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

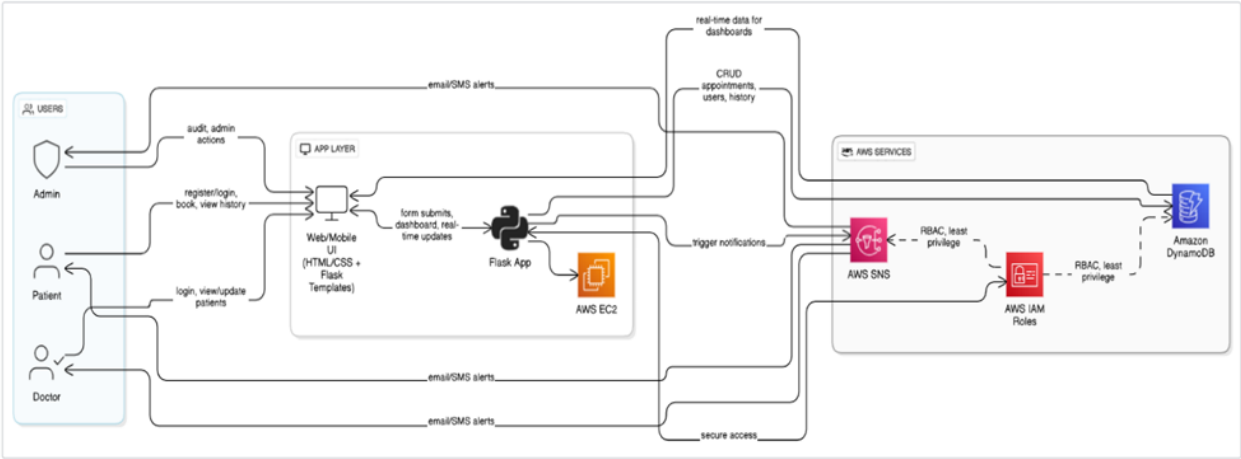
A patient login into the MedTrack web app, which is hosted on Amazon EC2 instances. They schedule an appointment, and the booking details are saved in Amazon DynamoDB, enabling fast retrieval and updates. Immediately, AWS SNS sends a confirmation SMS and email to the patient with appointment details.

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.

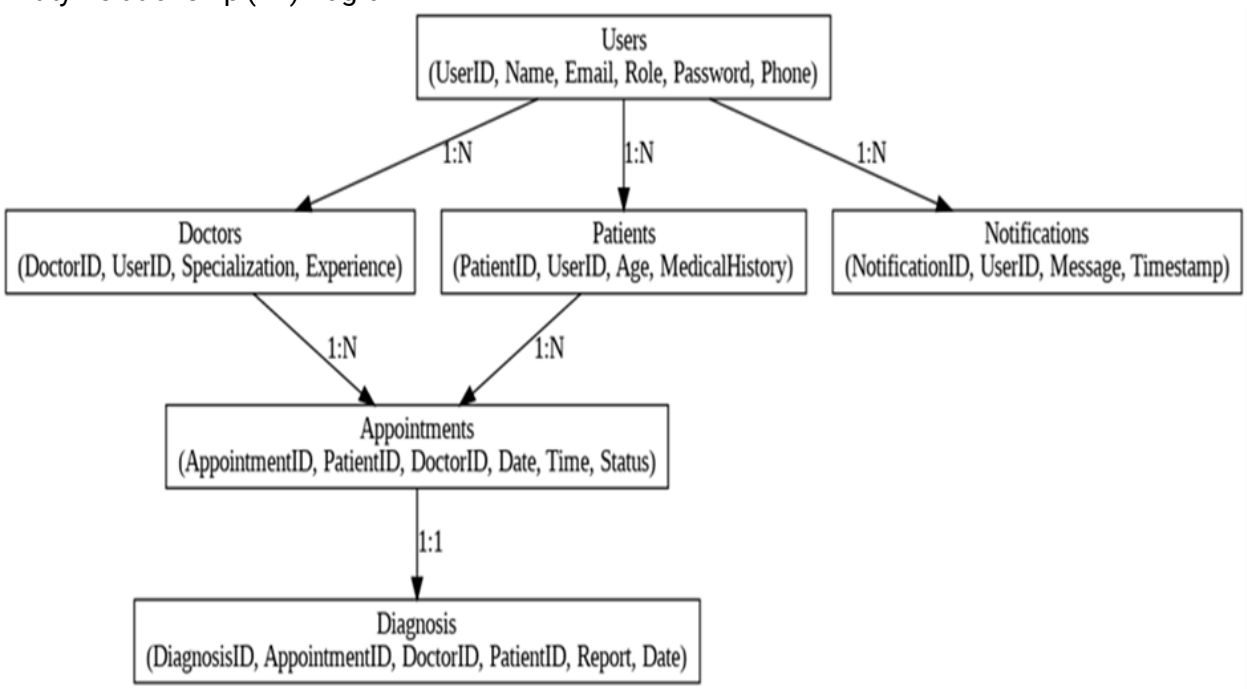
Scenario 3 – Emergency Alerts and Notifications

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.

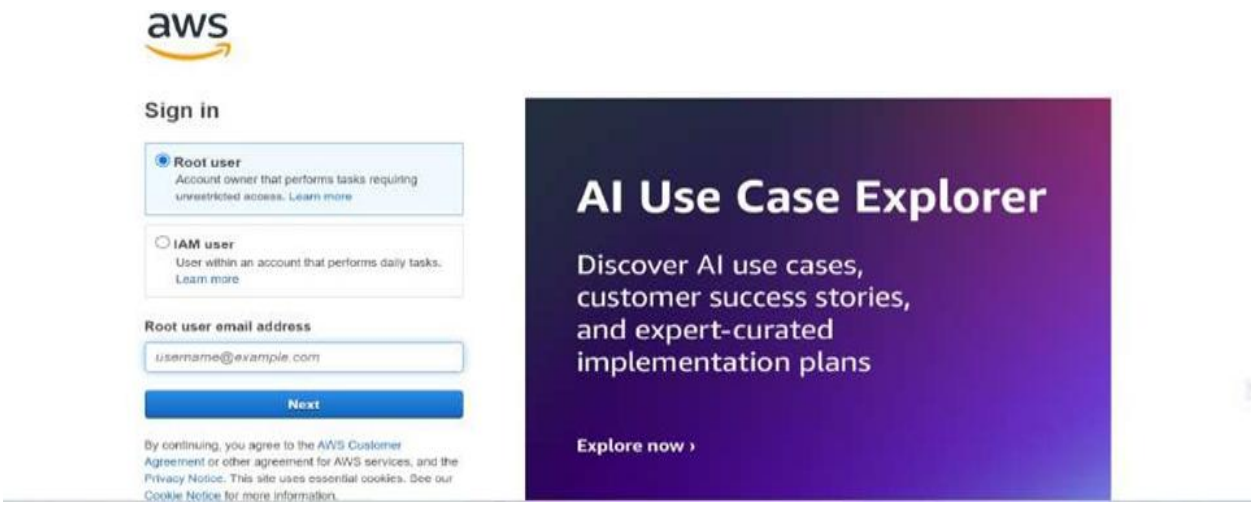
Milestone 1: AWS Account Setup and Login

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

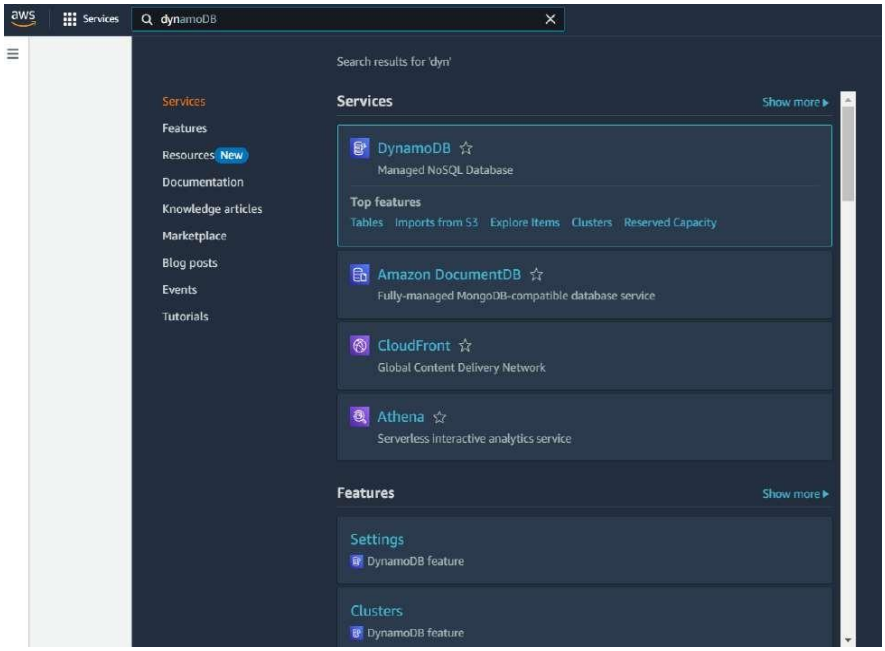
After setting up your account, log in to the AWS Management Console.

