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

**Hardware Required:**

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

**Software Required:**

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

**System Required:**

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

**Description:**

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

**Scenarios:**

**Scenario 1: Efficient Appointment Booking System for Patients**

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

**Scenario 2: Secure User Management with IAM**

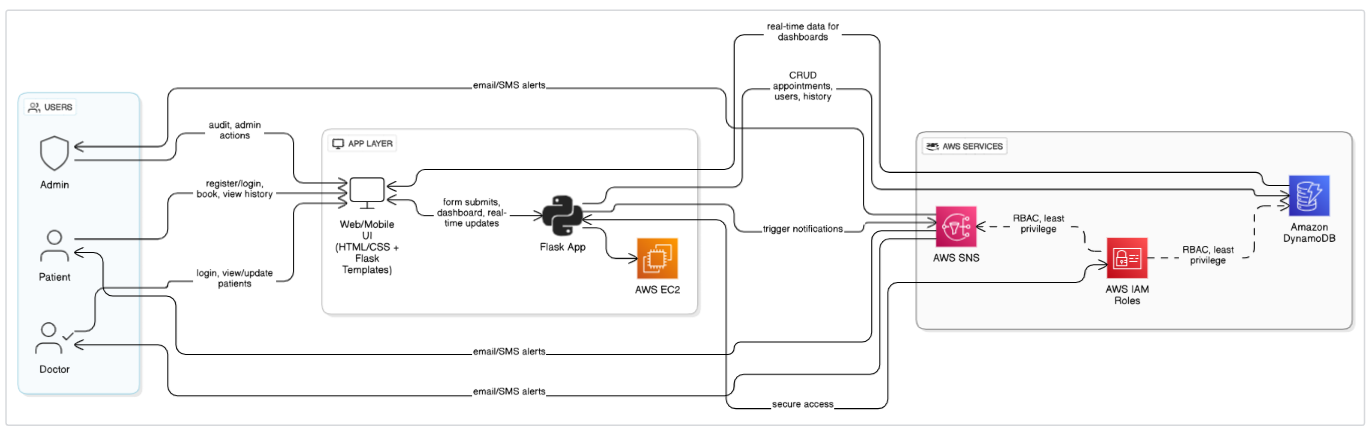
MedTrack utilizes AWS IAM to manage user permissions and ensure secure access to the system. For instance, when a new patient registers, an IAM user is created with specific roles and permissions to access only the features relevant to them. Doctors have their own IAM configurations, allowing them access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

**Scenario 3: Easy Access to Medical History and Resources**

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

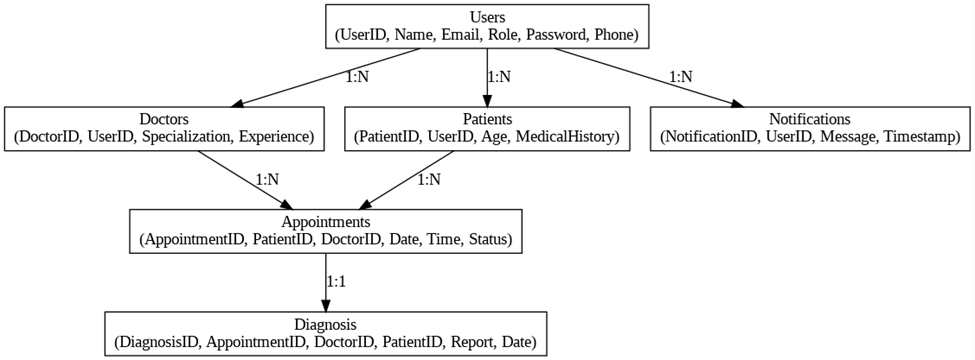
# AWS ARCHITECTURE

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



Entity Relationship (ER)Diagram:

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



**Pre-requisites:**

* AWS Account Setup:  
  <https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
* AWS IAM (Identity and Access Management):  
  <https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
* AWS EC2 (Elastic Compute Cloud):  
  <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
* AWS DynamoDB:   
  [https://docs.aws.amazon.com/amazondynamodb/Introduction.html](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html)
* Amazon SNS:   
  <https://docs.aws.amazon.com/sns/latest/dg/welcome.htm>l
* Git Documentation:   
  <https://git-scm.com/doc>
* VS Code Installation: (download the VS Code using the below link or you can get that in Microsoft store)  
  <https://code.visualstudio.com/download>

**Project WorkFlow:**

# Milestone 1. Web Application Development and Setup

# Develop the Backend Using Flask.

# Integrate AWS Services Using boto3.

# Milestone 2. AWS Account Setup and Login

# Set up an AWS account if not already done.

# Login to AWS Management Console.

# Milestone 3. DynamoDB Database Creation and Setup

# Create a DynamoDB Table.

# Configure Attributes for User Data and Book Requests.

# Milestone 4. SNS Notification Setup

# Create SNS topics for book request notifications.

# Subscribe users and library staff to SNS email notifications.

# Milestone 5. IAM Role Setup

# Create IAM Role

# Attach  Policies

# Milestone 6. EC2 Instance Setup

# Launch an EC2 instance to host the Flask application.

# Configure security groups for HTTP, and SSH access.

# Milestone 7. Deployment using EC2

# Upload Flask Files

# Run the Flask App

# Milestone 8. Testing and Deployment

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

**Milestone 1: Web Application Development and Setup**

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

**Please refer to this sample as a guide for local deployment :**[https://docs.google.com/document/d/1sFF7-tJ6IgWtRbawWoA4W3PkkxEFrSJZhKzULgLsjxo/edit?usp=sharing](https://skillwallet.smartinternz.com/Student/guided_project_info/Please%20refer%20to%20this%20sample%20as%20a%20guide%20for%20local%20deployment%20:%20https:/docs.google.com/document/d/1sFF7-tJ6IgWtRbawWoA4W3PkkxEFrSJZhKzULgLsjxo/edit?usp=sharing)

**Important Instructions:**

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

**Post Troven Access Activation:**

* Once Troven Labs access is provided (valid for 3 hours), you must immediately proceed with Milestone 1 of your Guided Project instructions.
* At this point, modify your app.py and replace local dictionary/list operations with AWS services (such as DynamoDB, RDS, or others as per project requirements).
* Using the temporary credentials provided by Troven Labs, securely connect your application to AWS resources.
* Since the AWS configuration is lightweight and already instructed in the milestones, you should be able to complete the cloud integration efficiently within the allotted time.

# Milestone 2: DynamoDB Database Creation and Setup

**Milestone 2: AWS Account Setup**

**Important Notice: Use Troven Labs for AWS Access**

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

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

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

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

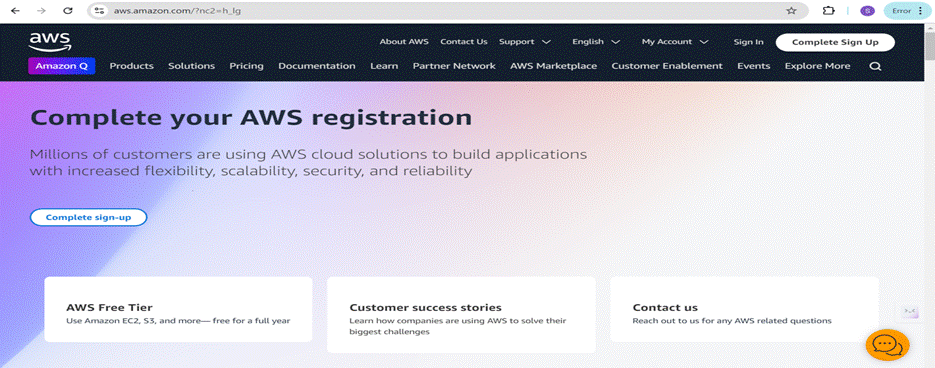
**Please follow the provided guidelines and access AWS exclusively through Troven to avoid unnecessary issues.**

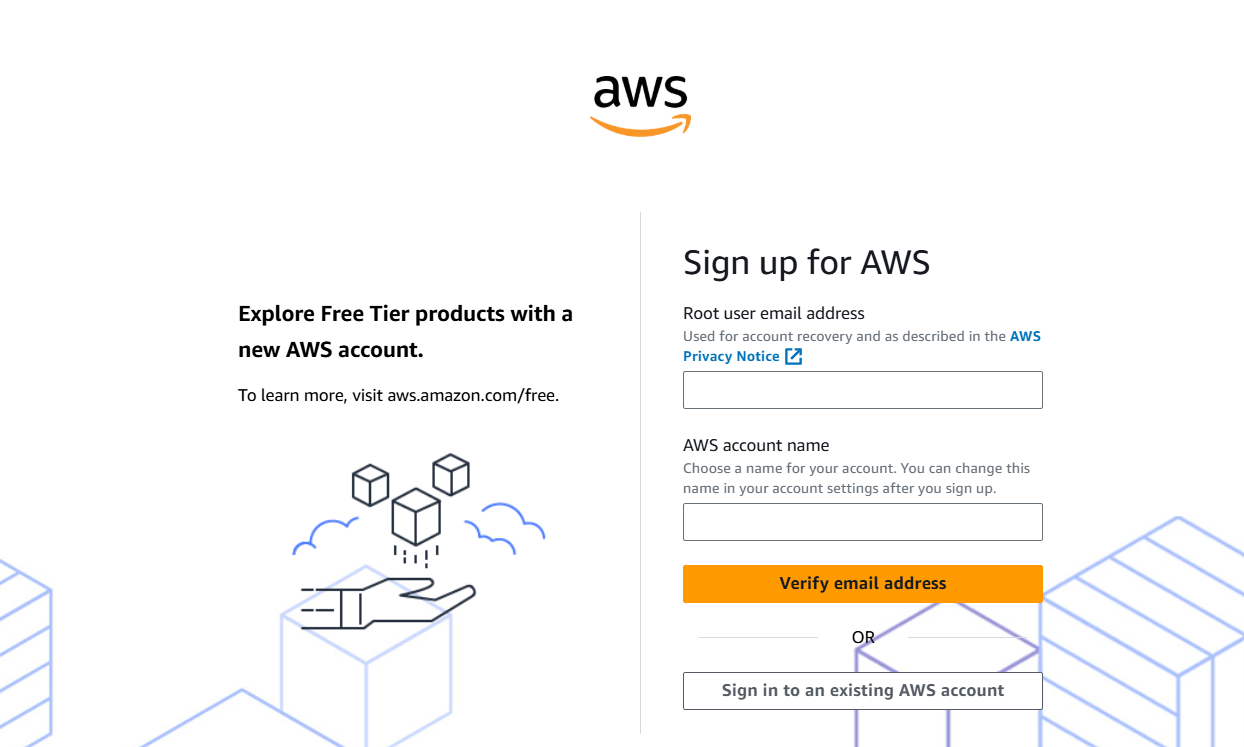
Please refer the below link -  
<https://drive.google.com/file/d/1HzWc7AMJ2BrxhV-uaw5s0vWtcd-28qgI/view?usp=sharing>

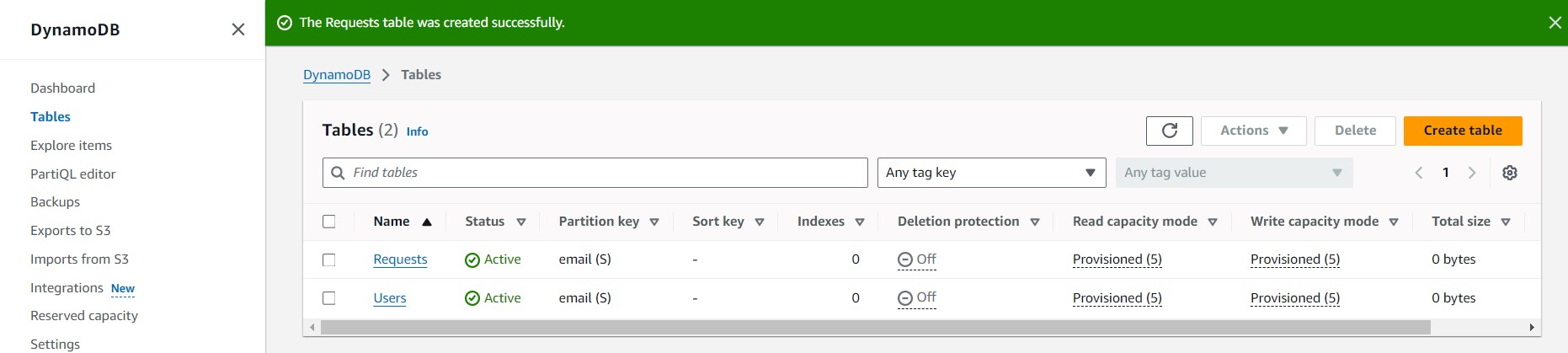
**AWS Account Setup and Login**

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

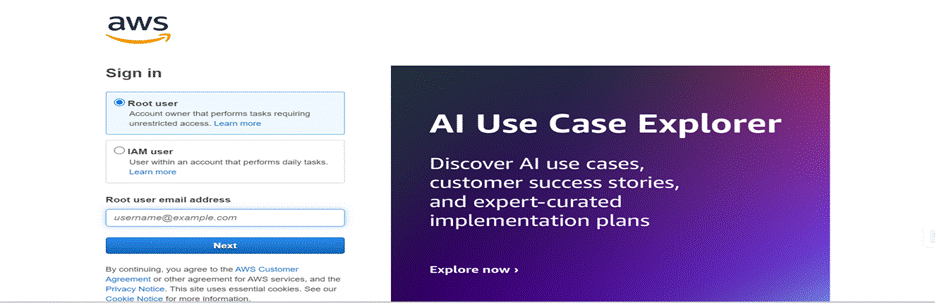
* Go to the AWS website (<https://aws.amazon.com/>).
* Click on the "Create an AWS Account" button.
* Follow the prompts to enter your email address and choose a password.
* Provide the required account information, including your name, address, and phone number.
* Enter your payment information. (Note: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







