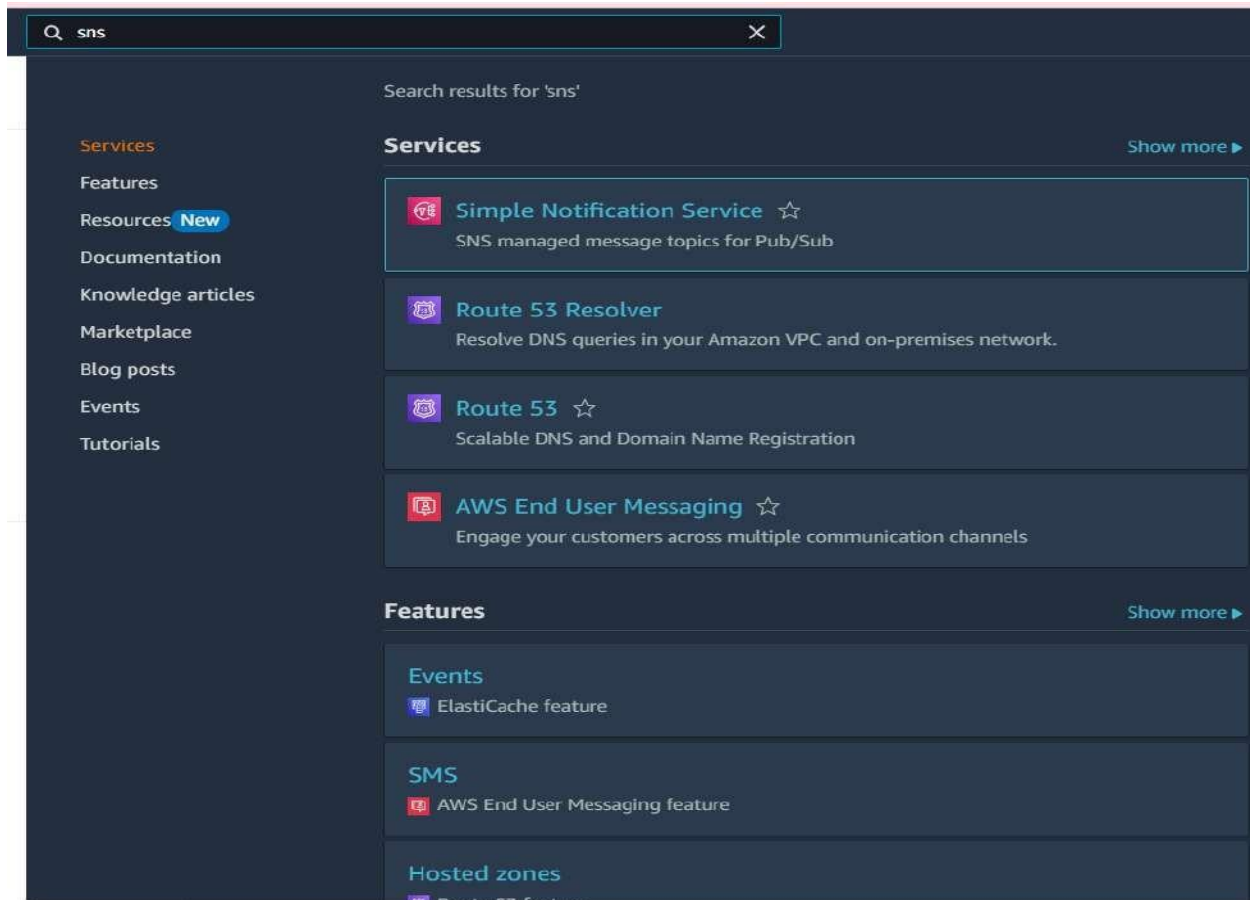
Create table





### Milestone 3: SNS Notification Setup

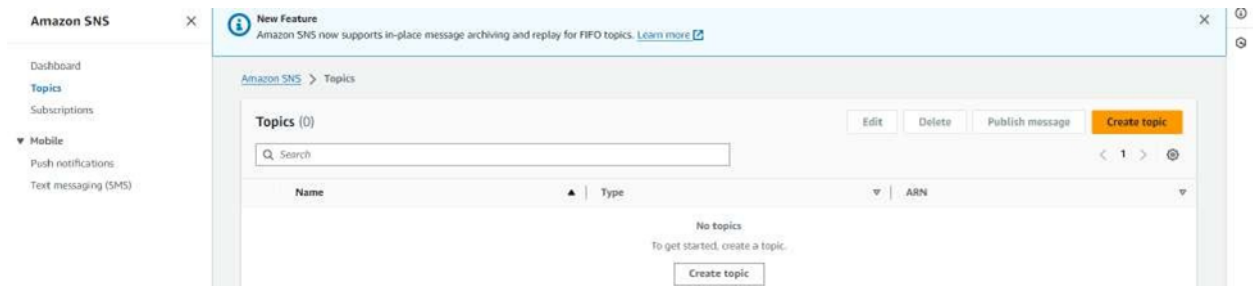
Activity 3.1: Create SNS topics for sending email notifications to users and library staff.



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



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](#) Createtopic

## Create topic

### Details

Type trrf0  
Topic type cannot be modified after topic is created

Smart Internz

- FJFO (first-in, first-out)

1. Strictly—preserved message ordering
2. Mostly-on-E message delivery
3. High throughput, up to 300 publishes/second
4. Subscription protocol: SQS

@ Standard

Best-effort message ordering at-least once message delivery

Highest throughput in publishes/second Subscription protocols: SQS, Lambda, HTTP, S3, email, mobile application endpoints


Name

BookRequeStNotificatiDr6

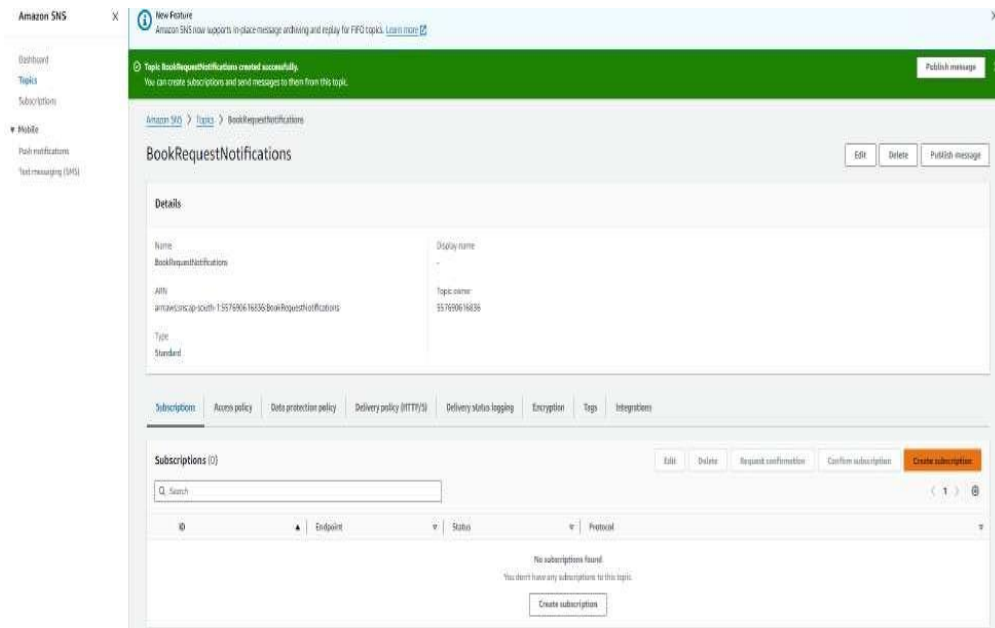
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores M.

Display name - optional [lrrfo](#)

To use this topic with 5M5 subscriptions, enter a display name. Only the first 10 characters are displayed in an Si:S message.  
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.

Configure the SNS topic and note down the Topic ARN.