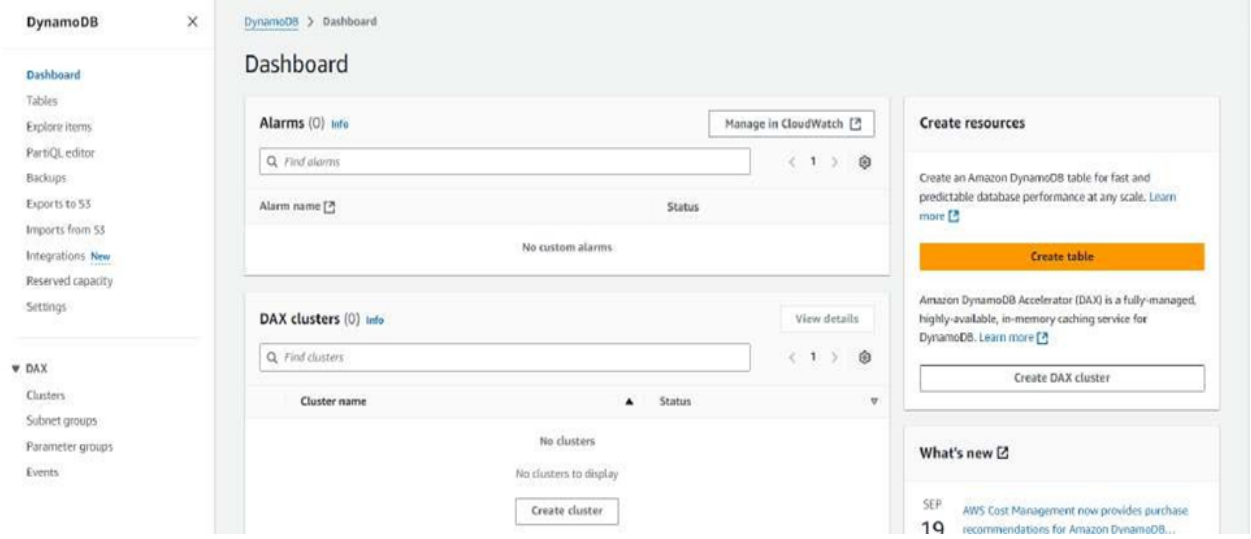


Milestone 2: Dynamo DB Database Creation and Setup

Activity 2.1: Navigate to the DynamoDB

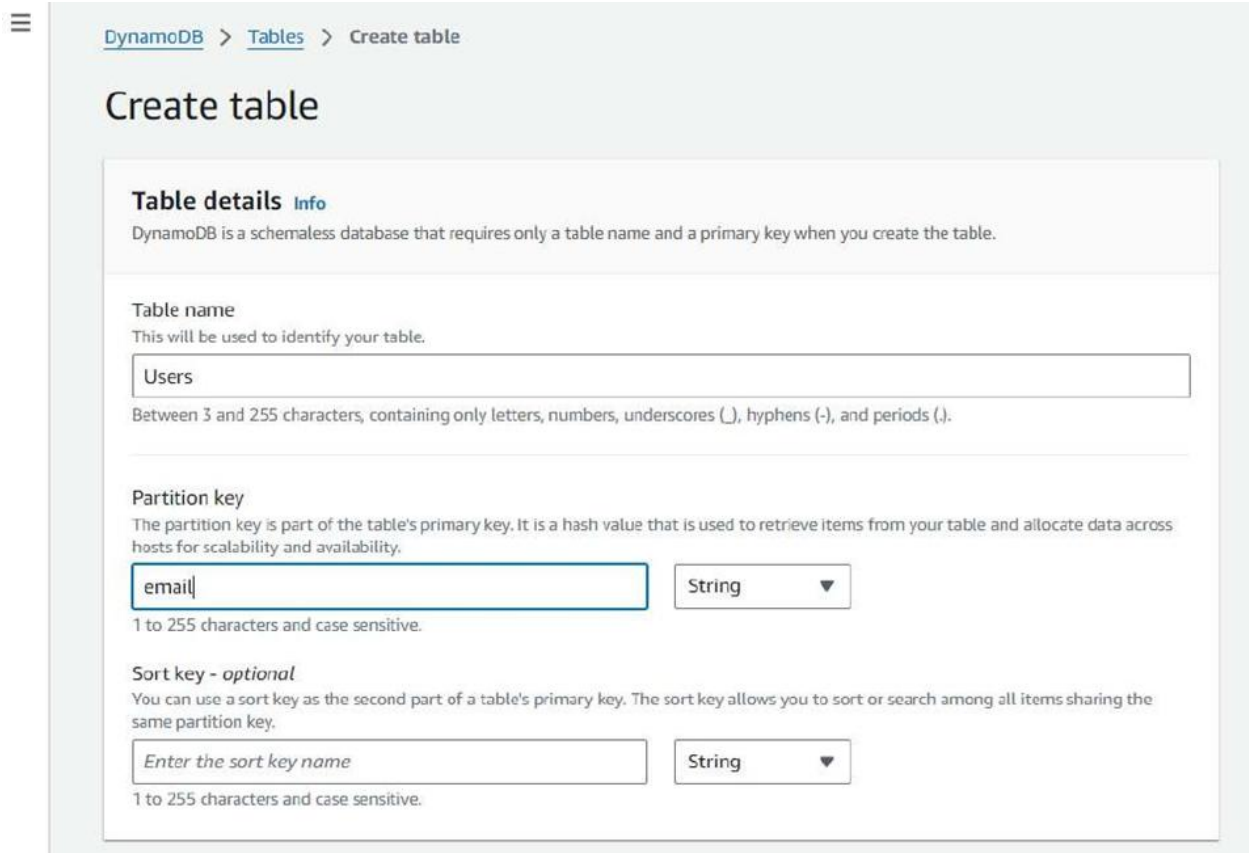
In the AWS Console, navigate to DynamoDB and click on create tables.





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.



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

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 toidentify your table.

Smart Internz

Requests

Between 3 and 2S5 characters, curyLaJning unly letters, rJumLt s, uryderscores , hmheris (-[, and p@iods (.).

Partition key

The partition key ispart of the table's primar'/ key. It isa hash value that is used to retrieve items from your table and allocate data across

email

1 ta 255 characters and case sensitive.

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

Enter the sort key nome

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

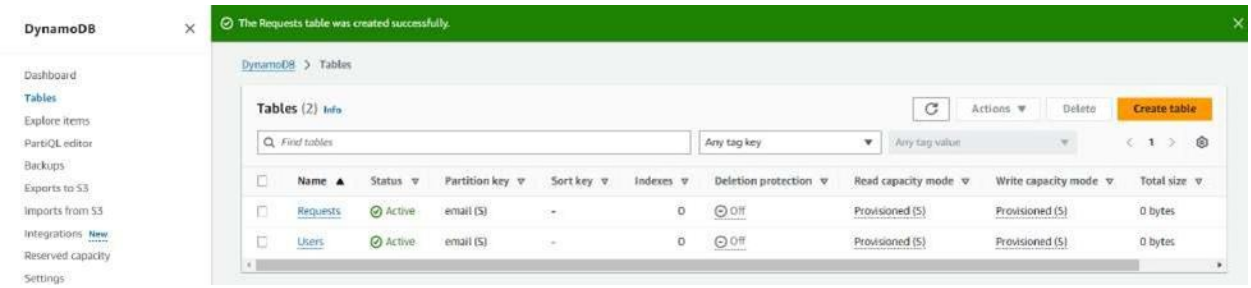
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

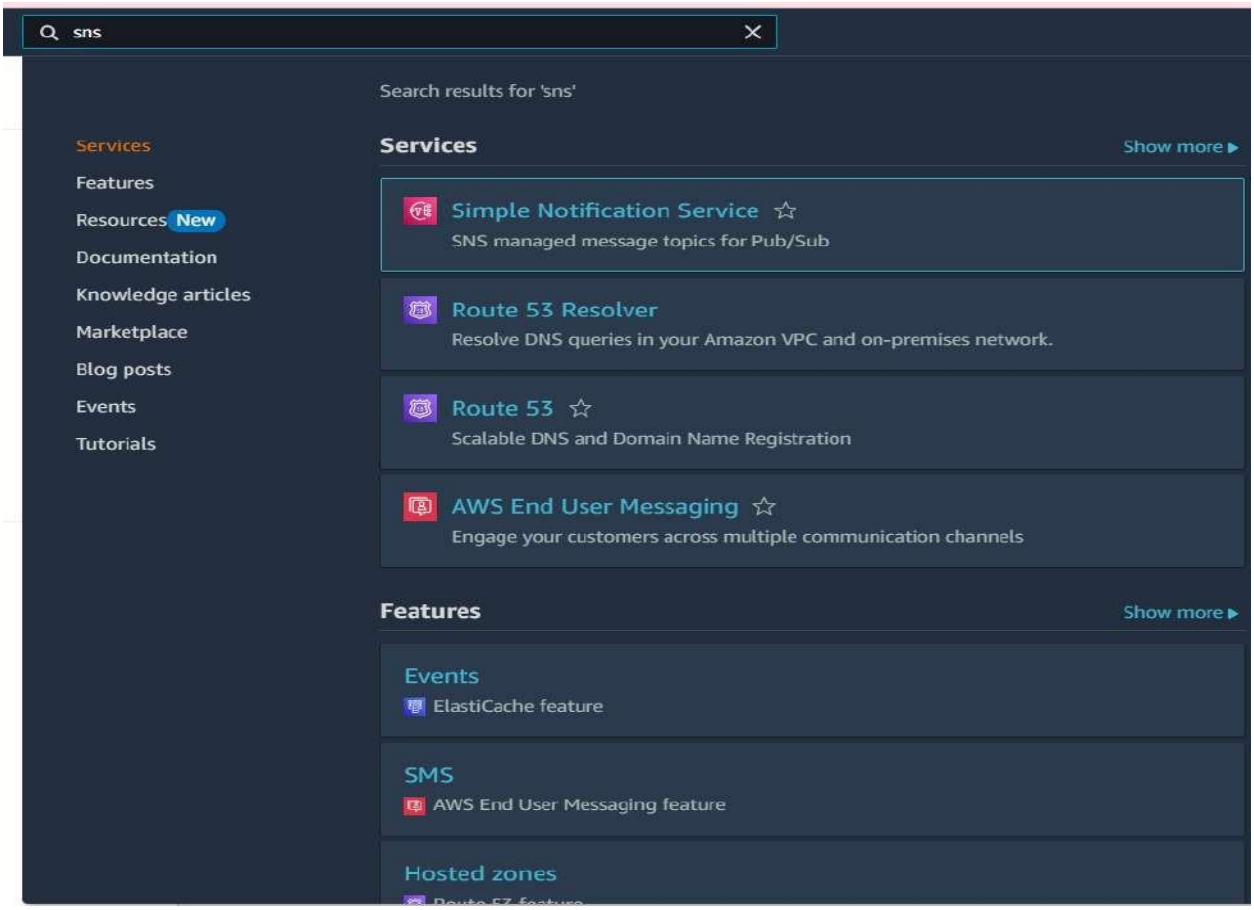
You can add 50 more tags.

Cancel Create table

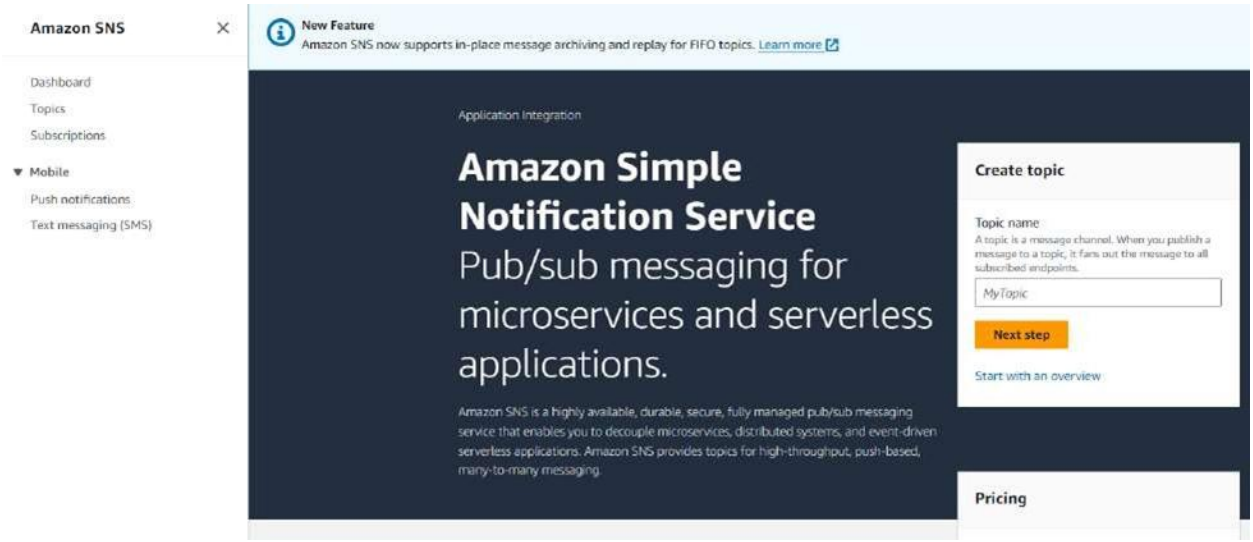


Milestone 3: SNS Notification Setup

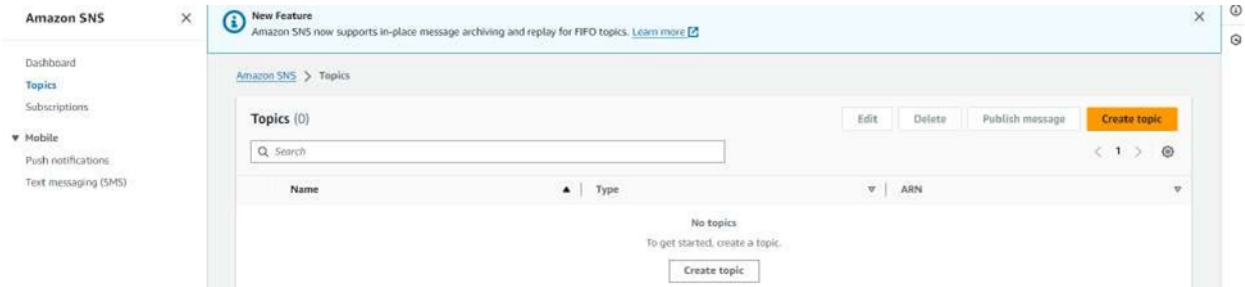
Activity 3.1: Create SNS topics for sending email notifications to users and library staff.