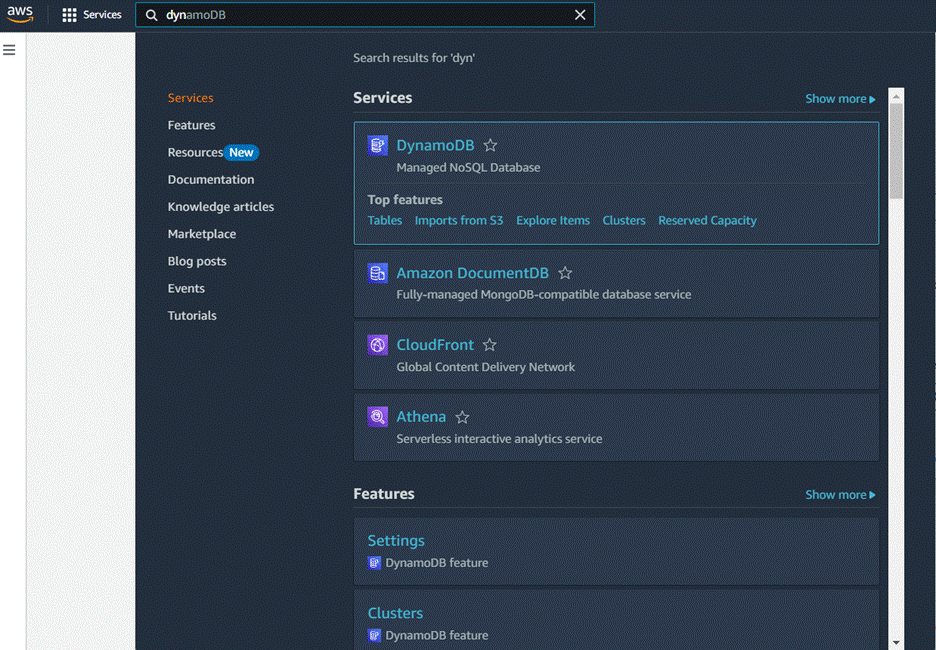
* **Log in to the AWS Management Console**
* **After setting up your account, log in to the**[**AWS Management Console**](https://aws.amazon.com/console/)**.**

****

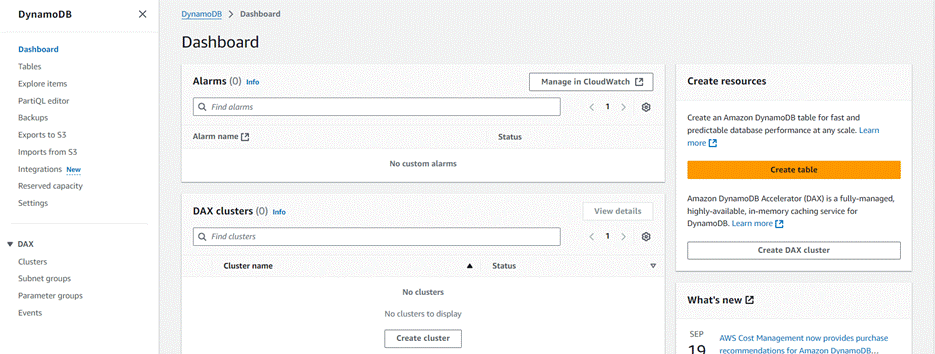
**Milestone 3: DynamoDB Database Creation and Setup**Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high-performance data storage for seamless backend operations.

**Navigate to the DynamoDB**

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



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

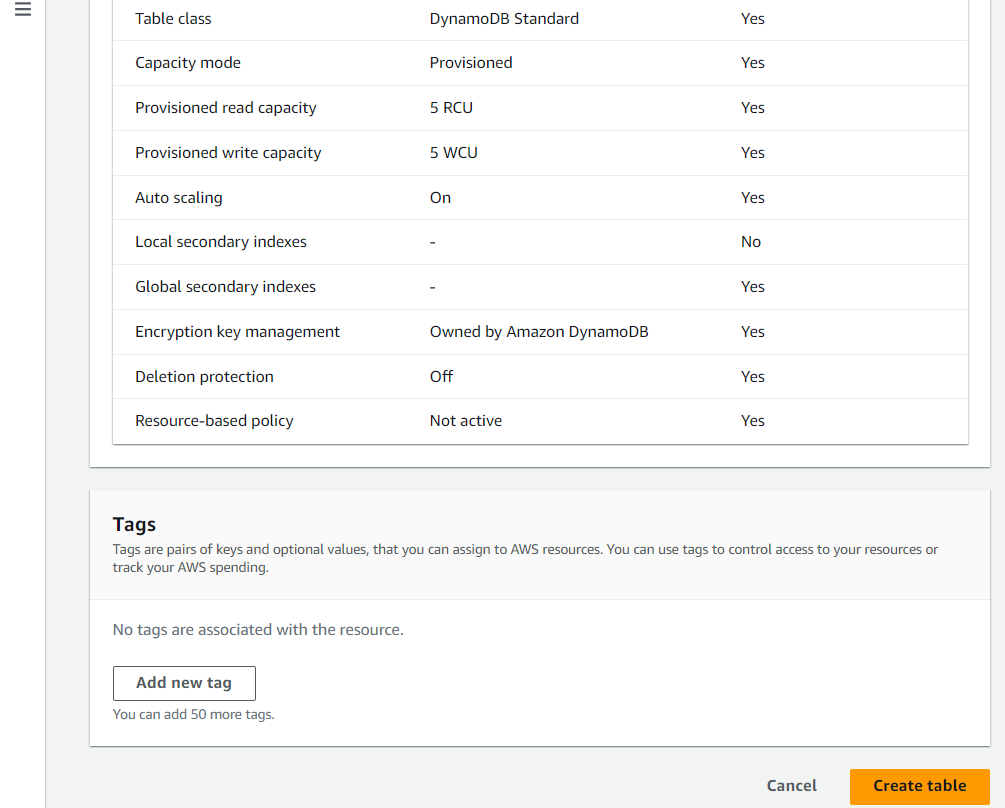




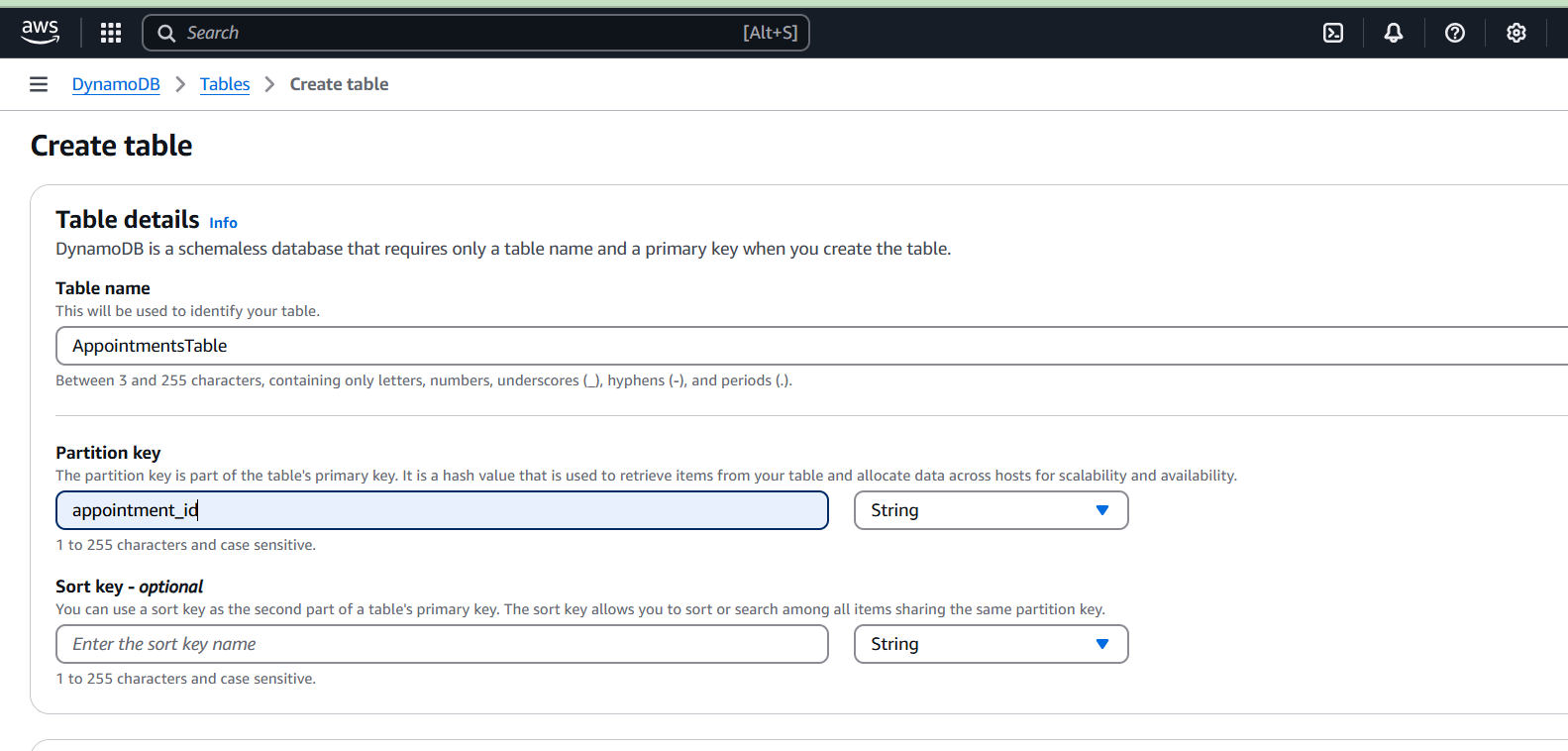
**Create a DynamoDB table for storing data**

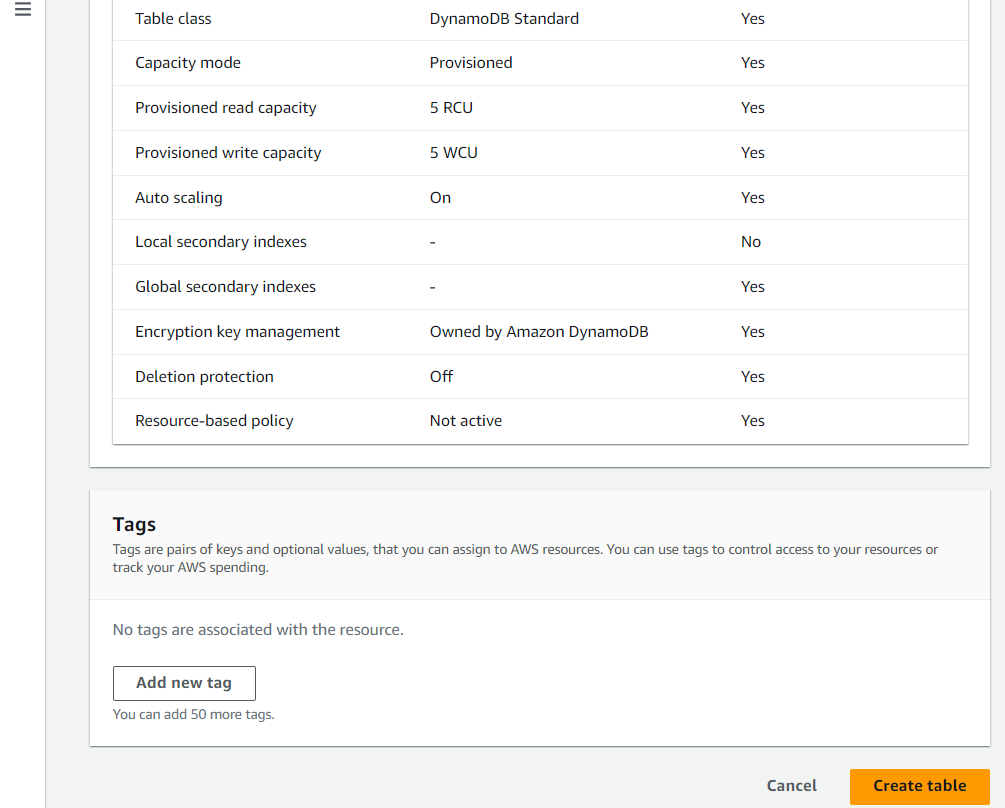
* Create Users table with partition key “Email” with type String and click on create tables.

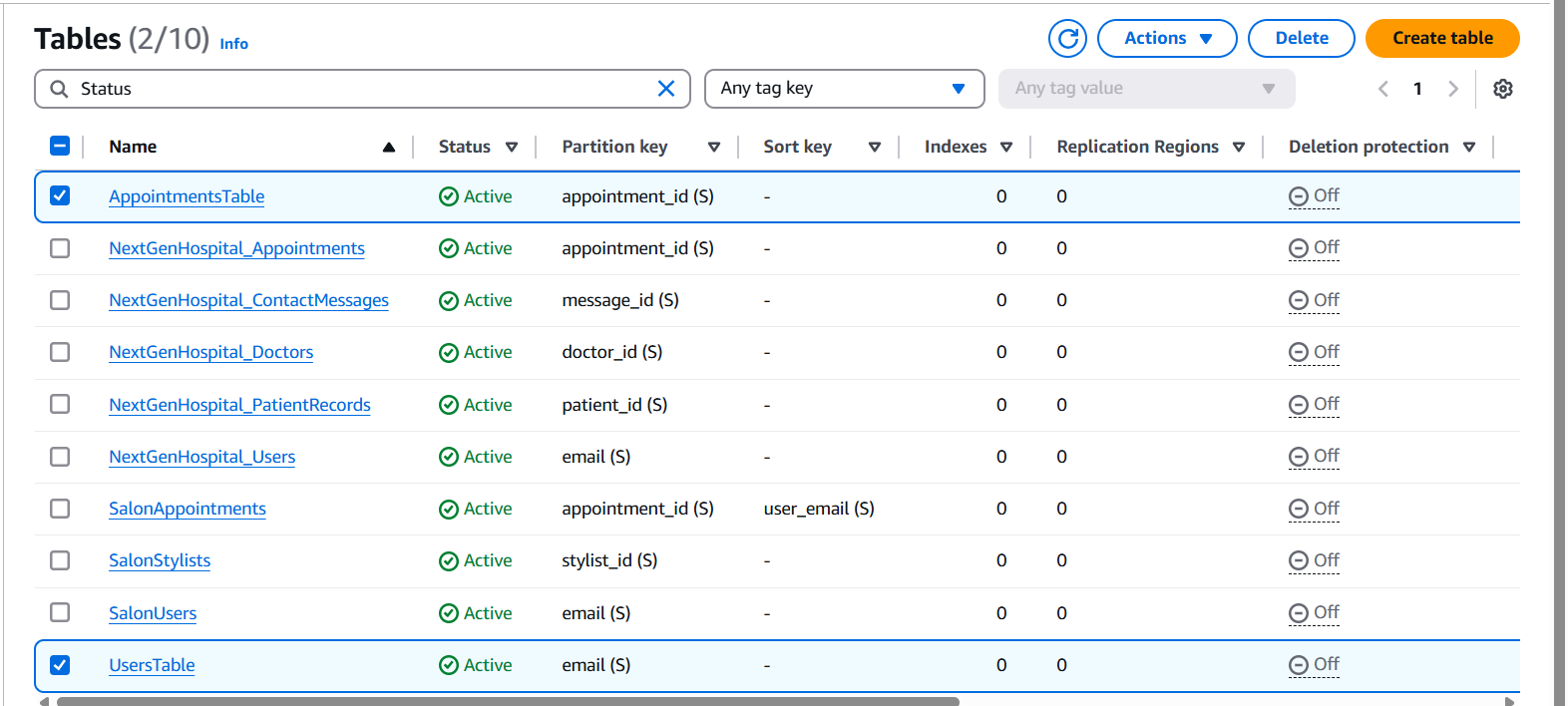
****

****

* Create Appointments Table with partition key “appointment\_id”  with type String and click on create tables.

