MedTrack: AWS Cloud-Enabled Healthcare Management System

Project Description:

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

Scenario 1: Efficient Appointment Booking System for Patients

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

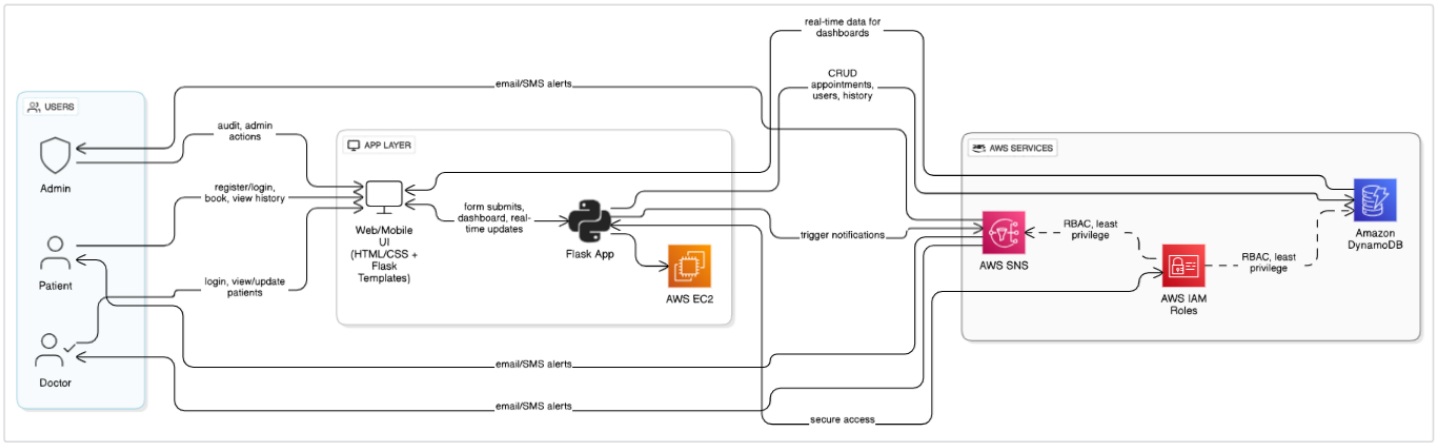
Scenario 2: Secure User Management with IAM

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

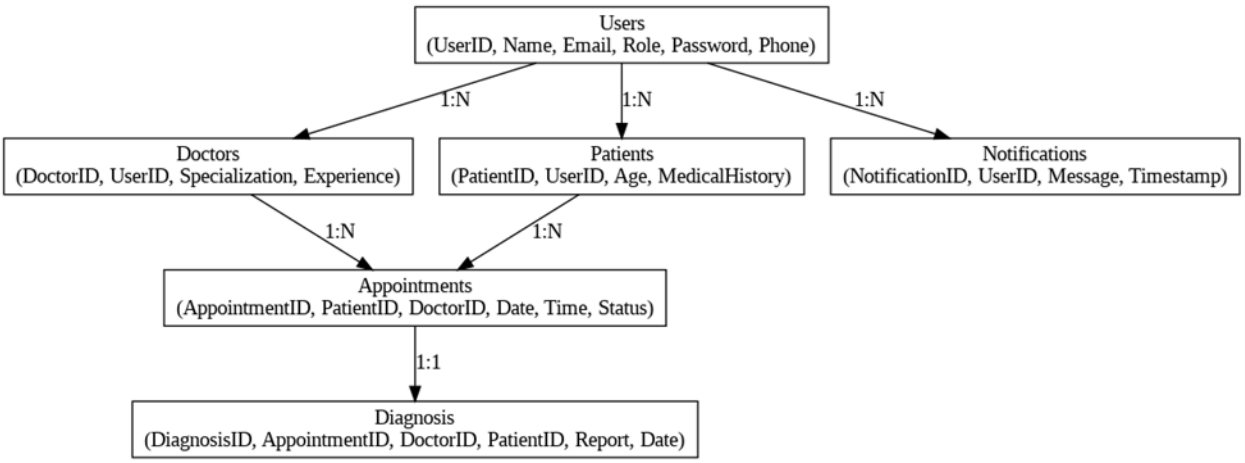
Scenario 3: Easy Access to Medical History and Resources

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

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Pre-requisites:

1. .**AWS Account Setup**: AWS Account Setup
2. **Understanding IAM**: IAM Overview
3. **Amazon EC2 Basics**: EC2 Tutorial
4. **DynamoDB Basics**: DynamoDB Introduction
5. **SNS Overview**: SNS Documentation
6. **Git Version Control**: Git Documentation

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 patients, doctors, and Book appointments.

3. SNS Notification Setup

Activity 3.1: Create SNS topics for Book Appointment notifications.

Activity 3.2: Subscribe patients and doctors to SNS email notifications.

4. Backend Development and Application Setup

Activity 4.1:Develop the Backend Using Flask.

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 Flask application.

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

7. Deployment on EC2

Activity 7.1:Upload Flask Files

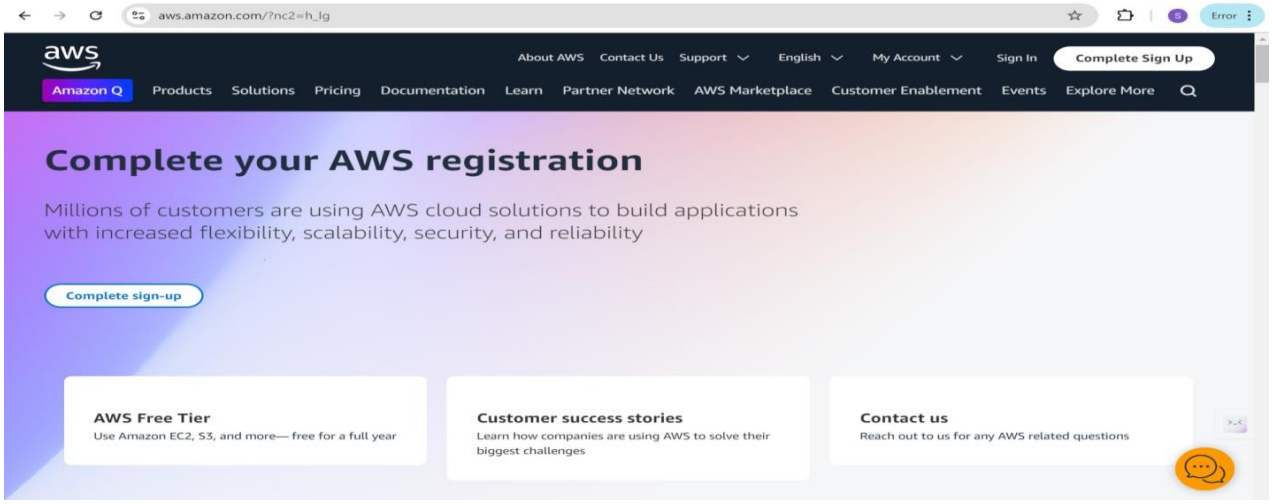
Activity 7.2: Run the Flask App

8. Testing and Deployment

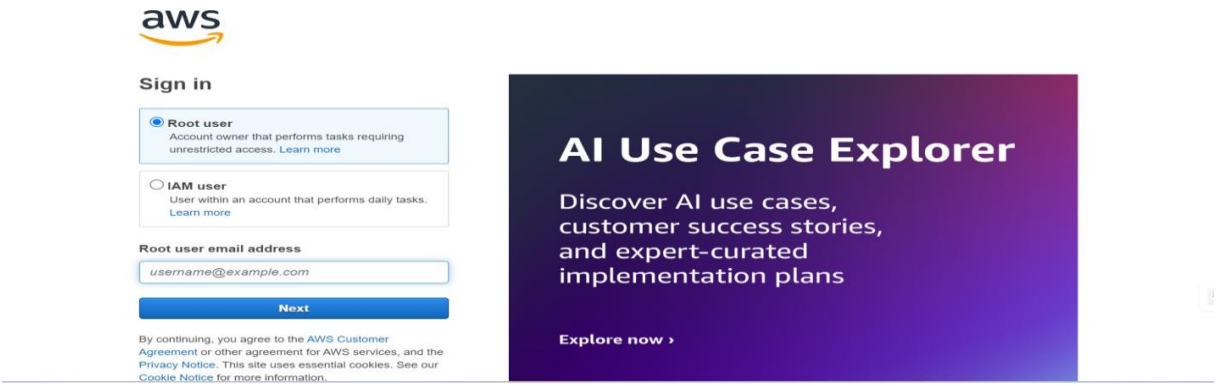
Activity 8.1: Conduct functional testing to verify user registration, login, dashboard, and notifications.

Milestone 1: AWS Account Setup and Login

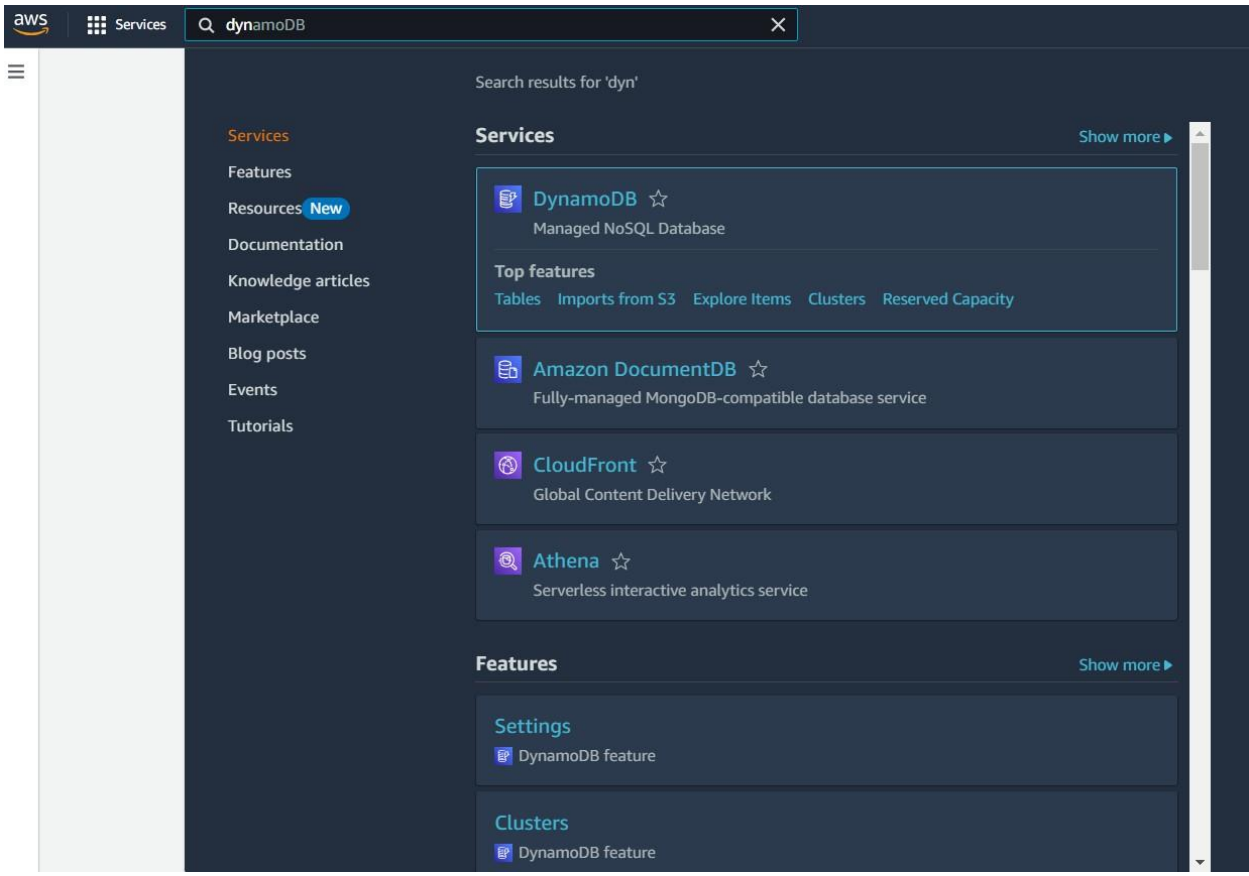
- **Activity 1.1: Set up an AWS account if not already done.**
  - Sign up for an AWS account and configure billing settings.

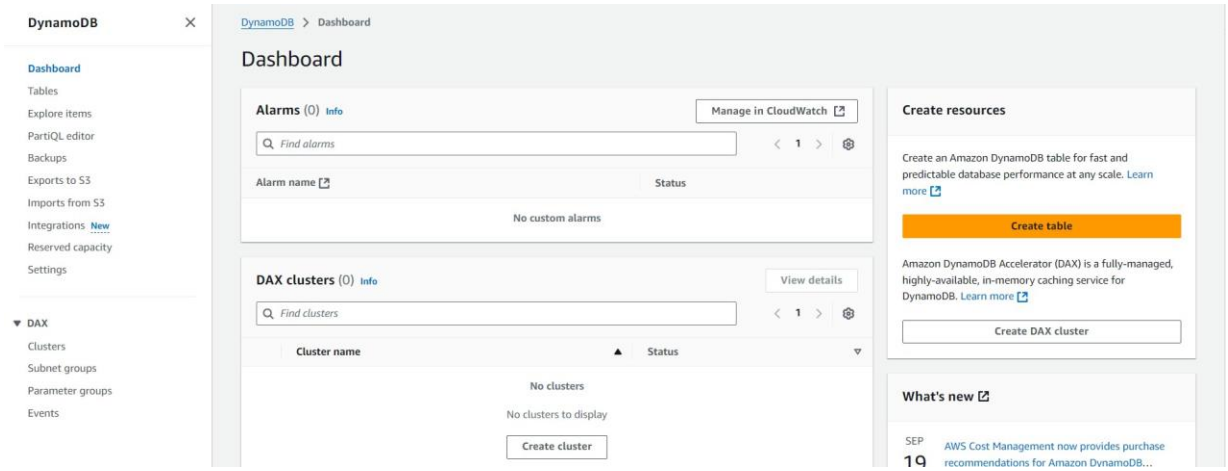


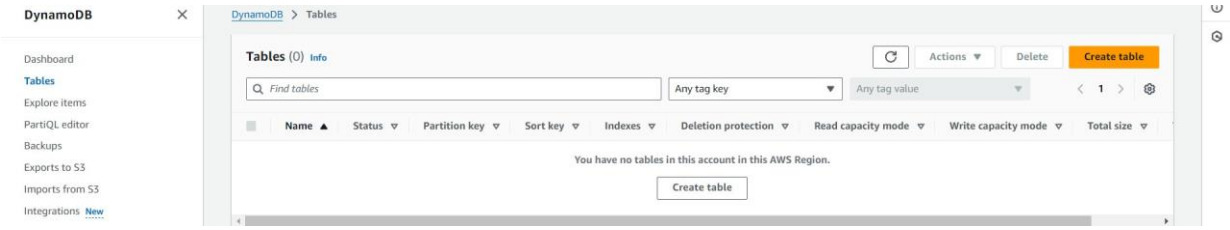
- **Activity 1.2: Log in to the AWS Management Console**
  - After setting up your account, log in to the AWS Management Console.



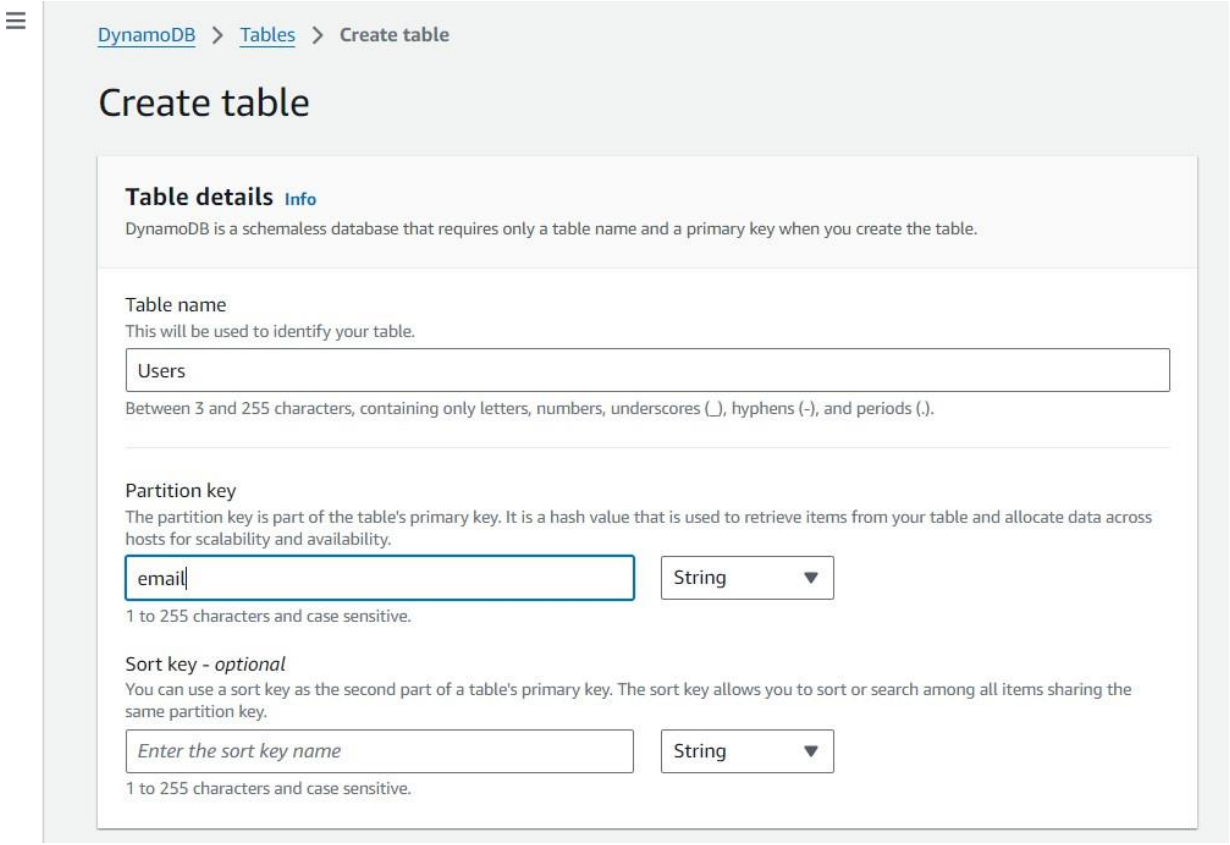
Milestone 2: DynamoDB Database Creation and Setup

- **Activity 2.1:Navigate to the DynamoDB**
    - In the AWS Console, navigate to DynamoDB and click on create tables.
    - 





- **Activity 2.2: Create a DynamoDB table for storing data**
    - Create Users table with partition key “Email” with type String and click on create tables.