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 5N5 \ Topics Createtopic

Create topic

Details

Type trrfo

Topic type cannot be modified atter topic is created

O FJFO (first-in, first-out)

1. Strictly—preserved message ordering
2. Mactly-onEe message delivery
3. High throughput, up to ZOO publishes/second
4. Subscription protocol: SQS

@ Standard

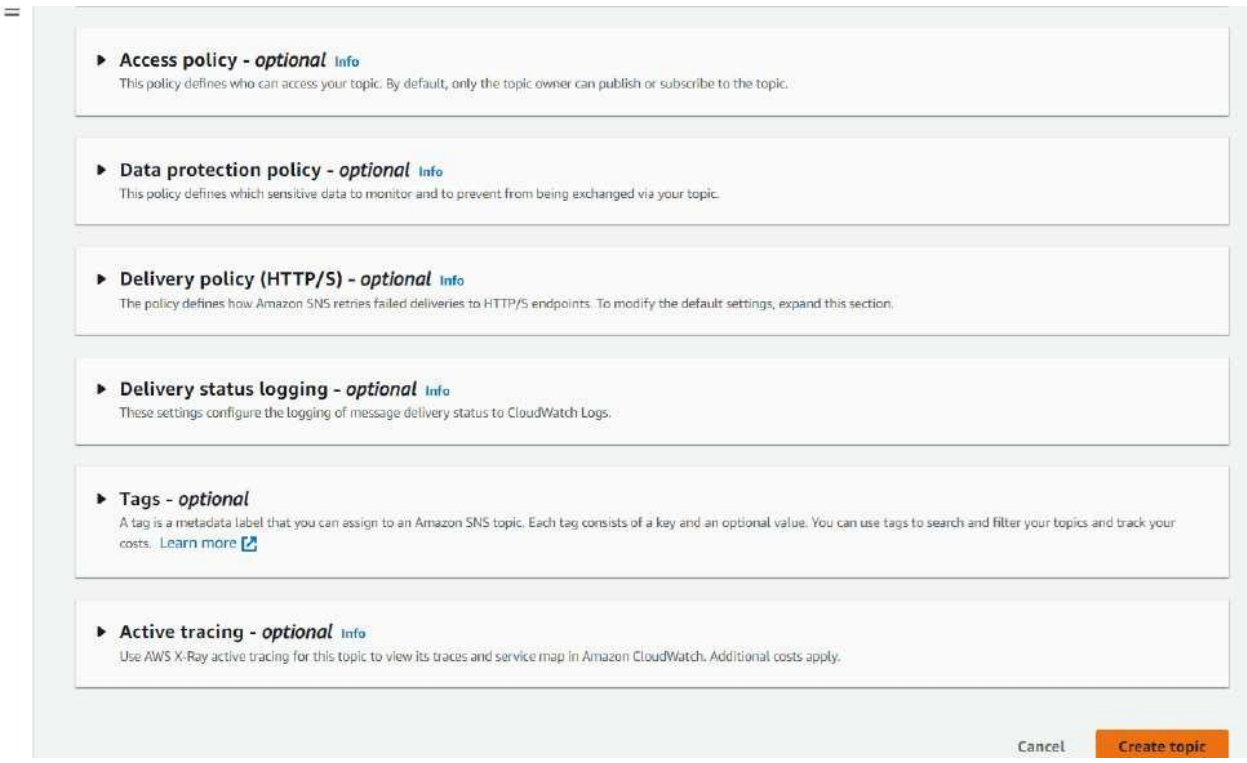
Best-effcrt message ordering at-least once message delivery
Highest throughput in publishes/second Subscription protocols: SQS, Lambda, HTTP, SI IS, email, mobile application endpoints

Name
BookRequeStNotificatiDr6

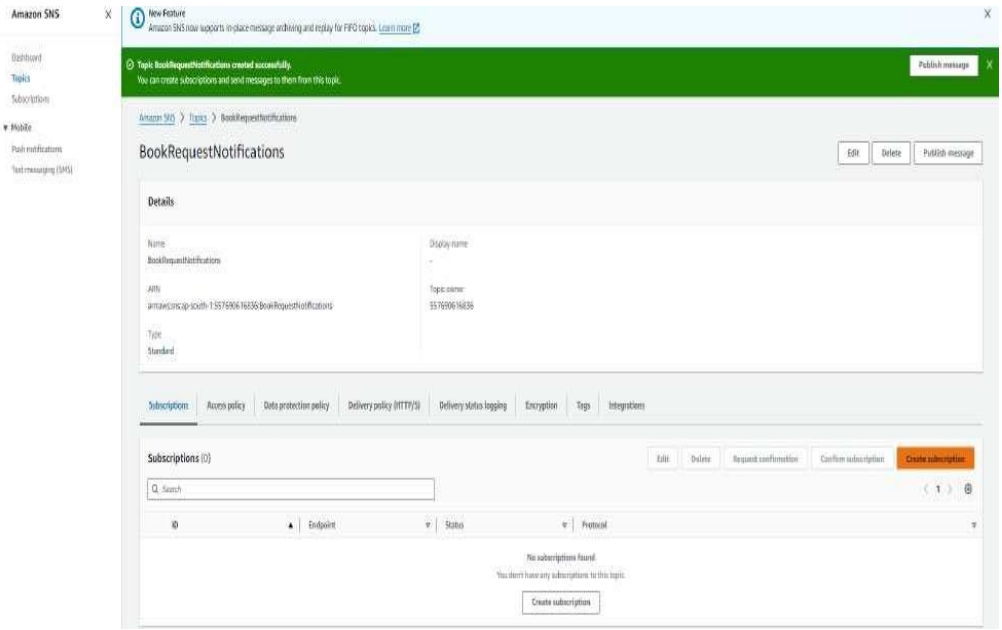
Maximum 256 characters. Can include alphanurr+eric characters, hyphens (-) and underscores M.

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 iS message. Maximum 1 00 characters

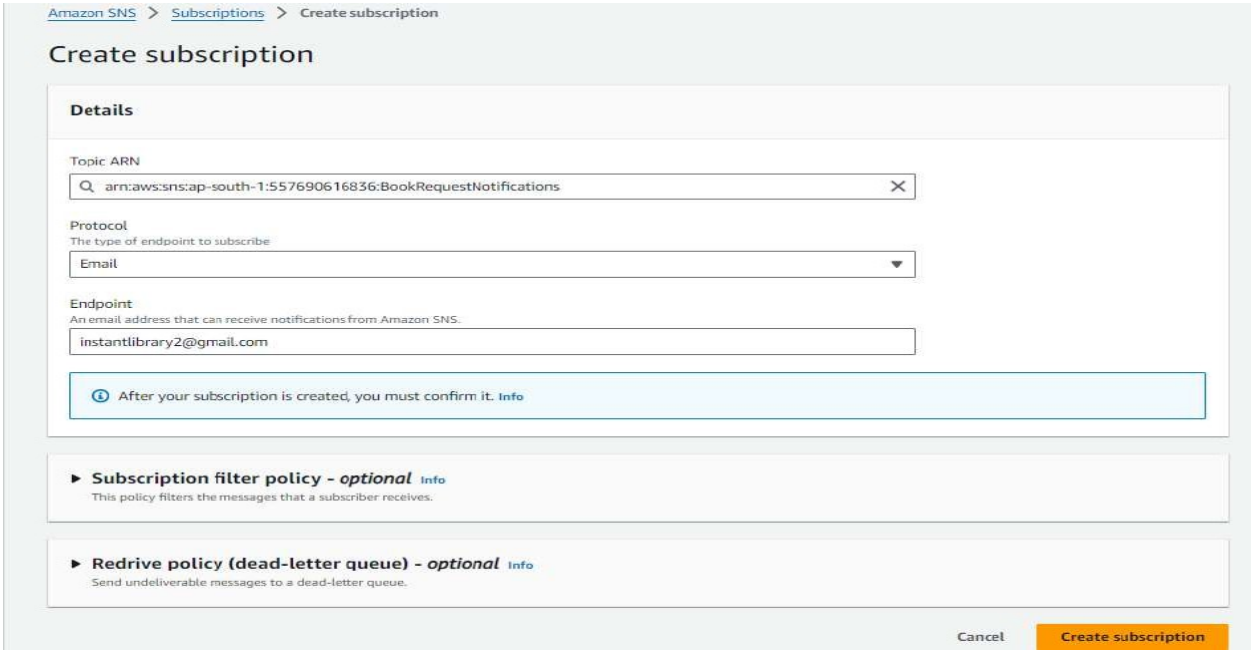


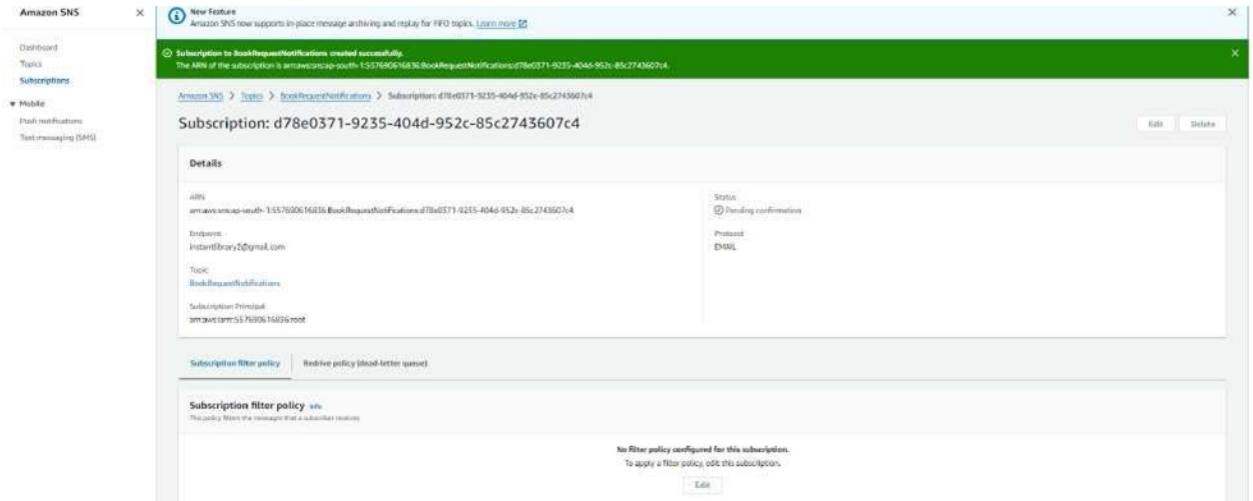
Configure the SNS topic and note down the Topic ARN.



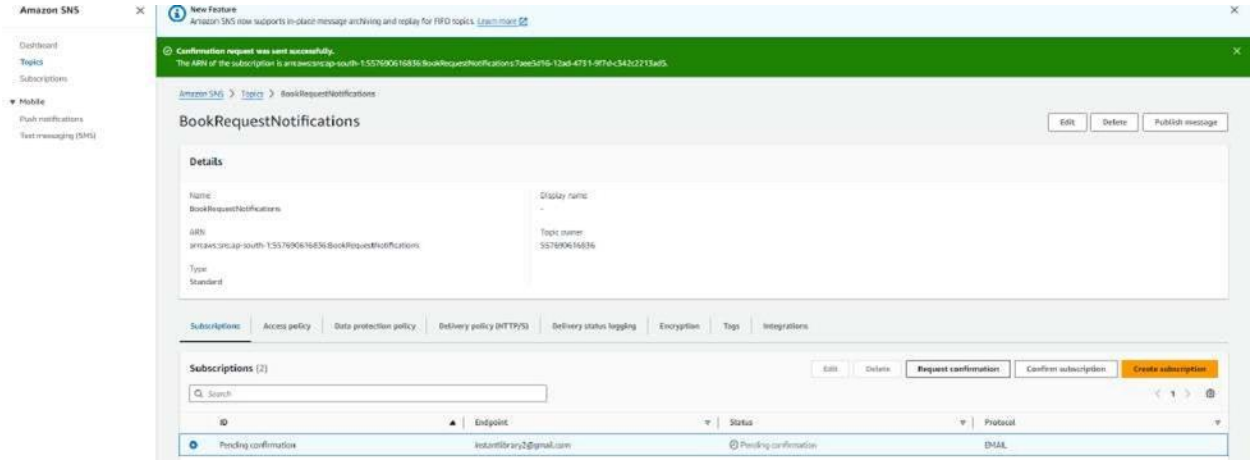
Activity 3.2: Subscribe users and staff to relevantSNS topics to receive real-time notifications when a book request is made.

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





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 ×

AWS Notifications <no-reply@sns.amazonaws.com> 9
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

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

aws

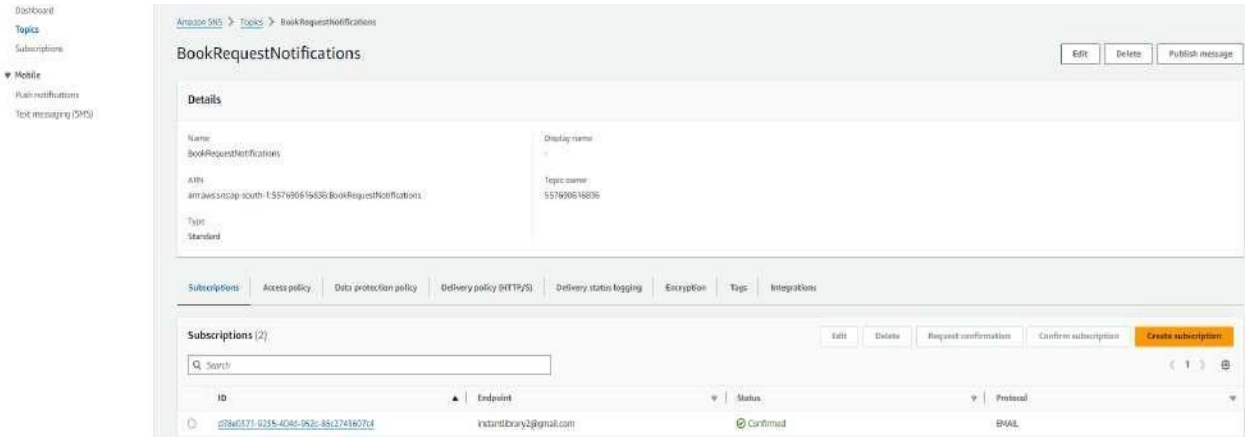
Simple Notification Service

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.