****

****

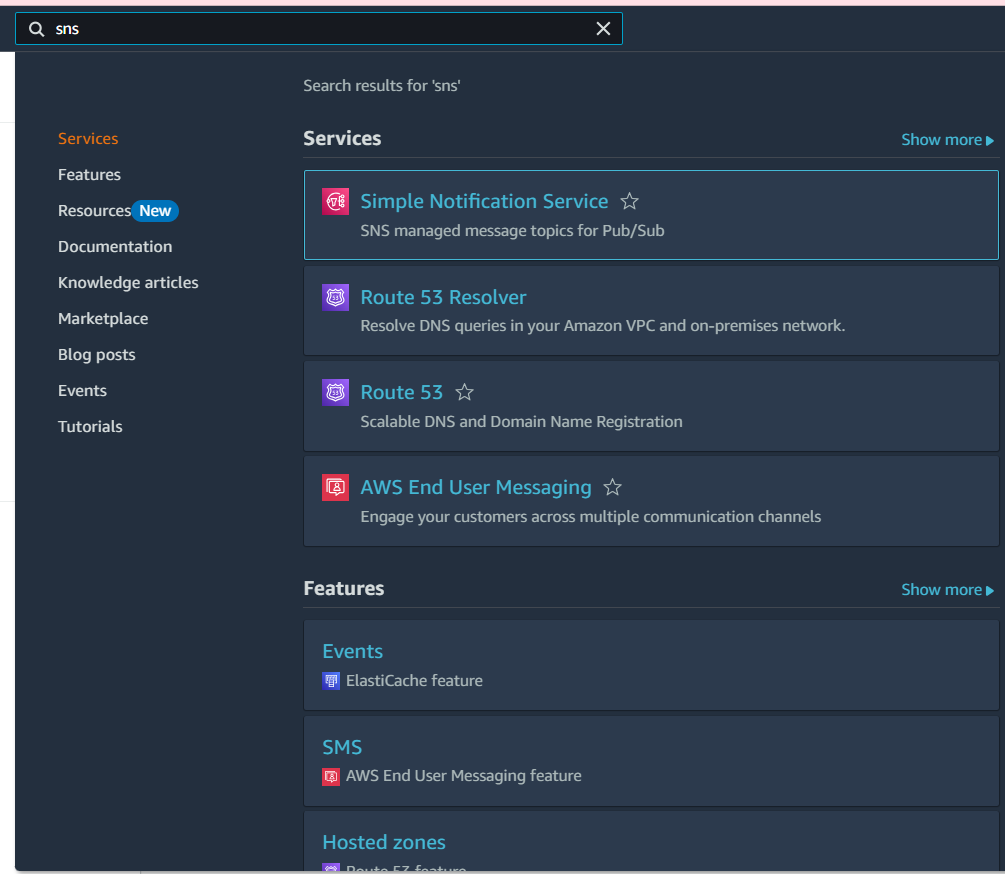
****

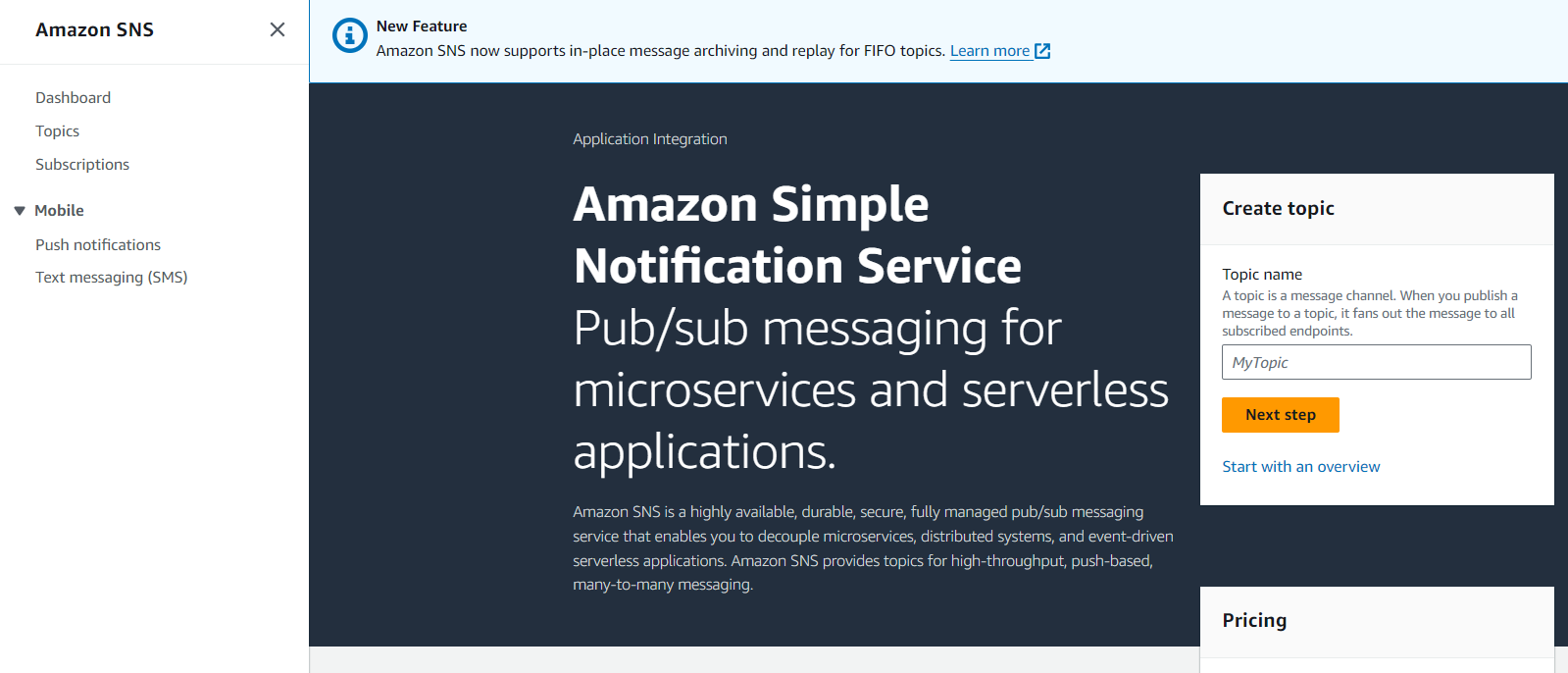
**Milestone 4 : SNS Notification Setup**