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

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

Amazon SNS > Subscriptions > Create subscription

## Create subscription

**Details**

Topic ARN

Protocol  
The type of endpoint to subscribe

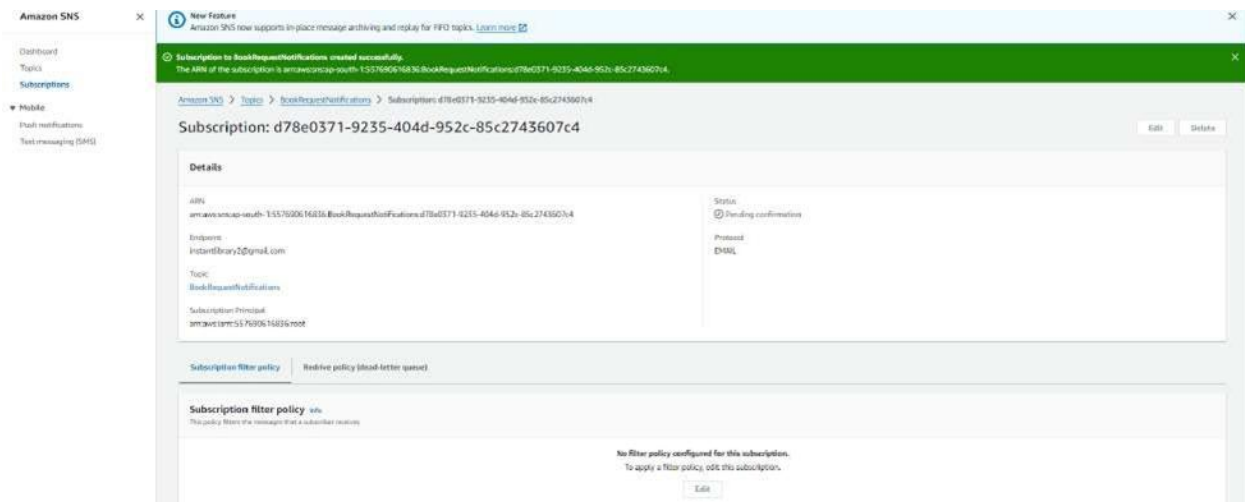
Endpoint  
An email address that can receive notifications from Amazon SNS.

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

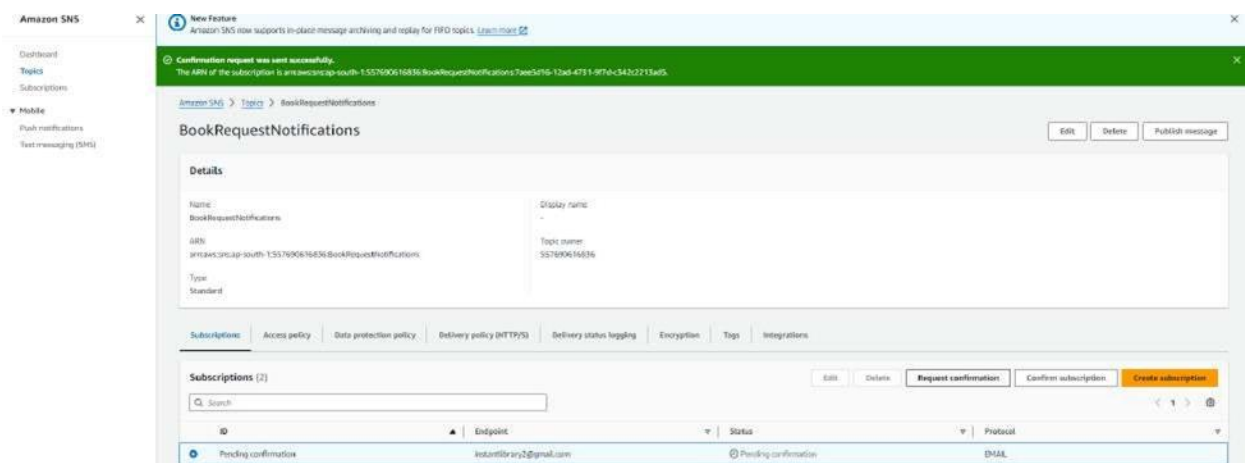
► **Subscription filter policy - optional** [Info](#)  
This policy filters the messages that a subscriber receives.

► **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



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

## AWS Notification – Subscription Confirmation Inbox x

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

9

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 [sns-opt-out](#)

**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 [sns-opt-out](#)



Simple Notification Service

### 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](#).

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

Amazon SNS > Topics > BookRequestNotifications

### BookRequestNotifications

Edit Delete Publish message

**Details**

Name  
BookRequestNotifications

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

Type  
Standard

Display name  
▼

Topic owner  
557690616836

Subscriptions

Access policy Data protection policy Delivery policy (HTTPS) Delivery status logging Encryption Topic Integrations

Subscriptions (2)

Edit Delete Request confirmation Confirm subscription Create subscription

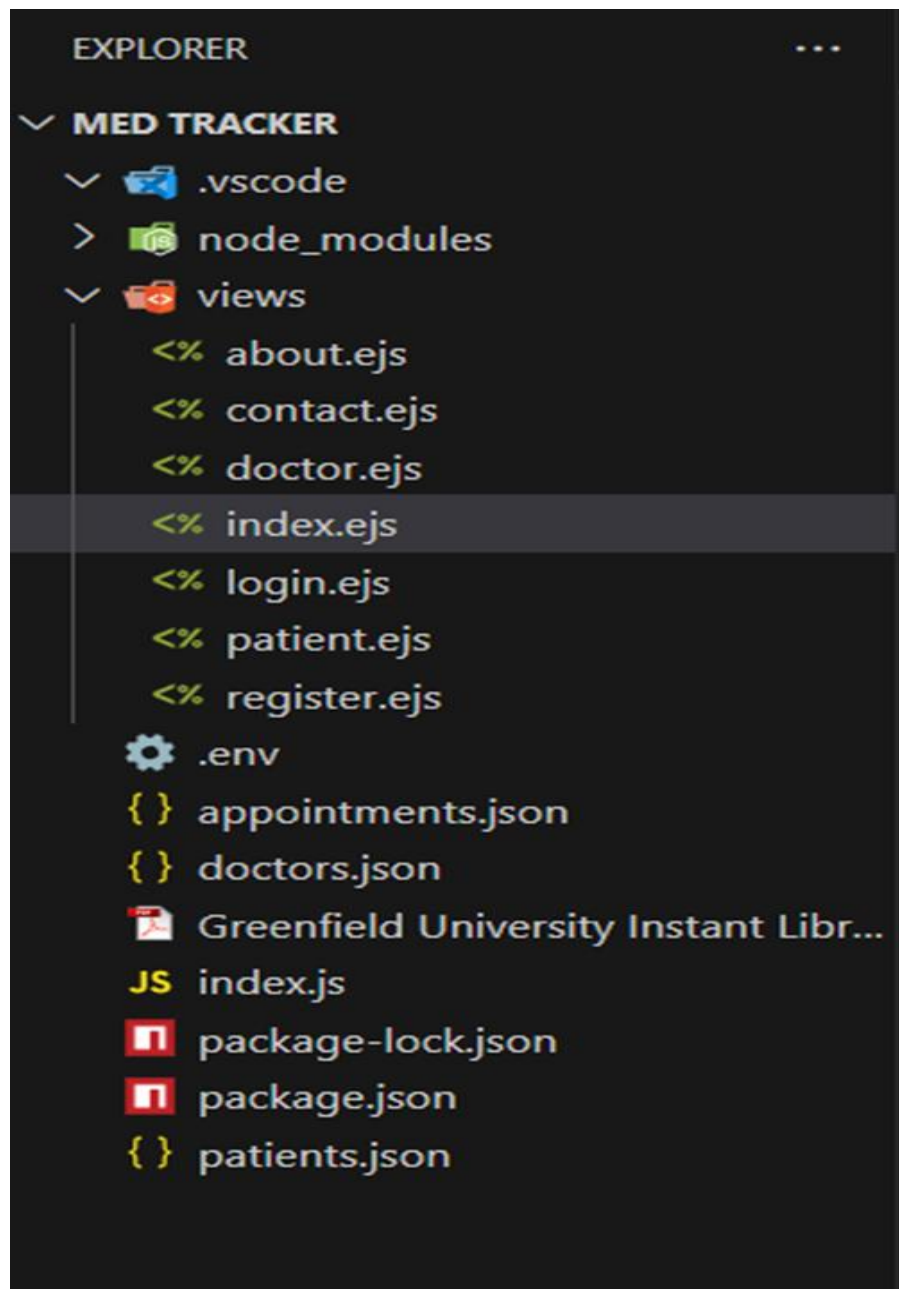
ID	Endpoint	Status	Protocol
d78e0371-9235-404d-952c-85c2743607c4	instantboxy2@gmail.com	Confirmed	EMAIL

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

#### Milestone 4: Backend Development and Application Setup

##### Activity 4.1: Develop the backend using JavaScript

###### File Explorer Structure



**Description:** set up the INSTANT LIBRARY project with an index.js file, a public/ folder for assets, and a views/ directory containing all required HTML pages like home, login, register,



subject-specific pages (e.g., computer\_science.html, data\_science.html), and utility pages (e.g., request-form.html, statistics.html).