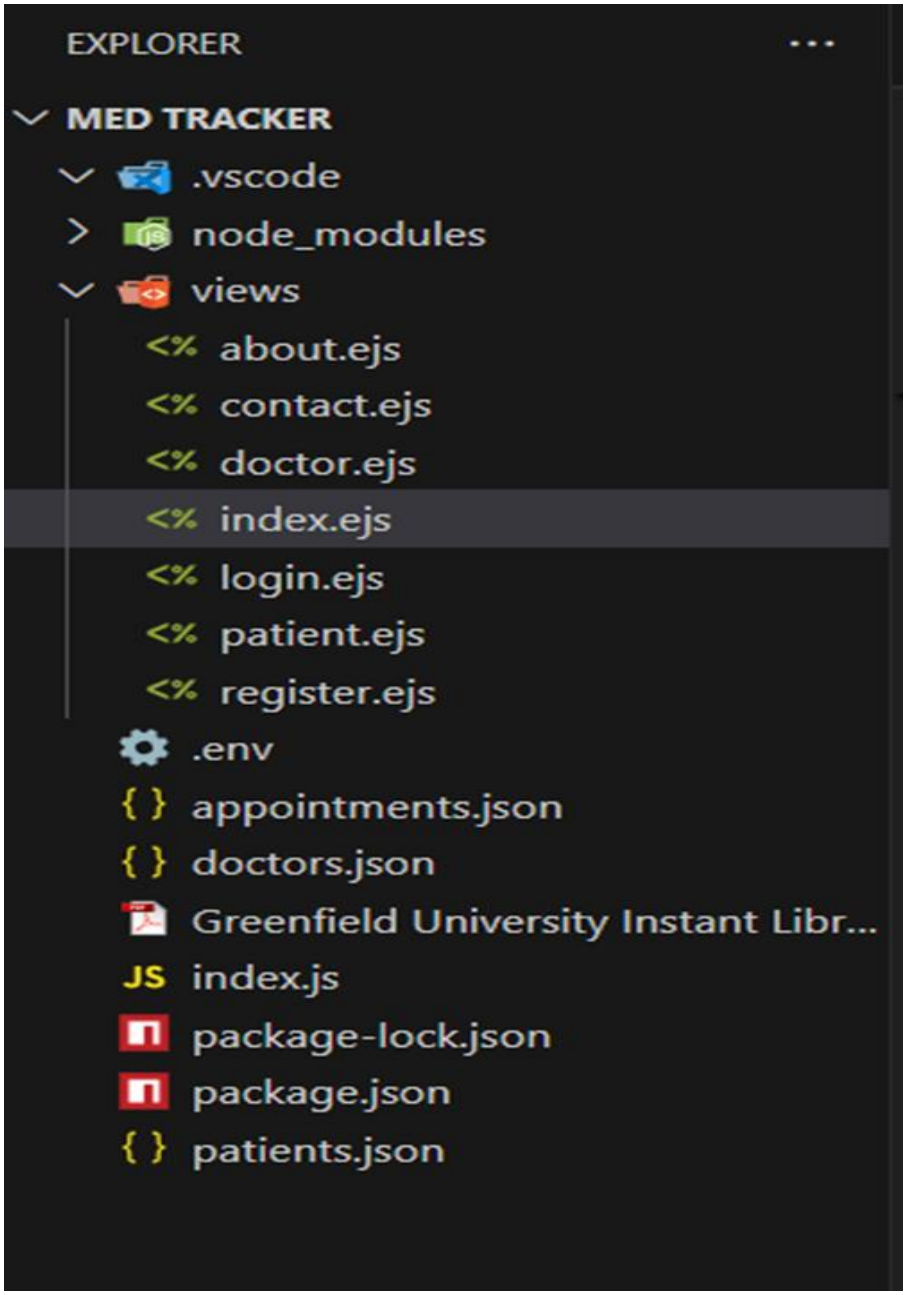


Successfully done with the SNS mail subscription and setup, now store the ARN link.

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 ×   .env   index.ejs
JS index.js > app.get("/") callback
import express from 'express';
import path from 'path';
import { fileURLToPath } from 'url';
import bcrypt from 'bcrypt';
import session from 'express-session';
import dotenv from 'dotenv';
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { SNSClient, PublishCommand } from '@aws-sdk/client-sns';
import { DynamoDBDocumentClient, GetCommand, PutCommand, QueryCommand, UpdateCommand } from '@aws-sdk/lib-dynamodb';
```

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.

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

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.

```
// Helper functions for DynamoDB
const getUserByEmail = async (email, role) => {
  const tableName = role === 'doctor' ? DOCTORS_TABLE : PATIENTS_TABLE;
  const command = new GetCommand({
    TableName: tableName,
    Key: { email }
  });
  const { Item } = await docClient.send(command);
  return Item;
};
```

Dynamo DB Setup:

```
// Helper functions for DynamoDB
const getUserByEmail = async (email, role) => {
  const tableName = role === 'doctor' ? DOCTORS_TABLE : PATIENTS_TABLE;
  const command = new GetCommand({
    TableName: tableName,
    Key: { email }
  });
  const { Item } = await docClient.send(command);
  return Item;
};
```

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

```
  // Send SNS notification
  if (SNS_TOPIC_ARN) {
    const snsCommand = new PublishCommand({
      TopicArn: SNS_TOPIC_ARN,
      Message: `New doctor registered: ${firstName} ${lastName} (${email}) - ${specialization}`,
      Subject: 'MedTrack Registration'
    });
    await snsClient.send(snsCommand).catch(err => console.error('SNS Error:', err));
  }

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

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:

```
app.get("/", (req, res) => res.render("index"));
app.get("/contactus", (req, res) => res.render("contactus"));
app.get("/about", (req, res) => res.render("about"));
app.get("/register", (req, res) => res.render("register"));
```