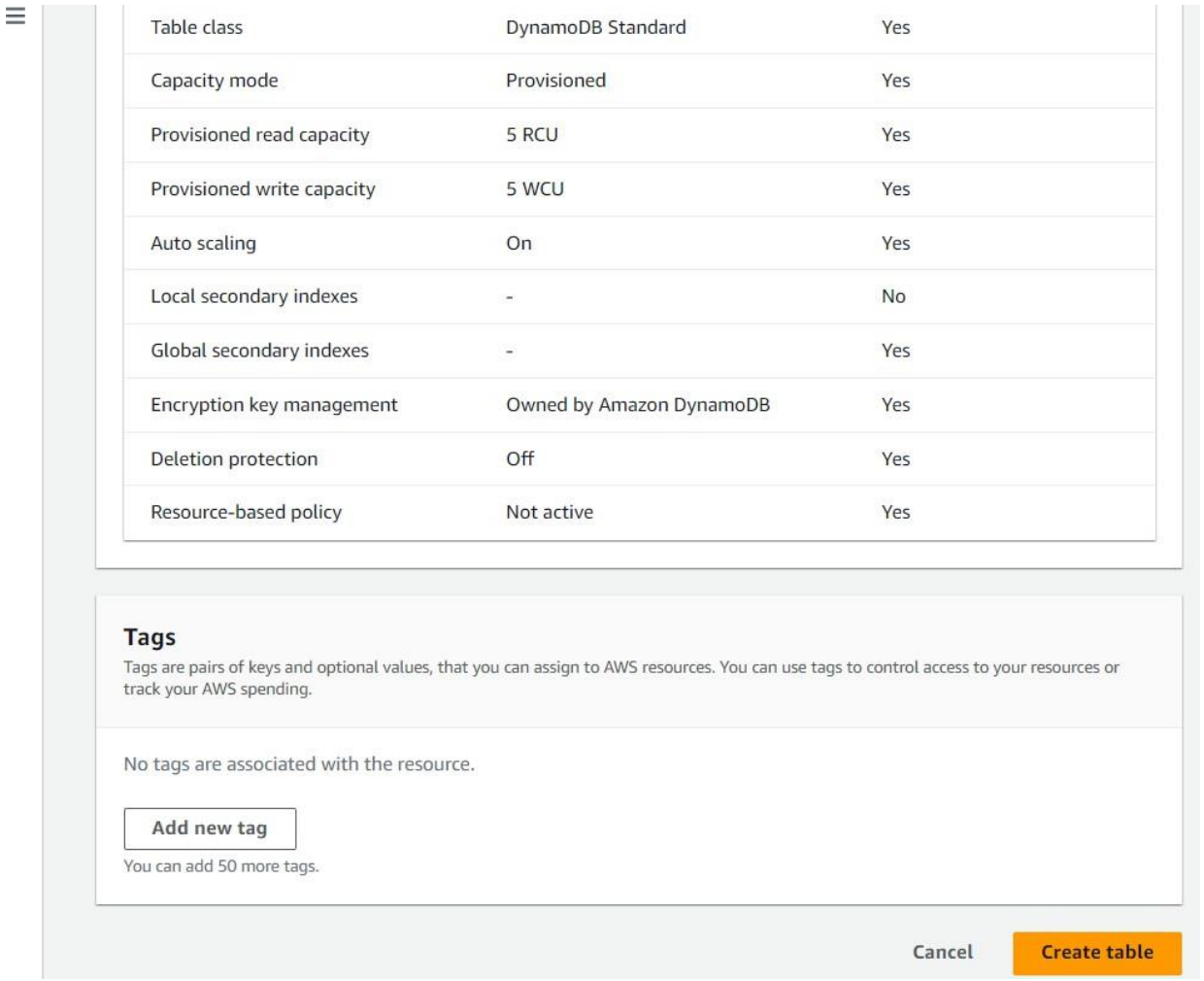


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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

Tags

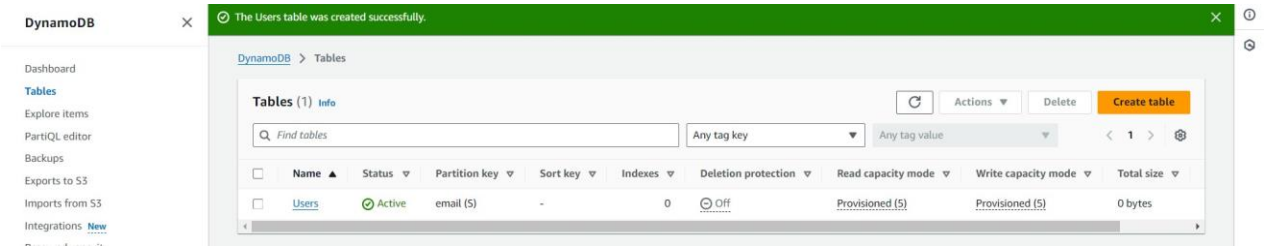
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

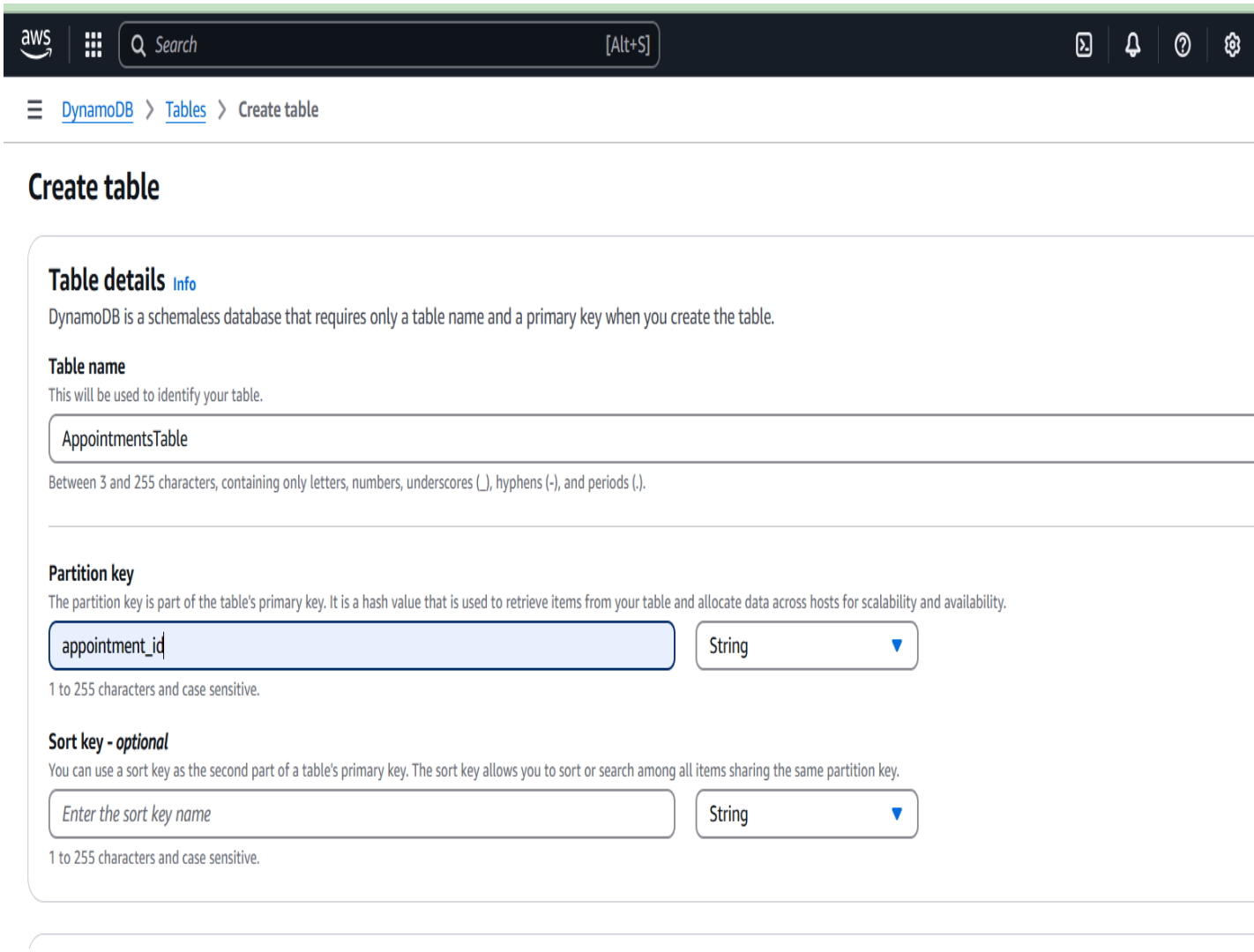
Add new tag

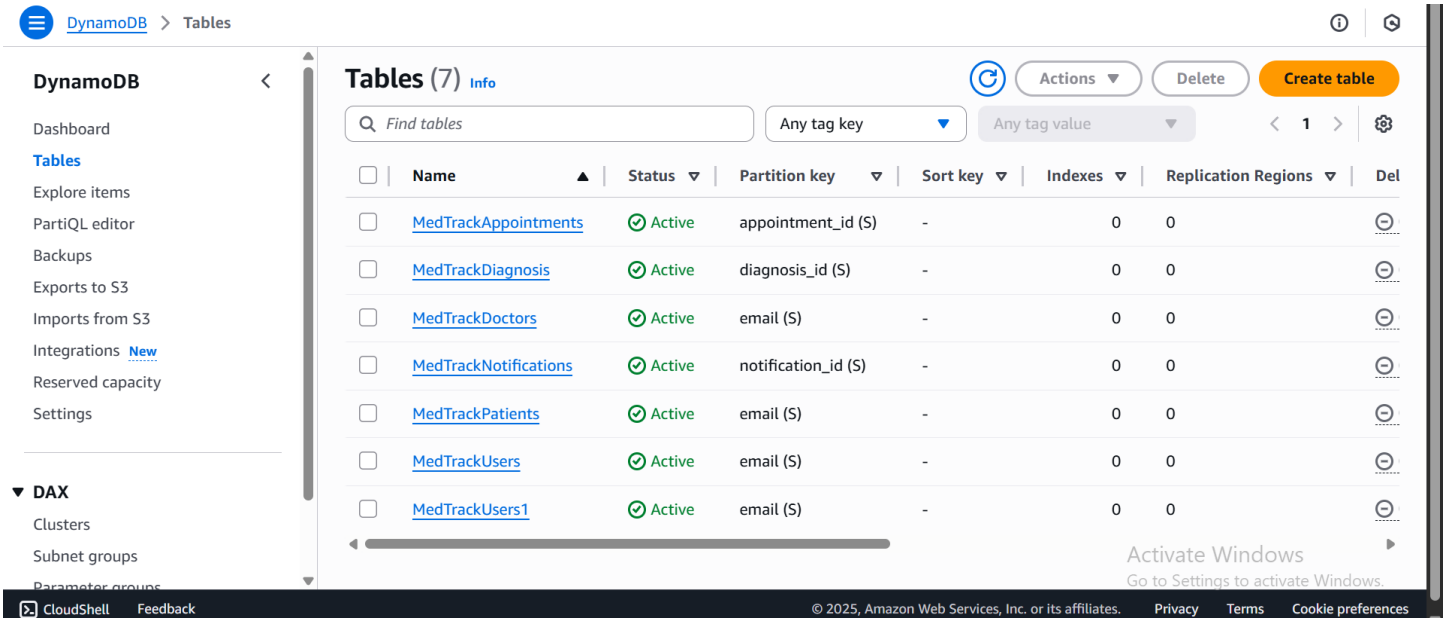
You can add 50 more tags.

Cancel Create table



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

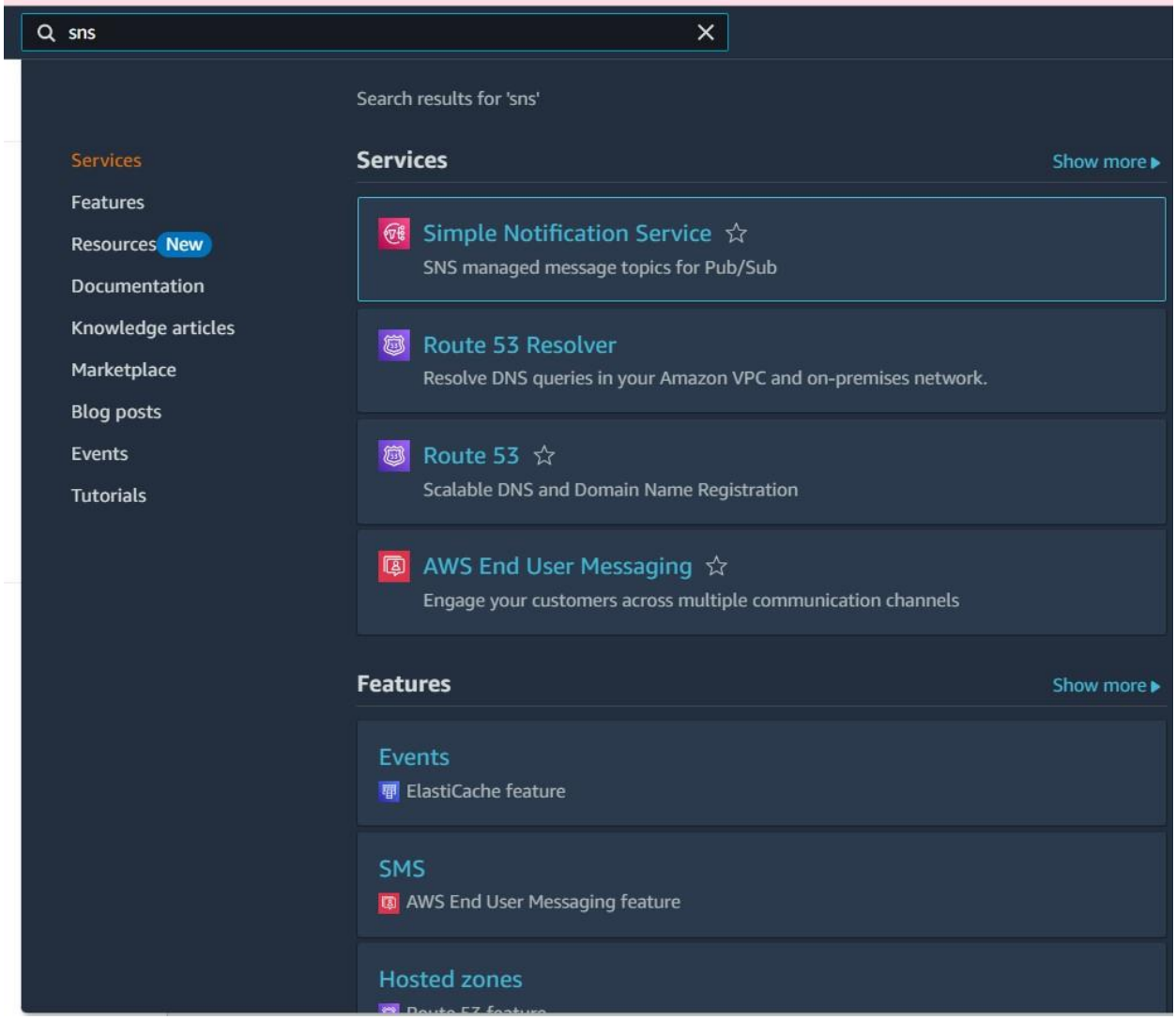

aws
Search
[Alt+S]
DynamoDB
Tables
Create table
Create table
Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.
Table name
This will be used to identify your table.
AppointmentsTable
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).
Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
appointment_id
String
1 to 255 characters and case sensitive.
Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
Enter the sort key name
String
1 to 255 characters and case sensitive.

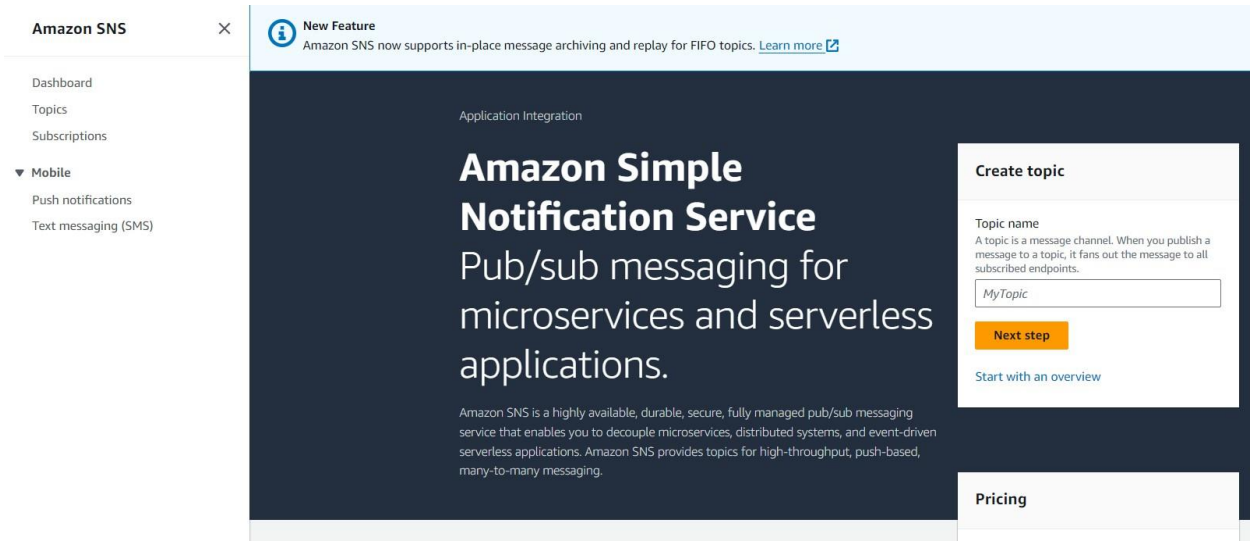


Milestone 3: SNS Notification Setup

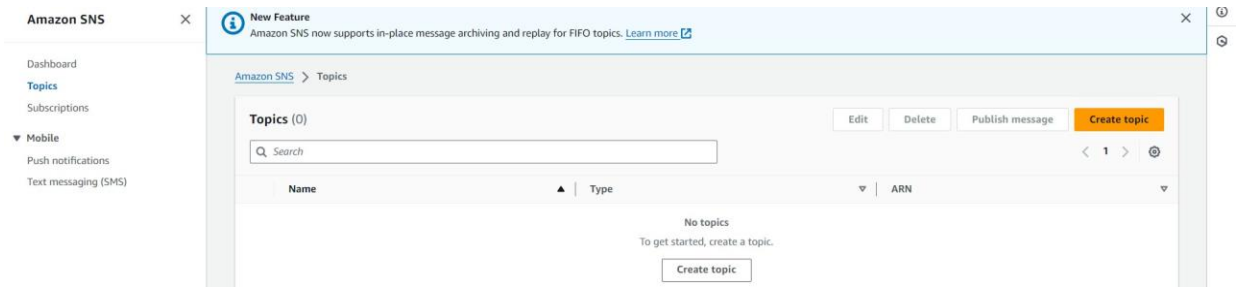
- **Activity 3.1: Create SNS topics for sending email notifications to patients and doctors.**