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

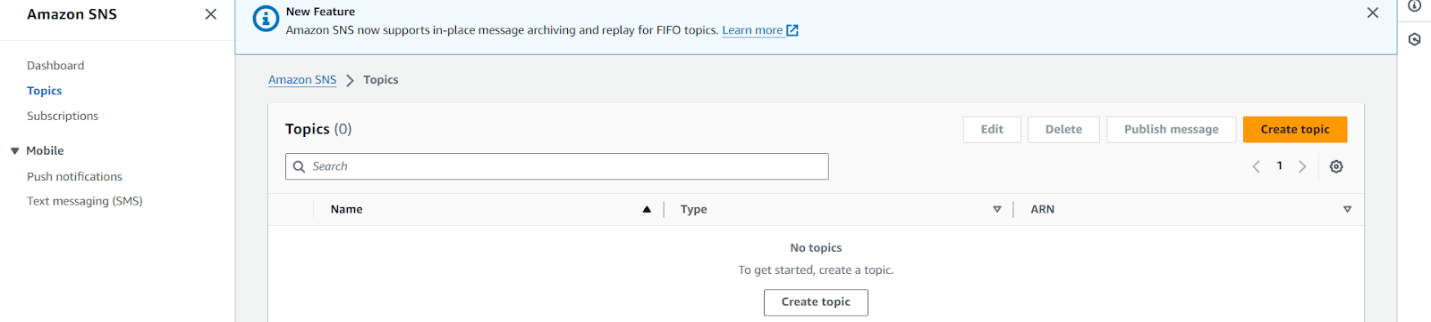
**SNS topics for email notifications**

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

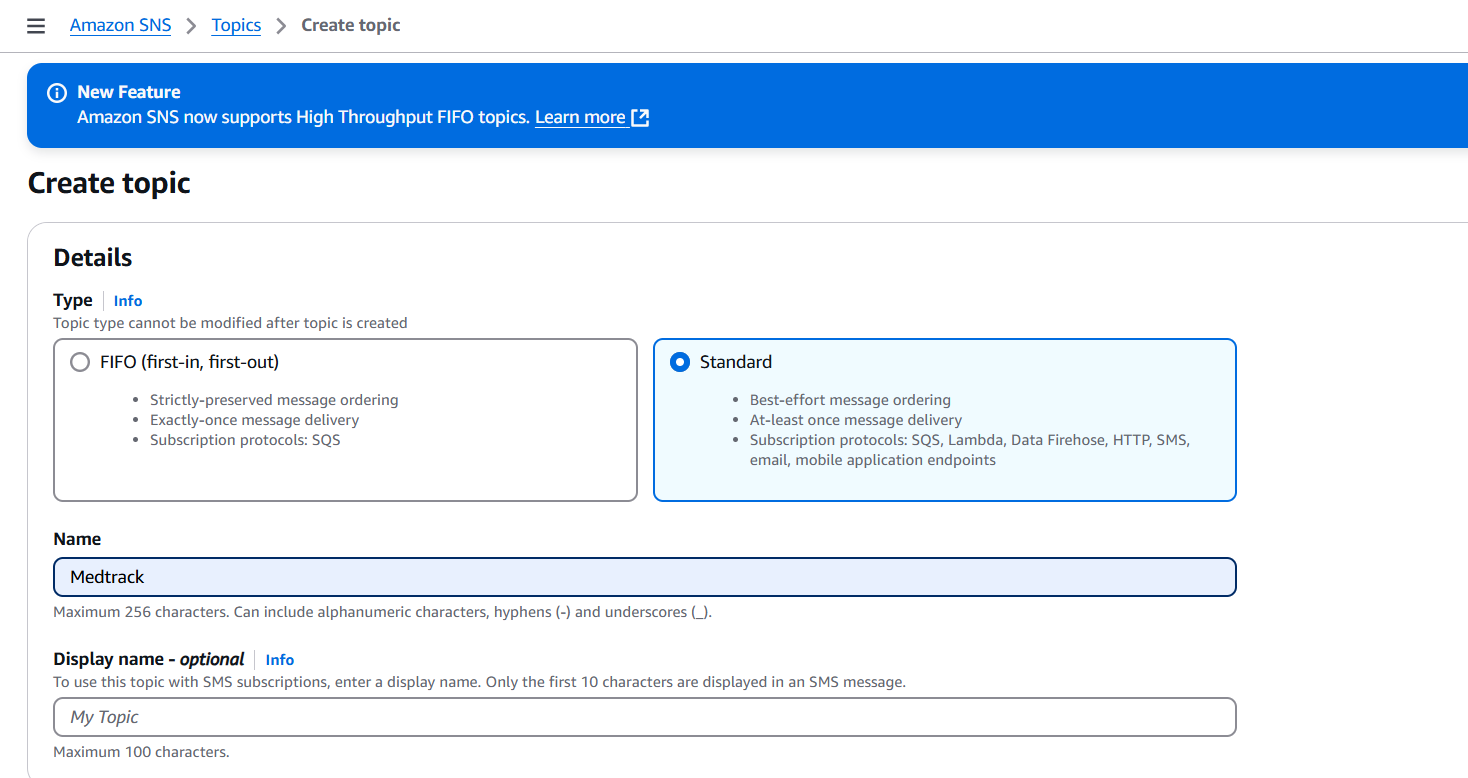
****

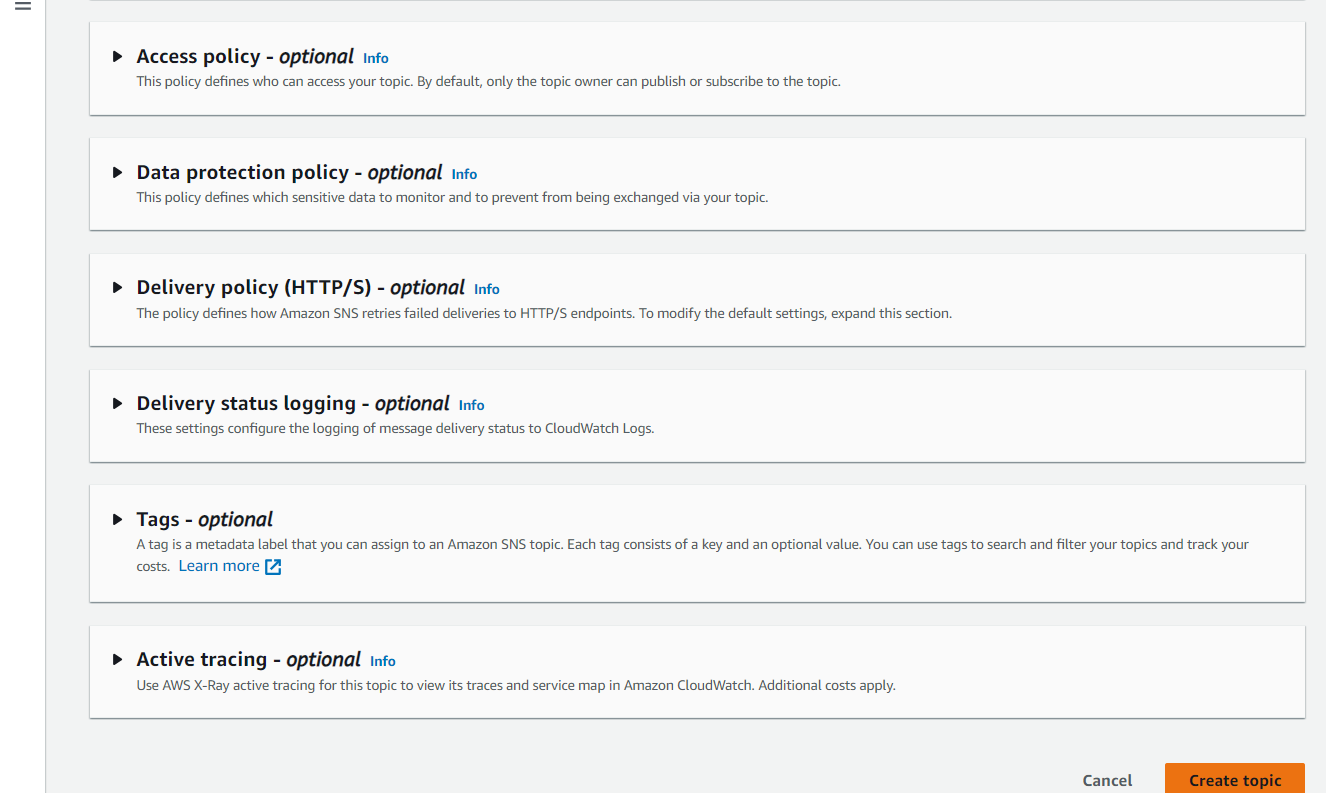
****

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

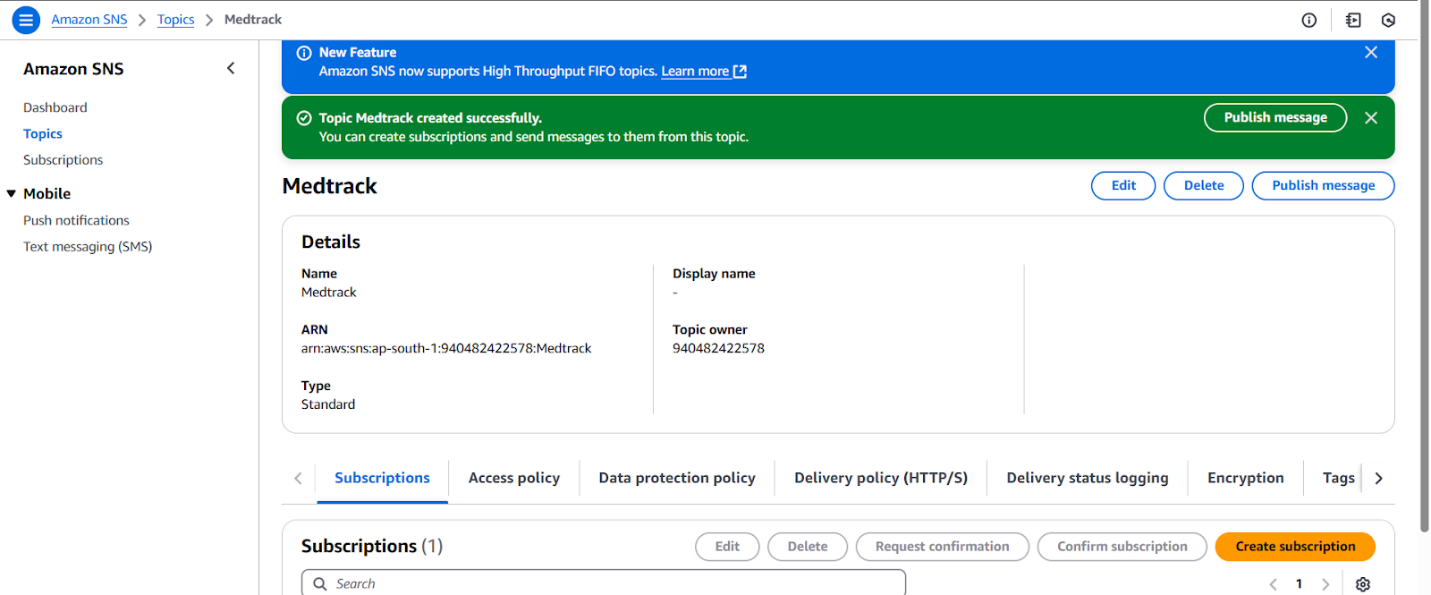
****

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

****

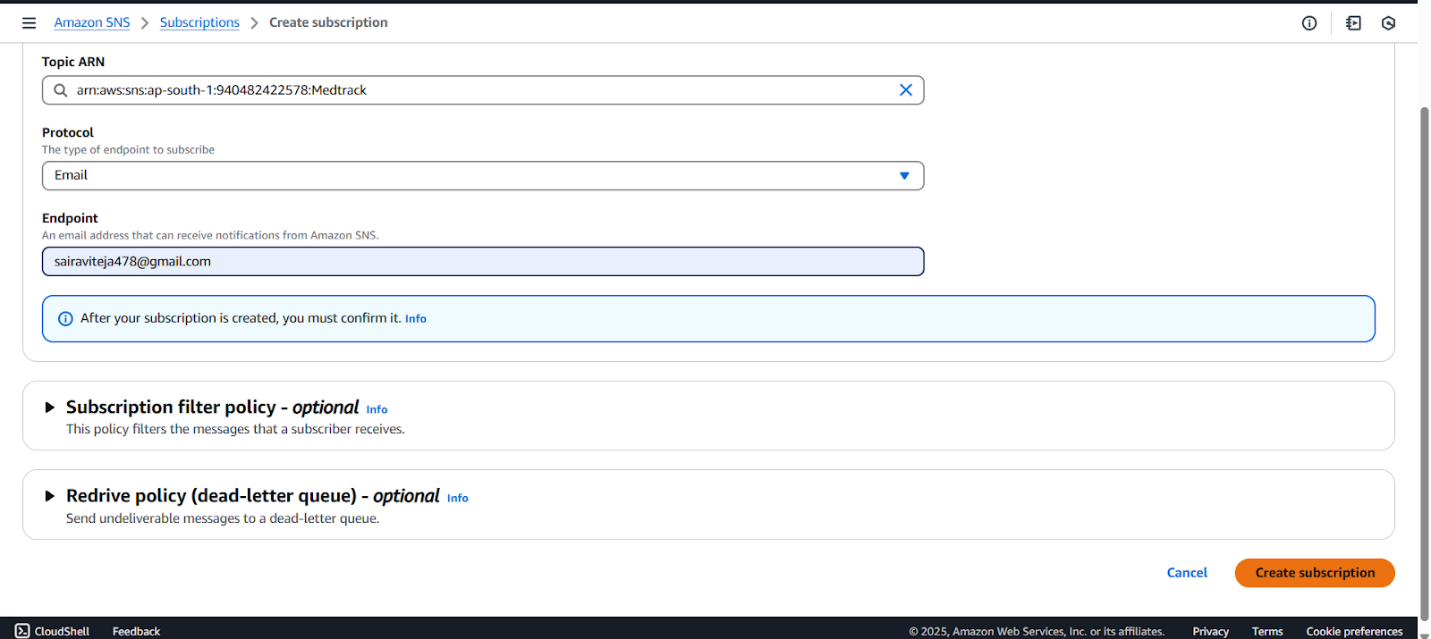
****

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

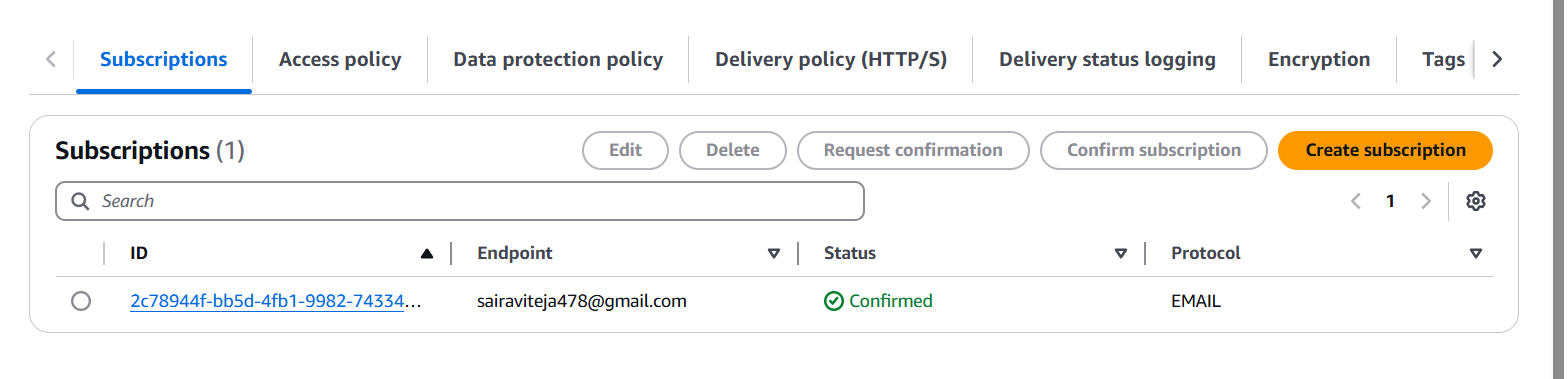
****

**Subscribe users and Admin**

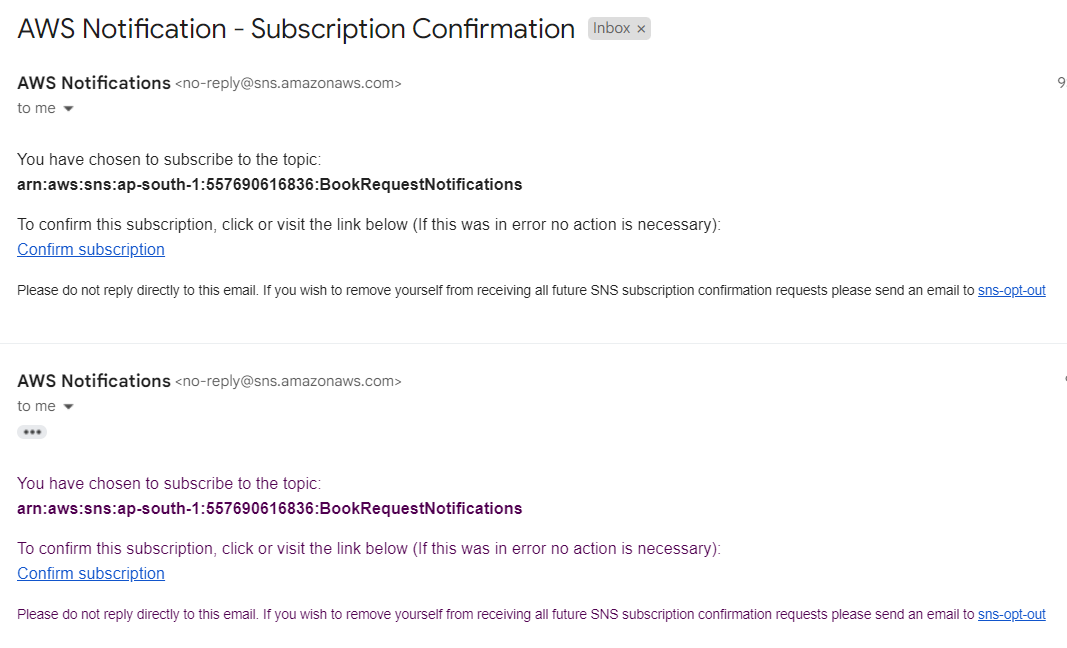
* **Subscribe users (or admin staff) to this topic via email. When a book request is made, notifications will be sent to the subscribed emails.**