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

1. Register Route:

```
app.post('/register/patient', async (req, res) => {
  const { firstName, lastName, dob, gender, email, phone, address, password } = req.body;

  // Check if patient exists
  const existingPatient = await getUserByEmail(email, 'patient');
  if (existingPatient) return res.send('Patient exists');

  // Create new patient
  await createUser({
    id: Date.now().toString(),
    name: `${firstName} ${lastName}`,
    dob,
    gender,
    email,
    phone,
    address,
    password: await bcrypt.hash(password, 10)
  }, 'patient');

  // Send SNS notification
  if (SNS_TOPIC_ARN) {
    const snsCommand = new PublishCommand({
      TopicArn: SNS_TOPIC_ARN,
      Message: `New patient registered: ${firstName} ${lastName} (${email})`,
      Subject: 'MedTrack Registration'
    });
    await snsClient.send(snsCommand).catch(err => console.error('SNS Error:', err));
  }

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

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 counton 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.

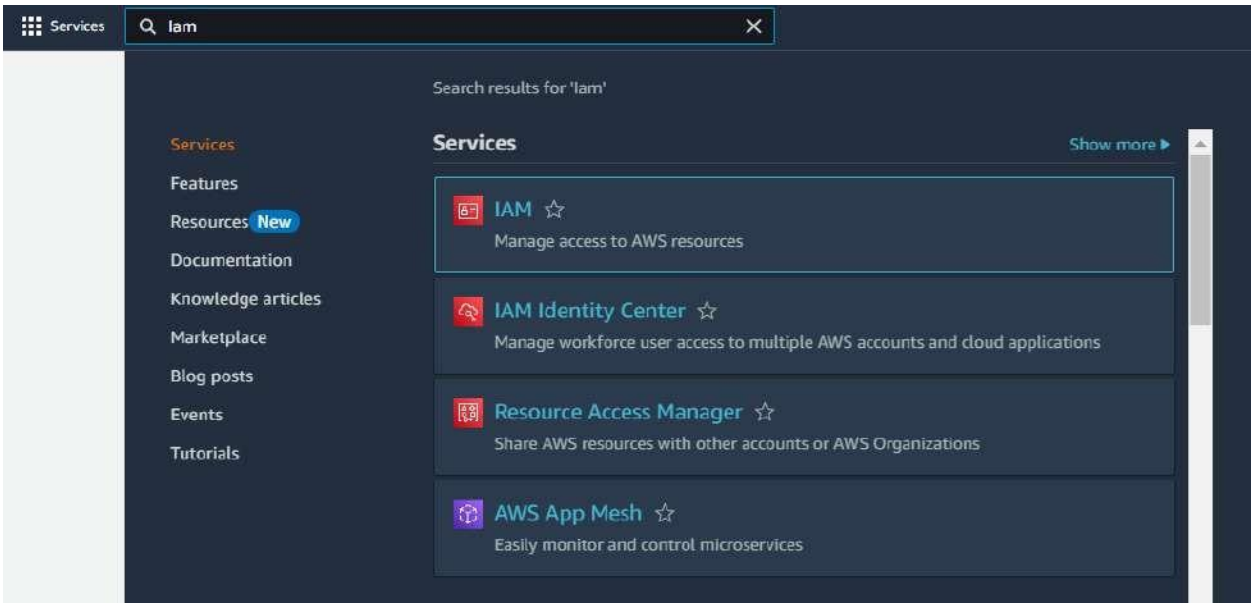
Deployment Code:

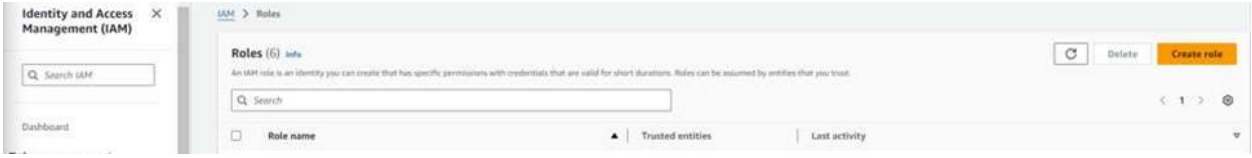
Start the Flask server to listen on all network interfaces (0.0.0.0) at port 80