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

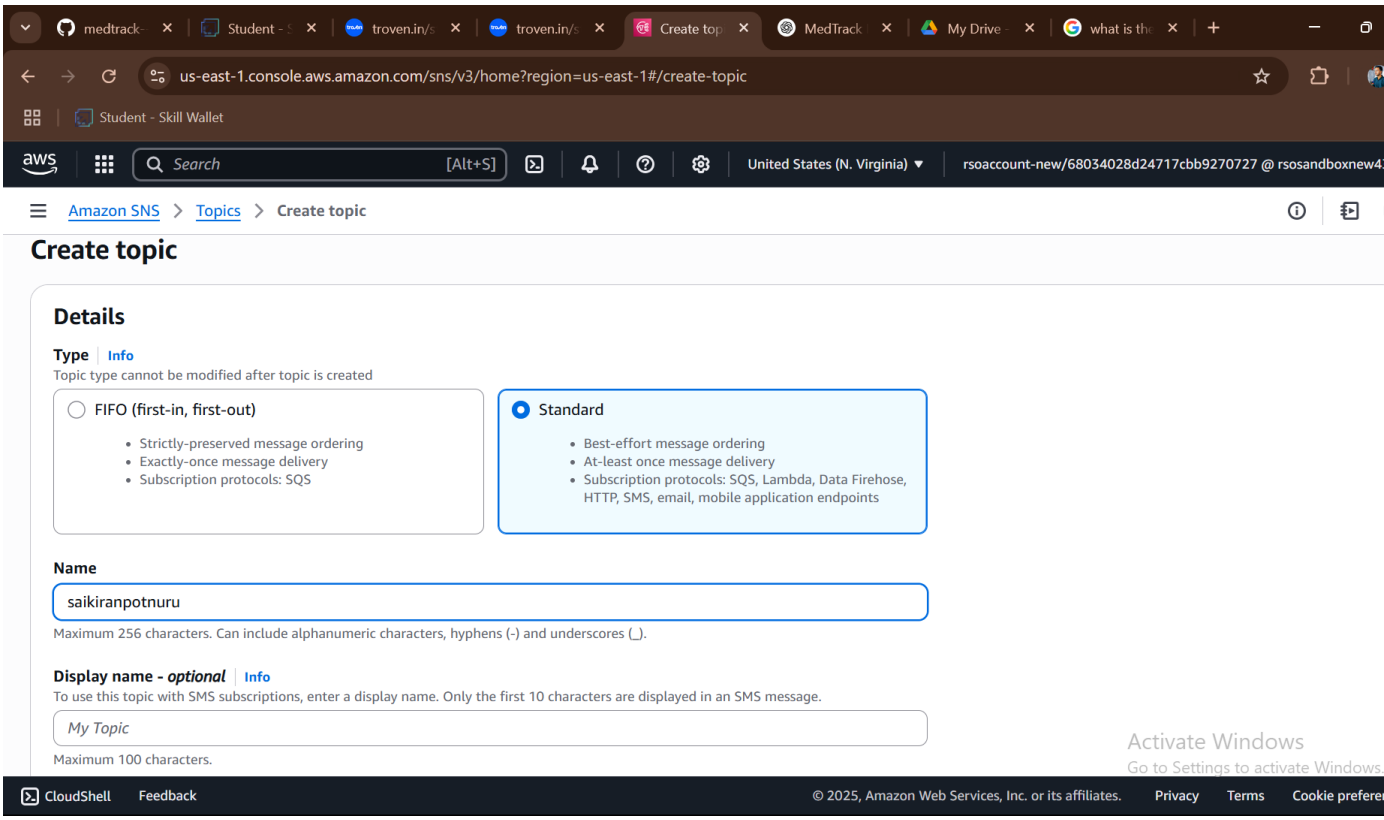


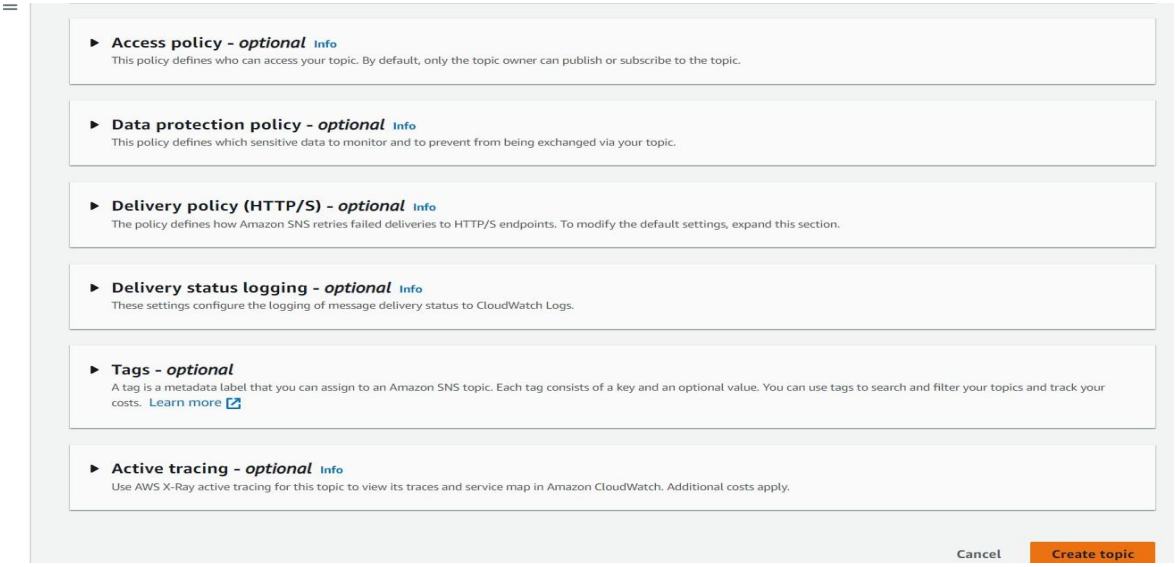


- Click on **Create Topic** and choose a name for the topic.

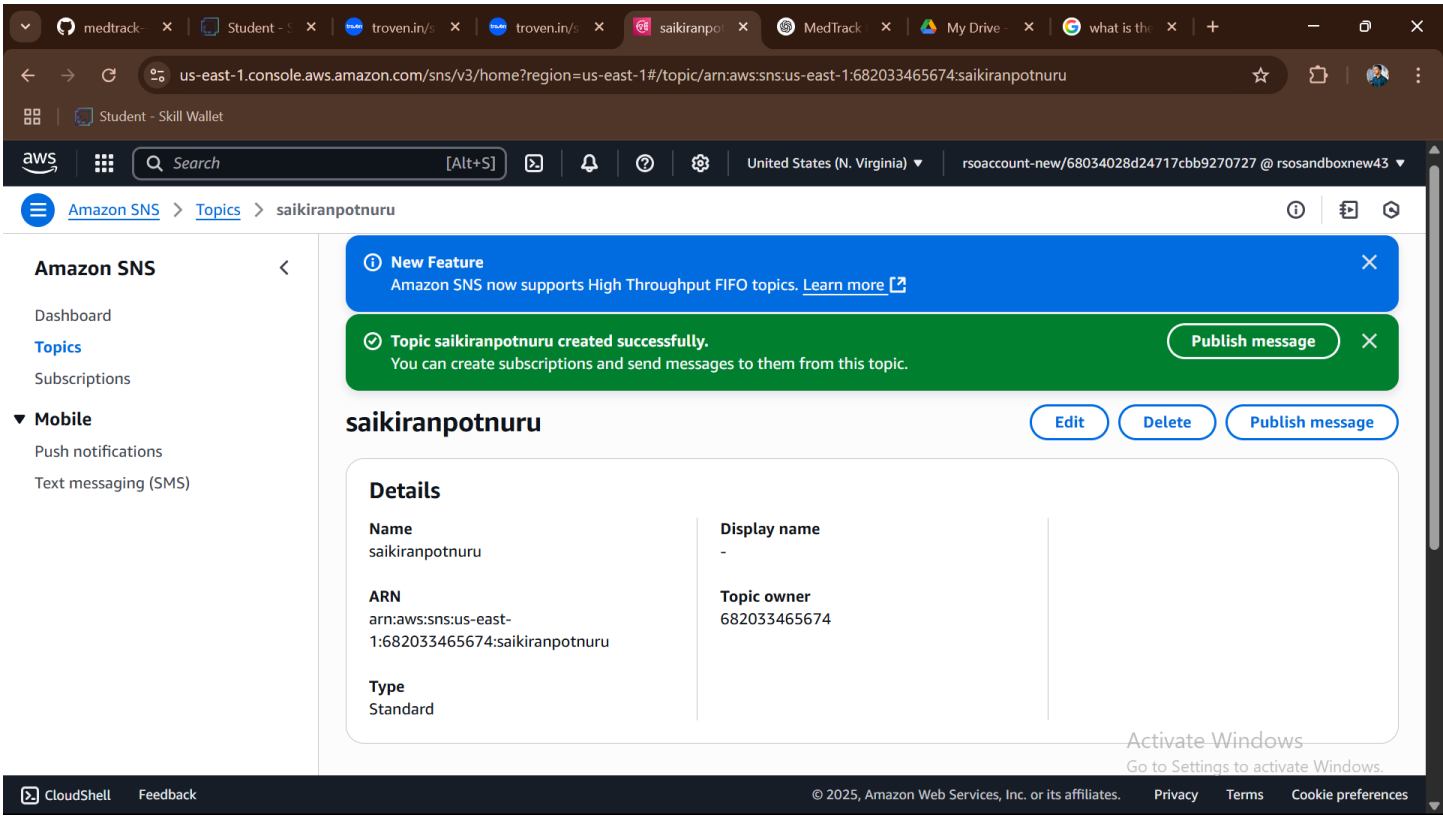


- Choose Standard type for general notification use cases and Click on Create Topic.


us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/create-topic
Student - Skill Wallet
Search [Alt+S]
United States (N. Virginia)
Amazon SNS > Topics > Create topic
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
saikiranpotnuru
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.
Activate Windows
Go to Settings to activate Windows.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms

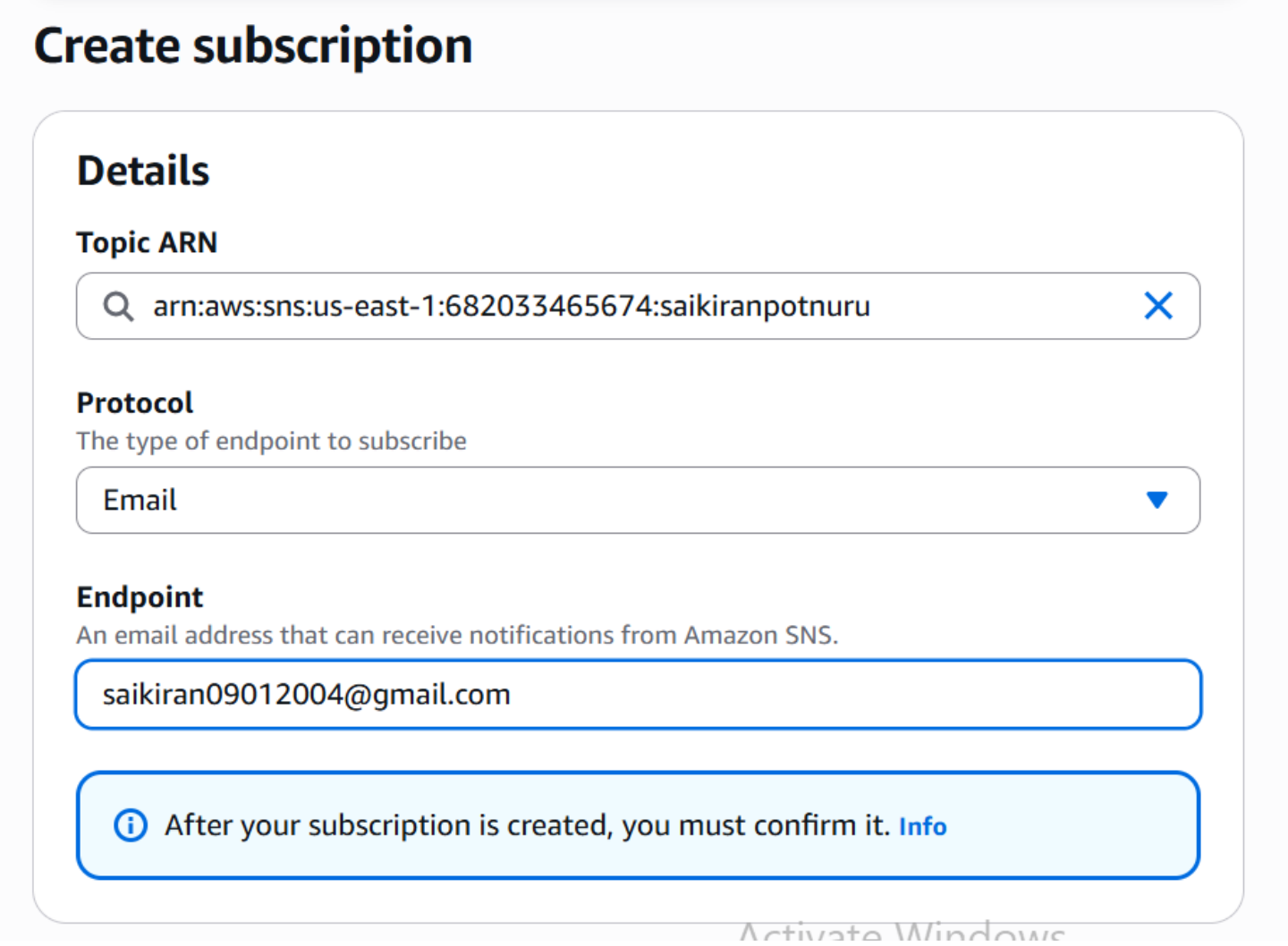


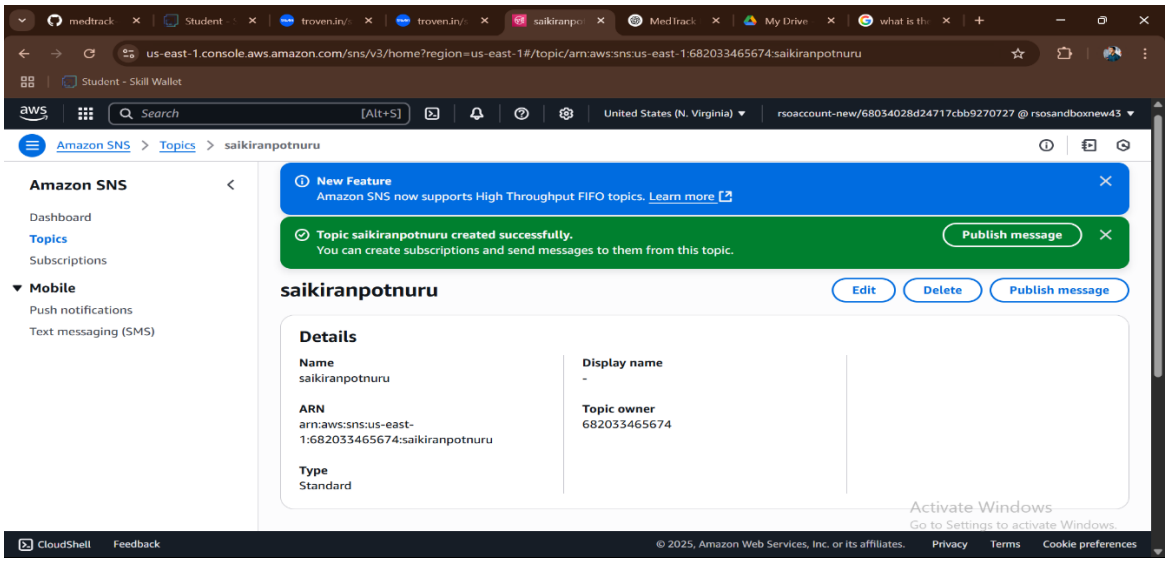
- Configure the SNS topic and note down the **Topic ARN**.



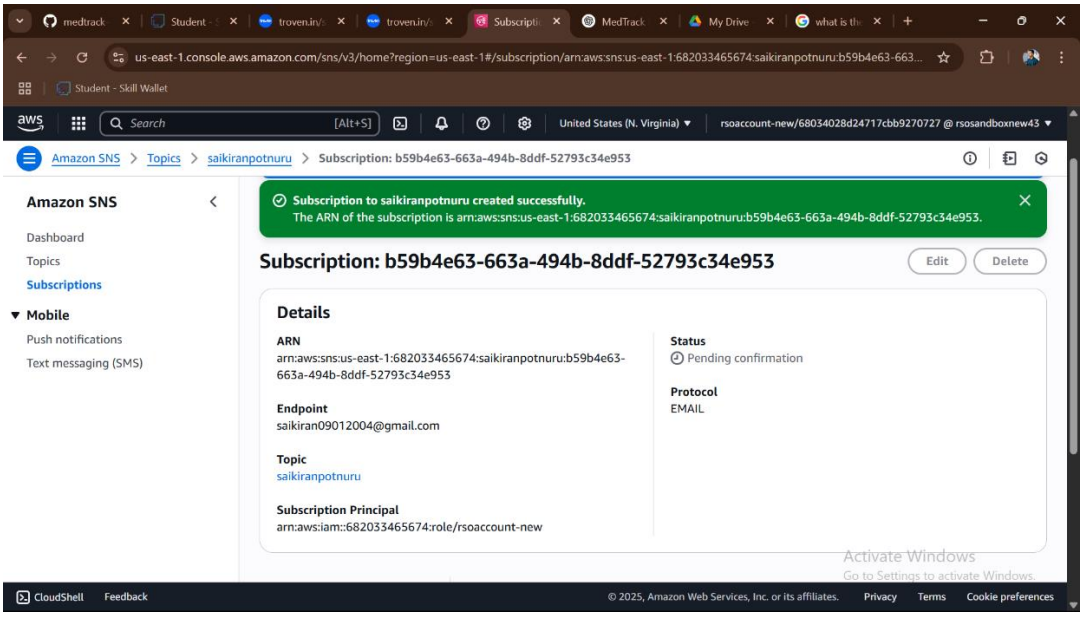
- **Activity 3.2: Subscribe users and Admin.**

- Subscribe users to this topic via email. When an Appointment is made, notifications will be sent to the corresponding doctor or patient emails.