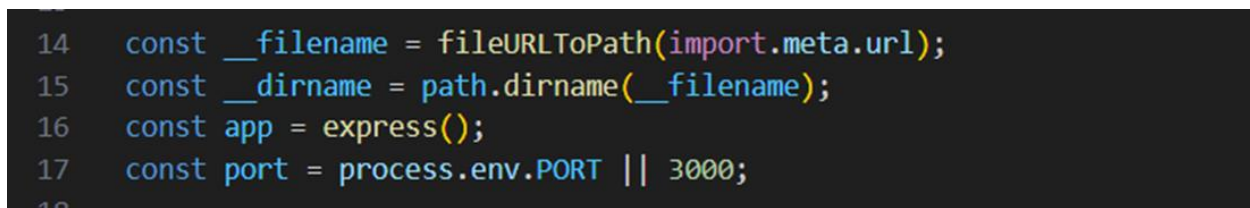
Description of the code :

Node js ( Express ) Initialization



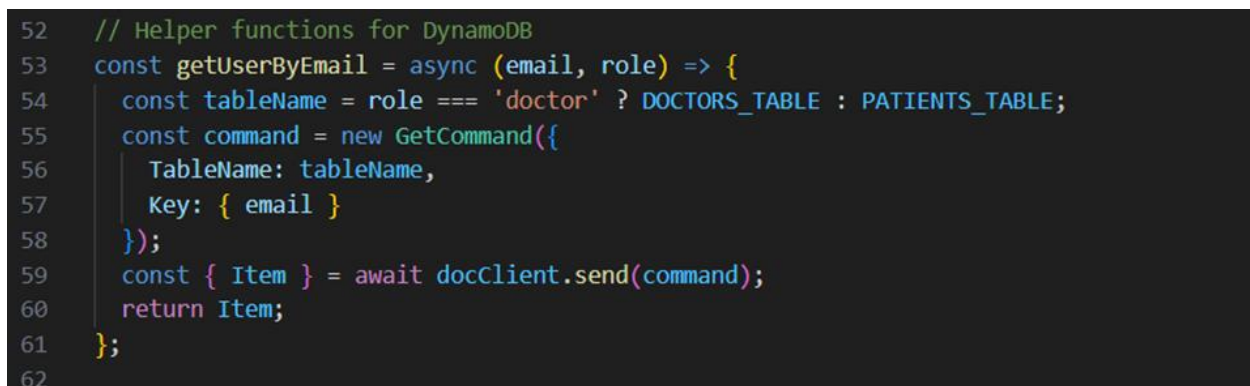
```
JS index.js x .env <% index.ejs
JS index.js > app.get("/") callback
1  import express from 'express';
2  import path from 'path';
3  import { fileURLToPath } from 'url';
4  import bcrypt from 'bcrypt';
5  import session from 'express-session';
6  import dotenv from 'dotenv';
7  import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
8  import { SNSClient, PublishCommand } from '@aws-sdk/client-sns';
9  import { DynamoDBDocumentClient, GetCommand, PutCommand, QueryCommand, UpdateCommand } from '@aws-sdk/lib-dynamodb';
10
```

Import essential libraries including Express utilities for routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification.



```
14  const __filename = fileURLToPath(import.meta.url);
15  const __dirname = path.dirname(__filename);
16  const app = express();
17  const port = process.env.PORT || 3000;
18
```

Initialize the Flask application instance using \_\_filename to get the path of the directory and app is used to get express to start building the web app.



```
52  // Helper functions for DynamoDB
53  const getUserByEmail = async (email, role) => {
54    const tableName = role === 'doctor' ? DOCTORS_TABLE : PATIENTS_TABLE;
55    const command = new GetCommand({
56      TableName: tableName,
57      Key: { email }
58    });
59    const { Item } = await docClient.send(command);
60    return Item;
61  };
62
```

Dynamo DB Setup:



```

52 // Helper functions for DynamoDB
53 const getUserByEmail = async (email, role) => {
54   const tableName = role === 'doctor' ? DOCTORS_TABLE : PATIENTS_TABLE;
55   const command = new GetCommand({
56     TableName: tableName,
57     Key: { email }
58   });
59   const { Item } = await docClient.send(command);
60   return Item;
61 };
62

```

Initialize the Dynamo DB resource for the ap-south-1 region and set up access to the Users and Requests tables for storing user details and book requests.

### 1. SNS Connection

```

214 // Send SNS notification
215 if (SNS_TOPIC_ARN) {
216   const snsCommand = new PublishCommand({
217     TopicArn: SNS_TOPIC_ARN,
218     Message: `New doctor registered: ${firstName} ${lastName} (${email}) - ${specialization}`,
219     Subject: 'MedTrack Registration'
220   });
221   await snsClient.send(snsCommand).catch(err => console.error('SNS Error:', err));
222 }
223
224 res.redirect('/login');
225 });
226

```

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.

### 2. Routes for Web Pages

Home Route:

```

140 app.get("/", (req, res) => res.render("index"));
141 app.get("/contactus", (req, res) => res.render("contactus"));
142 app.get("/about", (req, res) => res.render("about"));
143 app.get("/register", (req, res) => res.render("register"));
144

```

Define the home route / to automatically redirect users to the register page when they access the base URL.

#### 1. Register Route:

```

61 app.post('/register/patient', async (req, res) => {
62   const { firstName, lastName, dob, gender, email, phone, address, password } = req.body;
63
64   // Check if patient exists
65   const existingPatient = await getUserByEmail(email, 'patient');
66   if (existingPatient) return res.send('Patient exists');
67
68   // Create new patient
69   await createUser({
70     id: Date.now().toString(),
71     name: `${firstName} ${lastName}`,
72     dob,
73     gender,
74     email,
75     phone,
76     address,
77     password: await bcrypt.hash(password, 10)
78   }, 'patient');
79
80   // Send SNS notification
81   if (SNS_TOPIC_ARN) {
82     const snsCommand = new PublishCommand({
83       TopicArn: SNS_TOPIC_ARN,
84       Message: `New patient registered: ${firstName} ${lastName} (${email})`,
85       Subject: 'MedTrack Registration'
86     });
87     await snsClient.send(snsCommand).catch(err => console.error('SNS Error:', err));
88   }
89
90   res.redirect('/login');
91 });
92

```

Define /register route to validate registration form fields, hash the user password using Bcrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

Login Route (GET/POST):

```

app.post('/check', async (req, res) => {
  const { email, password, role } = req.body;

  const user = await getUserByEmail(email, role);
  if (!user || !(await bcrypt.compare(password, user.password))) {
    return res.render('login', { message: 'Invalid credentials' });
  }

  req.session.user = {
    id: user.id,
    name: user.name,
    email: user.email,
    role: user.role
  };
  res.redirect(`/${role}`);
});

```

Define /login route to validate user credentials against DynamoDB, check the password using Bcrypt, update the login count on successful authentication, and redirect users to the home page.

1. Home, E- book buttons and subject routes:

```
// Appointment actions
app.post('/doctor/appointment/:id/precautions', requireDoctor, async (req, res) => {
  await updateAppointment(req.params.id, {
    precautions: req.body.precautions,
    status: 'Completed',
    updatedAt: new Date().toISOString()
  });
  res.redirect('/doctor');
});

app.post('/doctor/appointment/:id/reschedule', requireDoctor, async (req, res) => {
  await updateAppointment(req.params.id, {
    date: req.body.date,
    time: req.body.time,
    status: 'Rescheduled',
    updatedAt: new Date().toISOString()
  });
  res.redirect('/doctor');
});

app.post('/doctor/appointment/:id/cancel', requireDoctor, async (req, res) => {
  await updateAppointment(req.params.id, {
    status: 'Cancelled',
    updatedAt: new Date().toISOString()
  });
  res.redirect('/doctor');
});
```

Define /home-page to render the main homepage, /ebook-buttons to handle subject selection and redirection, and /<subject>.html dynamic route to render subject-specific pages.