with debug mode enabled for development and testing.

Milestone 5: IAM Role Setup

Activity 5.1: Create IAM Role.

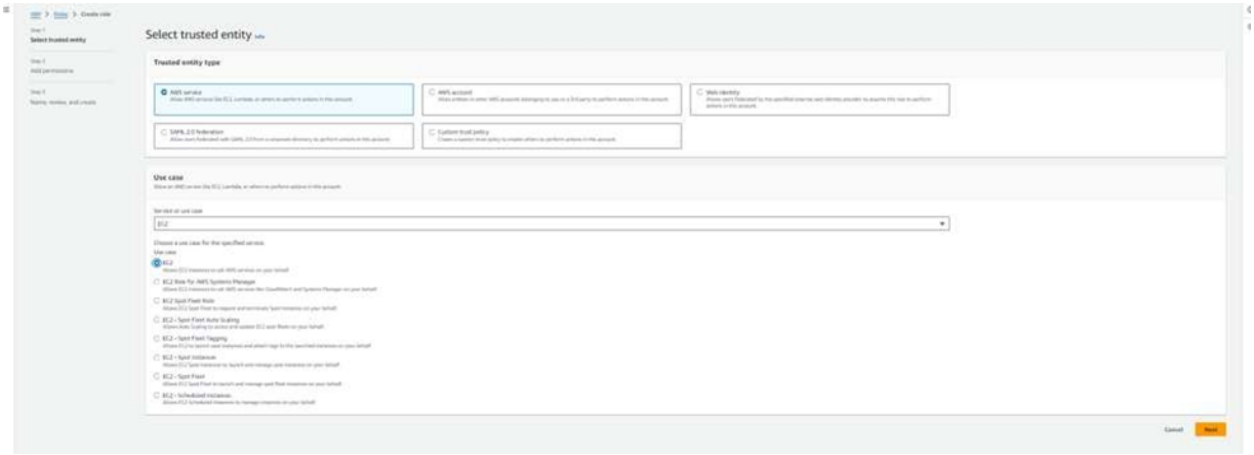
In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.





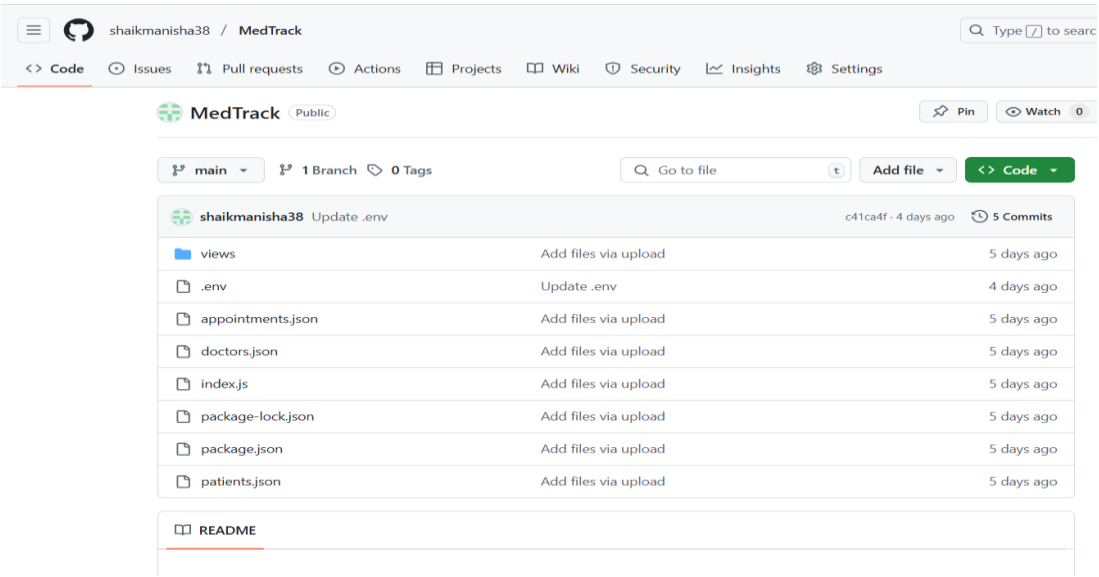
Activity 5.2: Attach Policies.

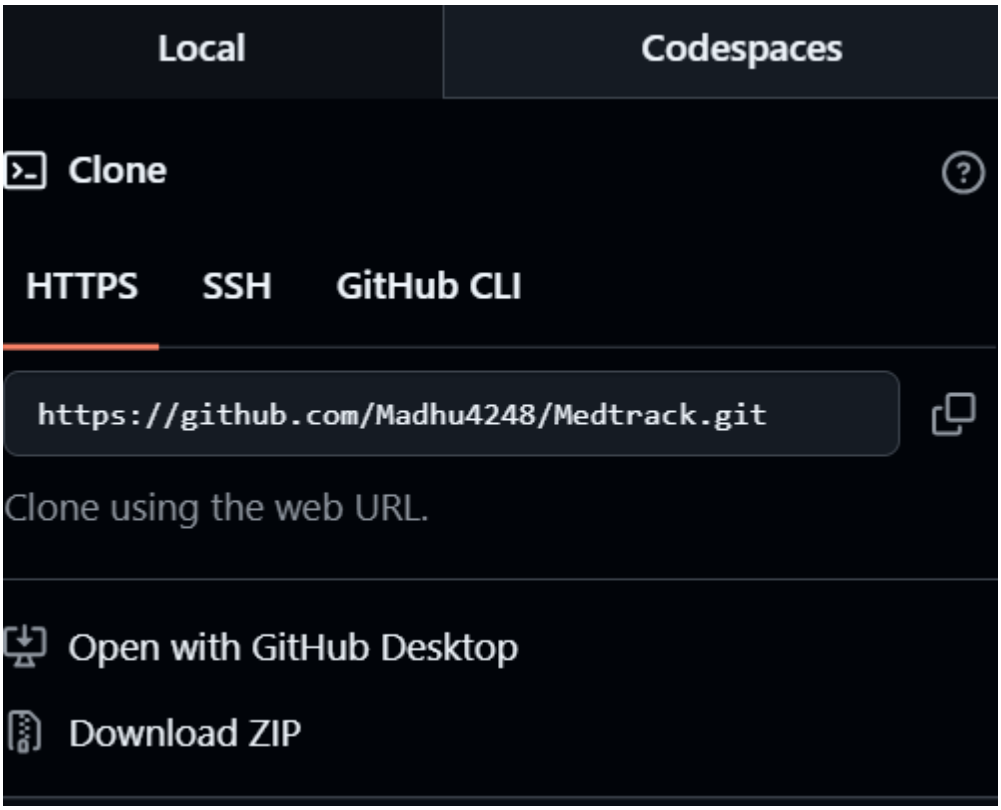
1. Amazon Dynamo DB Full Access: Allows EC2 to perform read/write operations on DynamoDB.
2. Amazon SNS Full Access: Grants EC2 the ability to send notifications via SNS.



Milestone 6: EC2 Instance Setup

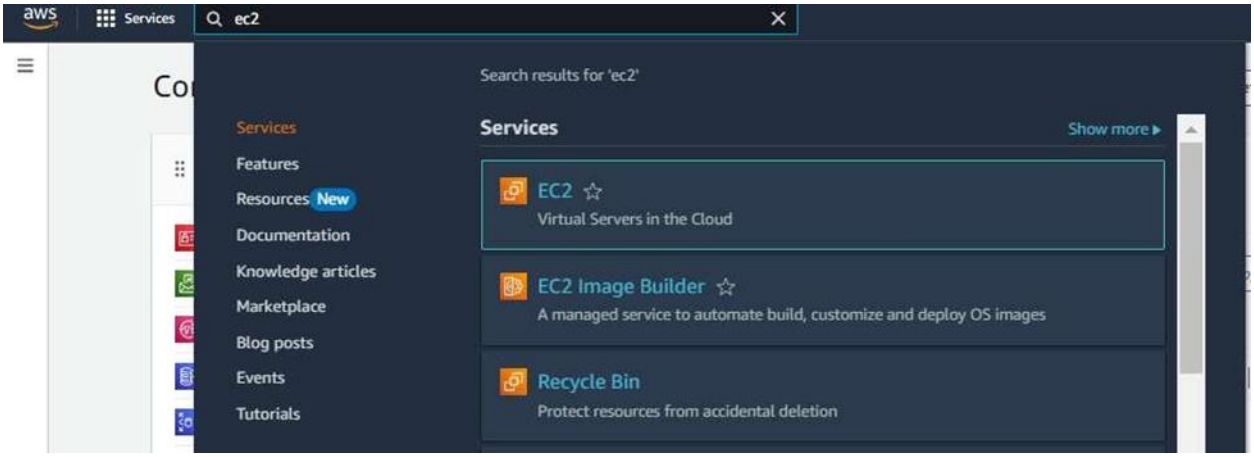
Load your index.jsand Html files into GitHubrepository.



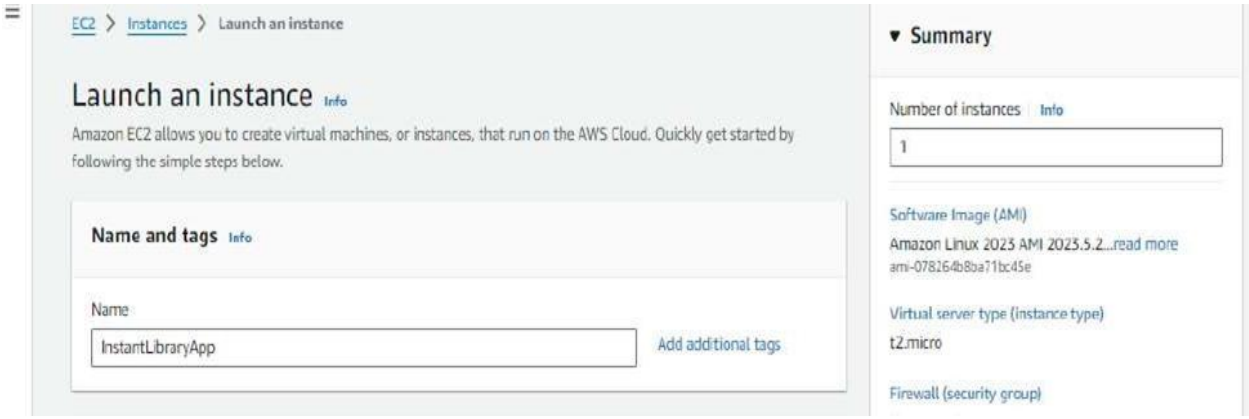


Activity 6.1: Launchan EC2 instance to host the Flask application.

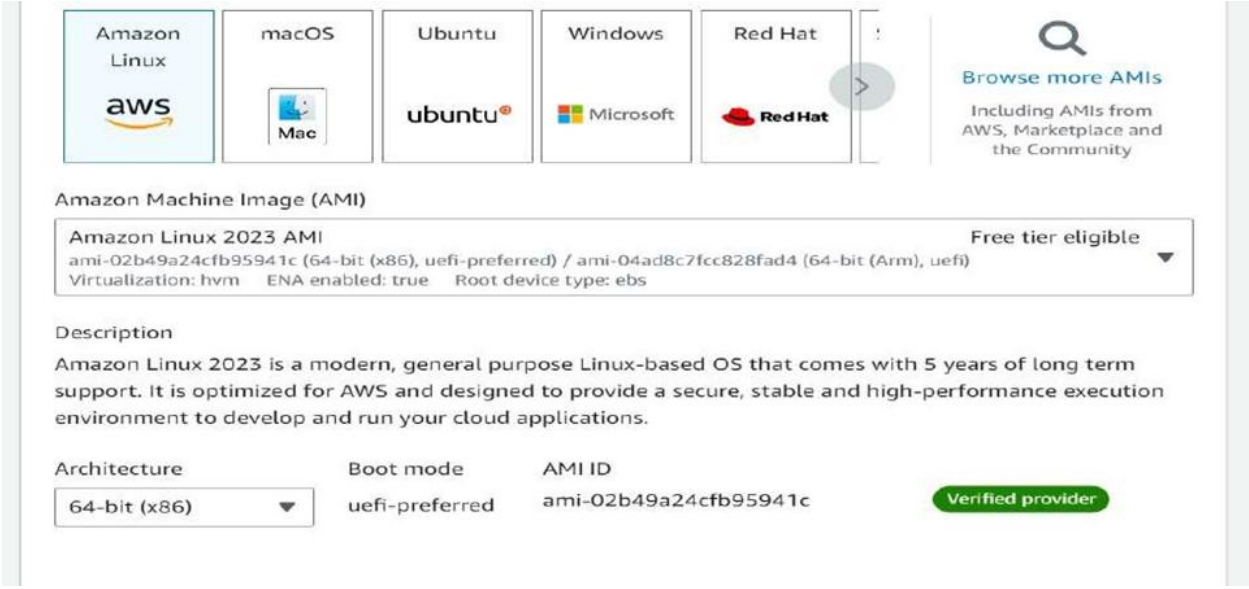
1. In the AWS Console, navigateto EC2 and launch a new instance.



2.Click on Launchinstance to launchEC2 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

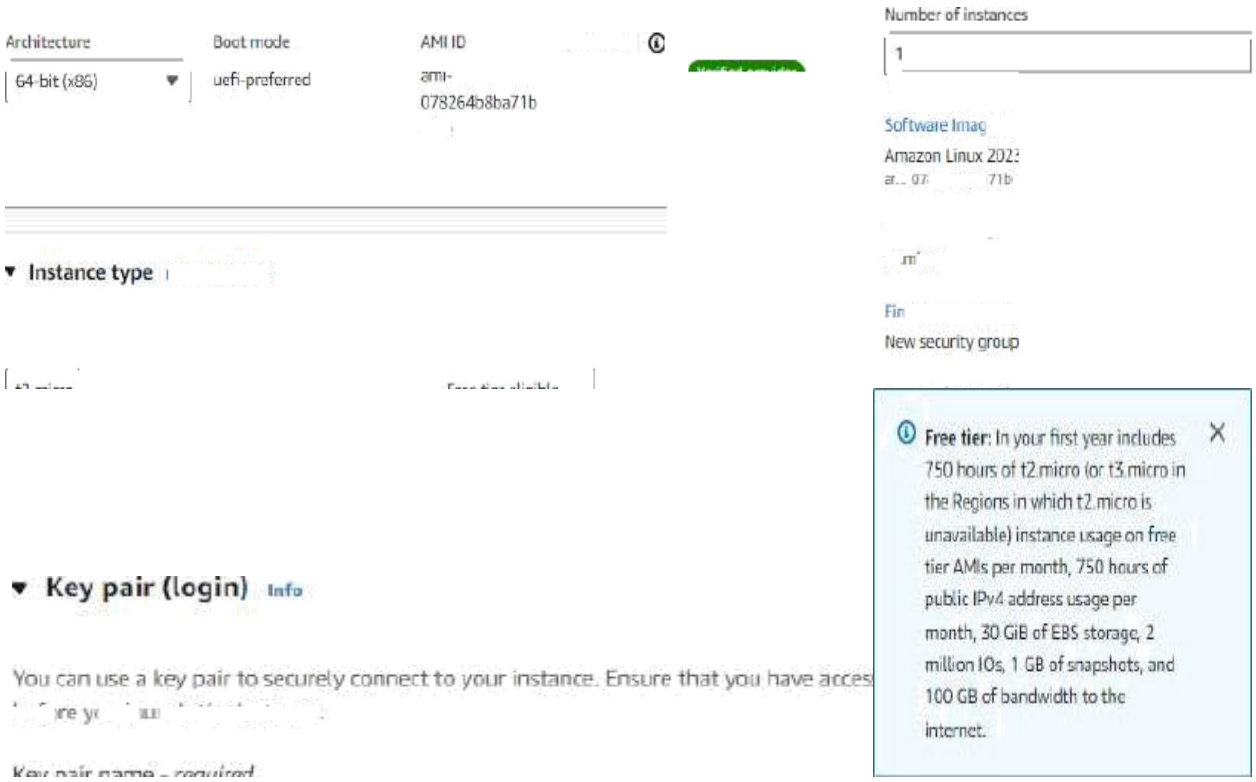
Private key file format

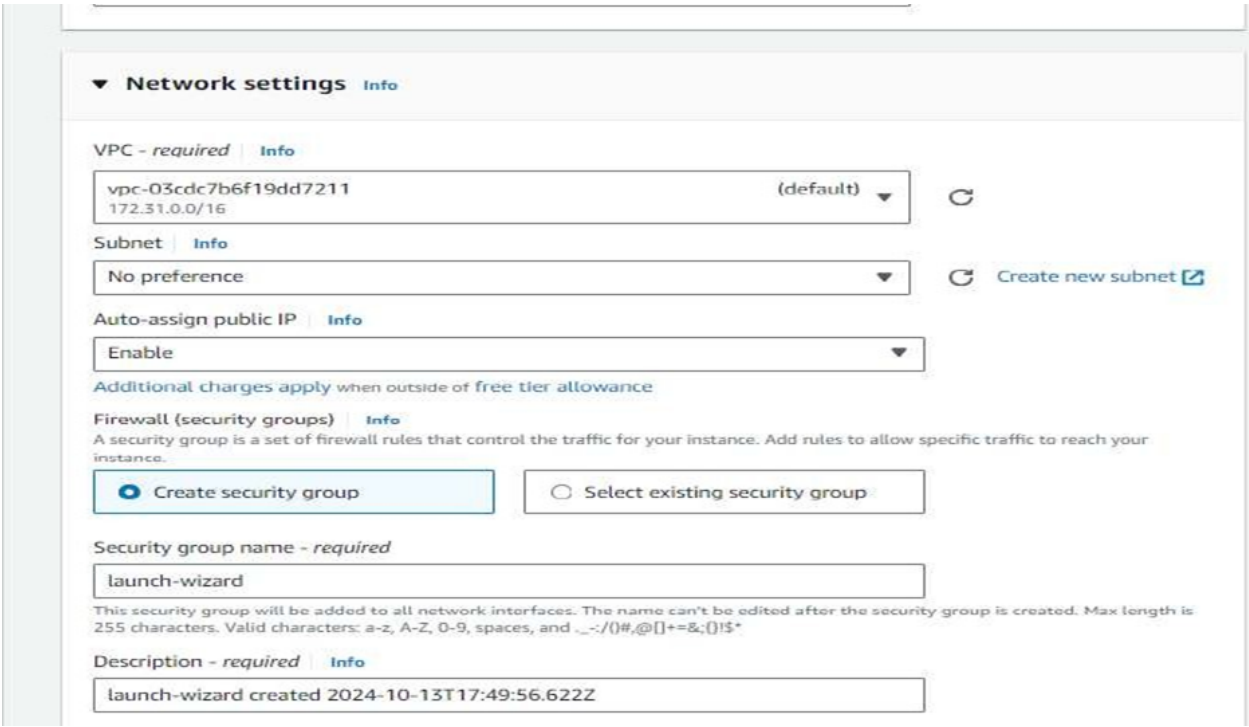
- .pem — For use with OpenSSH
- .ppk — For use with PuTTY

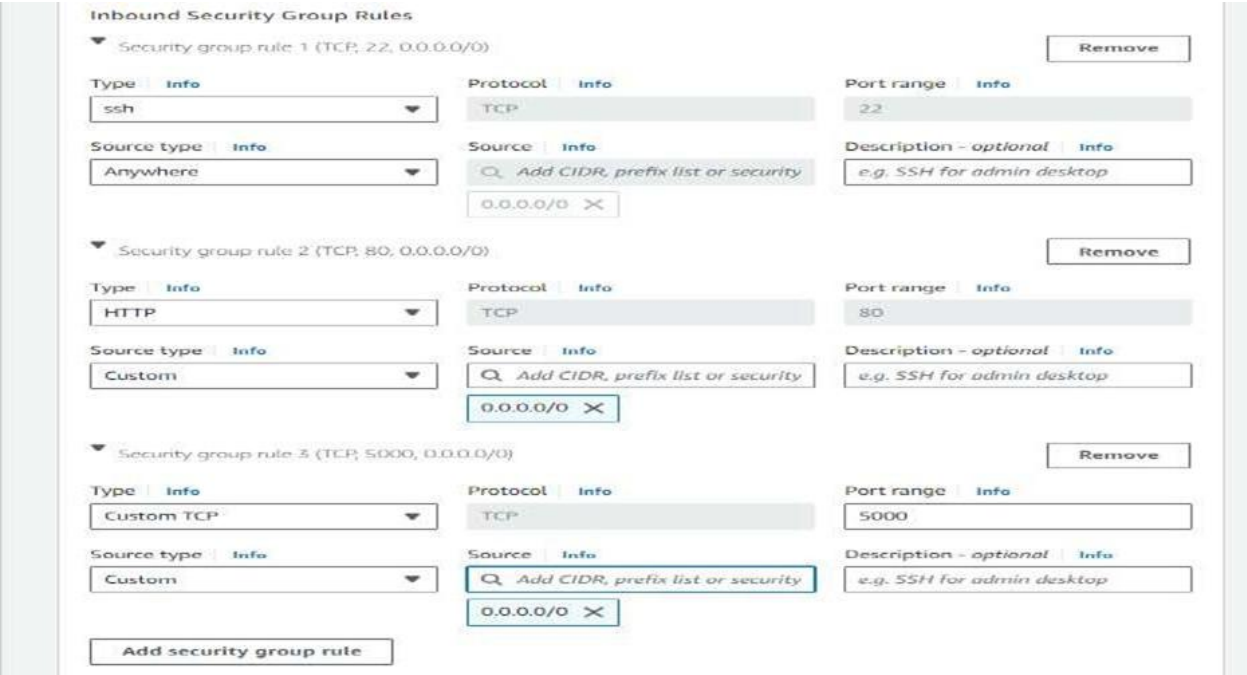
⚠ When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** Learn more

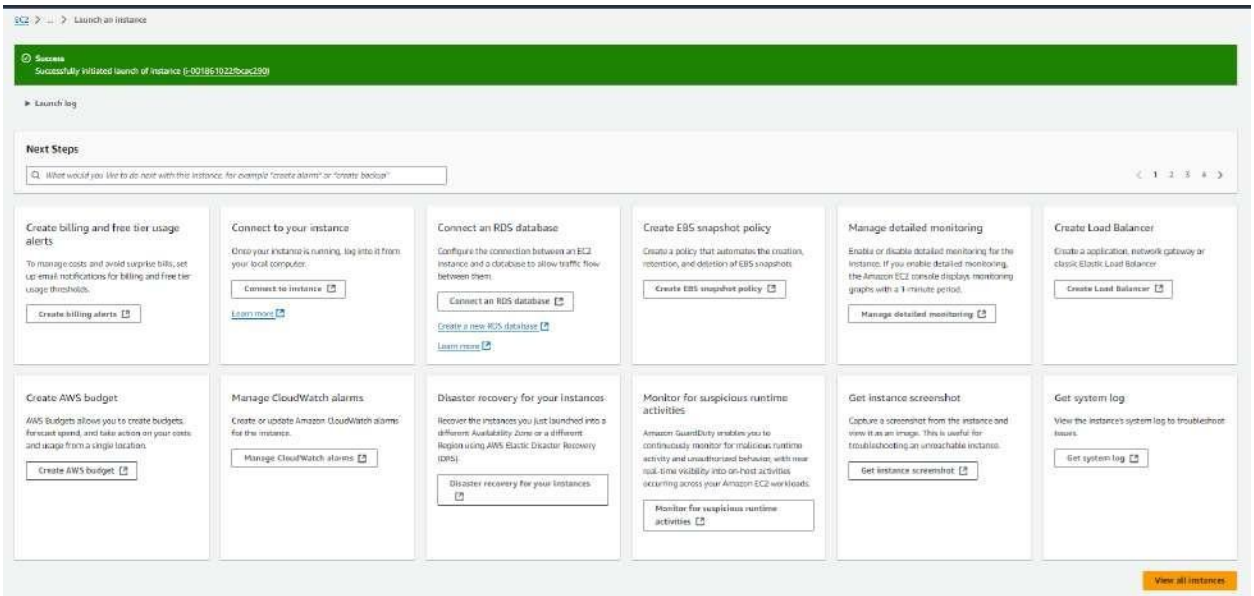