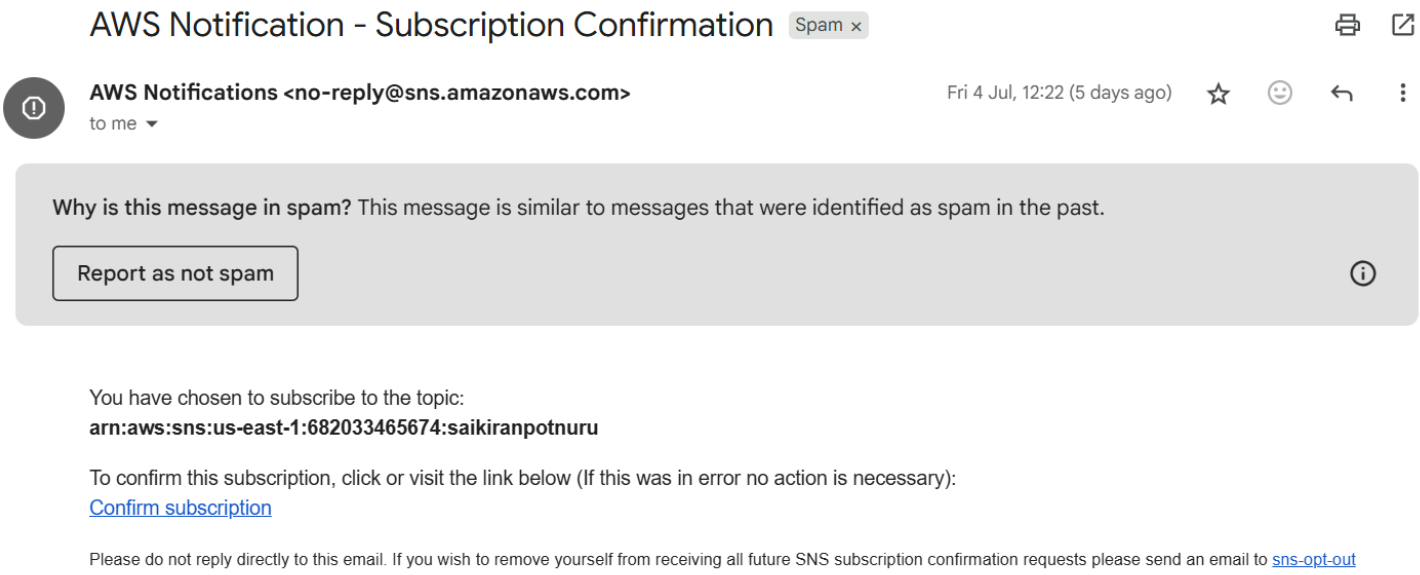


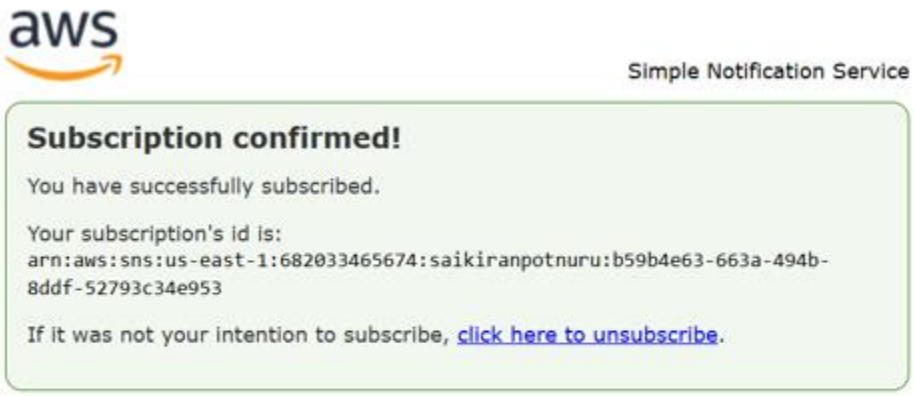


- After subscription request for the mail confirmation

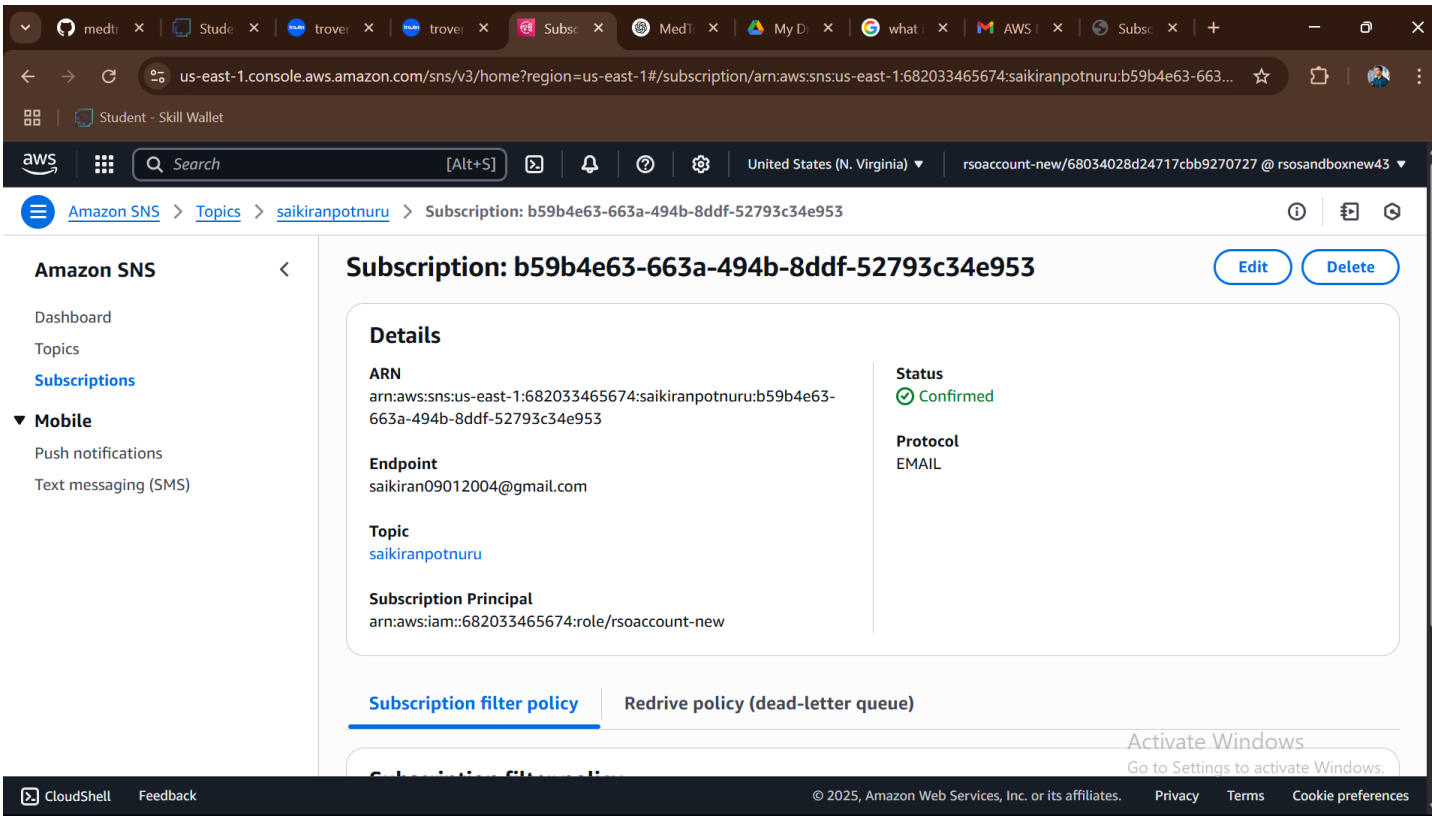


- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.



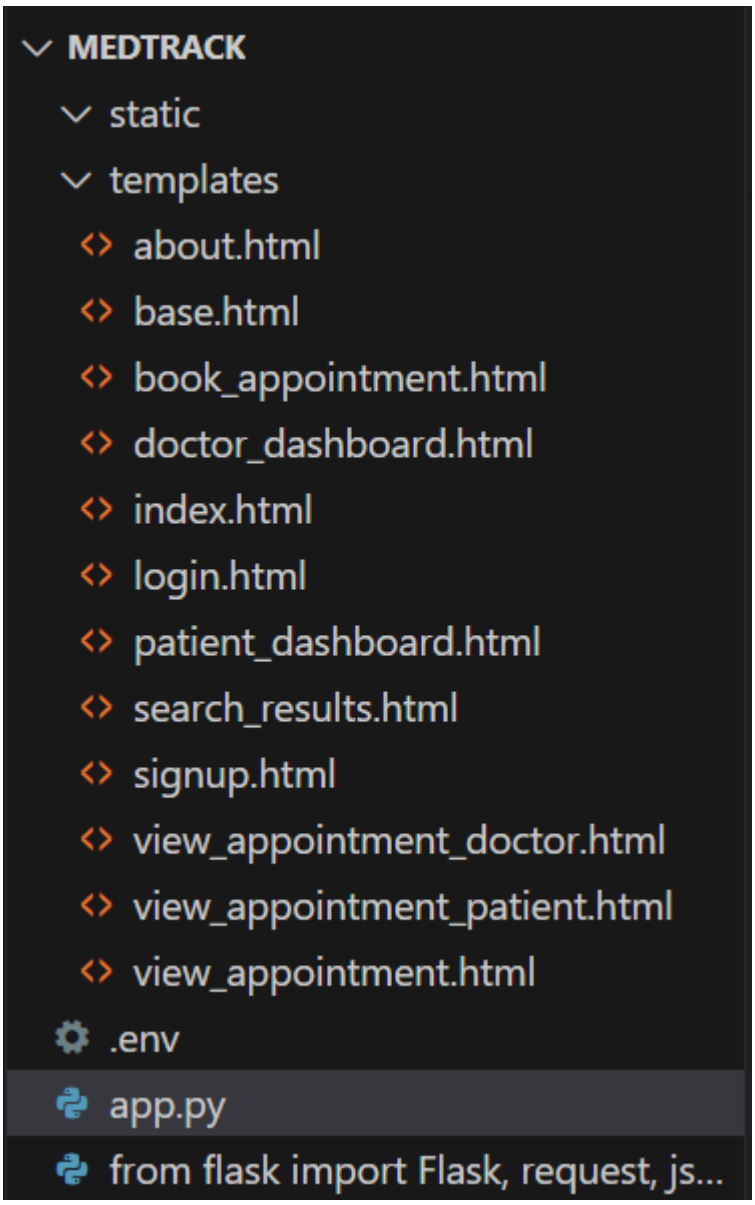


- Successfully done with the SNS mail subscription and setup, now store the ARN link.



Milestone 4:Backend Development and Application Setup

- **Activity 4.1: Develop the backend using Flask**
    - File Explorer Structure

- 
MEDTRACK
static
templates
about.html
base.html
book_appointment.html
doctor_dashboard.html
index.html
login.html
patient_dashboard.html
search_results.html
signup.html
view_appointment_doctor.html
view_appointment_patient.html
view_appointment.html
.env
app.py
from flask import Flask, request, js...

Description: 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.

Description of the code :

- **Flask App Initialization**

```
from flask import Flask, render_template, request, redirect, url_for
import boto3
from boto3.dynamodb.conditions import Key
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from bcrypt import hashpw, gensalt, checkpw
```

Description: import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification

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

Description: initialize the Flask application instance using Flask(__name__) to start building the web app.

- **Dynamodb Setup:**

```
# DynamoDB Table Names aligned with ER diagram
USERS_TABLE_NAME = os.environ.get('USERS_TABLE_NAME', 'MedTrackUsers')
DOCTORS_TABLE_NAME = os.environ.get('DOCTORS_TABLE_NAME', 'MedTrackDoctors')
PATIENTS_TABLE_NAME = os.environ.get('PATIENTS_TABLE_NAME', 'MedTrackPatients')
APPOINTMENTS_TABLE_NAME = os.environ.get('APPOINTMENTS_TABLE_NAME', 'MedTrackAppointments')
DIAGNOSIS_TABLE_NAME = os.environ.get('DIAGNOSIS_TABLE_NAME', 'MedTrackDiagnosis')
NOTIFICATIONS_TABLE_NAME = os.environ.get('NOTIFICATIONS_TABLE_NAME', 'MedTrackNotifications')
```

Description: initialize the DynamoDB resource for the ap-south-1 region and set up access to the Users and Requests tables for storing user details and book requests.

- **SNS Connection**

```python
def send_email_notification(to_email, subject, body):
    if not ENABLE_EMAIL or not SENDER_EMAIL:
        logger.info(f"Email notification would be sent: {subject}")
        return True

    try:
        msg = MIMEMultipart()
        msg['From'] = SENDER_EMAIL
        msg['To'] = to_email
        msg['Subject'] = subject
        msg.attach(MIMEText(body, 'plain'))

        server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        server.starttls()
        server.login(SENDER_EMAIL, SENDER_PASSWORD)
        server.send_message(msg)
        server.quit()

        logger.info(f"Email sent successfully to {to_email}")
        return True
    except Exception as e:
        logger.error(f"Failed to send email: {e}")
        return False
```

Description: Configure **SNS** to send notifications when a Appointment is booked 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.

- **Routes for Web Pages**
    - **Index Route:**

```python
# Route: Select role
@app.route('/')
def index():
    return render_template('index.html')
```

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

- **Signu Route:**

```python
@app.route('/signup/<role>', methods=['GET', 'POST'])
def signup(role):
    if role not in ('patient', 'doctor'):
        flash('Invalid role selected.', 'danger')
        return redirect(url_for('index'))

    if request.method == 'POST':
        name = request.form.get('name')
        email = request.form.get('email')
        password = request.form.get('password')

        if not name or not email or not password:
            flash('All fields are required.', 'warning')
            return render_template('signup.html', role=role)

        # Check if user already exists
        if dynamodb:
            user_table = get_users_table()
            response = user_table.get_item(Key={'email': email})
            if 'Item' in response:
                flash('User already exists.', 'danger')
                return render_template('signup.html', role=role)
        else:
            #if email in local_db['users']:
                #flash('User already exists.', 'danger')
                #return render_template('signup.html', role=role)
            flash('User already exists.', 'danger')
            return render_template('signup.html', role=role)

        # Create user
        user_id = str(uuid.uuid4())
        hashed_password = generate_password_hash(password)
        user_data = {
            'user_id': user_id,
            'name': name,
            'email': email,
            'password_hash': hashed_password,
            'role': role,
            'created_at': datetime.now().isoformat(),
            'is_active': True
        }

        if dynamodb:
            user_table.put_item(Item=user_data)
        else:
            #local_db['users'][email] = user_data
            flash('DynamoDB is required for production. Please enable it.', 'danger')
            return render_template('signup.html', role=role)


        flash('Signup successful! Please log in.', 'success')
        return redirect(url_for('login', role=role))

    return render_template('signup.html', role=role)
```

Description: 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)**:

```python
login_attempts = {}  # Rate-limiting support
@app.route('/login/<role>', methods=['GET', 'POST'])
def login(role):
    if role not in ('patient', 'doctor'):
        flash("Invalid role.", "danger")
        return redirect(url_for('index'))

    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')

        if not email or not password:
            flash('Email and password are required.', 'warning')
            return render_template('login.html', role=role)

        # Rate-limiting check
        client_ip = request.remote_addr
        if client_ip in login_attempts:
            attempt_data = login_attempts[client_ip]
            if attempt_data['count'] >= 5:
                if datetime.now() - attempt_data['last_attempt'] < timedelta(minutes=15):
                    flash('Too many login attempts. Try again later.', 'danger')
                    return render_template('login.html', role=role)
                else:
                    login_attempts[client_ip] = {'count': 0, 'last_attempt': datetime.now()}

        # Fetch user data
        user_data = None
        if dynamodb:
            user_table = get_users_table()
            try:
```

Description: 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

- **doctor and patient dashboards routes:**

```python
""" def get_patient_dashboard_data(user_email):···

def get_patient_dashboard_data(user_email):···


"""def get_doctor_dashboard_data(user_email):···


def get_doctor_dashboard_data(user_email):···


@app.route('/patient_dashboard')
@login_required(role='patient')
def patient_dashboard():···

@app.route('/doctor_dashboard')
@login_required(role='doctor')
def doctor_dashboard():···
# ...existing code...
```

Description: 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 like Mathematics or English.

- **Book Appointment Routes:**

```python
def book_appointment():
    if request.method == 'POST':
        doctor_email = request.form['doctor']
        date = request.form['date']
        time = request.form['time']
        title = request.form.get('title', 'Consultation')
        location = request.form.get('location', 'Office 203')
        color = request.form.get('color', '#3498db')
        if dynamodb:
            user_table = get_users_table()
            appointments_table = get_appointments_table()
            doctor_resp = user_table.get_item(Key={'email': doctor_email})
            doctor_name = doctor_resp.get('Item', {}).get('name', 'Doctor')
            patient_email = session['user']
            patient_resp = user_table.get_item(Key={'email': patient_email})
            patient_name = patient_resp.get('Item', {}).get('name', 'Patient')
            appointment_id = str(uuid.uuid4())
            appointment = {
                'appointment_id': appointment_id,
                'patient': patient_email,
                'patient_name': patient_name,
                'doctor': doctor_email,
                'doctor_name': doctor_name,
                'title': title,
                'date': date,
                'time': time,
                'location': location,
                'color': color
            }
            appointments_table.put_item(Item=appointment)
            flash('Appointment booked successfully!', 'success')
```

⊗ 0 ⚠ 0 saikiranpo

Description: 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.

Logout Route:

```python
@app.route('/api/logout', methods=['POST'])
@login_required(api=True)  # Ensures API-style authentication response
def logout():
    try:
        user_email = session.get('email')
        user_name = session.get('name')

        # Clear session
        session.clear()

        logger.info(f"User logged out: {user_email}")

        # Optionally send notification (email or SNS)
        message = f"{user_name} has logged out from MediTrack."
        send_sns_notification(message)
        # Optionally: send_email_notification(user_email, "Logout Alert", message)

        return jsonify({'message': 'Logged out successfully'}), 200

    except Exception as e:
        logger.error(f"Logout failed: {e}")
        return jsonify({'error': 'Logout failed'}), 500
```

Description: define /Logout route to render the exit.html page when the user chooses to leave or close the application.

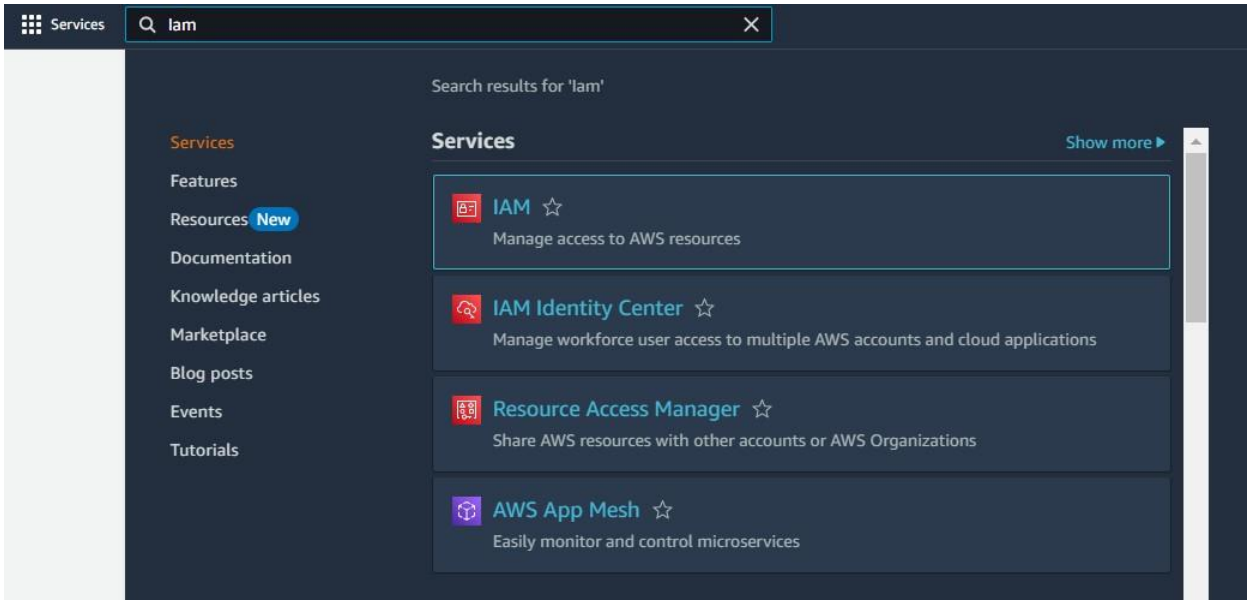
Deployment Code:

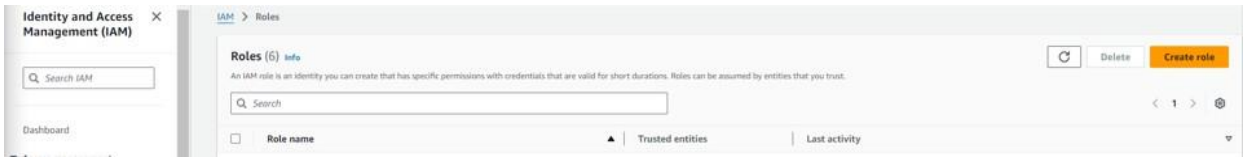
```
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

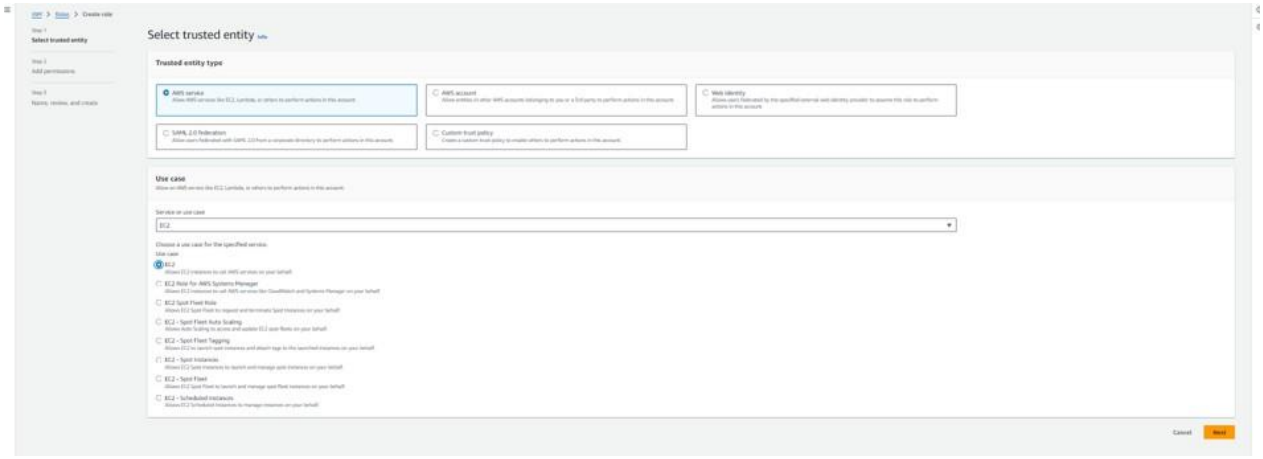
Description: start the Flask server to listen on all network interfaces (`0.0.0.0`) at port `80` with debug mode enabled for development and testing.

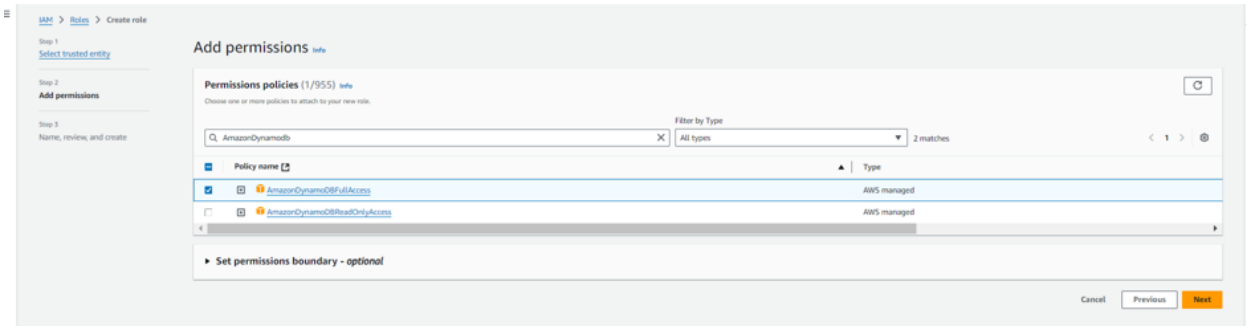
Milestone 5: IAM Role Setup

- **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.





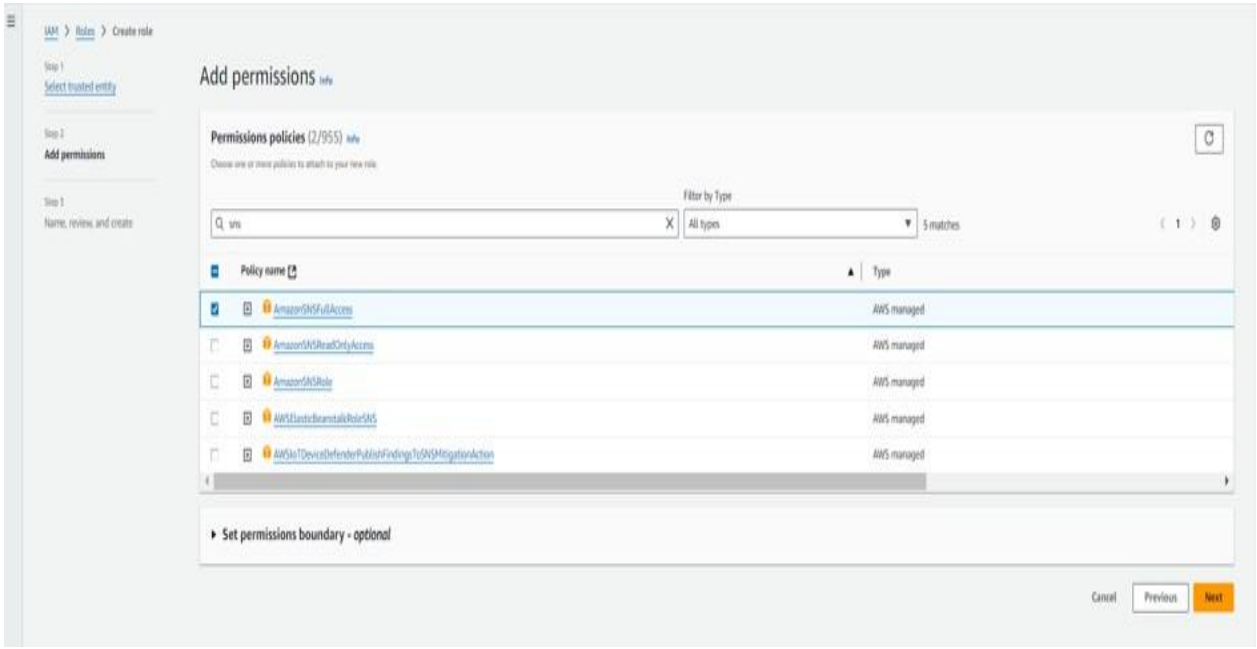


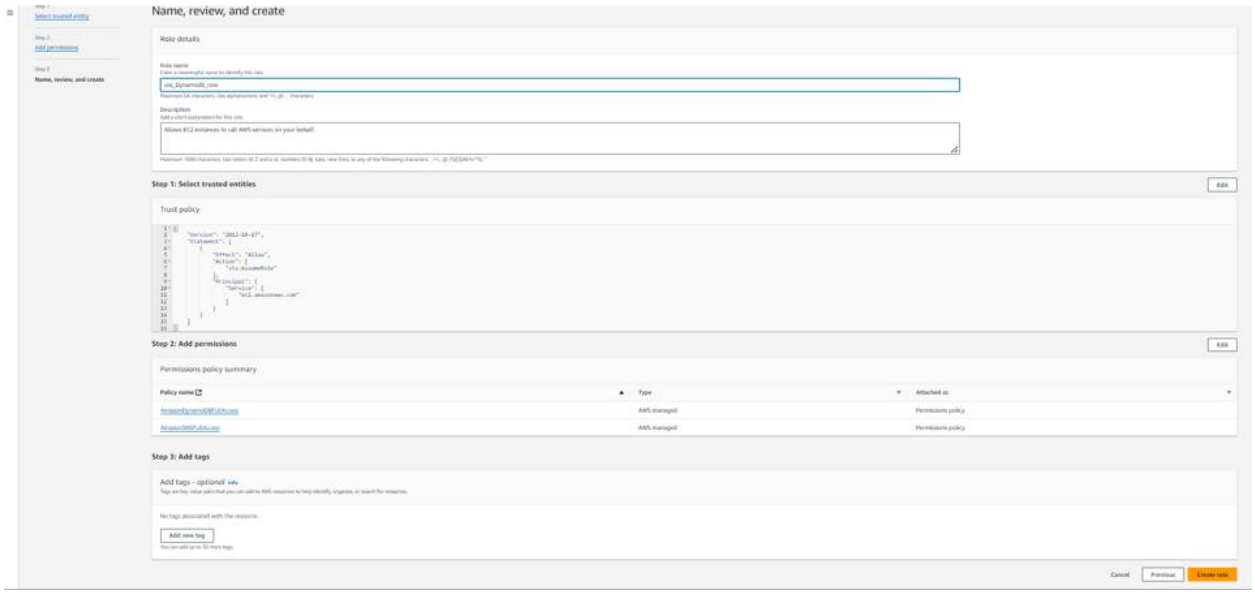


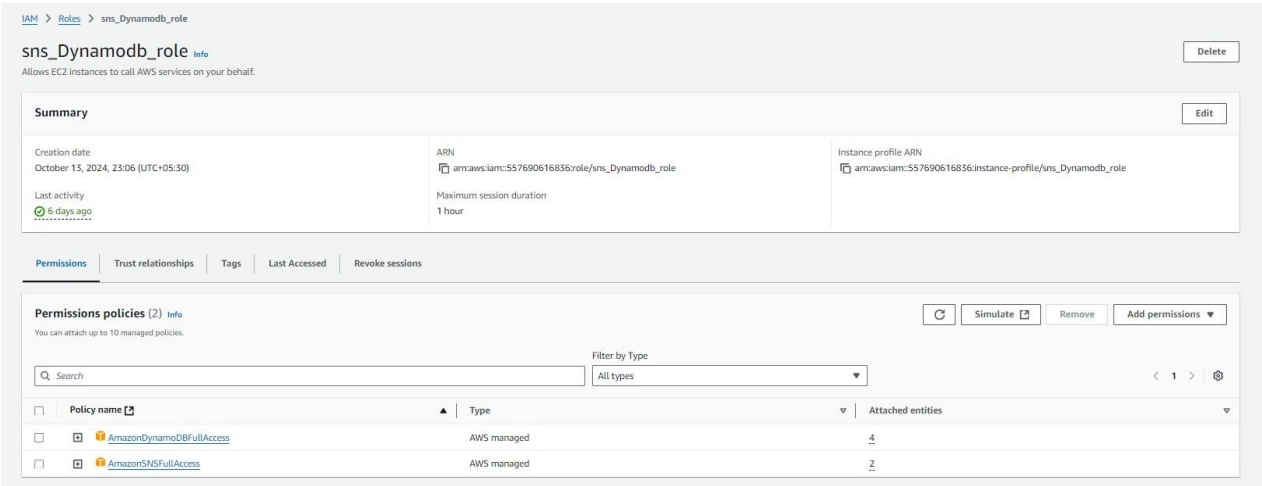
- **Activity 5.2: Attach Policies.**

  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.

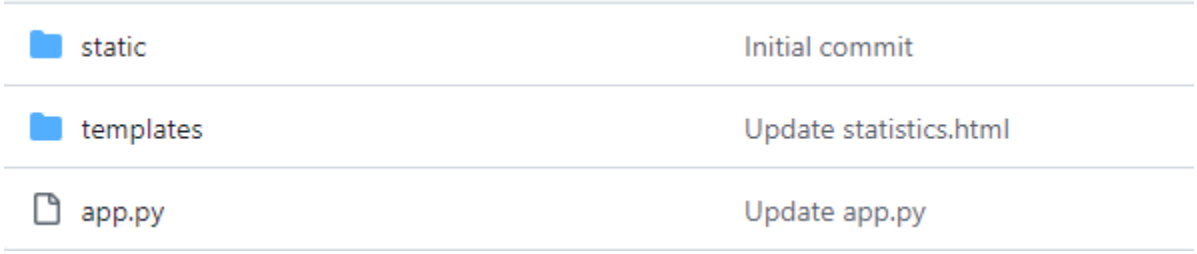

IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (2/955) Info
Choose one or more policies to attach to your new role.
Filter by Type
sns
All types
5 matches
Policy name
Type
AmazonSNSFullAccess
AWS managed
AmazonSNSReadOnlyAccess
AWS managed
AmazonSNSRole
AWS managed
AWSElasticBeanstalkRoleSNS
AWS managed
AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction
AWS managed
Set permissions boundary - optional
Cancel
Previous
Next

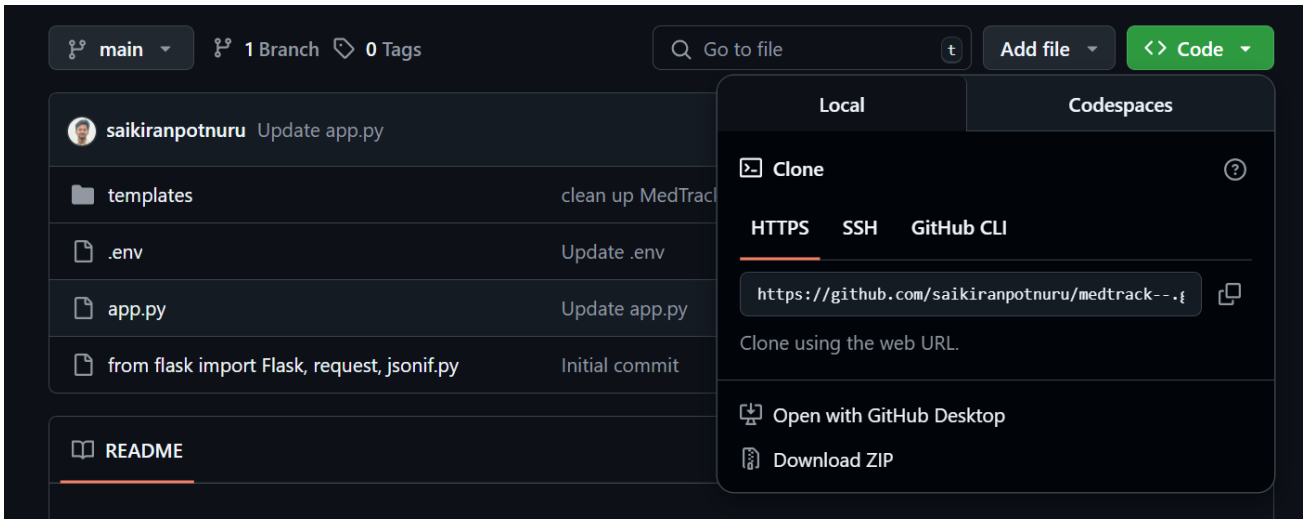

Name, review, and create
Role details
Step 1: Select trusted entities
Trust policy
Step 2: Add permissions
Permissions policy summary
Step 3: Add tags
Add tags - optional
Add new tag
Cancel
Previous
Create role



Milestone 6: EC2 Instance Setup

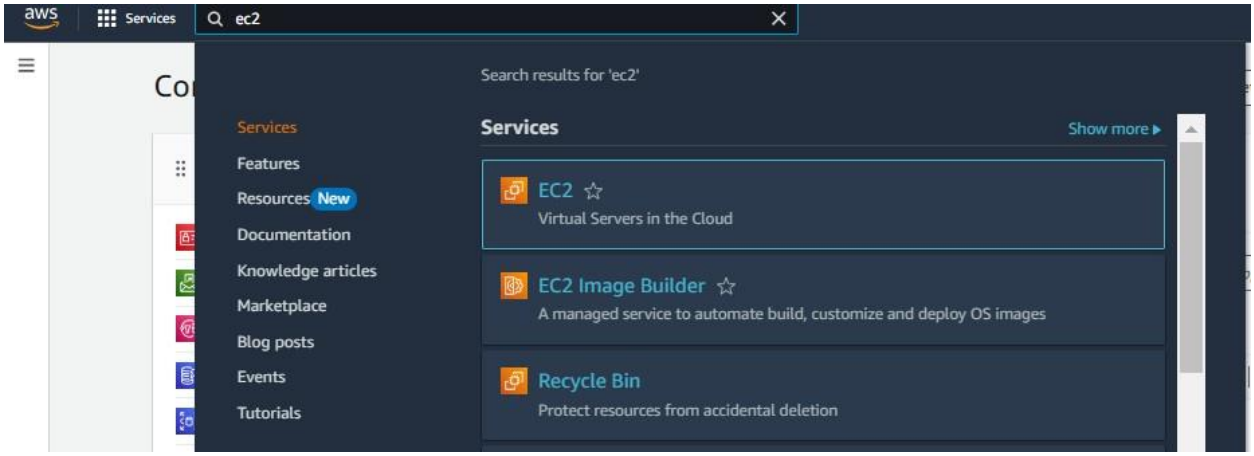
- **Note: Load your Flask app and Html files into GitHub repository.**



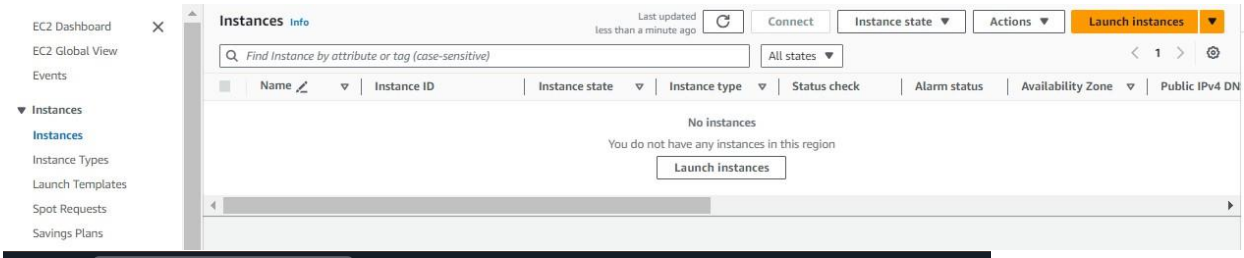


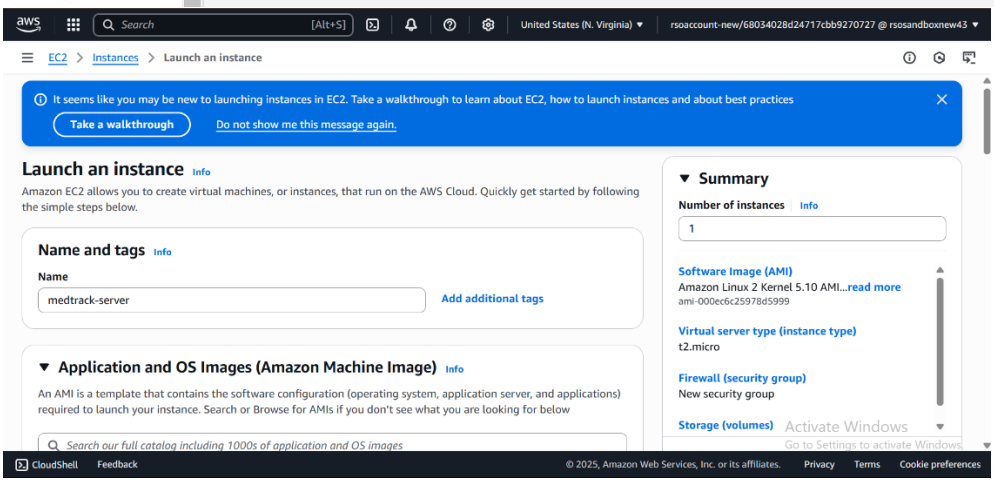
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

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

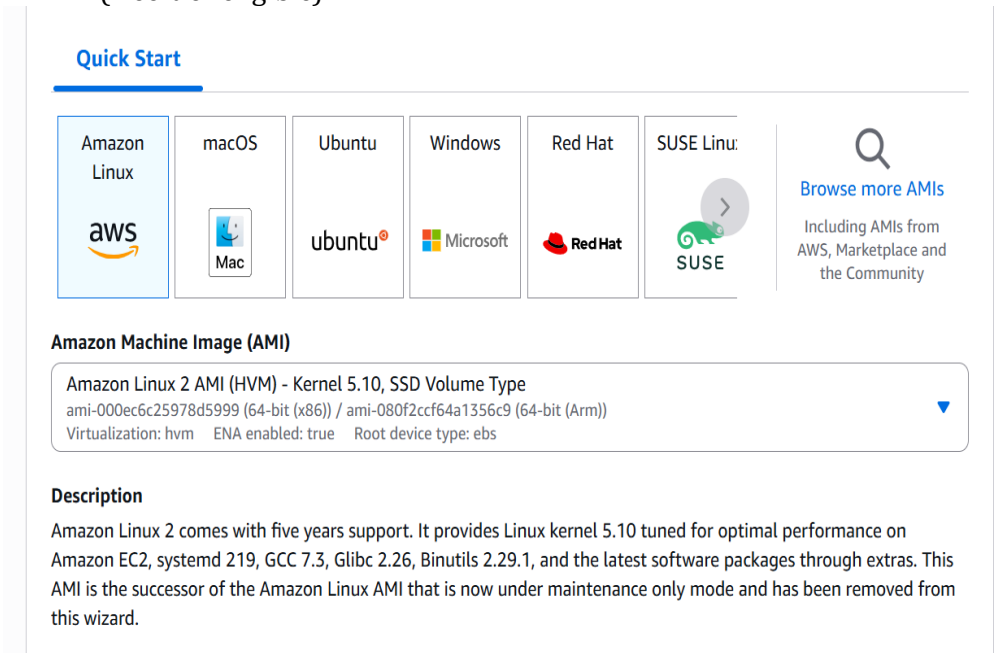


- Click on Launch instance to launch EC2 instance

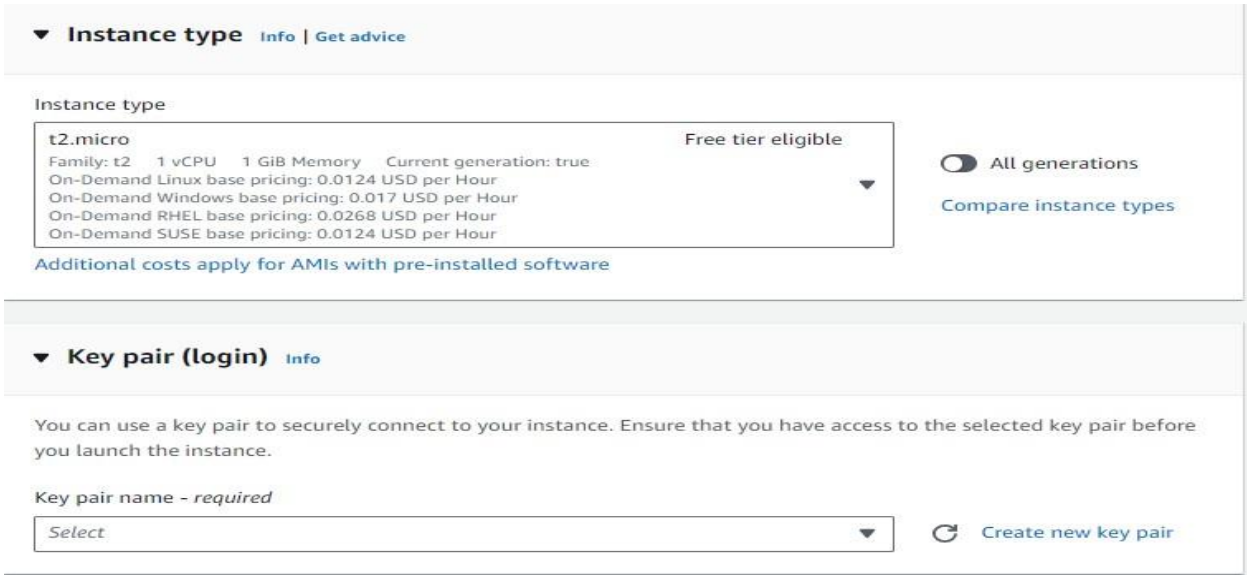


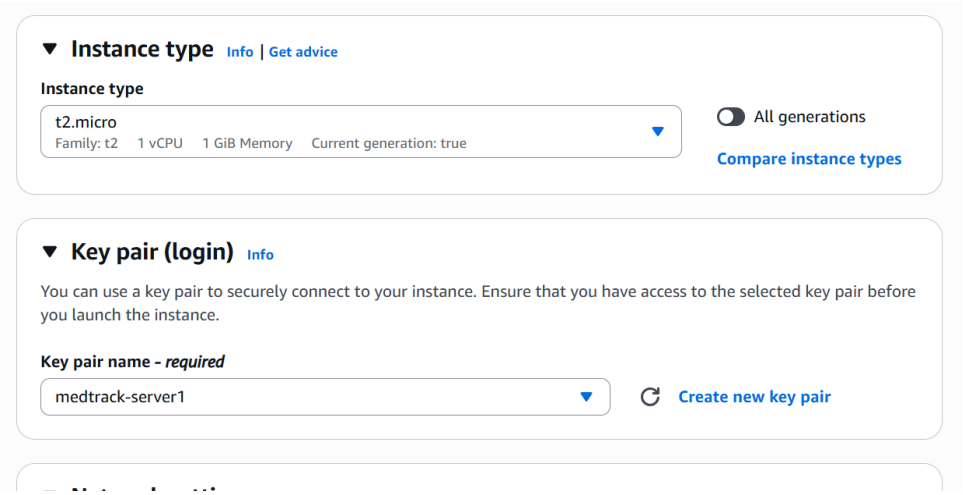


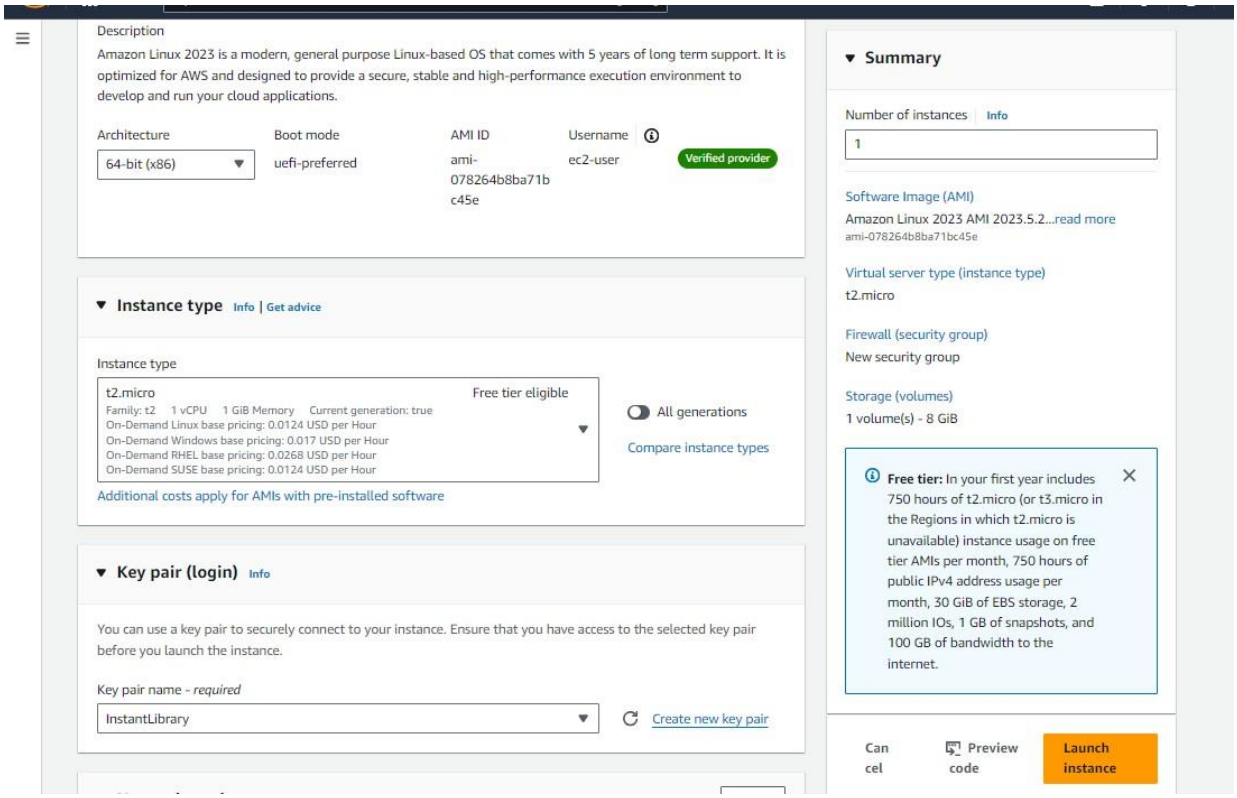
- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



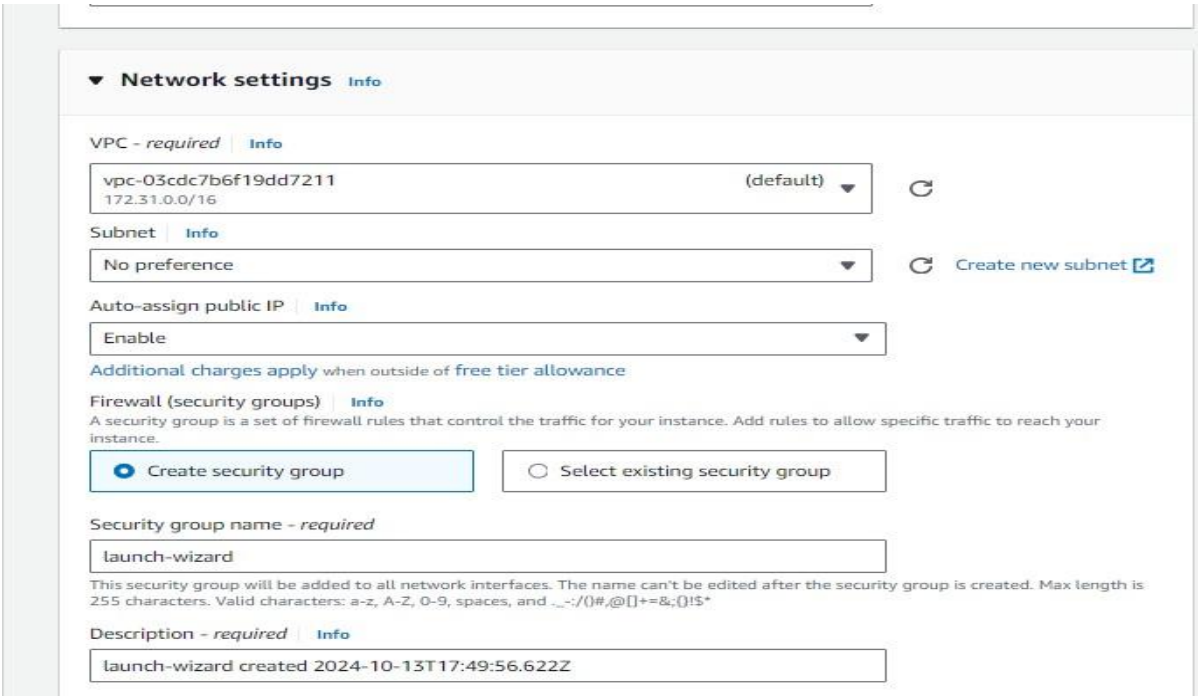