1. Request Routes:

```
// Create appointment
await createAppointment({
  doctorId,
  doctorName: doctor.name,
  specialty: doctor.specialization,
  patientId: req.session.user.id,
  patientName: req.session.user.name,
  date,
  time,
  reason
});

res.redirect('/patient');
});
```

Define /request-form route to capture book request details from users, store the request in DynamoDB, send a thank-you email to the user, notify the admin, and confirm submission with a success message.

## 2.Exit Route:

```
app.get('/logout', (req, res) => {  
  req.session.destroy(() => res.redirect('/'));  
});
```

Define /exit route to render the exit.html page when the user chooses to leave or close the application.

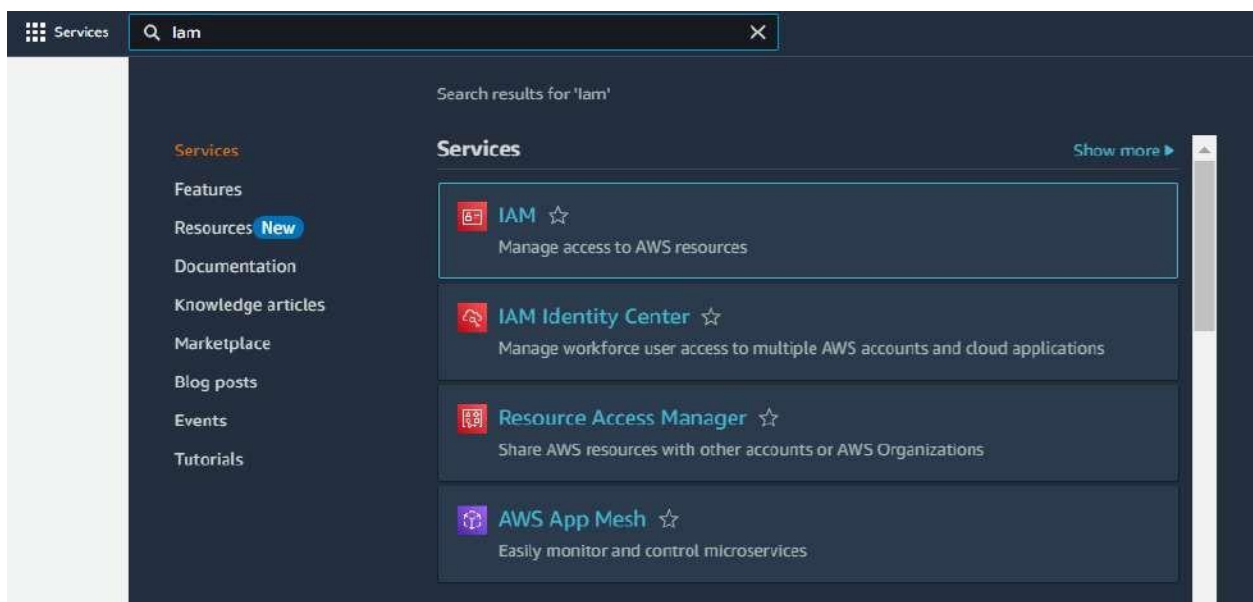
## Deployment Code:

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

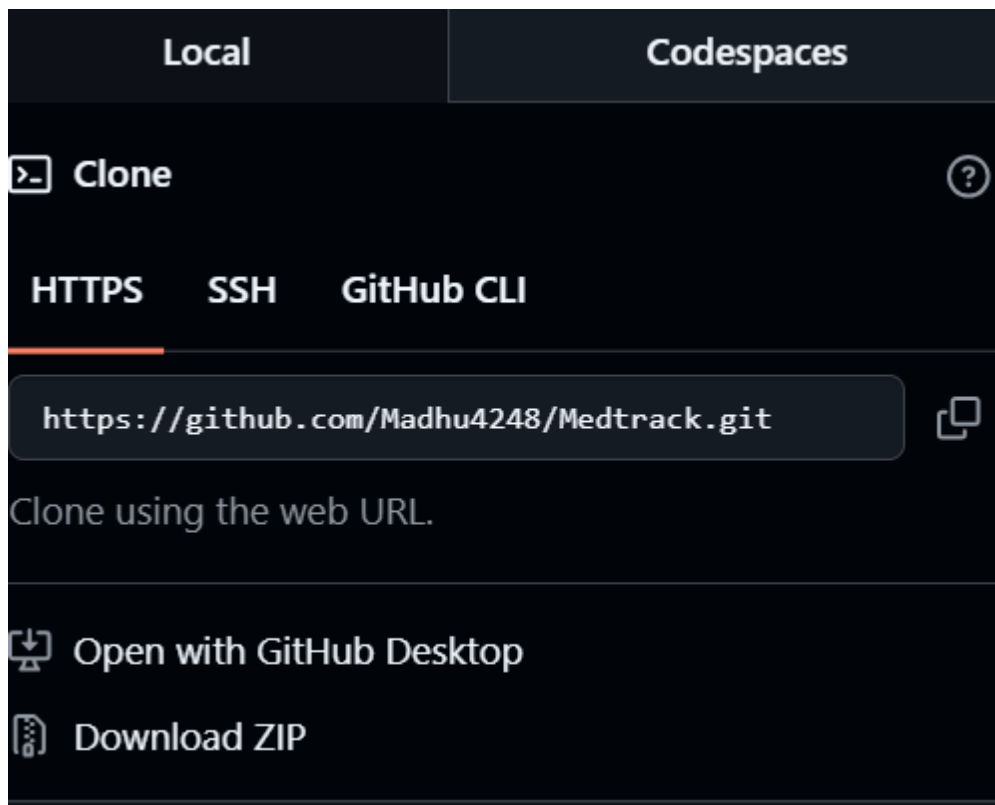
## Milestone 5: IAM Role Setup

### Activity 5.1: Create IAM Role.

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

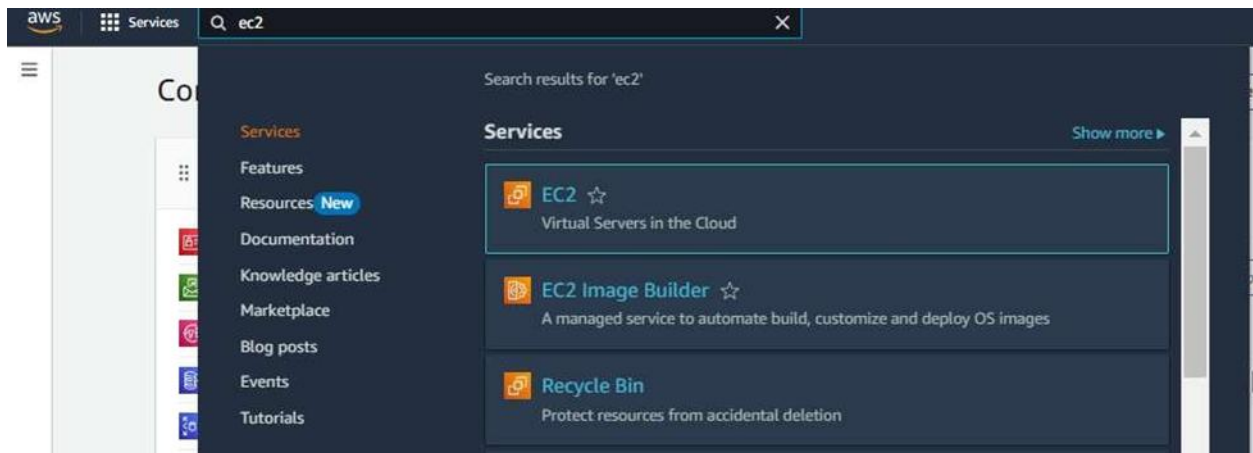






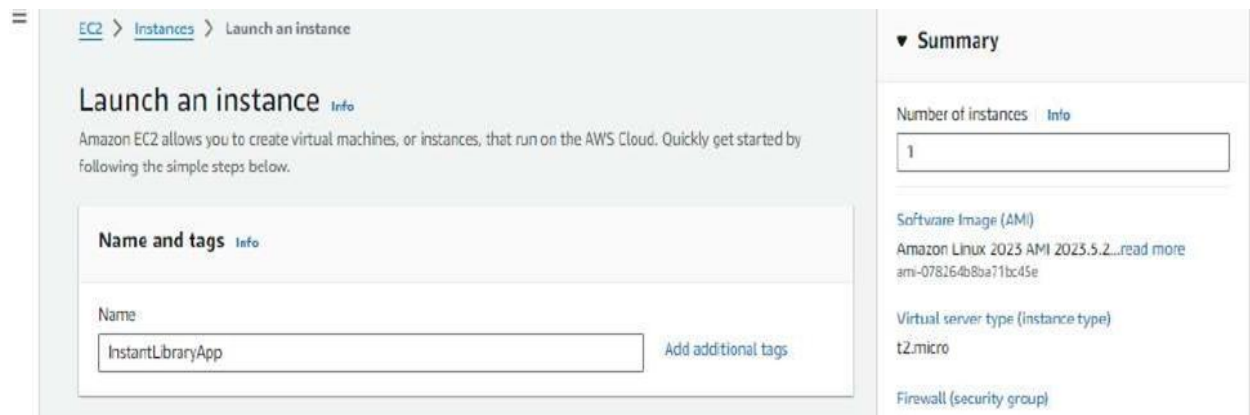
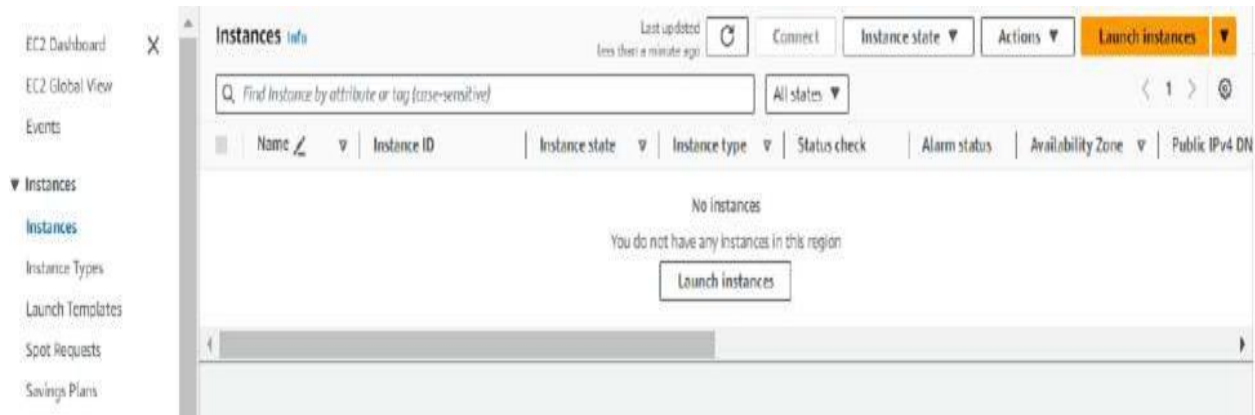
Activity 6.1: Launching EC2 instance to host the Flask application.

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

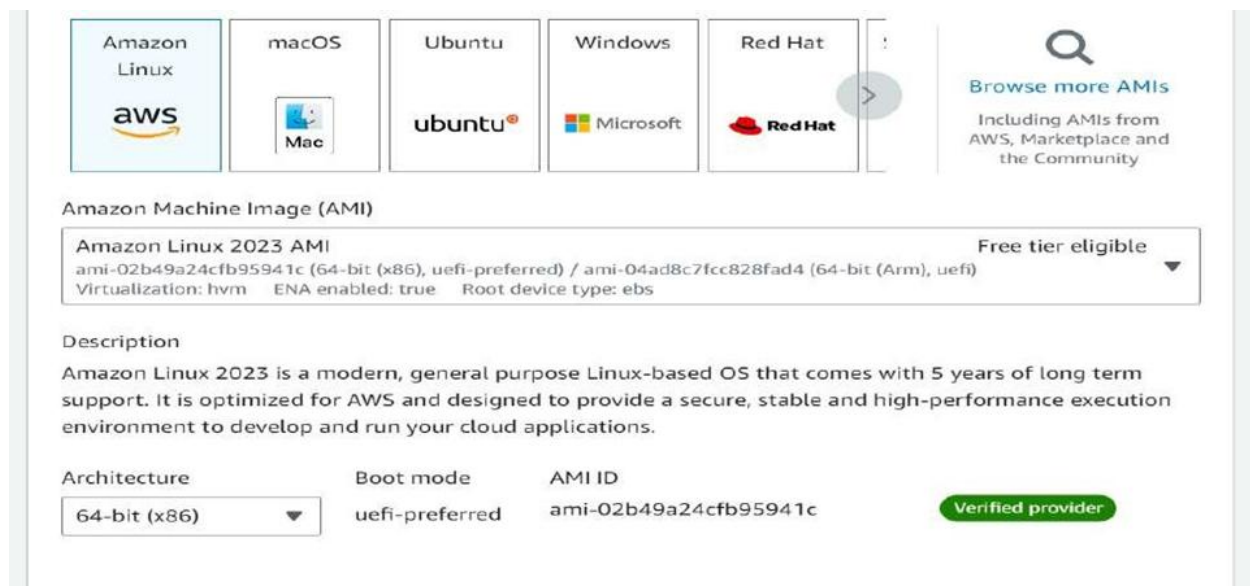


2. Click on Launch instance to launch EC2 instance





3. Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instancetype (free-tier eligible).



4. Create and download the key pair for Server access.

## ▼ 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](#)

## Create key pair



### Key pair name

Key pairs allow you to connect to your instance securely.

InstantLibrary

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

### Key pair type

☒ RSA

RSA encrypted private and public key pair

☐ ED25519

ED25519 encrypted private and public key pair

### Private key file format

☒ .pem

For use with OpenSSH

☐ .ppk

For use with PuTTY



When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** [Learn more](#)

Cancel

Create key pair



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

Architecture

64-bit (x86)

Boot mode

uefi-preferred

AMI ID

ami-078264b8ba71b

Number of instances

1

Software Image

Amazon Linux 2022

Instance type

m5.xlarge

Key pair (login)

Info

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

Key pair name - required

Free tier

In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

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 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

The image shows two screenshots from the AWS Management Console. The top screenshot displays the 'Inbound Security Group Rules' configuration page. It lists three rules:

- Security group rule 1 (TCP, 22, 0.0.0.0/0):** Type: ssh, Protocol: TCP, Port range: 22, Source type: Anywhere, Source: 0.0.0.0/0, Description: e.g. SSH for admin desktop.
- Security group rule 2 (TCP, 80, 0.0.0.0/0):** Type: HTTP, Protocol: TCP, Port range: 80, Source type: Custom, Source: 0.0.0.0/0, Description: e.g. SSH for admin desktop.
- Security group rule 3 (TCP, 5000, 0.0.0.0/0):** Type: Custom TCP, Protocol: TCP, Port range: 5000, Source type: Custom, Source: 0.0.0.0/0, Description: e.g. SSH for admin desktop.

The bottom screenshot shows the 'Launch an instance' completion screen. It displays a green success banner: 'Successfully initiated launch of instance (i-0018610225uc2190)'. Below this, there are 'Next Steps' and a grid of recommended actions:

- Create billing and free tier usage alerts
- Connect to your instance
- Connect an RDS database
- Create EBS snapshot policy
- Manage detailed monitoring
- Create Load Balancer
- Create AWS budget
- Manage CloudWatch alarms
- Disaster recovery for your instances
- Monitor for suspicious runtime activities
- Get instance screenshot
- 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.