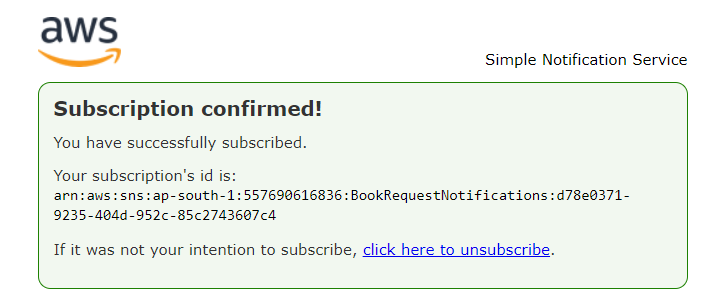
****

* **After subscription request for the mail confirmation**

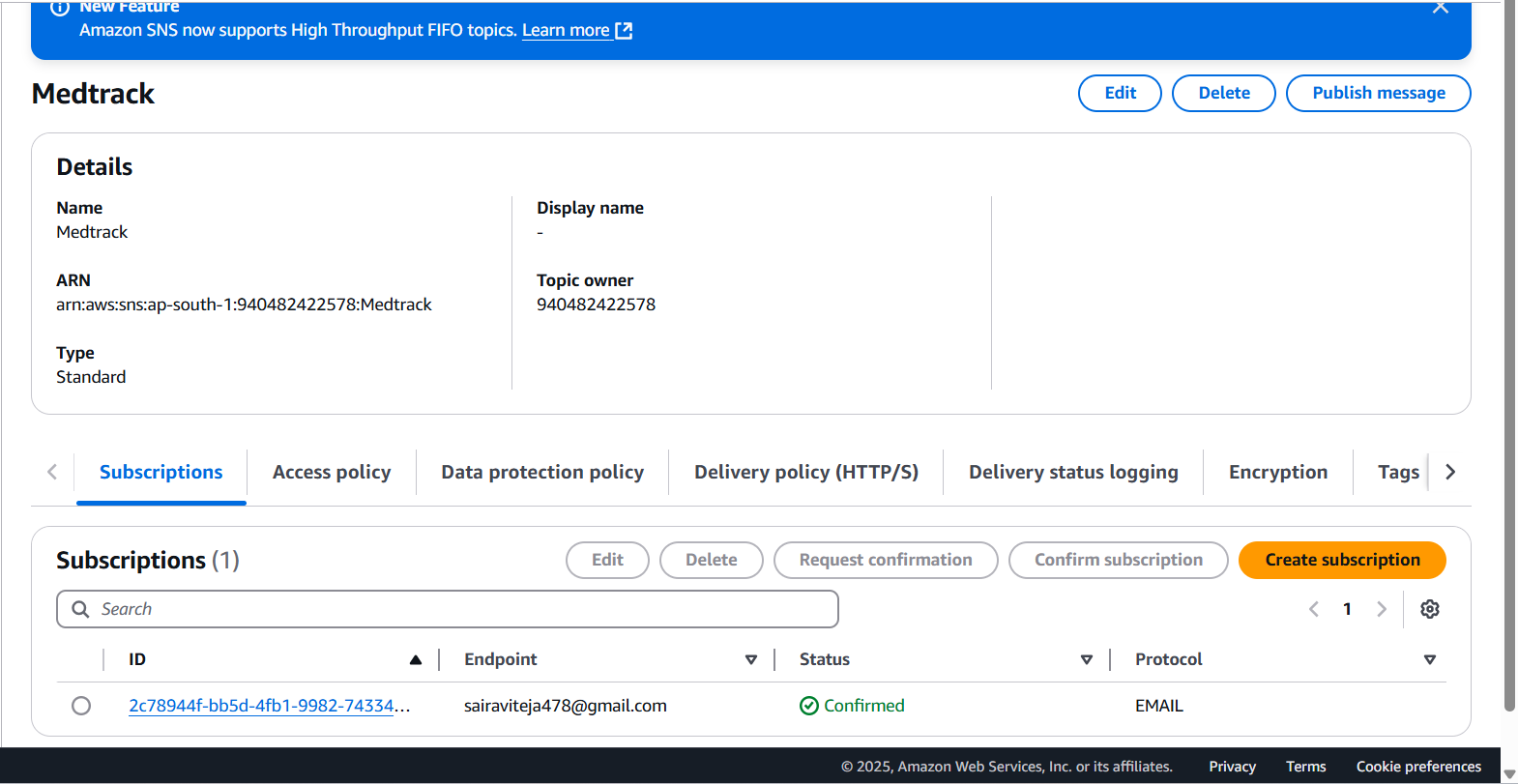
****

* **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.**

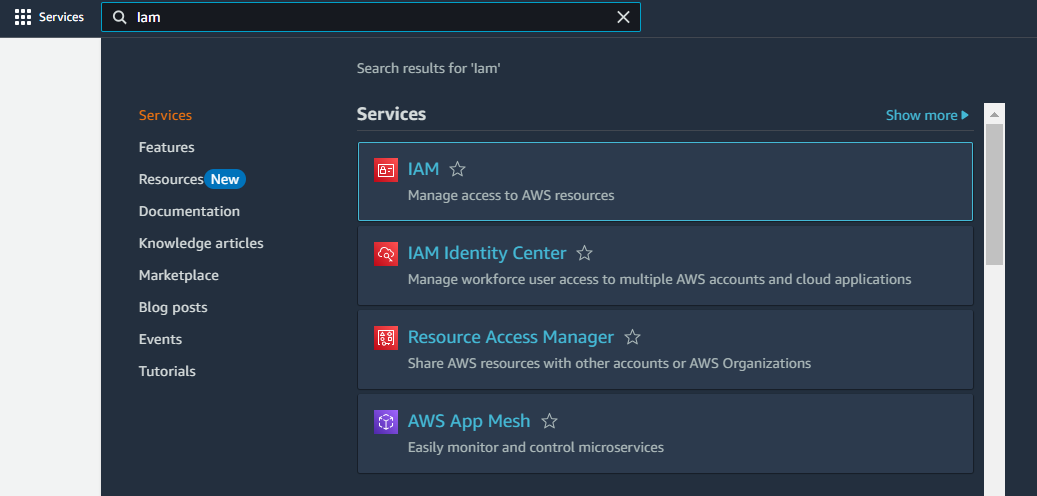
****

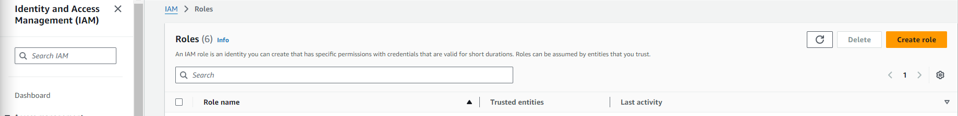
**Milestone 5: IAM Role Setup**

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

**Create IAM Role.**

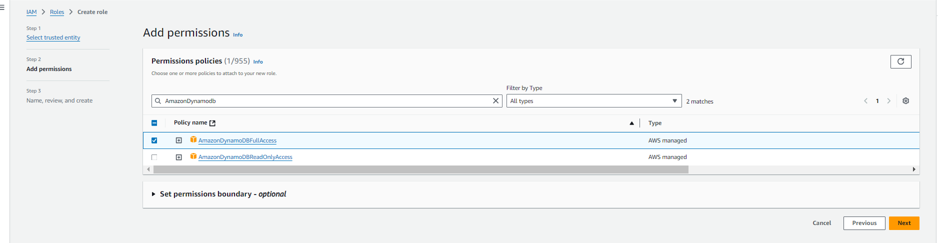
* **In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.**

****

****

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

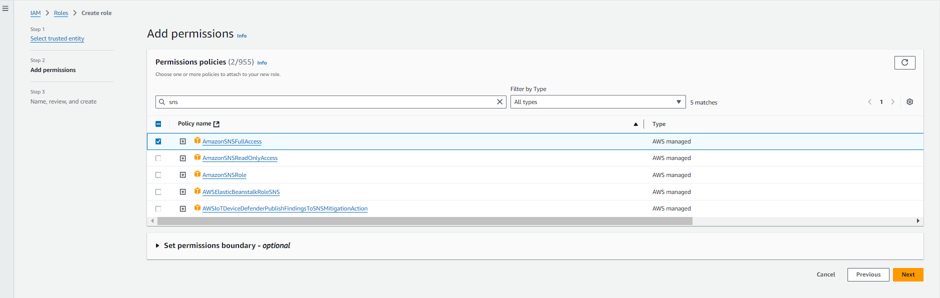
****

****

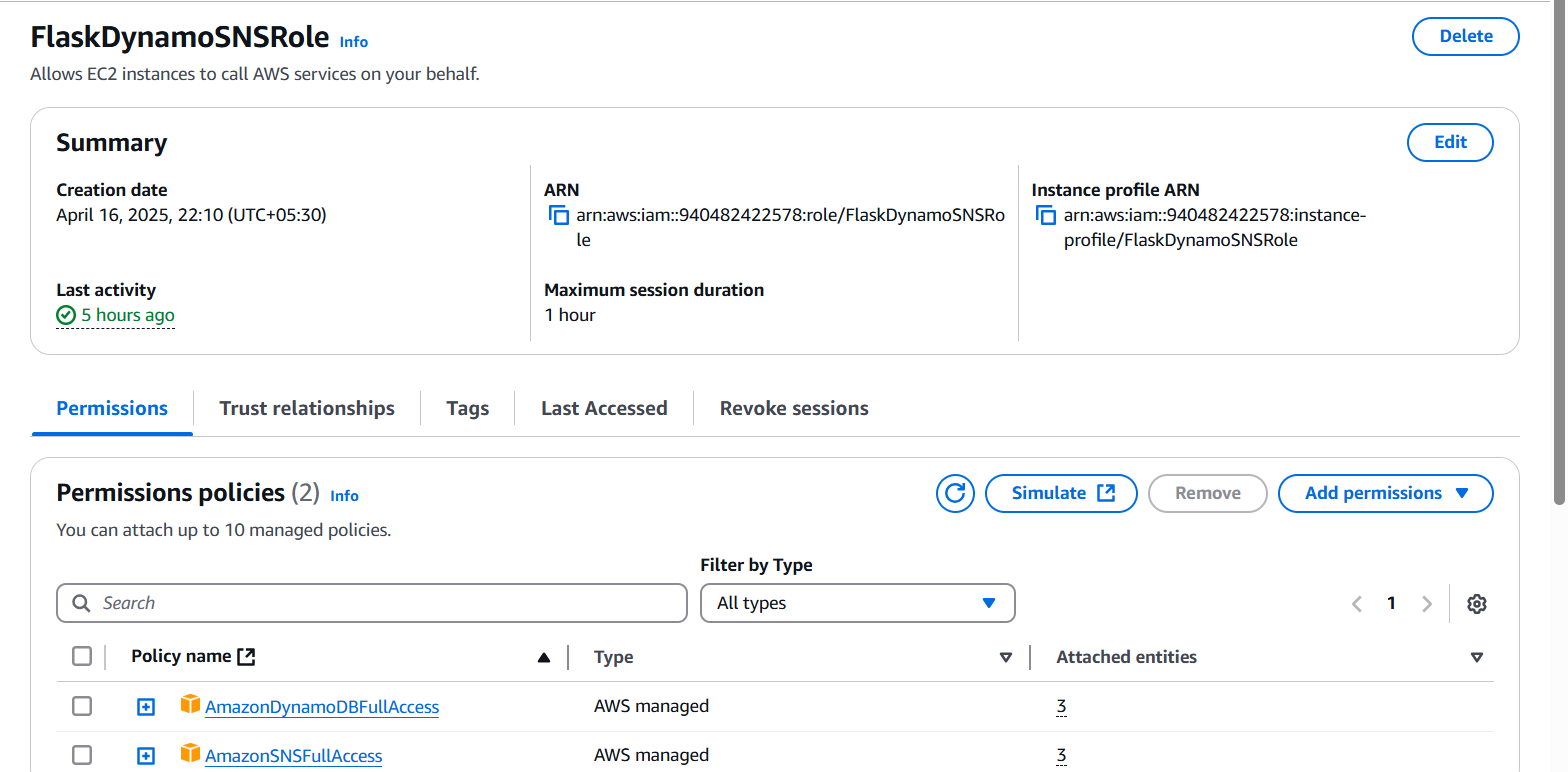
**Attach Policies.**

**Attach the following policies to the role:**

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

****

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

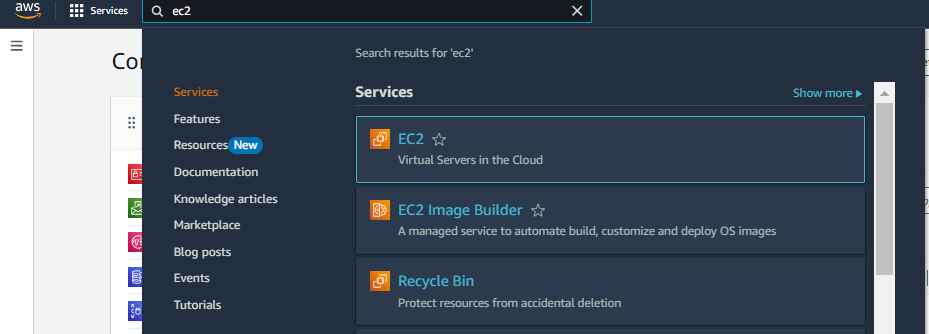
****

**Milestone 6: EC2 Instance setup**

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

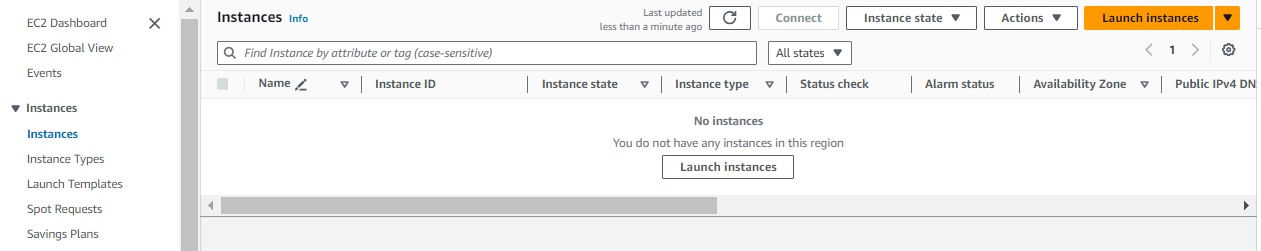
**Launch an EC2 instance to host the Flask application.**

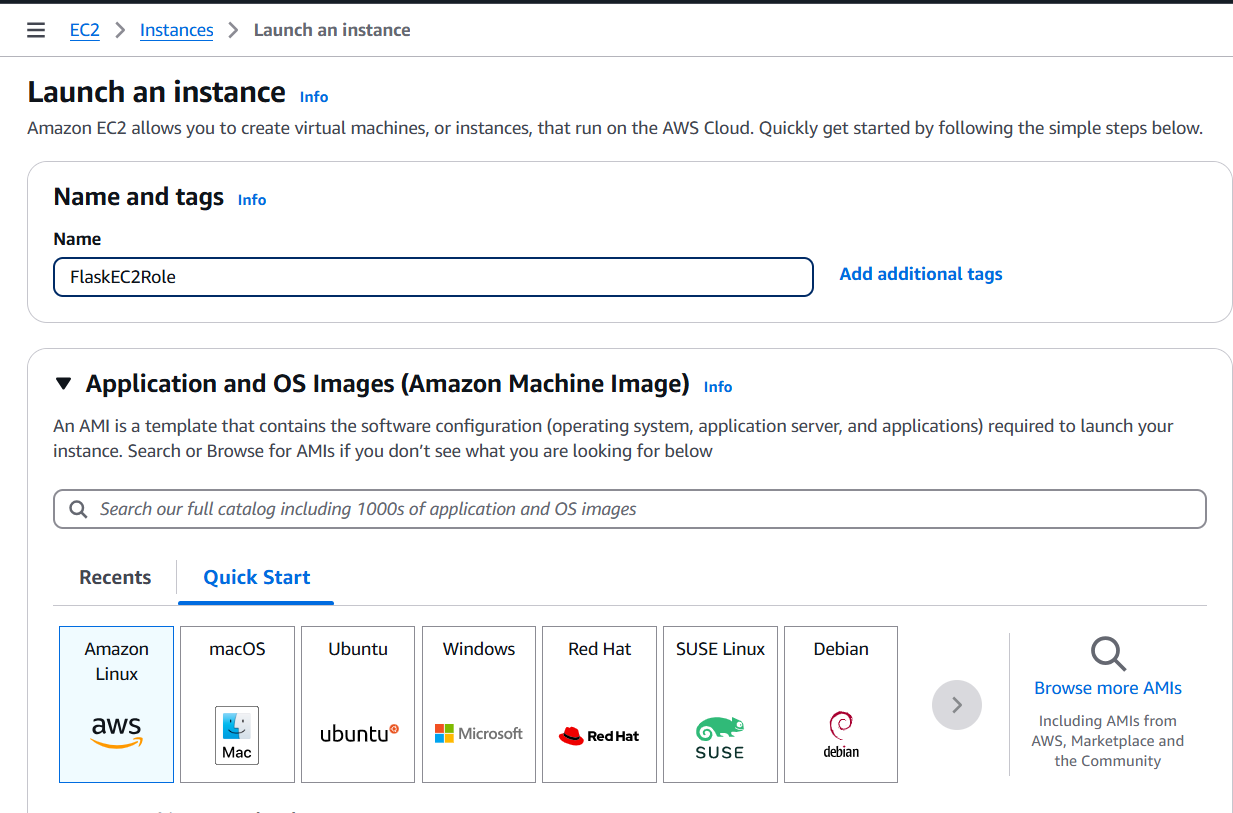
* **Launch EC2 Instance**
  + In the AWS Console, navigate to EC2 and launch a new instance.