- Create and download the key pair for Server access.

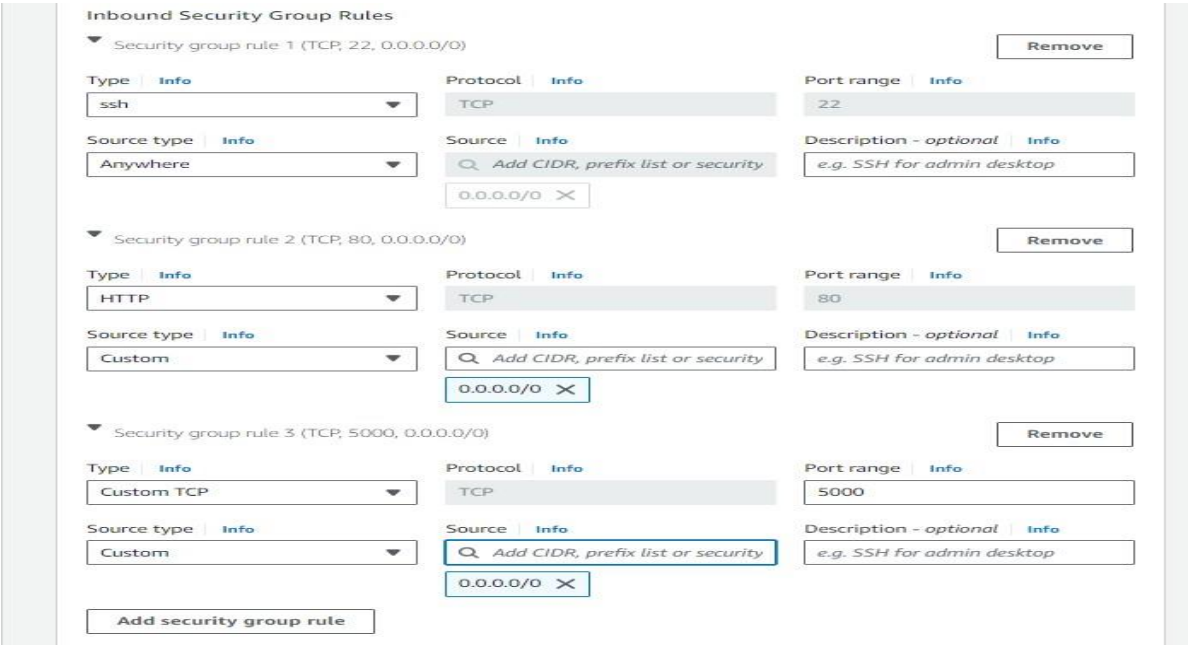


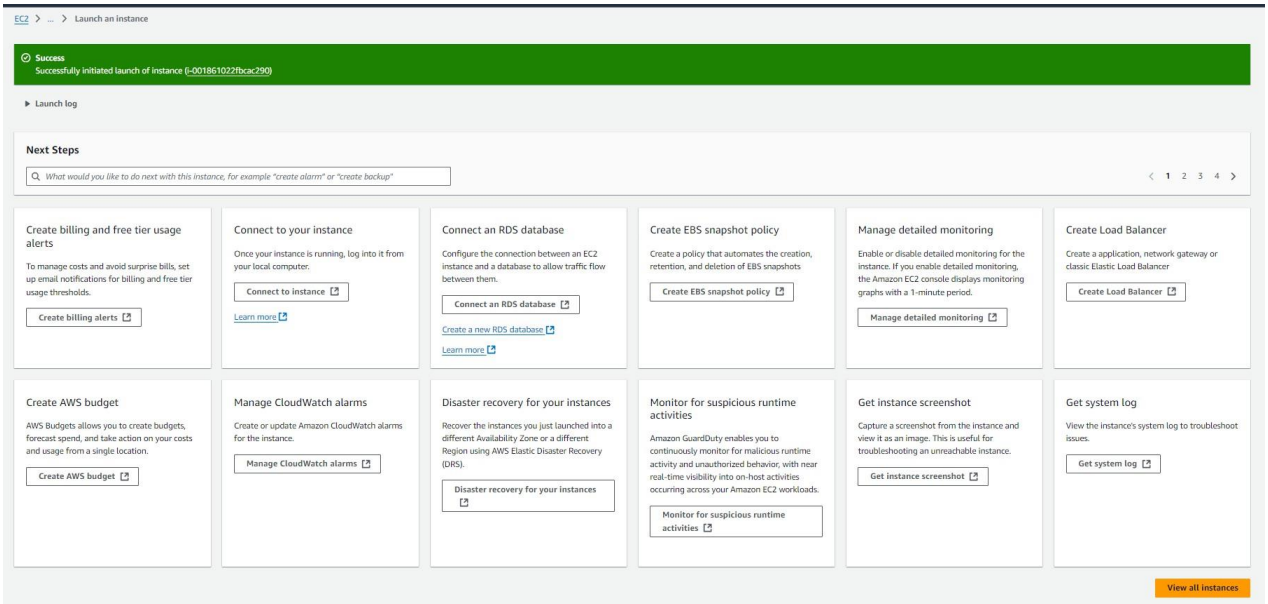



Description
Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.
Architecture
64-bit (x86)
Boot mode
uefi-preferred
AMI ID
ami-078264b8ba71bc45e
Username
ec2-user
Verified provider
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
InstantLibrary
Create new key pair
Summary
Number of instances Info
1
Software Image (AMI)
Amazon Linux 2023 AMI 2023.5.2...read more
ami-078264b8ba71bc45e
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
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 IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.
Cancel
Preview code
Launch instance

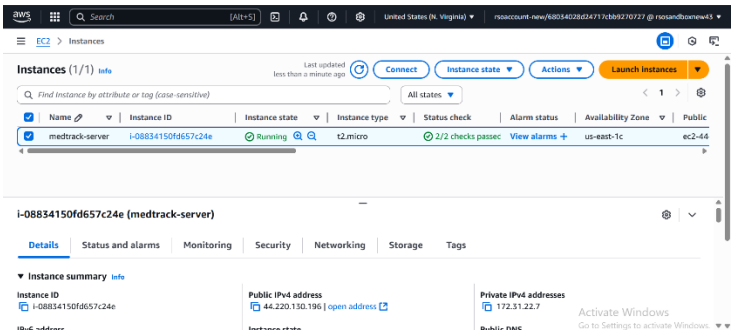
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**

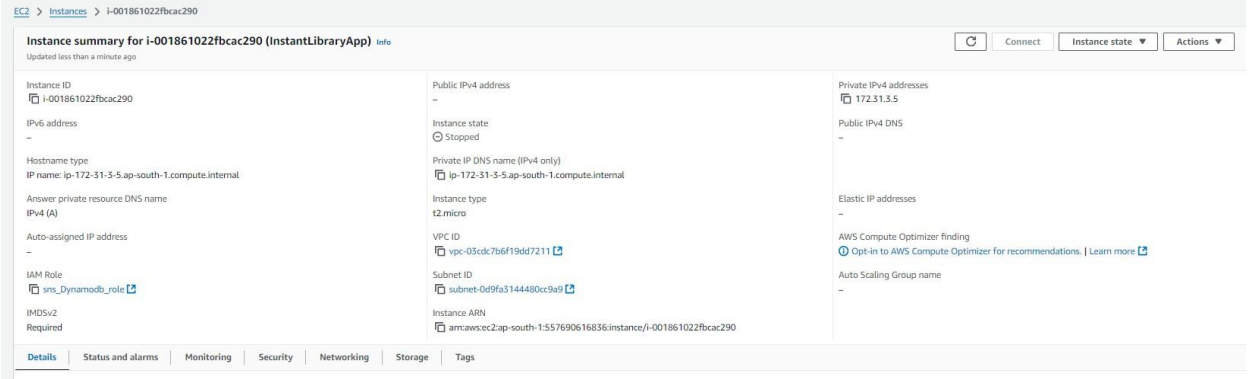


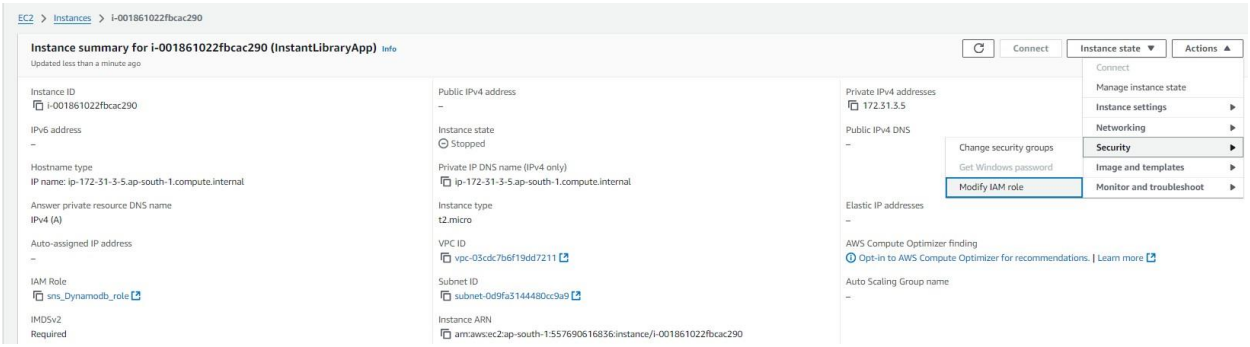


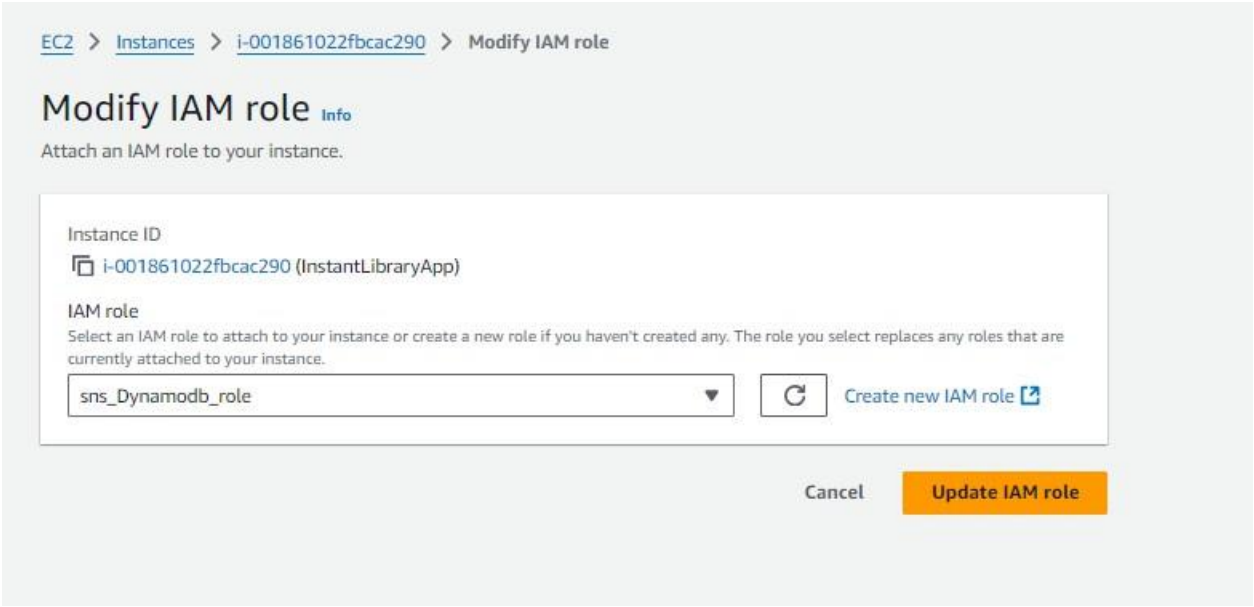


- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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

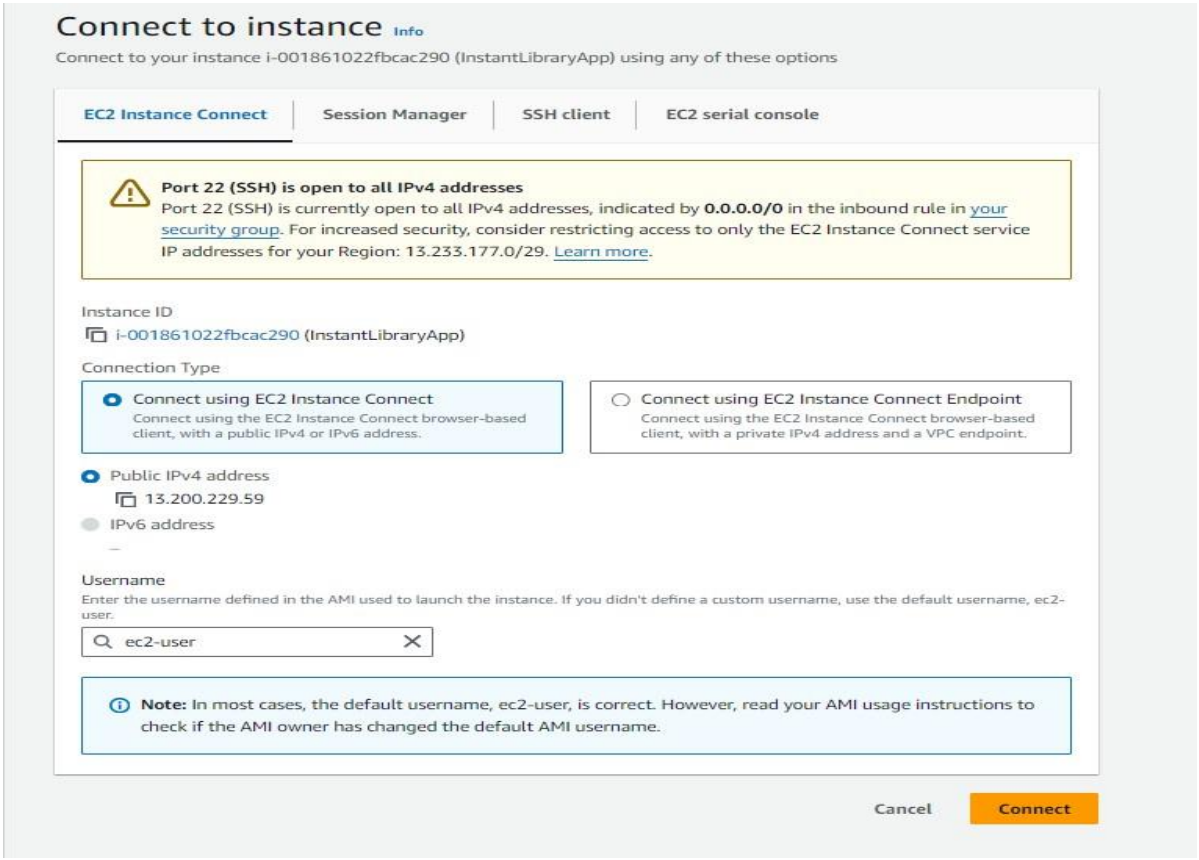


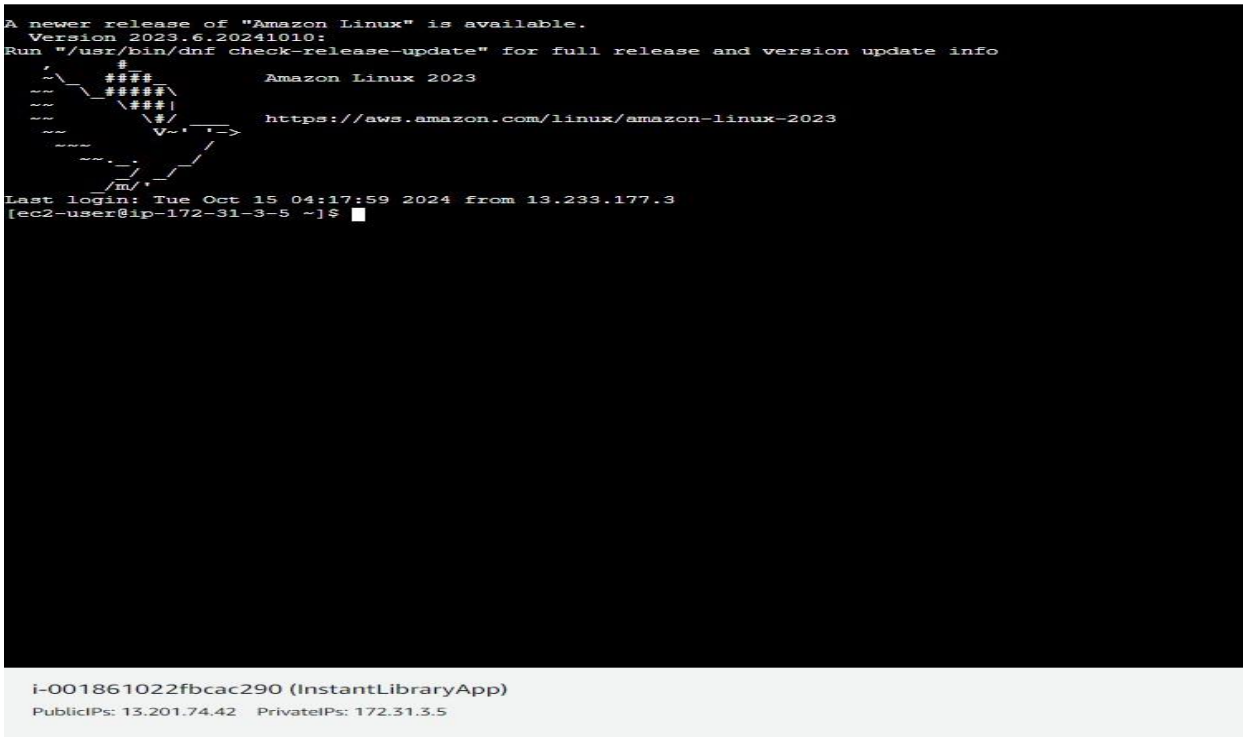






- Now connect the EC2 with the files


Connect to instance Info
Connect to your instance i-001861022fbcac290 (InstantLibraryApp) using any of these options
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Port 22 (SSH) is open to all IPv4 addresses
Port 22 (SSH) is currently open to all IPv4 addresses, indicated by 0.0.0.0/0 in the inbound rule in your security group. For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 13.233.177.0/29. Learn more.
Instance ID
i-001861022fbcac290 (InstantLibraryApp)
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
13.200.229.59
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.
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


A newer release of "Amazon Linux" is available.
Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$
i-001861022fbcac290 (InstantLibraryApp)
PublicIPs: 13.201.74.42 PrivateIPs: 172.31.3.5

Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