Instances (1/2) info										
<div> <div>Find instance by attribute or tag (case-sensitive)</div> <div>All states</div> </div>										
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4	Elastic IP	IPv6 IPs
InstantLibraryApp	i-001861022fbcac290	Stopped	t2.micro	-	View alarms	ap-south-1b	-	-	-	-

EC2 > Instances > i-001861022fbcac290

Instance summary for i-001861022fbcac290 (InstantLibraryApp)

Updated less than a minute ago

Instance ID

i-001861022fbcac290

IPv6 address

-

Hostname type

IP name: ip-172-31-3-5-ap-south-1-compute.internal

Answer private resource DNS name

IPv4 (A)

Auto-assigned IP address

-

IAM Role

sns\_Dynamodb\_role

IMDSv2

Required

Public IPv4 address

-

Instance state

stopped

Private IP DNS name (IPv4 only)

ip-172-31-3-5-ap-south-1-compute.internal

Instance type

t2.micro

VPC ID

vpc-03c0c7b6f19a07211

Subnet ID

subnet-0d9fa2144480c92d9

Instance ARN

arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290

Private IPv4 addresses

172.31.3.5

Public IPv4 DNS

-

Elastic IP addresses

-

AWS Compute Optimizer finding

Opt-in to AWS Compute Optimizer for recommendations. Learn more

Auto Scaling Group name

-

Details

Status and alarms

Monitoring

Security

Networking

Storage

Tags

EC2 > Instances > i-001861022fbcac290

Instance summary for i-001861022fbcac290 (InstantLibraryApp)

Updated less than a minute ago

Instance ID

i-001861022fbcac290

IPv6 address

-

Hostname type

IP name: ip-172-31-3-5-ap-south-1-compute.internal

Answer private resource DNS name

IPv4 (A)

Auto-assigned IP address

-

IAM Role

sns\_Dynamodb\_role

IMDSv2

Required

Public IPv4 address

-

Instance state

Stopped

Private IP DNS name (IPv4 only)

ip-172-31-3-5-ap-south-1-compute.internal

Instance type

t2.micro

VPC ID

vpc-03c0c7b6f19a07211

Subnet ID

subnet-0d9fa2144480c92d9

Instance ARN

arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290

Private IPv4 addresses

172.31.3.5

Public IPv4 DNS

-

Elastic IP addresses

-

AWS Compute Optimizer finding

Opt-in to AWS Compute Optimizer for recommendations. Learn more

Auto Scaling Group name

-

Connect

Change security groups

Get Windows password

Modify IAM role

Connect

Manage instance state

Instance settings

Networking

Security

Image and templates


Monitor and troubleshoot

EC2 > Instances > i-001861022fbcac290 > Modify IAM role

## Modify IAM role Info

Attach an IAM role to your instance.


Instance ID

 i-001861022fbcac290 (InstantLibraryApp)

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.

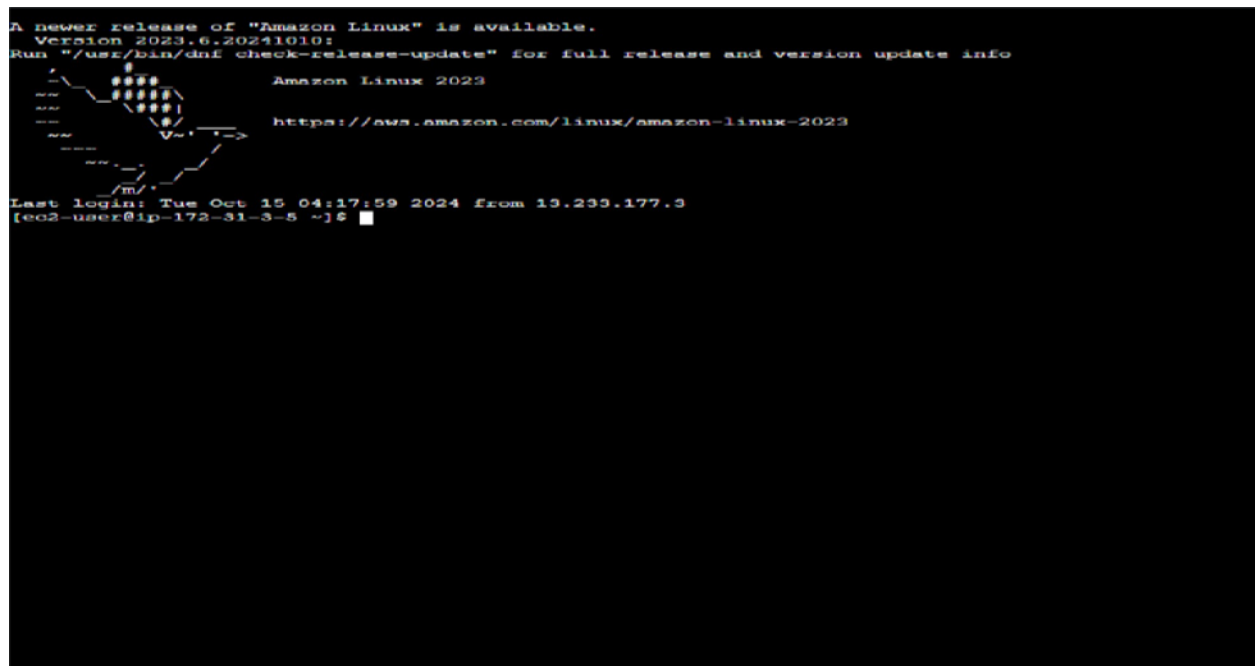
sns\_Dynamodb\_role

Create new IAM role 

Cancel

Update IAM role

Now connect the EC2 with the files



## Milestone 7: Deployment on EC2

### Activity 7.1: Install Software on the EC2 Instance

# Update system and install Node.js, npm, git

```
sudo yum update -y
```

```
curl -fsSL https://rpm.nodesource.com/setup_18.x | sudo bash -
```

```
sudo yum install -y nodejs git
```