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

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

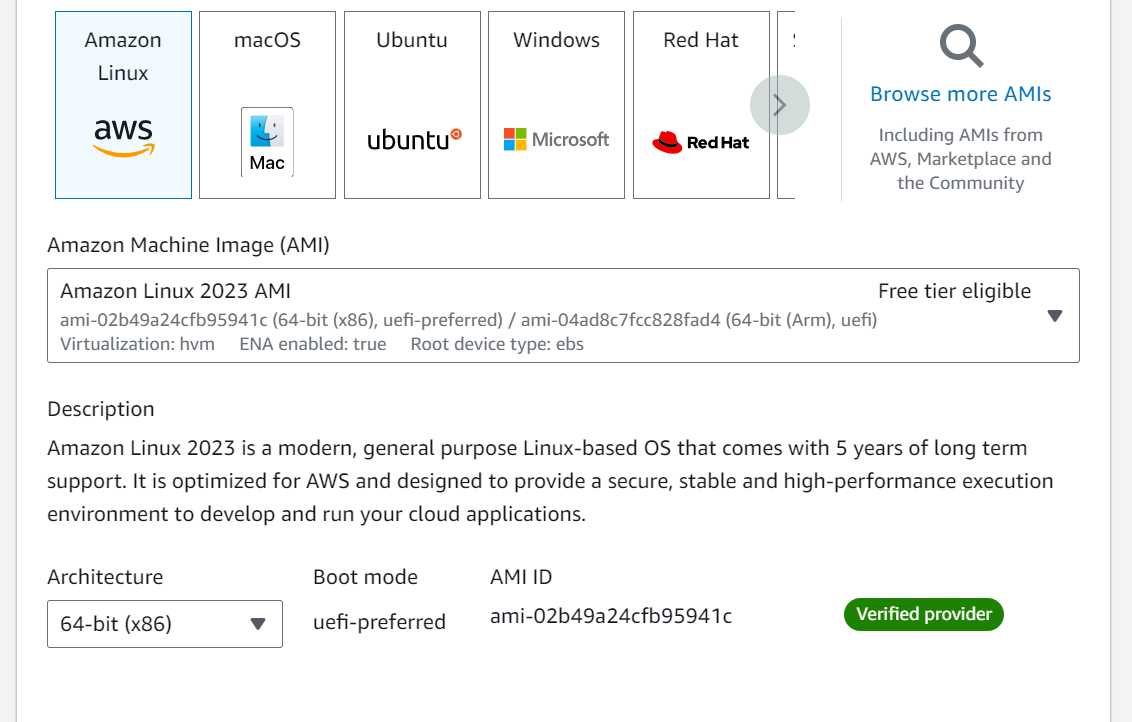




To launch an EC2 instance with **Amazon Linux 2** or **Ubuntu** as the AMI and **t2.micro** as the instance type (free-tier eligible):

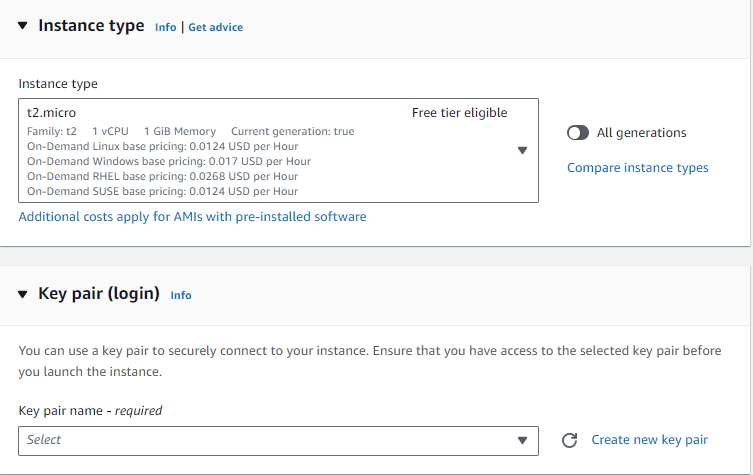
1. In the **Launch Instance** wizard, choose **Amazon Linux 2** or **Ubuntu** from the available AMIs.
2. Select **t2.micro** as the instance type, which is free-tier eligible, and continue with the configuration and launch steps.

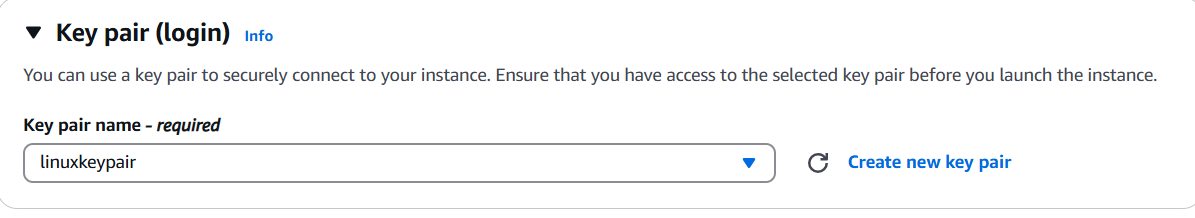


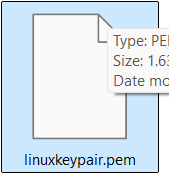


To create and download the key pair for server access:

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



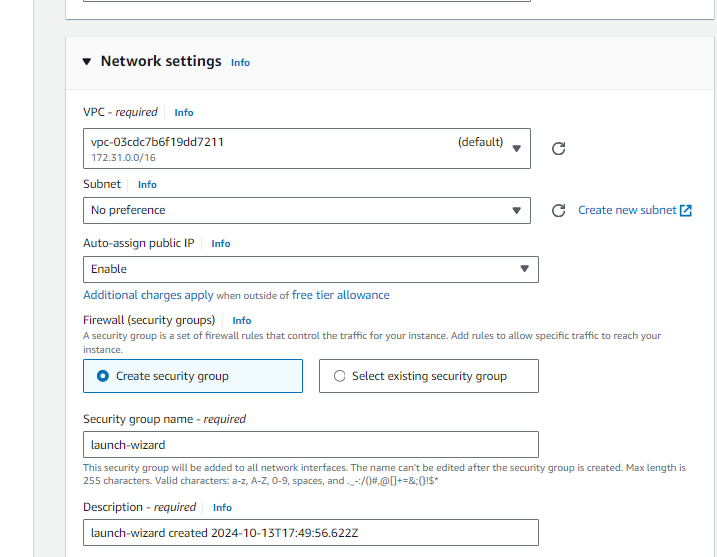


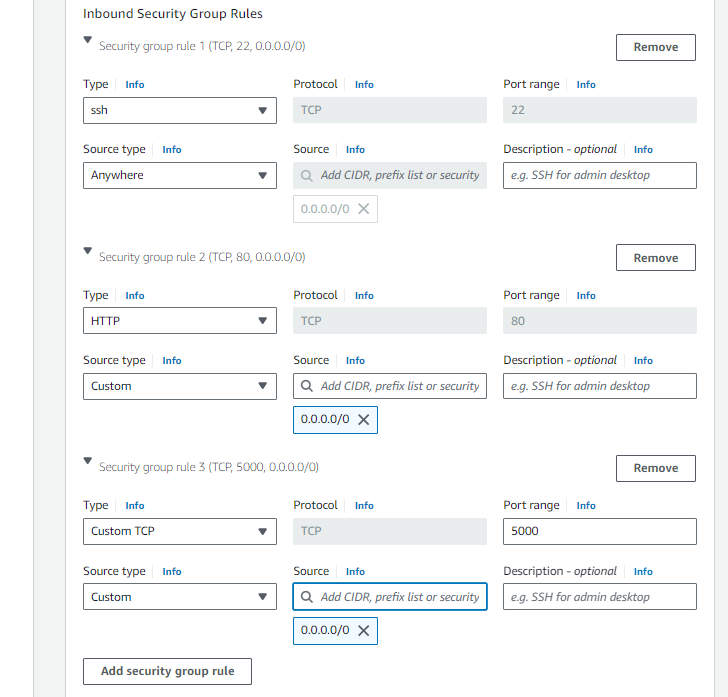


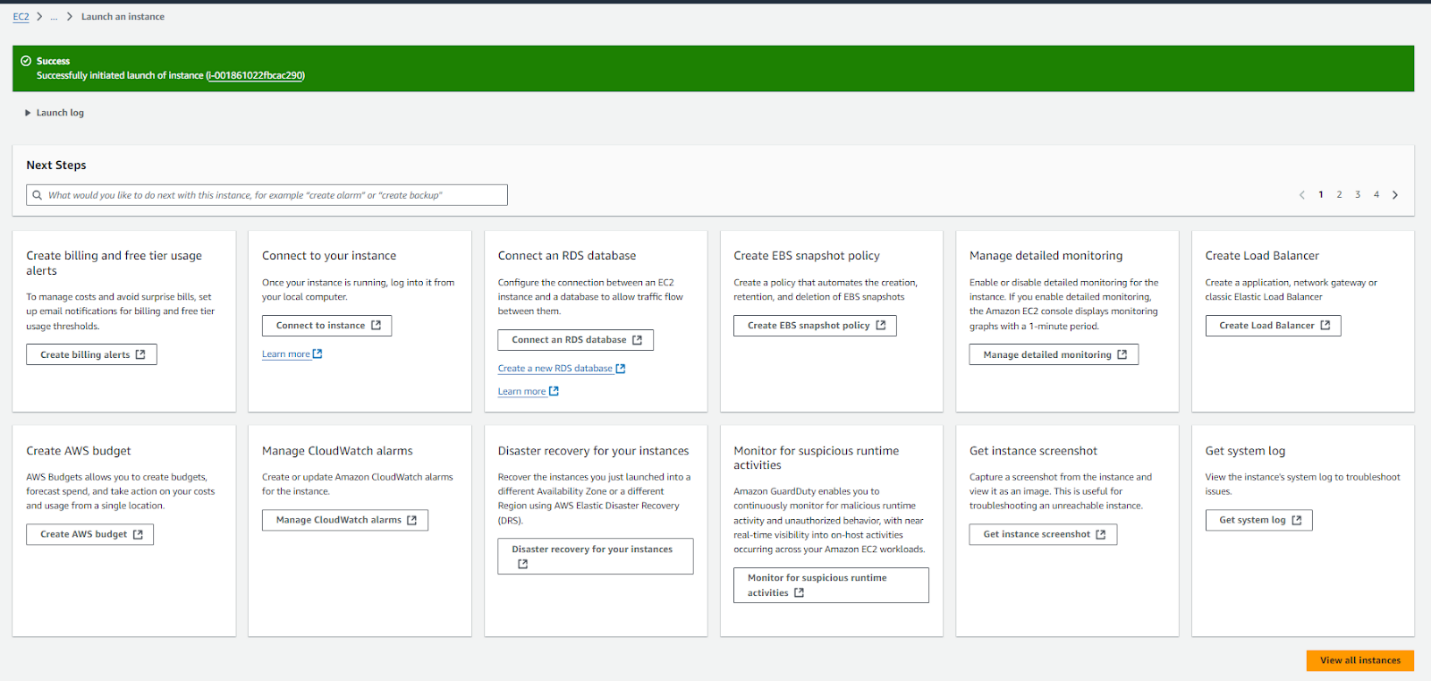
**Configure security groups for HTTP, and SSH access.**

For network settings during EC2 instance launch:

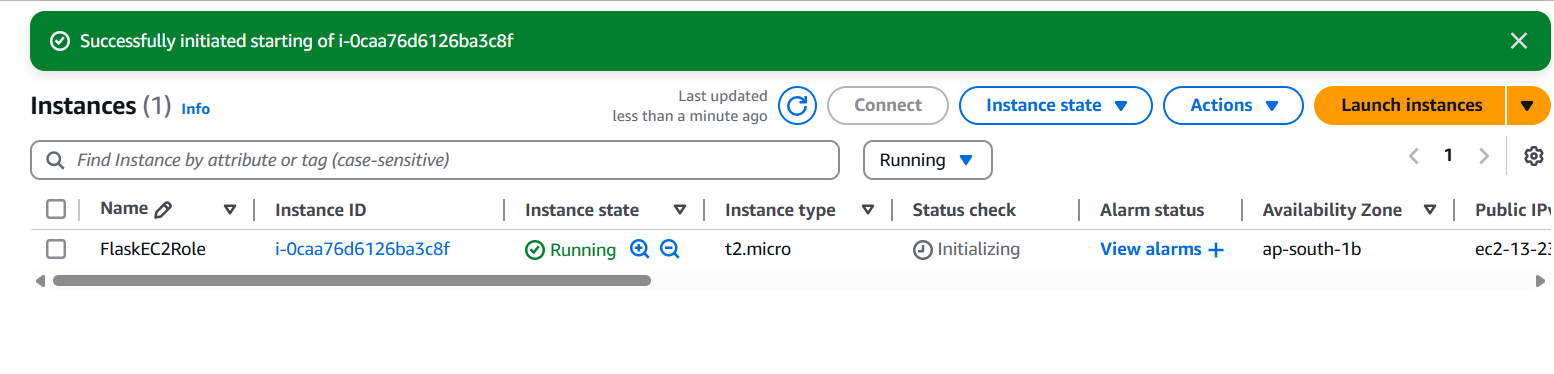
1. In the **Network Settings** section, select the **VPC** and **Subnet** you wish to use (if unsure, the default VPC and subnet should work).
2. Ensure **Auto-assign Public IP** is enabled so your instance can be accessed from the internet.
3. In **Security Group**, either select an existing one or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



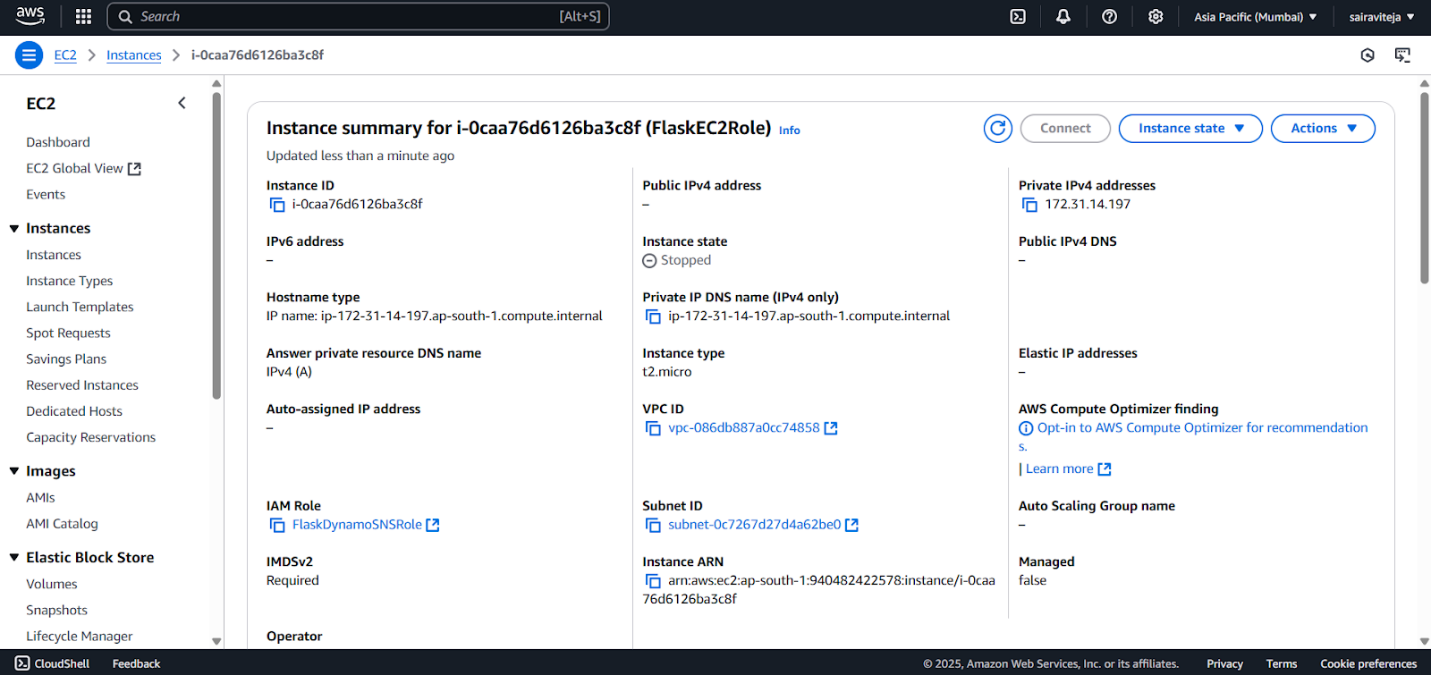




* To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected, navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the EC2 Dashboard, click the Connect button. From the connection methods presented, choose EC2 Instance Connect. Finally, click Connect again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