```
sudo yum update -y

sudo yum install python3 git

sudo pip3 install flask boto3
```

Verify Installations:

```
flask --version

git --version
```

Activity 7.2:Clone Your Flask Project from GitHub

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

Run: 'git clone https://github.com/your-github-username/your-repository-name.git'

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

here: 'git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git'

- This will download your project to the EC2 instance.

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

```
cd InstantLibrary
```

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

Run the Flask Application

```
sudo flask run --host=0.0.0.0 --port=80
```

```
A newer release of "Amazon Linux" is available.
  Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$ git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git
fatal: destination path 'InstantLibrary' already exists and is not an empty directory.
[ec2-user@ip-172-31-3-5 ~]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ flask run --host=0.0.0.0 --port=80
 * Debug mode: off
Permission denied
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[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
^C[ec2-user@ip-172-31-3-5 InstantLibrary]$
[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 -
```

i-001861022fbcac290 (InstantLibraryApp)

PublicIPs: 13.201.74.42 PrivateIPs: 172.31.3.5

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 -
```

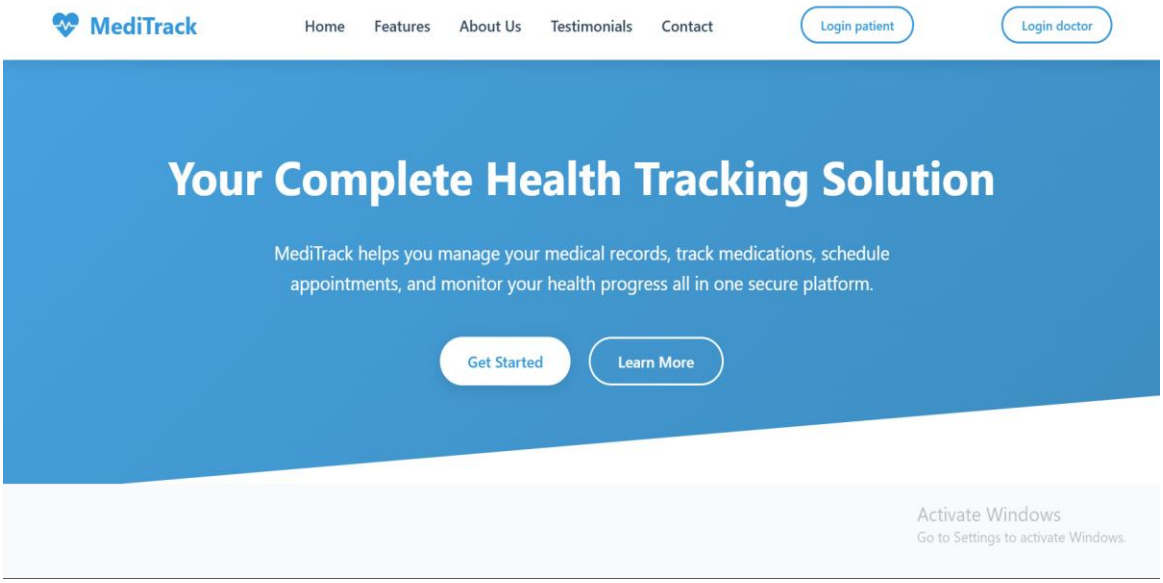
Access the website through:

PublicIPs: https://13.201.74.42/

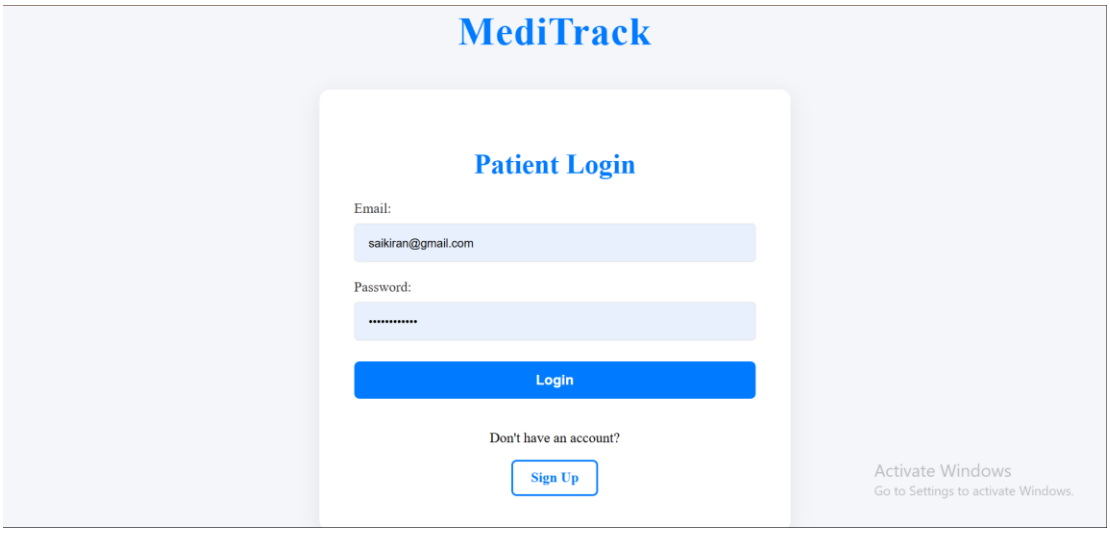
Milestone 8: Testing and Deployment

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

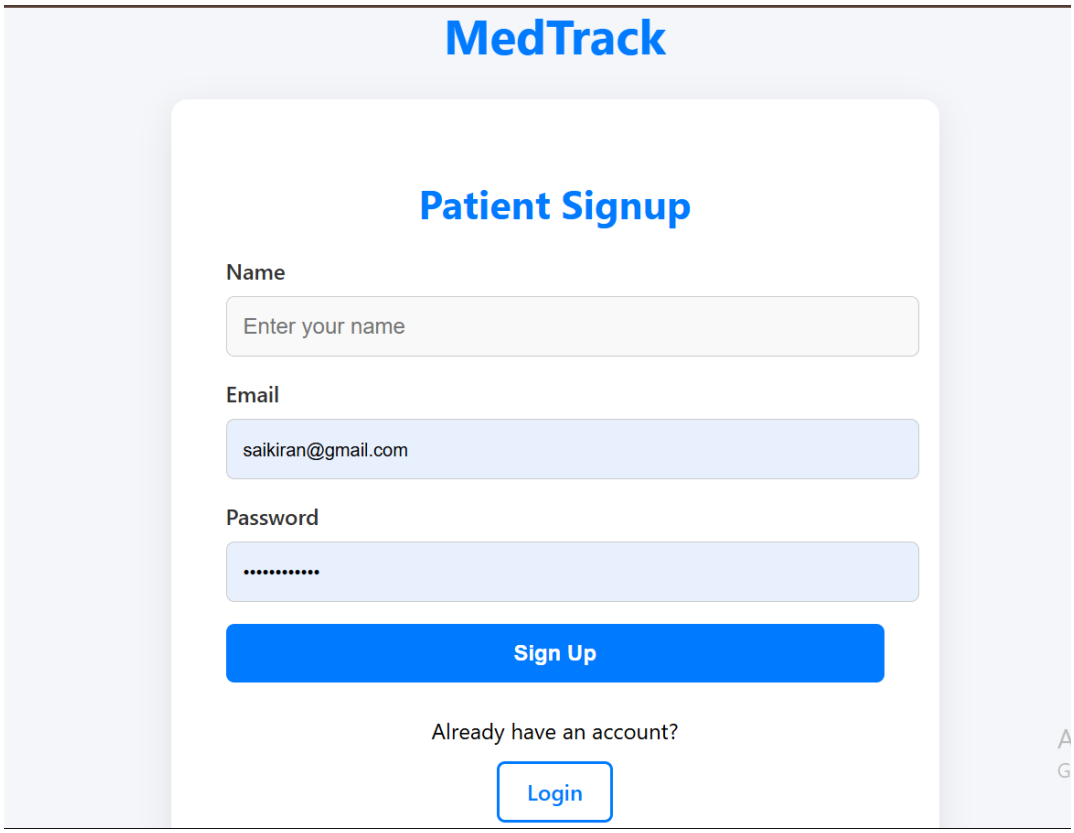
Index page:



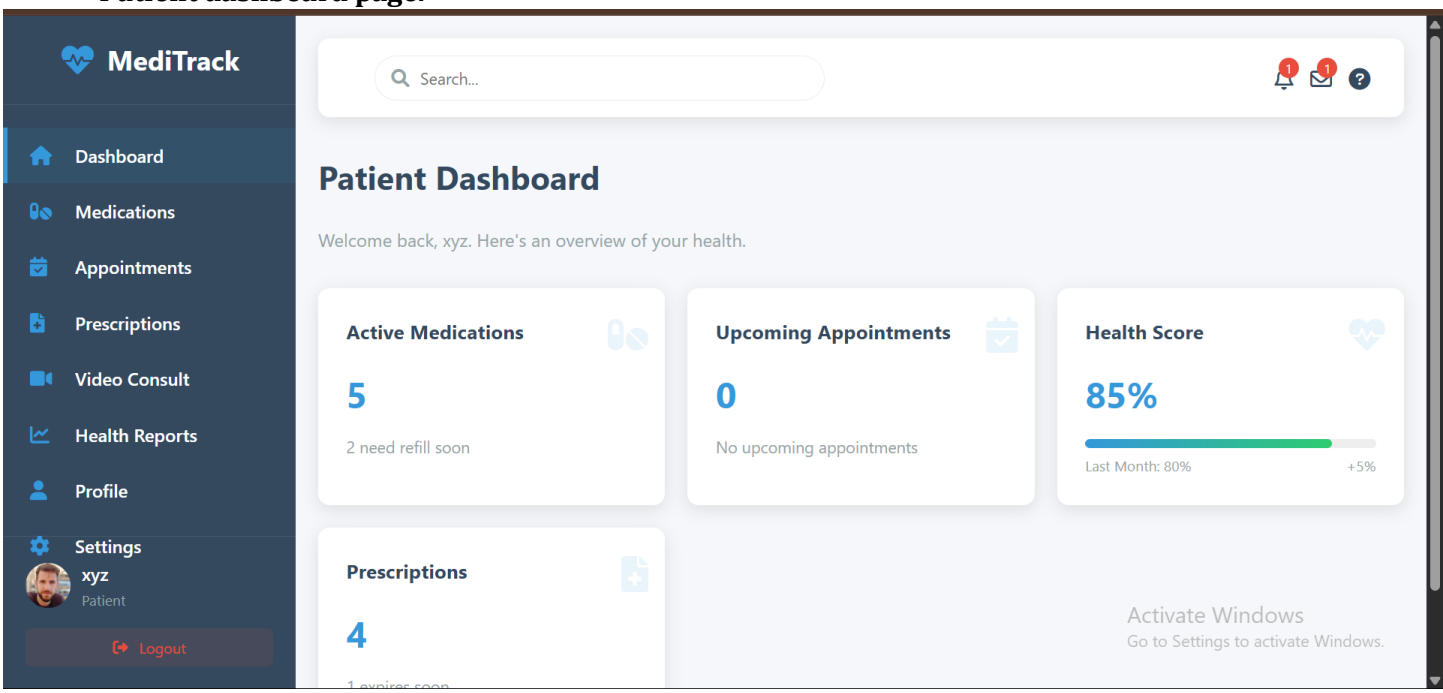
Patient Login Page:



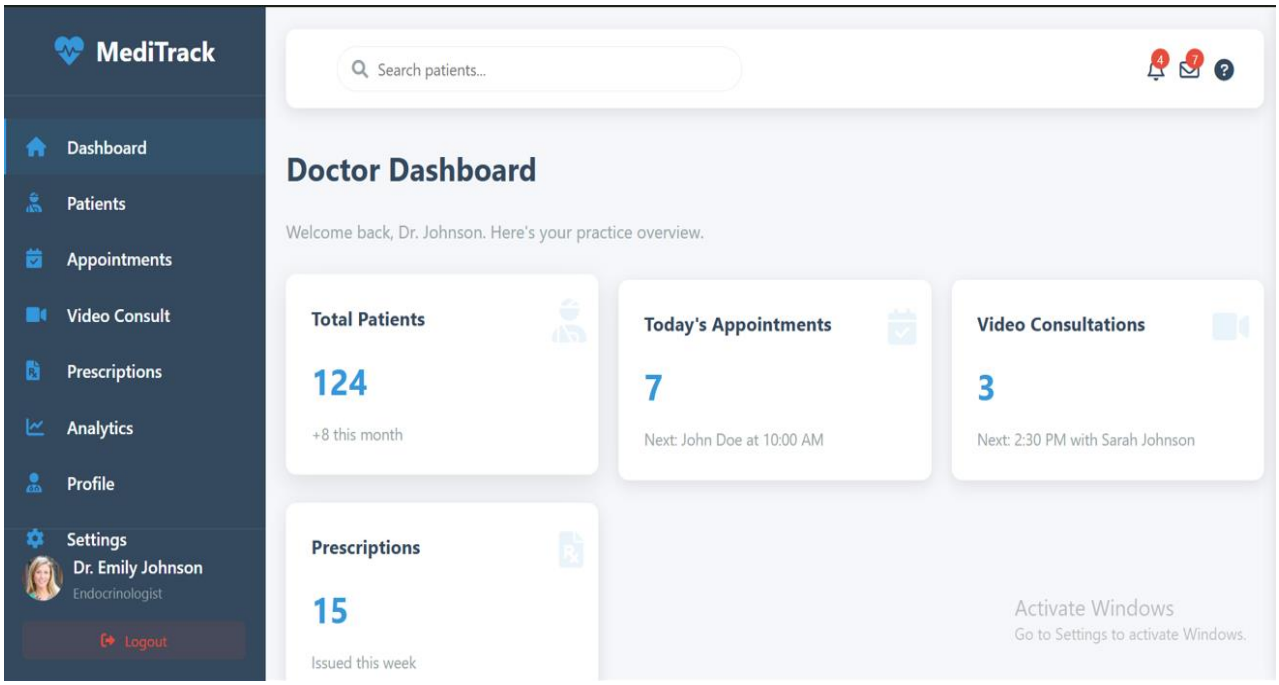
Sign up Page:



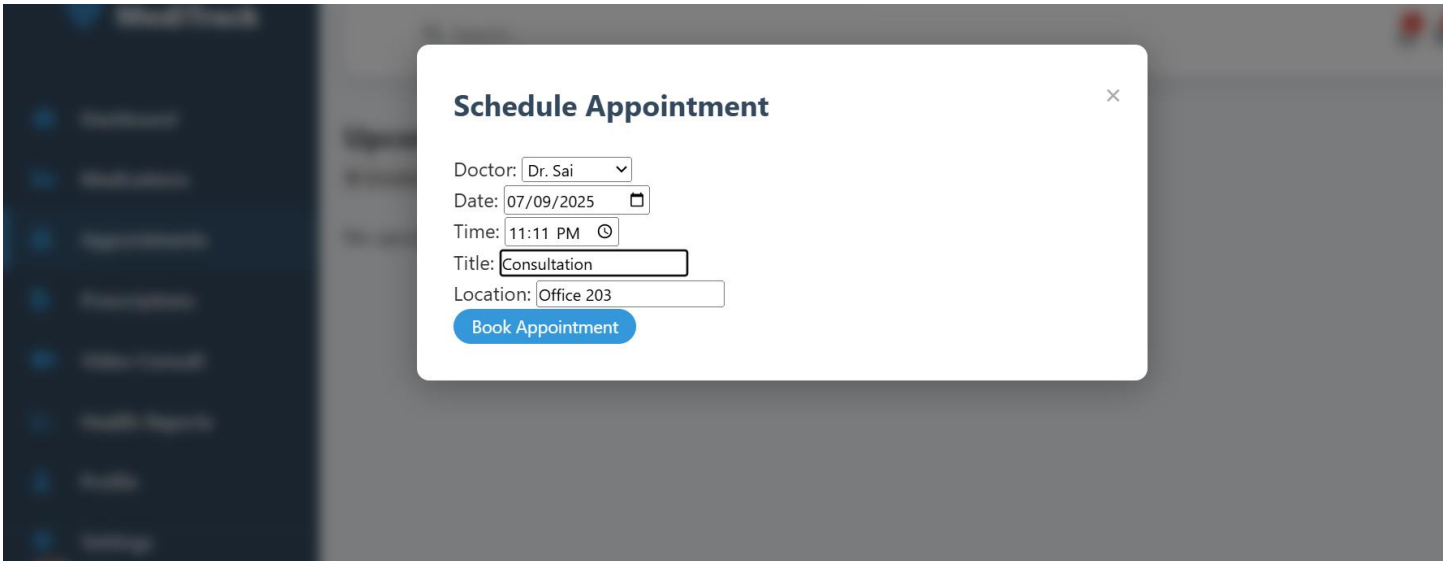
Patient dashboard page:



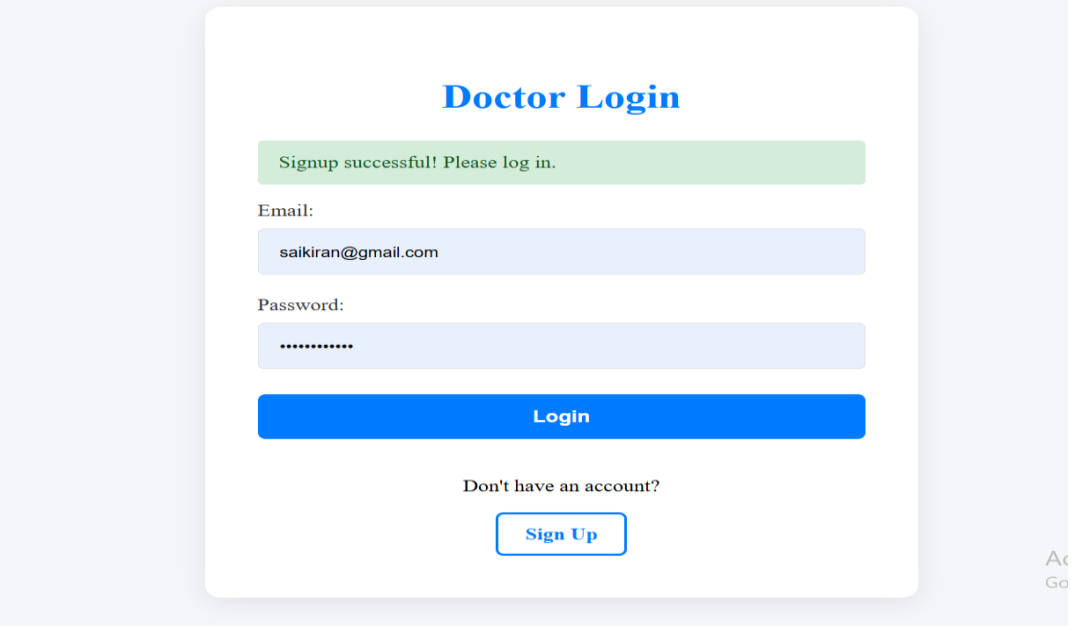
Doctor dashboard page:



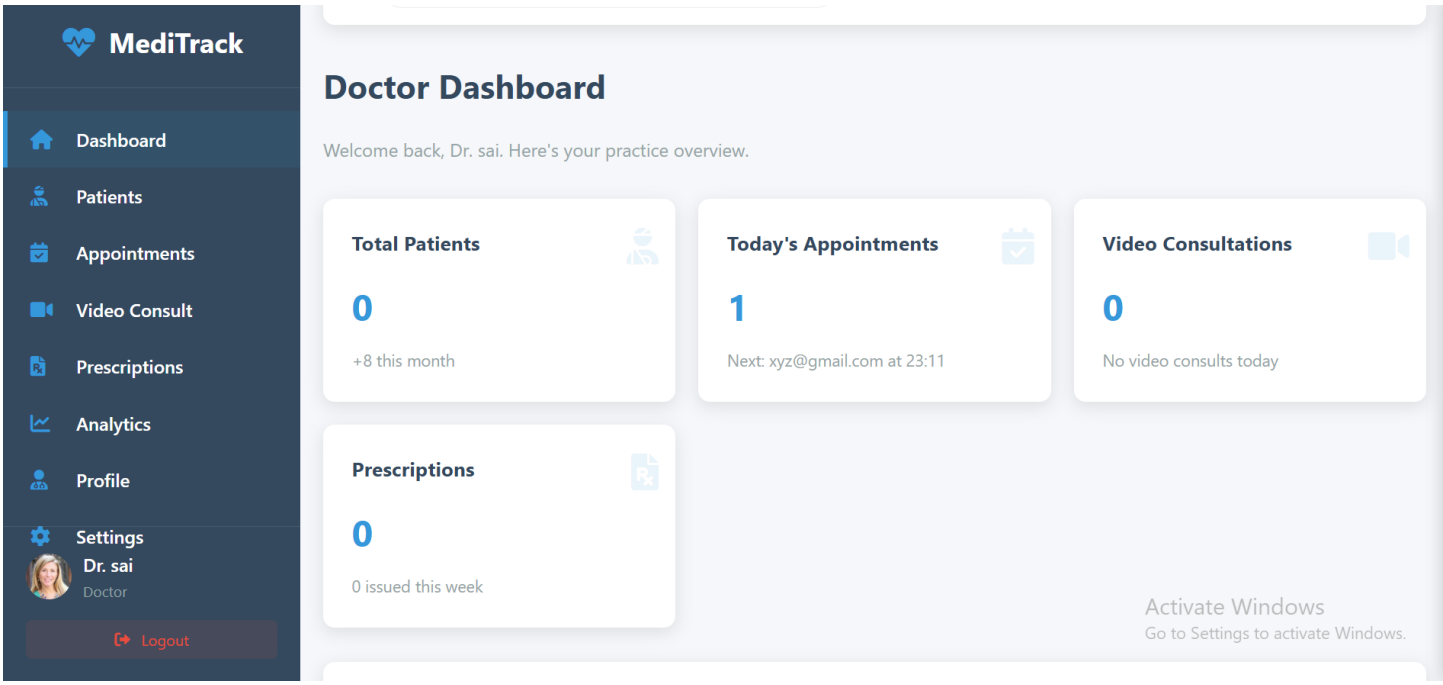
Bookappointment Page:



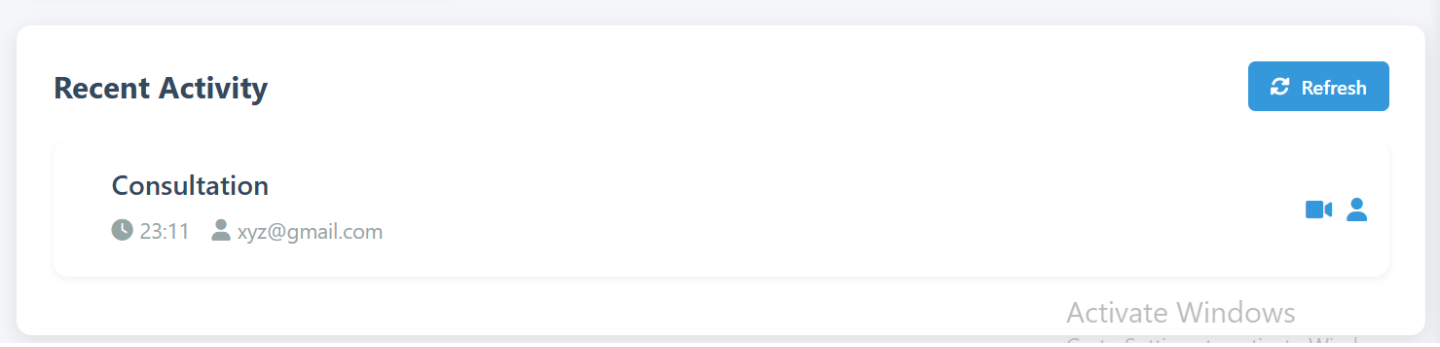
Doctor login :



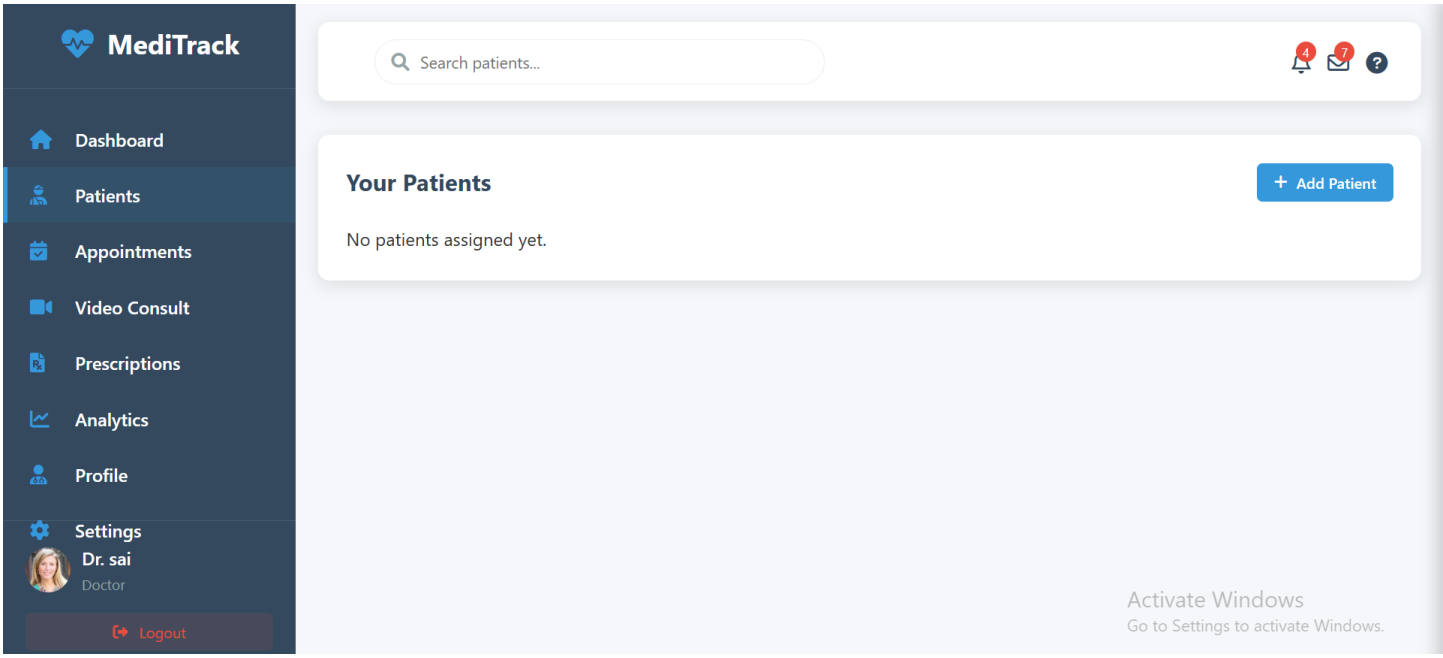
Doctor dashboard Page:



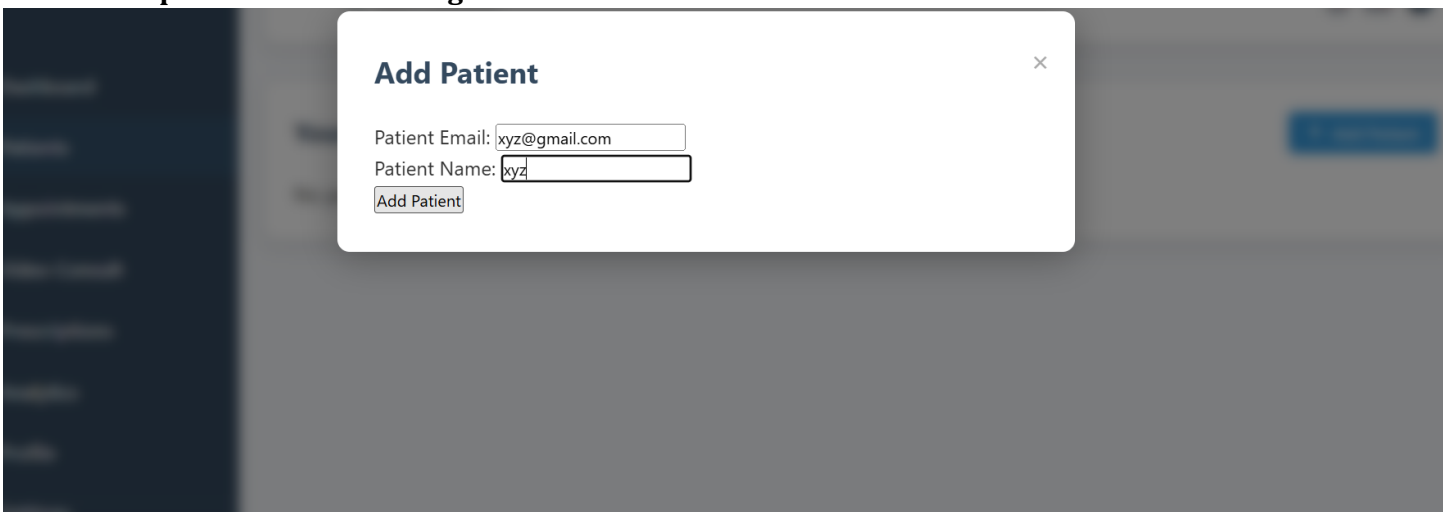
Upcoming Appointments Page:



Add patient Page:



Add patient with Email Page:



Patient Dash board Page:

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

Patient Dashboard

Welcome back, xyz. Here's an overview of your health.

Active Medications

5

2 need refill soon

Upcoming Appointments

1

Next: saikiran@gmail.com on 2025-07-09

Health Score

85%

Last Month: 80%

Prescriptions

4

1 expires soon

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 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.