### Activity 7.2: Clone Your Flask Project from GitHub

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

# Clone your Node.js project from GitHub

```
git clone https://github.com/prasannakumar133/MedTrack-repo.git
```

# Install project dependencies

```
npm install
```

Note: change your-github-username and your-repository-name with your

2. This will download your project to the EC2 instance.

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

```
cd MedTrack-repo
```

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

Run the Flask Application

# Install project dependencies

npm install

# Install PM2 globally to keep the app running

sudo npm install -g pm2

# Start the app using PM2 (replace 'app.js' with your entry point if different)

pm2 start app.js

pm2 save

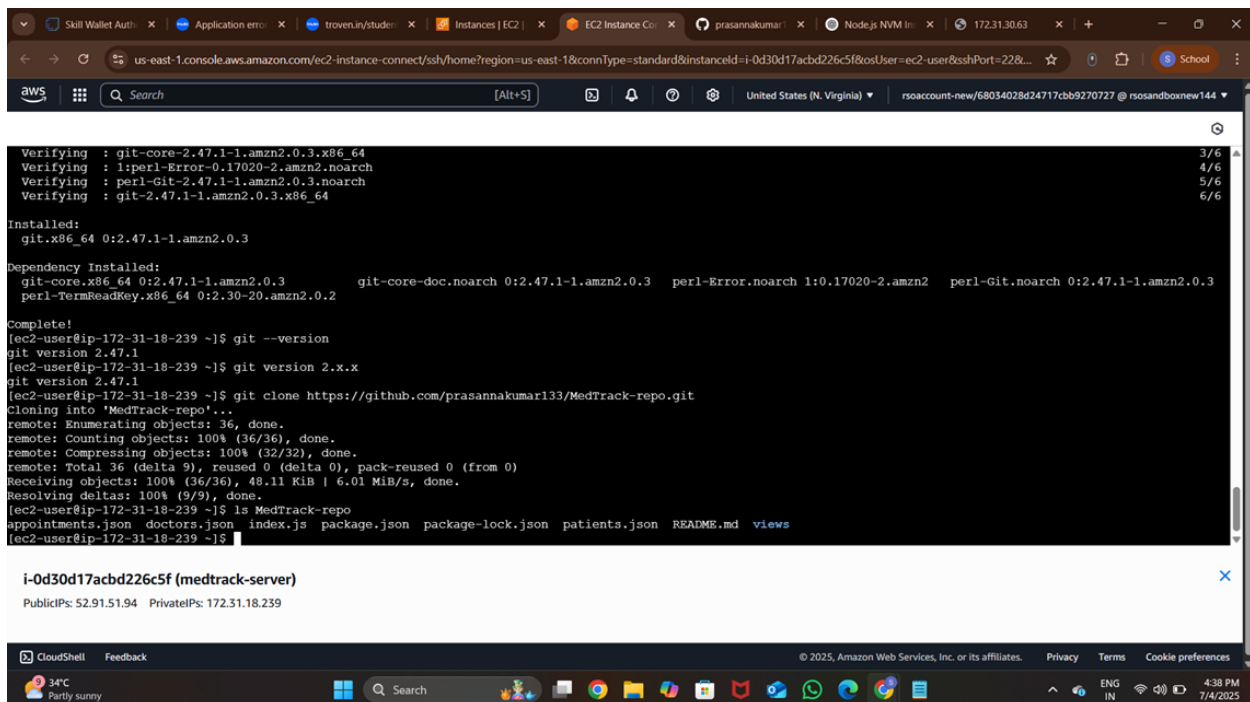
pm2 startup

# Copy and run the command shown by the previous line (for startup on reboot)

# (Optional) Allow the app port (3000) through firewall

sudo firewall-cmd --zone=public --permanent --add-port=3000/tcp

sudo firewall-cmd --reload



The screenshot shows a terminal window with the following output:

```
Verifying : git-core-2.47.1-1.amzn2.0.3.x86_64 3/6
Verifying : 1:perl-Error-0.17020-2.amzn2.noarch 4/6
Verifying : perl-Git-2.47.1-1.amzn2.0.3.noarch 5/6
Verifying : git-2.47.1-1.amzn2.0.3.x86_64 6/6

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

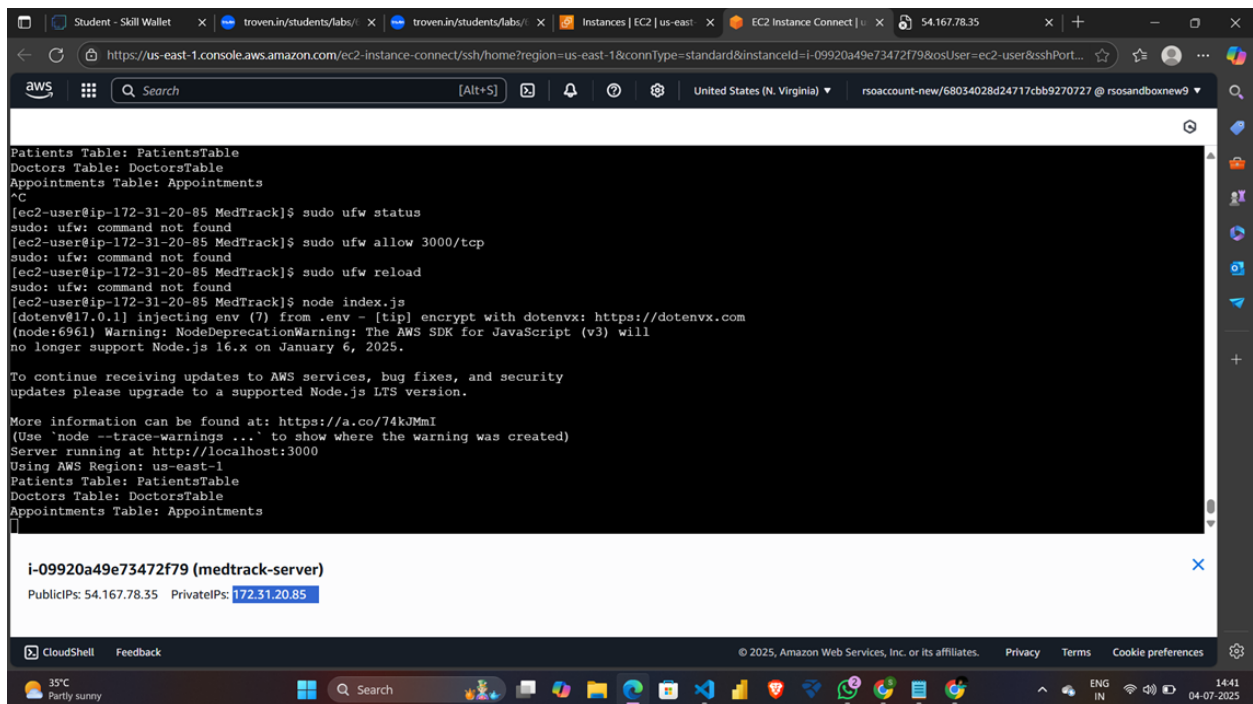
Dependency Installed:
git-core.x86_64 0:2.47.1-1.amzn2.0.3 git-core-doc.noarch 0:2.47.1-1.amzn2.0.3 perl-Error.noarch 1:0.17020-2.amzn2 perl-Git.noarch 0:2.47.1-1.amzn2.0.3
perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2

Complete!
[ec2-user@ip-172-31-18-239 ~]$ git --version
git version 2.47.1
[ec2-user@ip-172-31-18-239 ~]$ git version 2.x.x
git version 2.47.1
[ec2-user@ip-172-31-18-239 ~]$ git clone https://github.com/prasannakumar133/MedTrack-repo.git
Cloning into 'MedTrack-repo'...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (32/32), done.
remote: Total 36 (delta 9), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (36/36), 48.11 KiB | 6.01 MiB/s, done.
Resolving deltas: 100% (9/9), done.
[ec2-user@ip-172-31-18-239 ~]$ ls MedTrack-repo
appointments.json doctors.json index.js package.json package-lock.json patients.json README.md views
[ec2-user@ip-172-31-18-239 ~]$
```

Below the terminal output, the instance details are shown:

i-0d30d17acbd226c5f (medtrack-server)  
PublicIPs: 52.91.51.94 PrivateIPs: 172.31.18.239

Verify the Nodejs app is running: Done! Your app is running at <http://<your-ec2-public ip>:3000>  
Run the Nodejs app on the EC2 instance.



Access the website through:

Public IPs: <http://52.91.51.94:3000/>

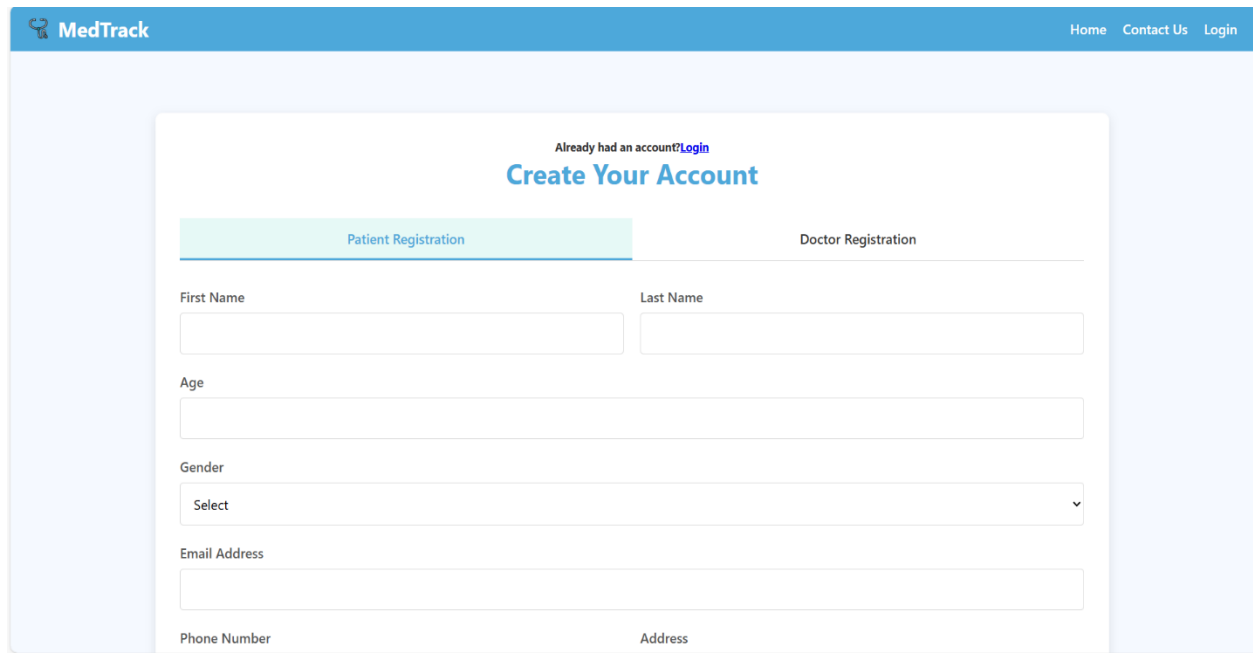
## Milestone 8: Testing and Deployment

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

Login Page:

The image shows a screenshot of the MedTrack login page. The page has a blue header with the MedTrack logo and navigation links: About, Contact Us, Register, Home. The main content area is light blue and contains a white login form. The form has two tabs: 'Patient Login' (selected) and 'Doctor Login'. Below the tabs are two input fields: 'Email' and 'Password'. Below these fields is a blue button labeled 'Login as Patient'. At the bottom of the form, there is a link: 'Don't have an account? Signup'. The footer of the page is blue and contains the text: '© 2025 MedTrack. All rights reserved.'