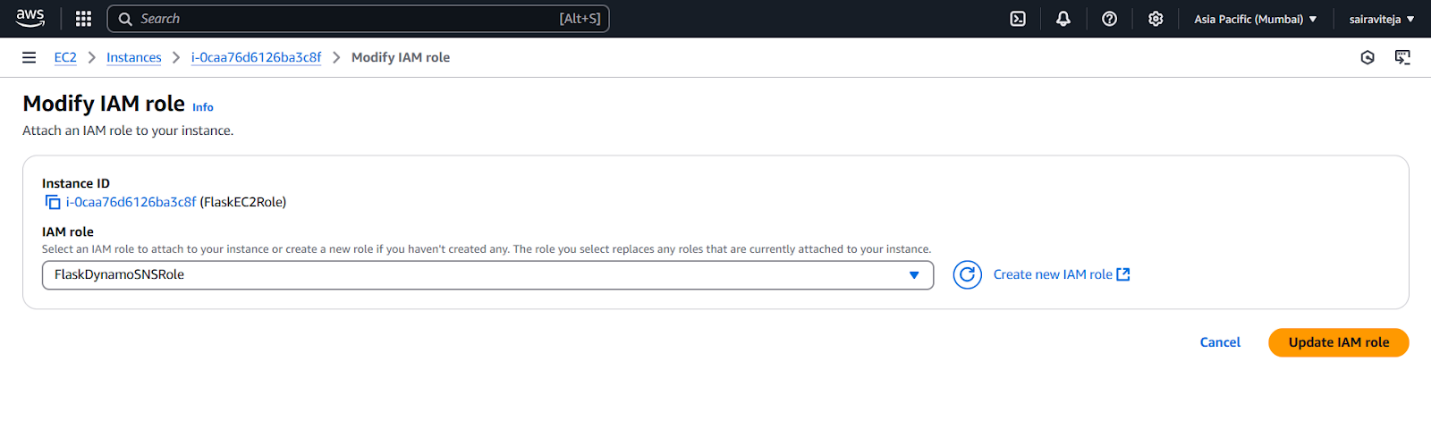


* The EC2 instance you are launching is configured with Amazon Linux 2 or Ubuntu as the AMI, t2.micro as the instance type (free-tier eligible), and flaskec2role IAM role for appropriate permissions. The flaskkeypair key pair is created for secure server access via SSH, and the instance is set to auto-assign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



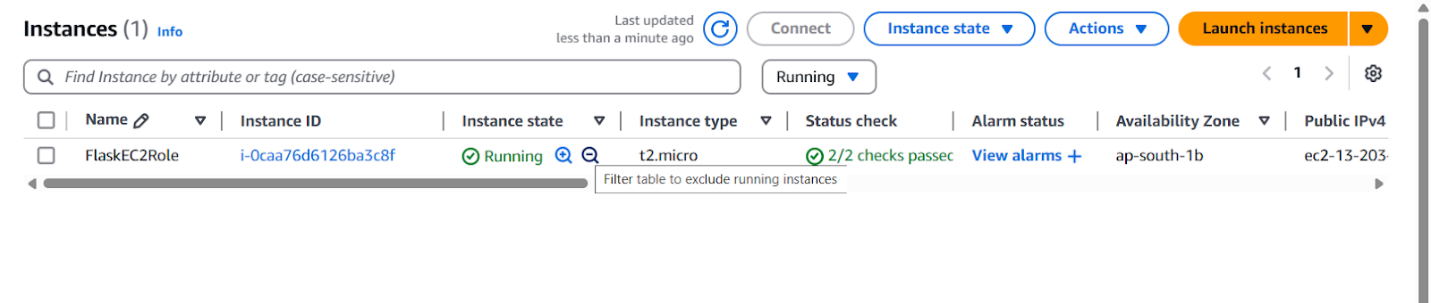
To modify the **IAM role** for your EC2 instance:

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

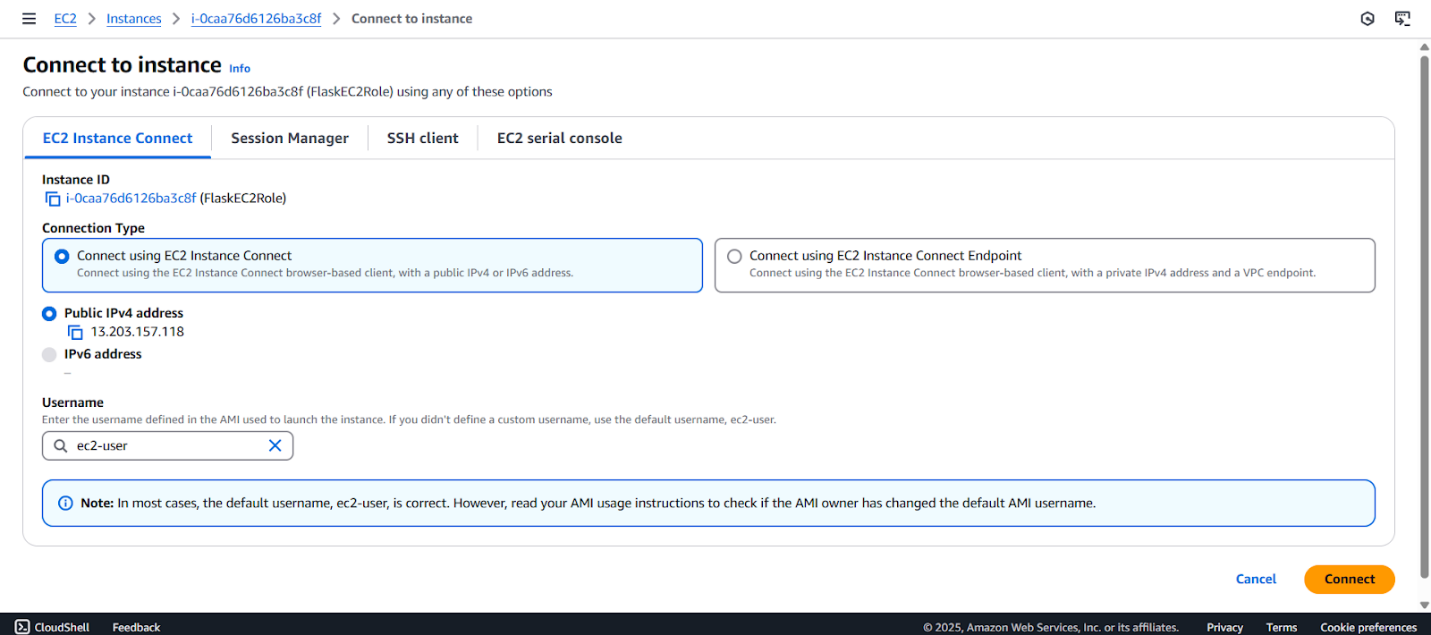


To connect to your EC2 instance:

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



* Now connect the EC2 with the files





**Milestone 7 : Deployment on EC2**

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

**Install Software on the EC2 Instance**

**Install Python3, Flask, and Git:**

**On Amazon Linux 2:**

sudo yum update -y

sudo yum install python3 git

sudo pip3 install flask boto3

**Verify Installations:**

flask --version

git --version

**Clone Your Flask Project from GitHub**

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

Run: ‘git clone <https://github.com/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/Ravi-teja-777/medtrack\_app.git

* This will download your project to the EC2 instance.

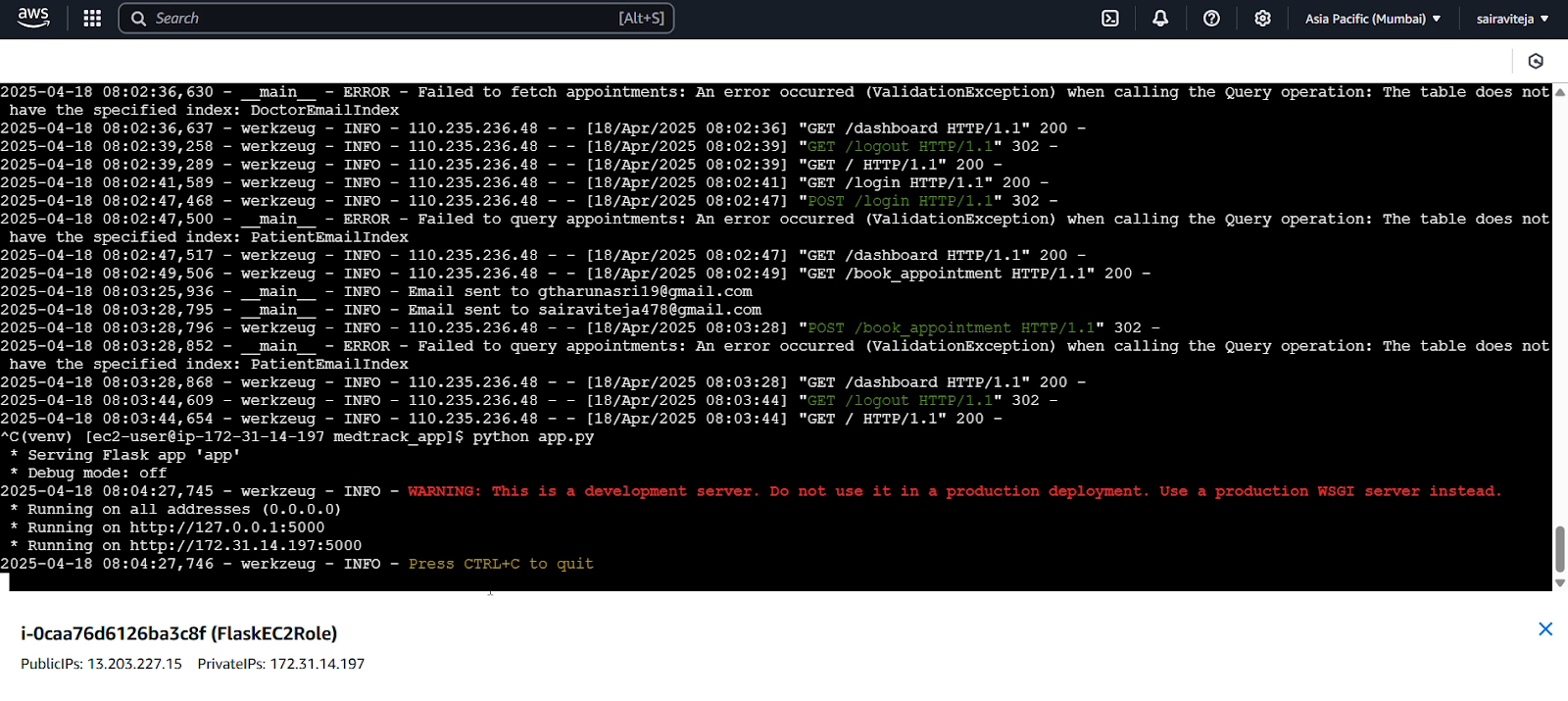
**To navigate to the project directory, run the following command:**

cd Medtrack

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

**Run the Flask Application**

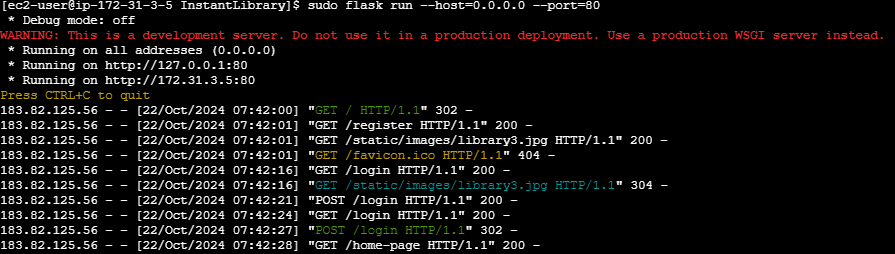
sudo flask run --host=0.0.0.0 --port=5000



**Verify the Flask app is running**:

http://your-ec2-public-ip

* Run the Flask app on the EC2 instance



**Access the website through:**

**PublicIPs:** https//52.73.204.225

**Milestone 8: Testing and Deployment**

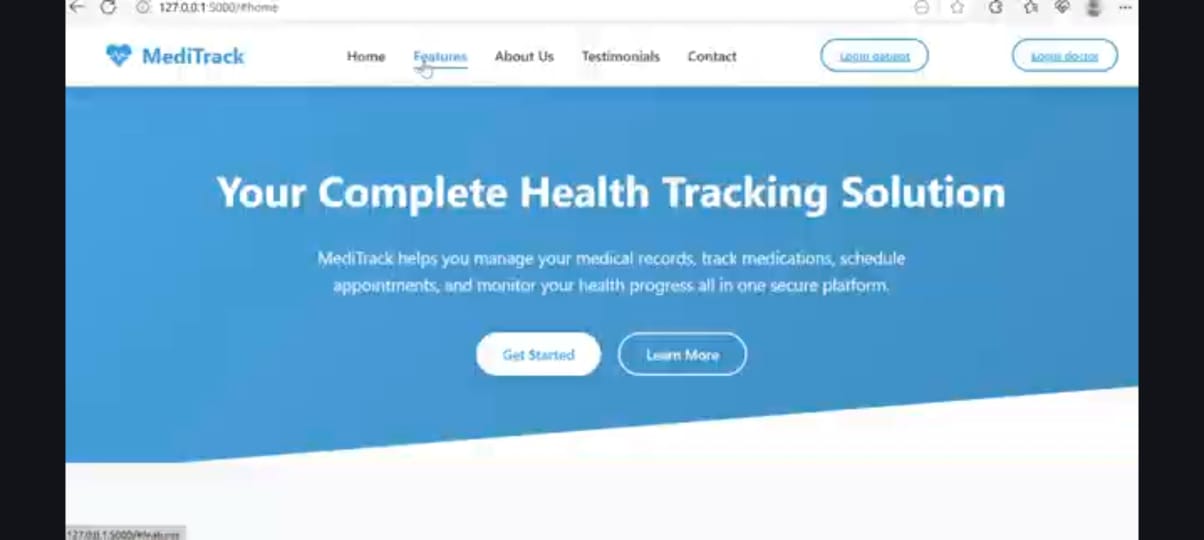
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and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