Cancel | Create key pair

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



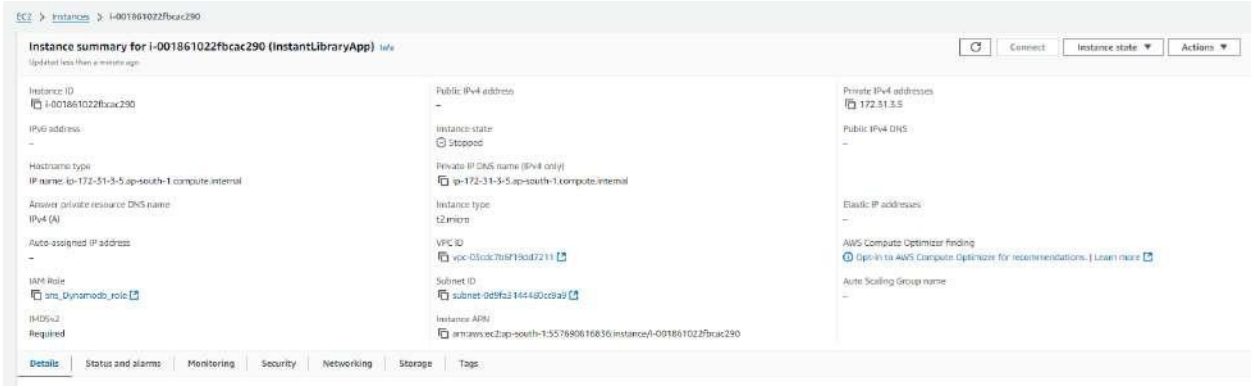


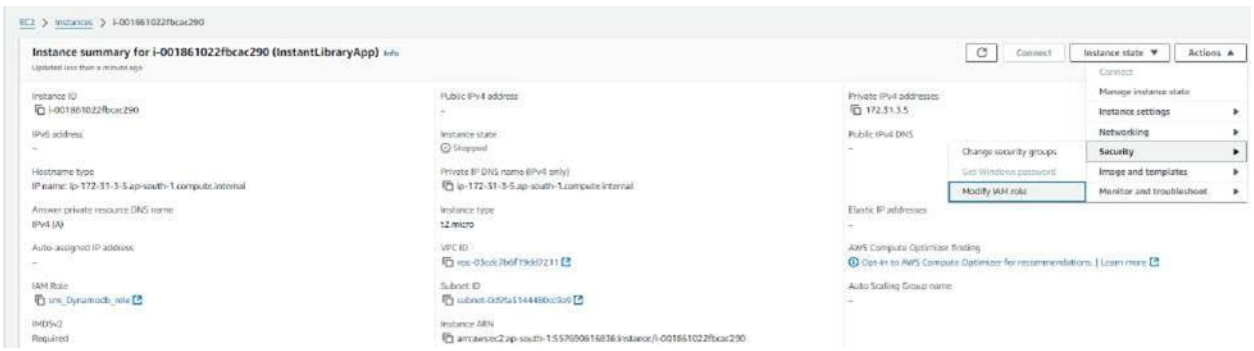


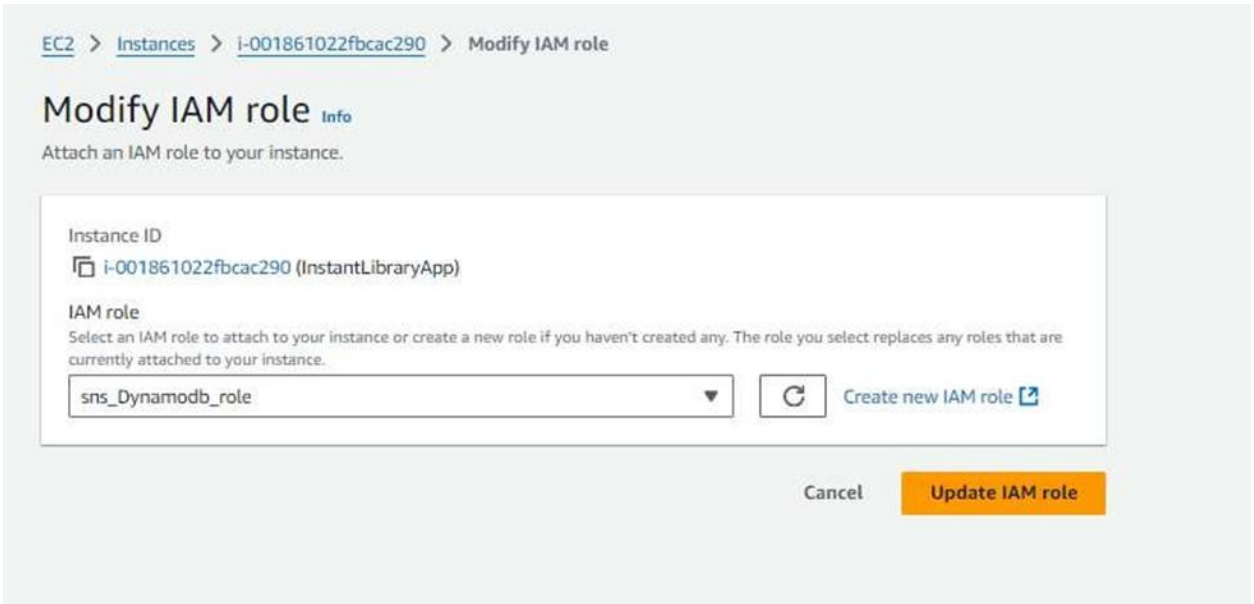


To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected, 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.

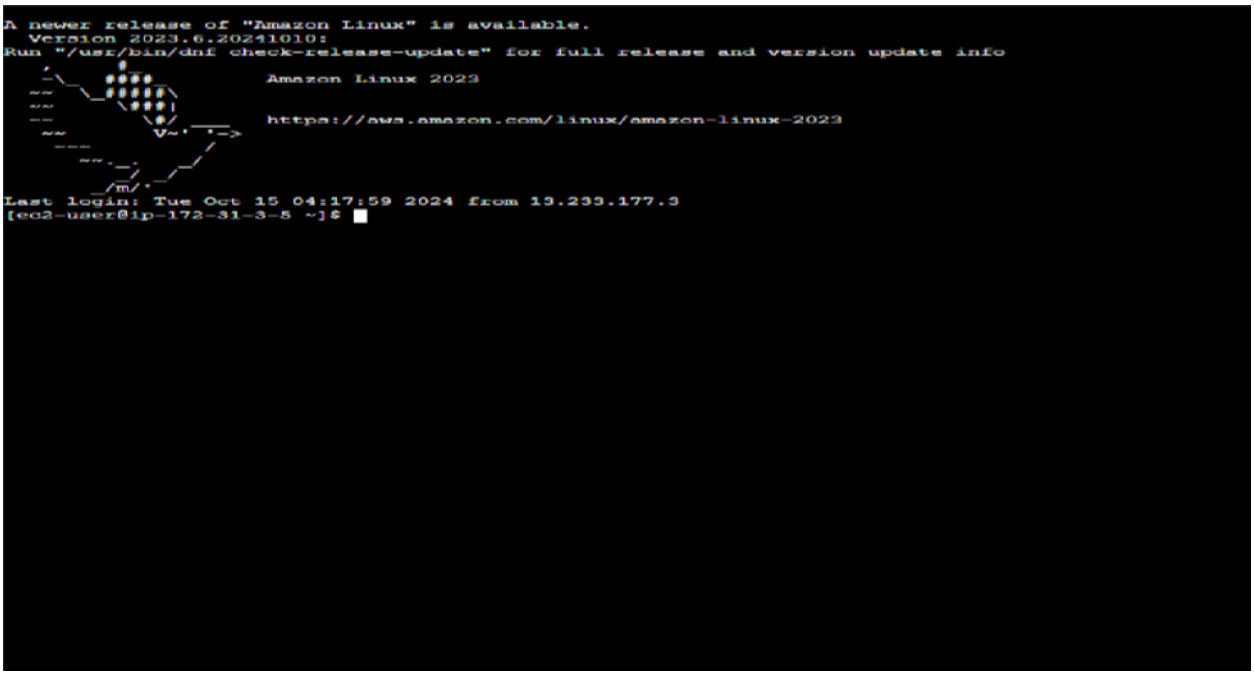

Instances (1/2) Info
Connect
Instance state
Actions
Launch instances
Find instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
Elastic IP
IPv6 IPs
Monitoring
Security g
InstantLibrary...
i-001861022fbcac290
Stopped
t2.micro
View alarms +
ap-south-1b
disabled


EC2 > Instances > i-001861022fbcac290
Instance summary for i-001861022fbcac290 (InstantLibraryApp) Info
Connect
Instance state
Actions
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
Subnet ID
Instance ARN
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) Info
Connect
Instance state
Actions
Connect
Manage instance state
Instance settings
Networking
Security
Image and templates
Monitor and troubleshoot
Change security groups
Get Windows password
Modify IAM role


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 Softwareon 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 Projectfrom GitHub