Register Page:



The register page features a blue header with the MedTrack logo and navigation links: Home, Contact Us, and Login. The main content area is a white card with the heading "Create Your Account" and a link for users who already have an account. Below the heading are two tabs: "Patient Registration" (active) and "Doctor Registration". The form includes input fields for First Name, Last Name, Age, Email Address, Phone Number, and Address. A gender dropdown menu is also present.

Already had an account? [Login](#)

## Create Your Account

Patient Registration

Doctor Registration

First Name

Last Name

Age

Gender 

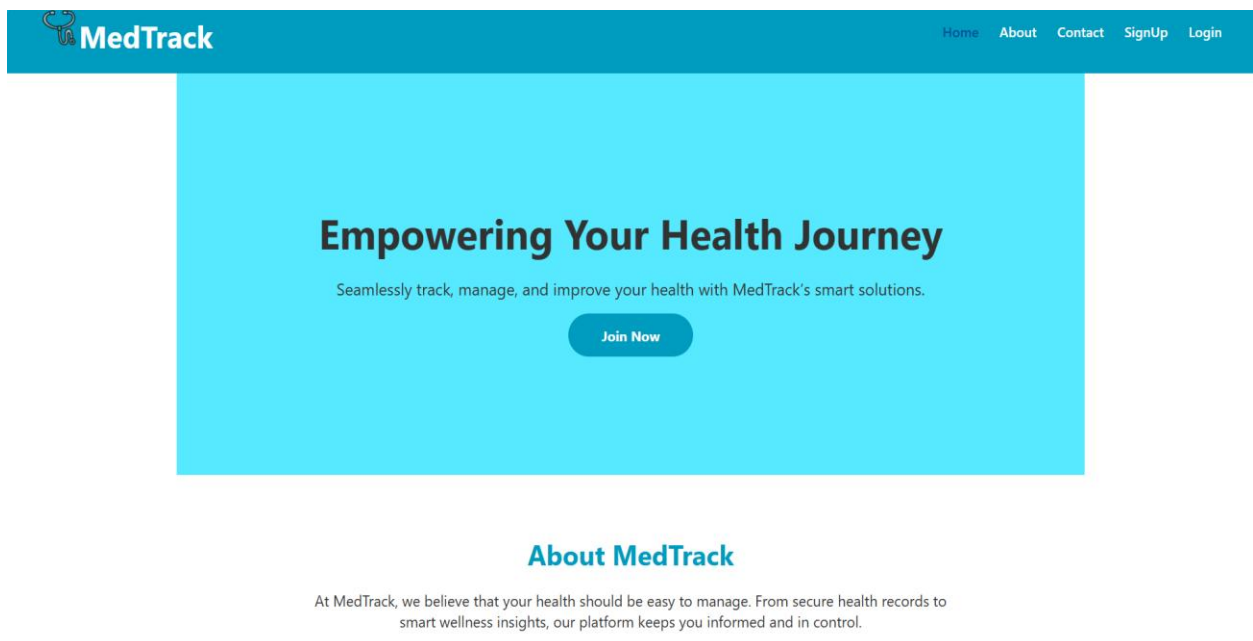
Select

Email Address

Phone Number

Address

Home page:



The home page has a blue header with the MedTrack logo and navigation links: Home, About, Contact, SignUp, and Login. The main content area is a large blue rectangle with the heading "Empowering Your Health Journey" and a subheading "Seamlessly track, manage, and improve your health with MedTrack's smart solutions." Below this is a "Join Now" button. At the bottom, there is an "About MedTrack" section with a paragraph about the platform's mission.

Home About Contact SignUp Login

## Empowering Your Health Journey

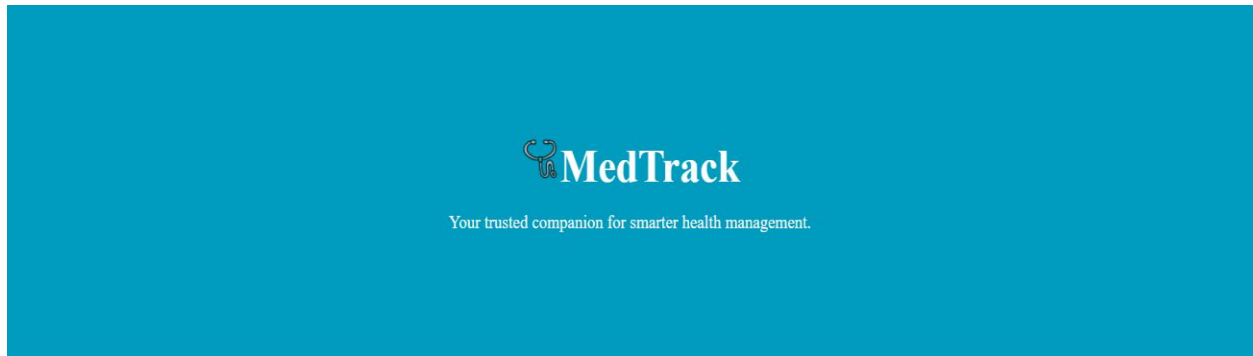
Seamlessly track, manage, and improve your health with MedTrack's smart solutions.

[Join Now](#)

### About MedTrack

At MedTrack, we believe that your health should be easy to manage. From secure health records to smart wellness insights, our platform keeps you informed and in control.

About page:



## Our Story

Founded with a vision to simplify healthcare management, MedTrack empowers you to take control of your well-being with intelligent tools and secure access to your health data. Our journey began with a simple goal — to make health tracking effortless, accurate, and accessible to everyone.

## Our Mission

We aim to bridge the gap between you and your healthcare providers, ensuring that every piece of medical information is right at your fingertips, whenever you need it.

Contact Page:

The image shows a contact form on a website. At the top, there is a blue header bar with the MedTrack logo on the left, the tagline "Your Health , Our Main Priority." in the center, and links for "Home", "Signup", and "Login" on the right. Below the header, the contact form is centered on a light blue background. The form has a white background and a rounded border. It is titled "Contact Us" in blue. It contains four input fields: "Name" with placeholder text "Your name", "Email" with placeholder text "Your email", "Subject" with placeholder text "Write your reason here...", and "Message" with placeholder text "Write your message here...". Each field has a small icon in the bottom right corner. At the bottom of the form is a blue button labeled "Send Message".



## **Conclusion:**

MedTrack illustrates the power of leveraging AWS cloud services to build a modern, reliable healthcare management system. By combining Amazon EC2 for scalable application hosting, DynamoDB for fast and flexible data storage, SNS for instant notifications, and IAM for fine-grained access control, MedTrack ensures secure, efficient, and responsive healthcare operations.

The system streamlines the entire patient care process—from appointment scheduling and telemedicine consultations to emergency access to records and automated alerts—reducing administrative overhead and improving the patient experience. With high availability, data encryption, and compliance-ready infrastructure, MedTrack empowers healthcare providers to focus on delivering quality care while maintaining trust and data security.

Overall, this project demonstrates how cloud-native solutions can transform traditional healthcare environments into agile, scalable, and patient-centric ecosystems.