Clone your projectrepository from GitHubinto 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 installNote: change your-github-username and your-repository-name with your

2. This will downloadyour project to the EC2 instance.

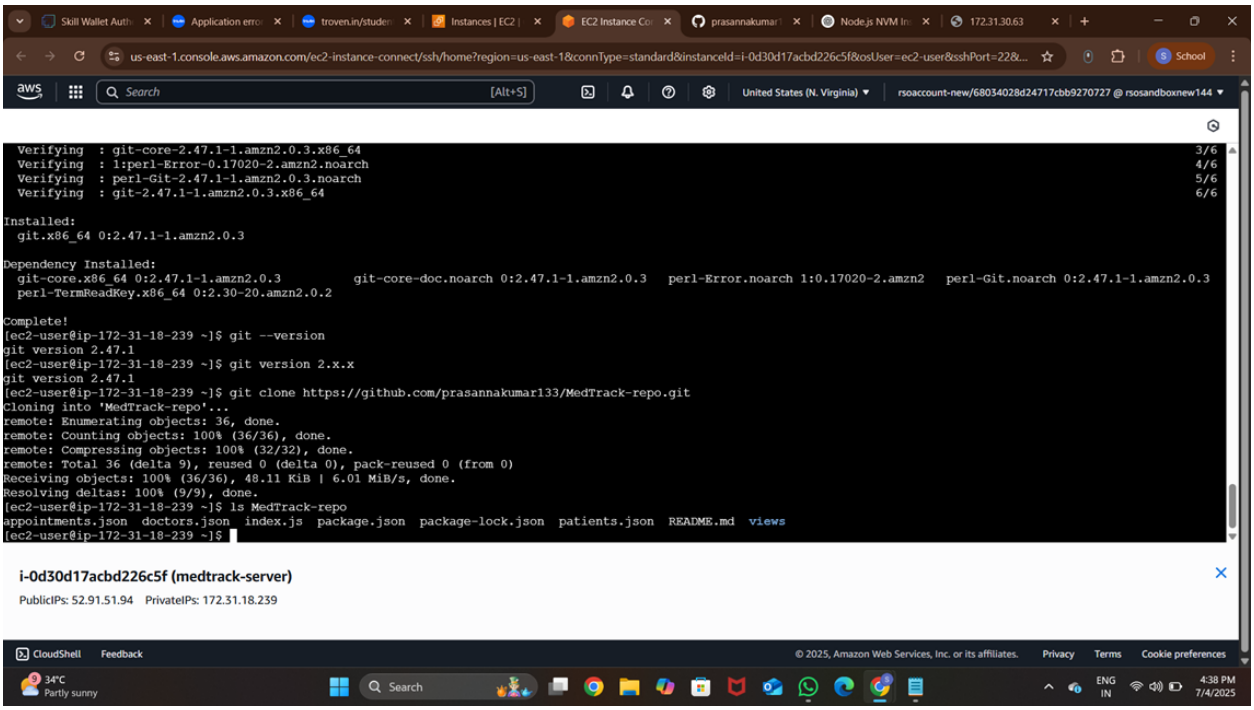
To navigateto the project directory, run the following command:
cd MedTrack-repo
Once inside the projectdirectory, configure and run the Nodejs application by executing the following command with elevated privileges:
Run the Flask Application
Install project dependencies

npm install

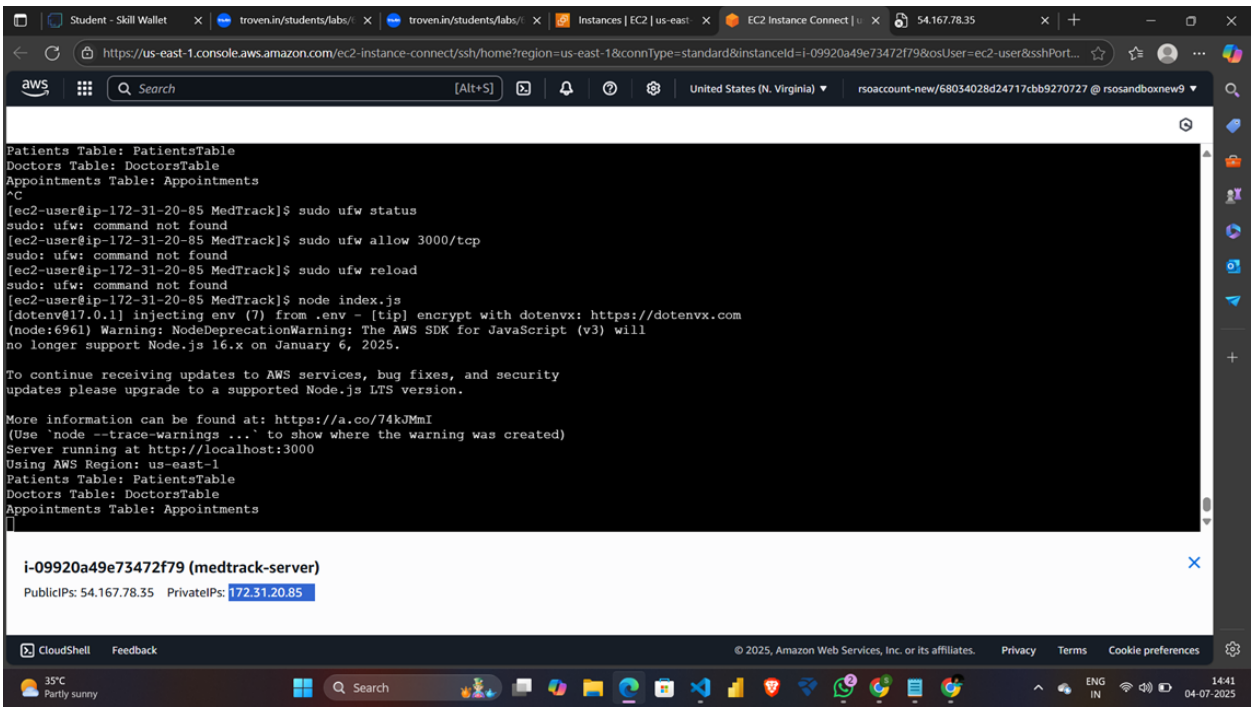
# Install PM2 globally to keep the app running
sudo npm install -g pm2

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



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.

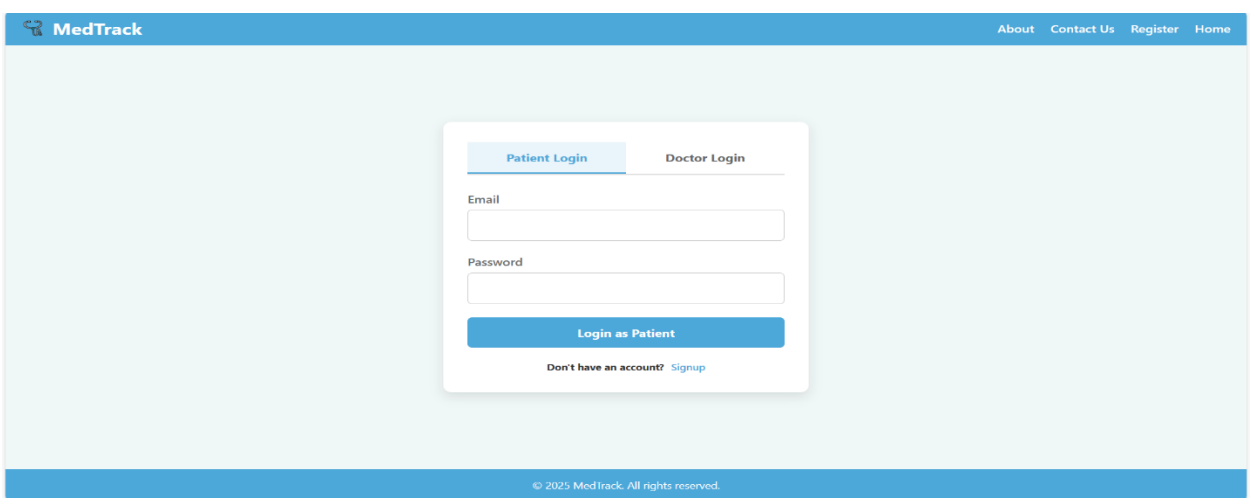


Access the website through:
Public IPs: http://52.91.51.94:3000/

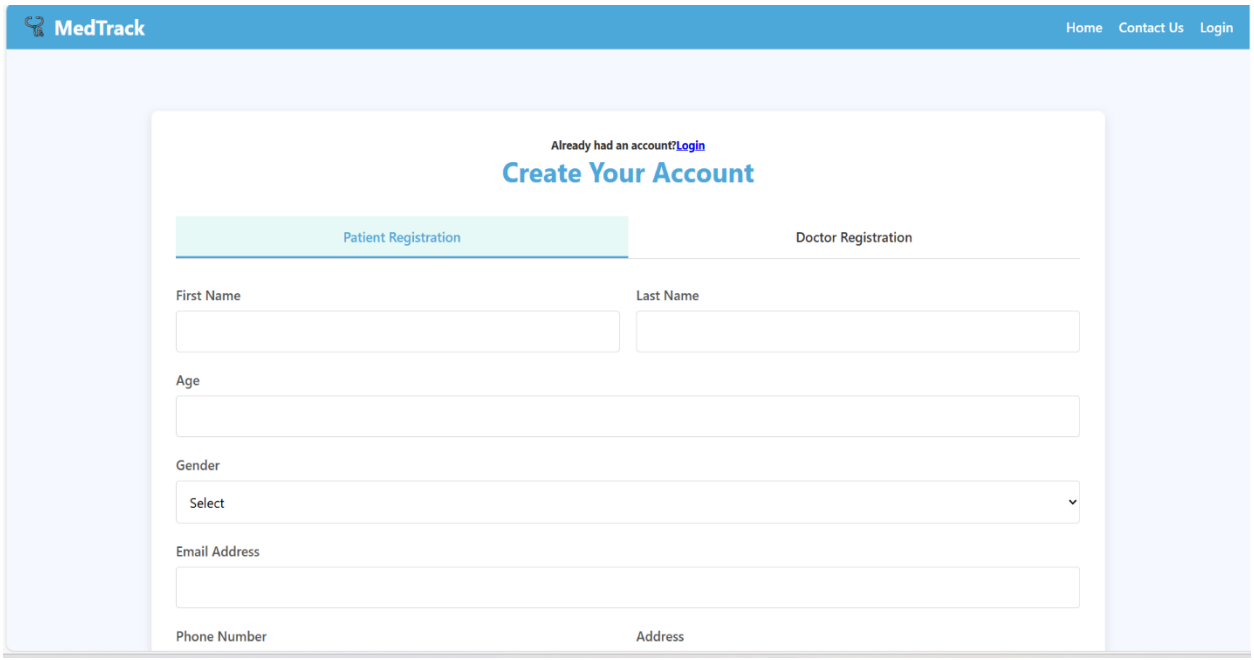
Milestone 8: Testing and Deployment

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

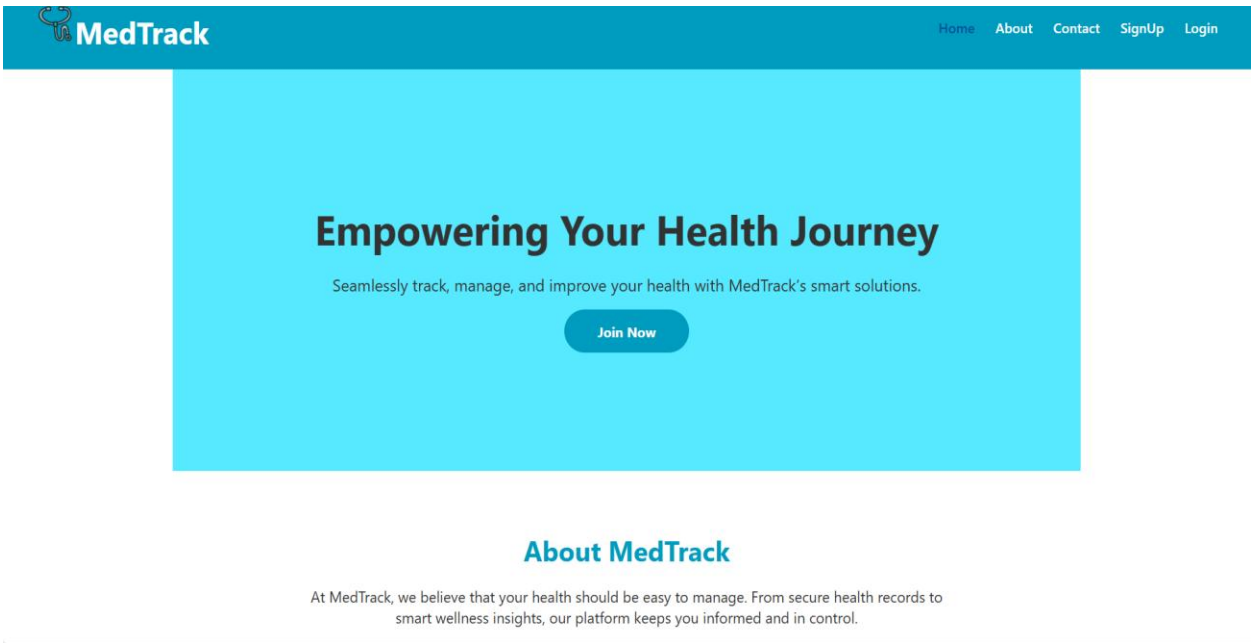
Login Page:



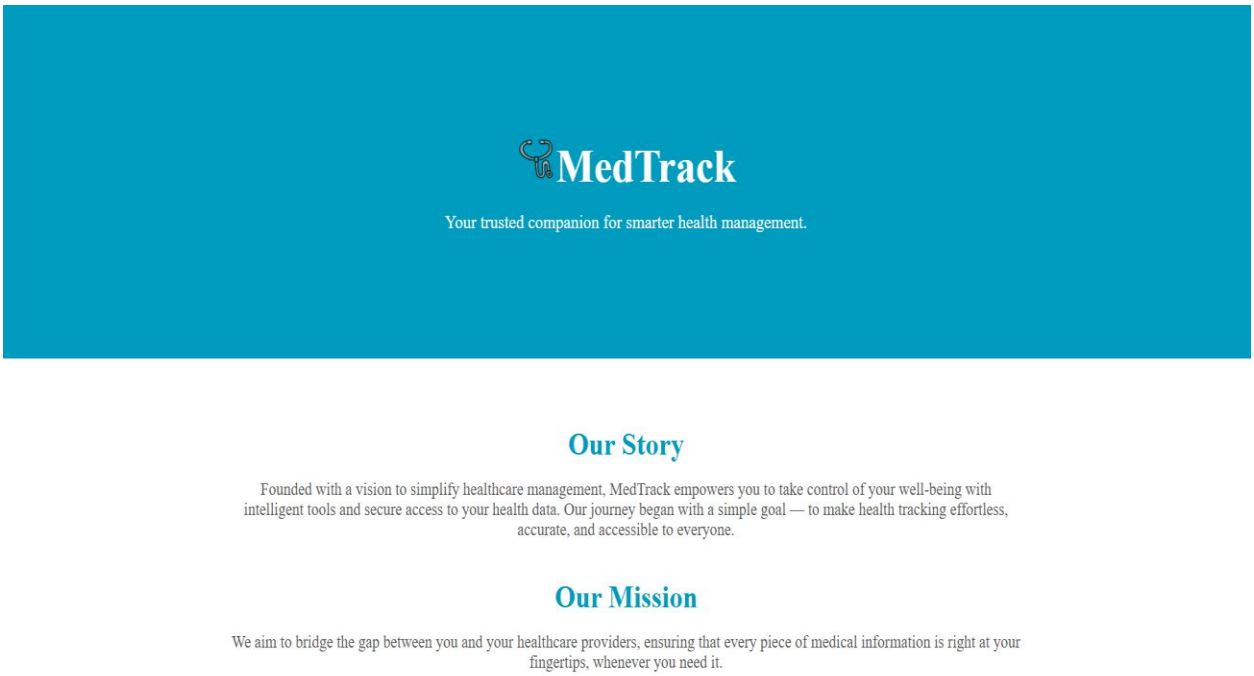
Register Page:



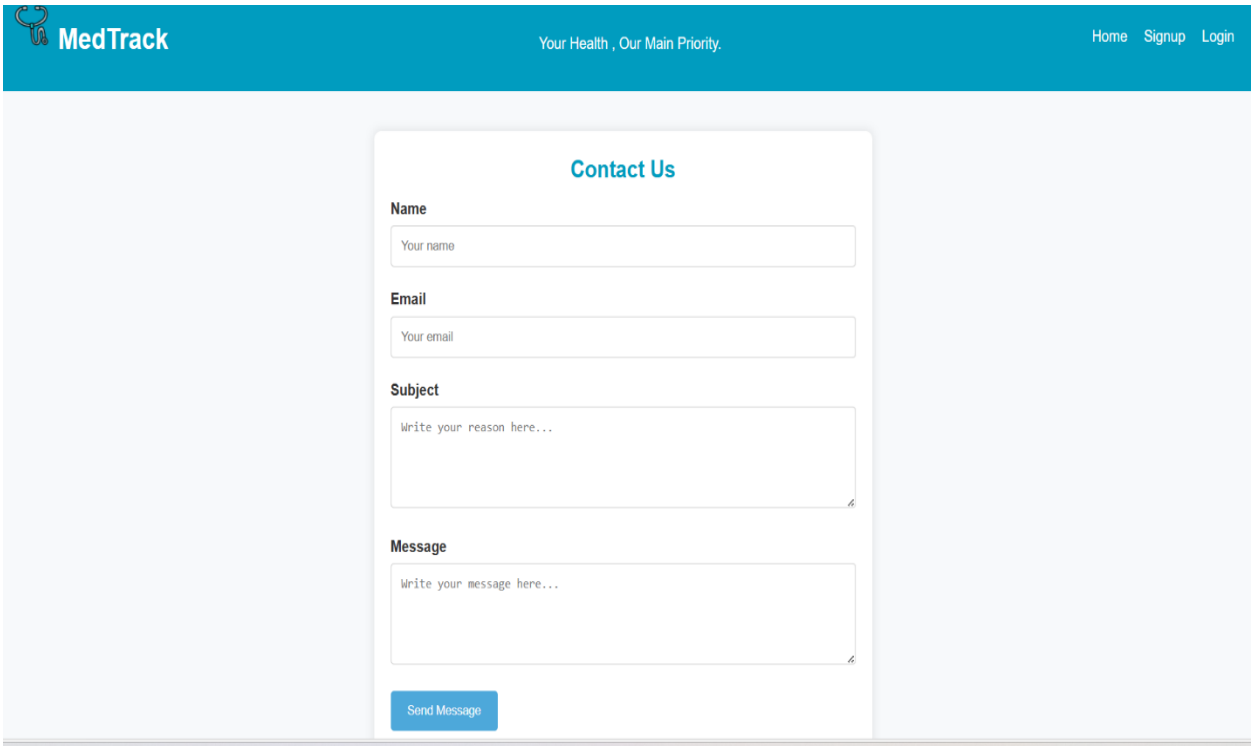
Home page:



About page:



Contact Page:



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.