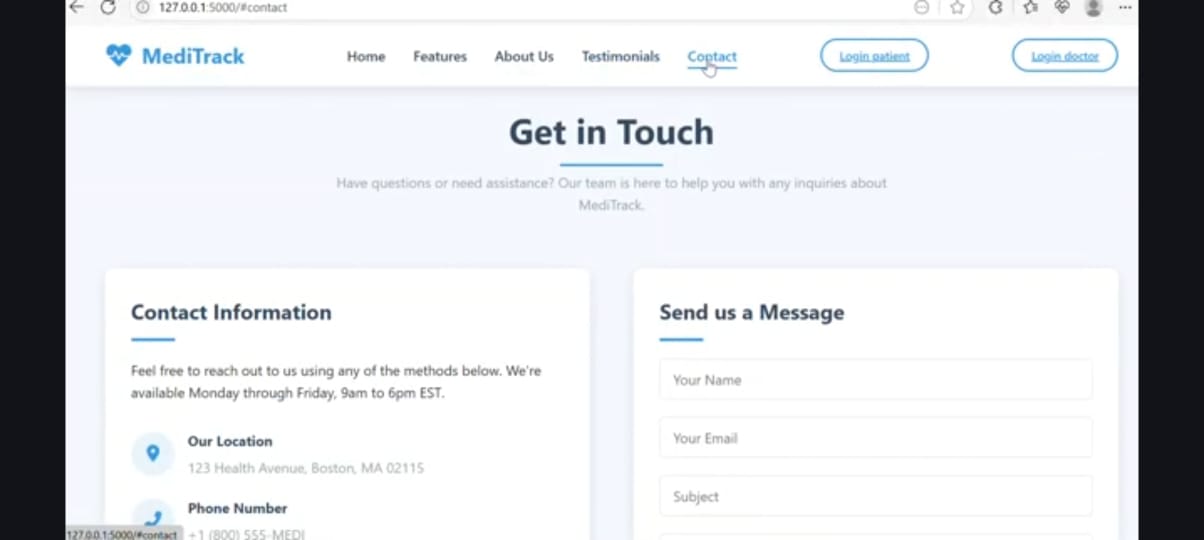
**Functional testing to verify the project**

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

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

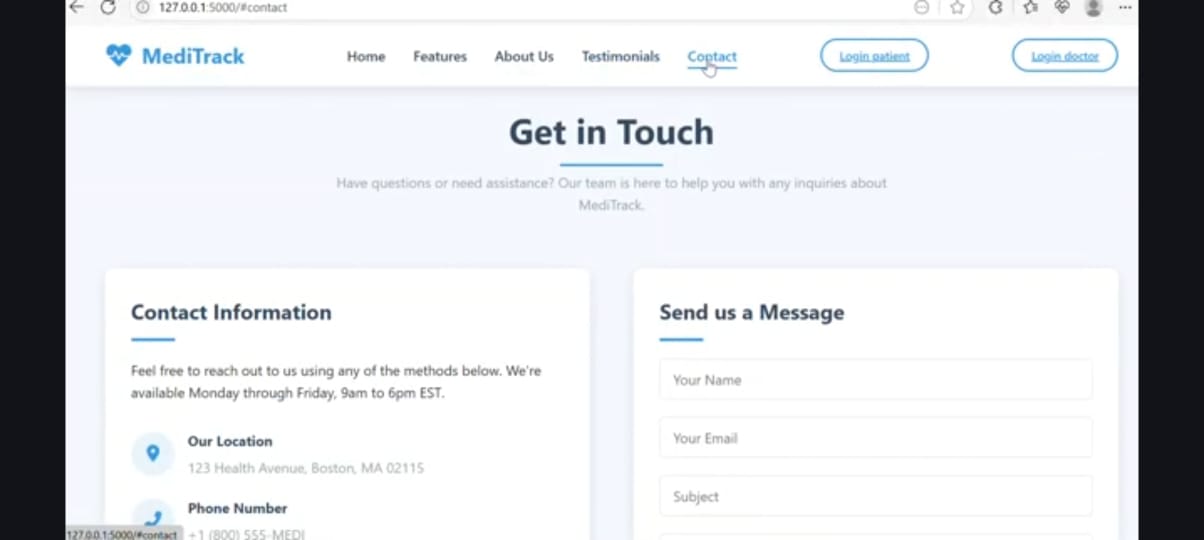
**The Home Page serves as the initial interface that directs users to the key functionalities of your web application.**

****



**DOCTOR AND PATIENT REGISTRATION PAGE :**

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

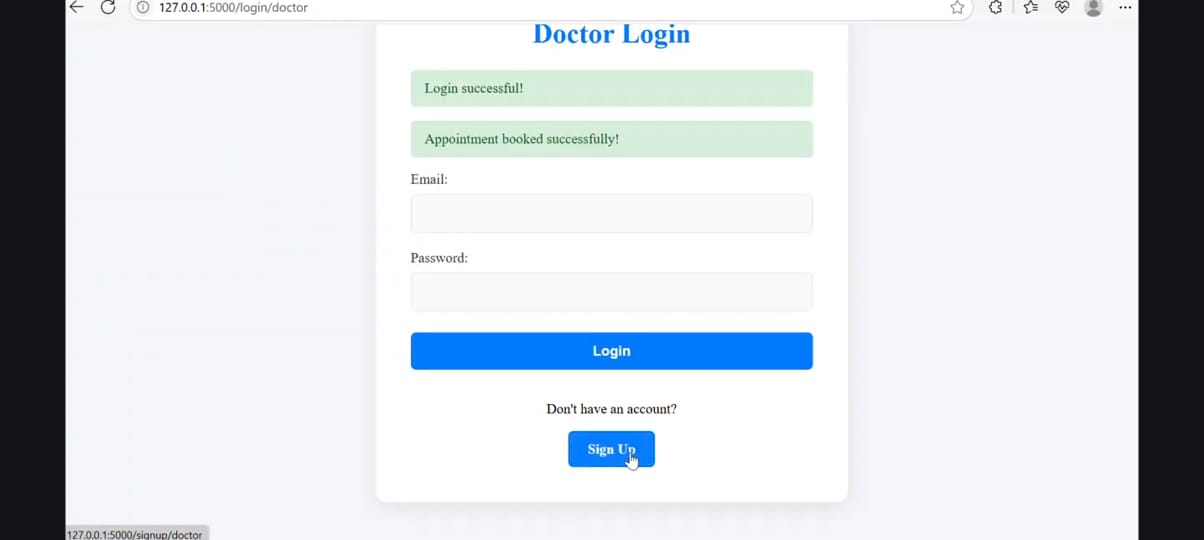
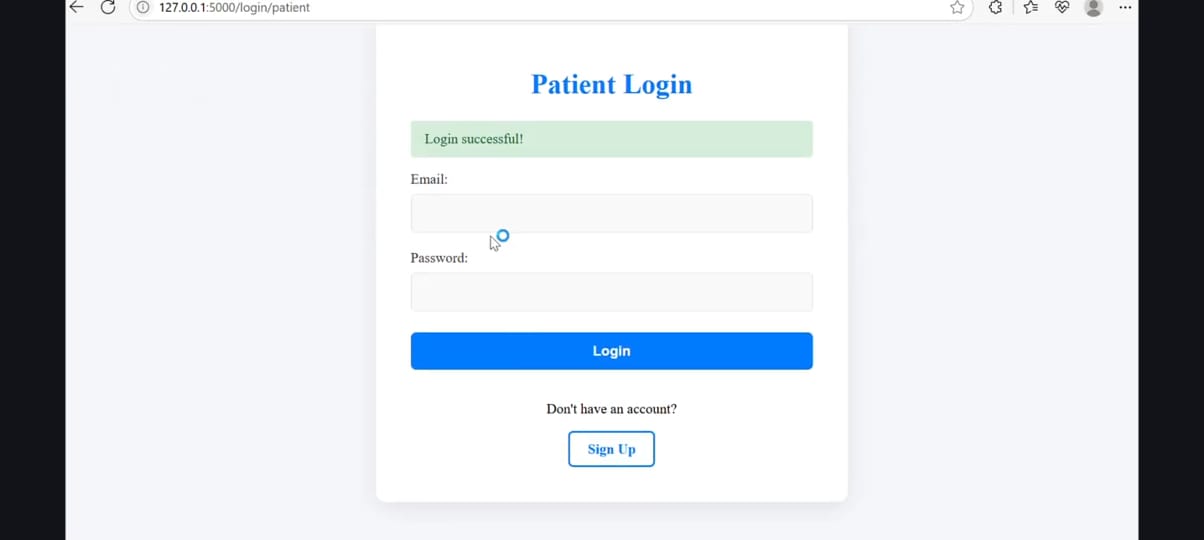
1. **Input Fields: For doctor details such as name, specialty, qualifications, and contact information.**
2. **Login Credentials: Fields for setting a username and password for secure access.**
3. **Submit Button: A button to submit the registration details, which will then be stored in the database after validation.**
4. 

**PATIENT AND DOCTOR LOGIN PAGES:**

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

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

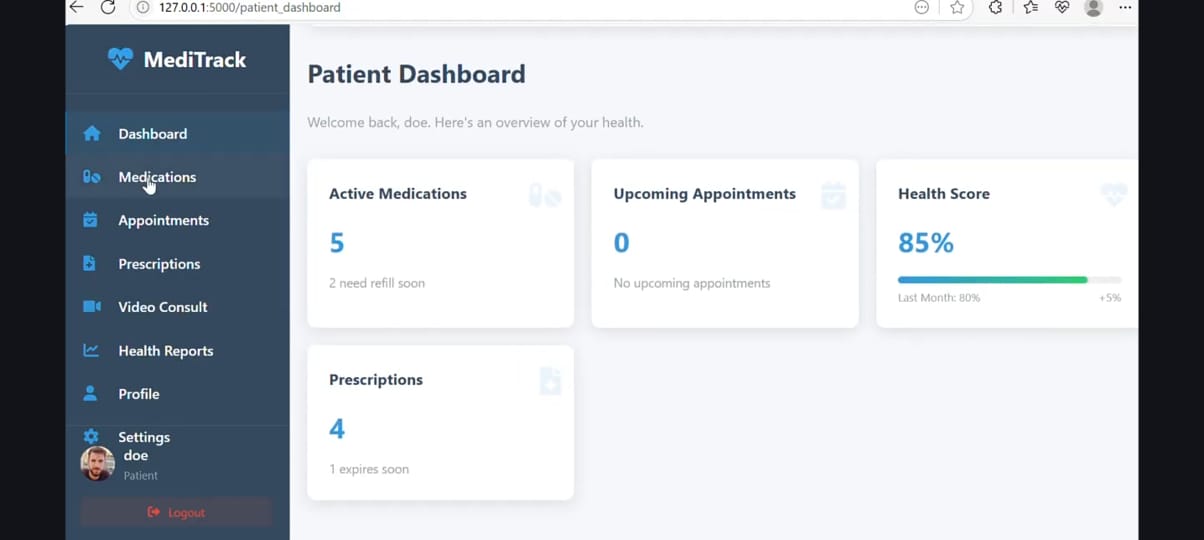
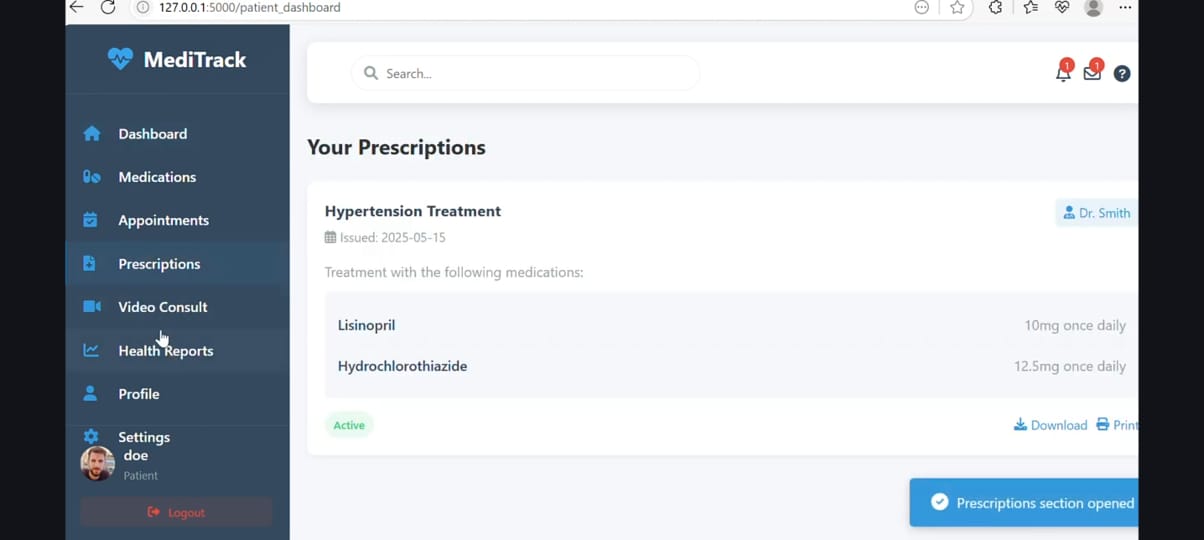
**Once logged in, patients and doctors are redirected to their respective dashboards to manage appointments, medical records, and other relevant tasks.**

**Patient Dashboard:**

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

1. **Book Appointment Section: A form for selecting a doctor, choosing an appointment time, and submitting the request.**
2. **Appointment Status: A section showing the current status of appointments (e.g., confirmed, pending, or completed) with options to view details or cancel.**
3. **Upcoming Appointments: A list of future appointments with relevant details such as doctor name, date, and time.**

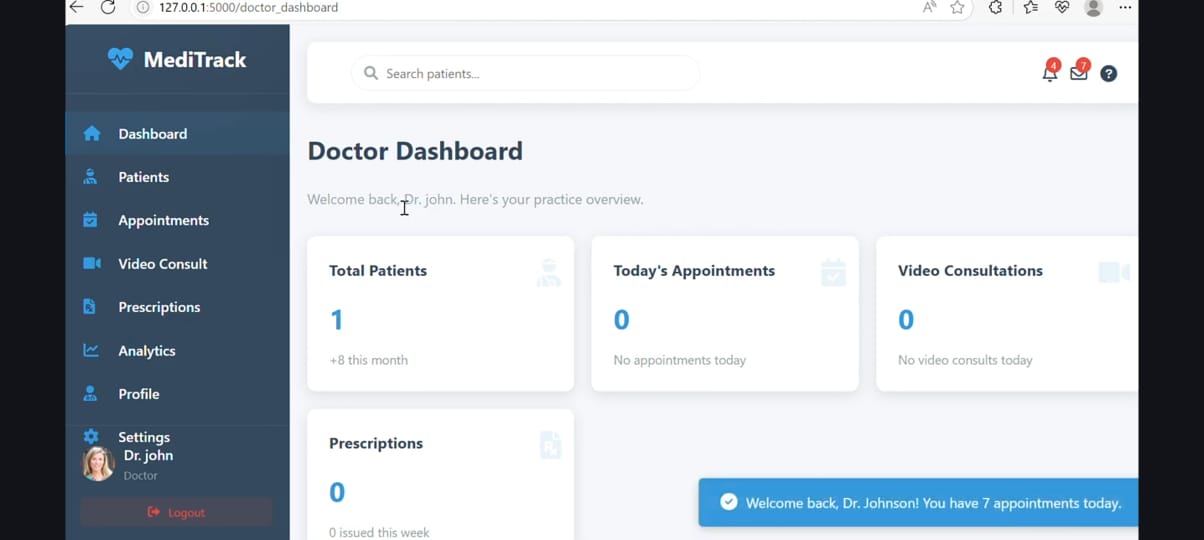
**This dashboard helps patients book new appointments and keep track of their healthcare schedules.**

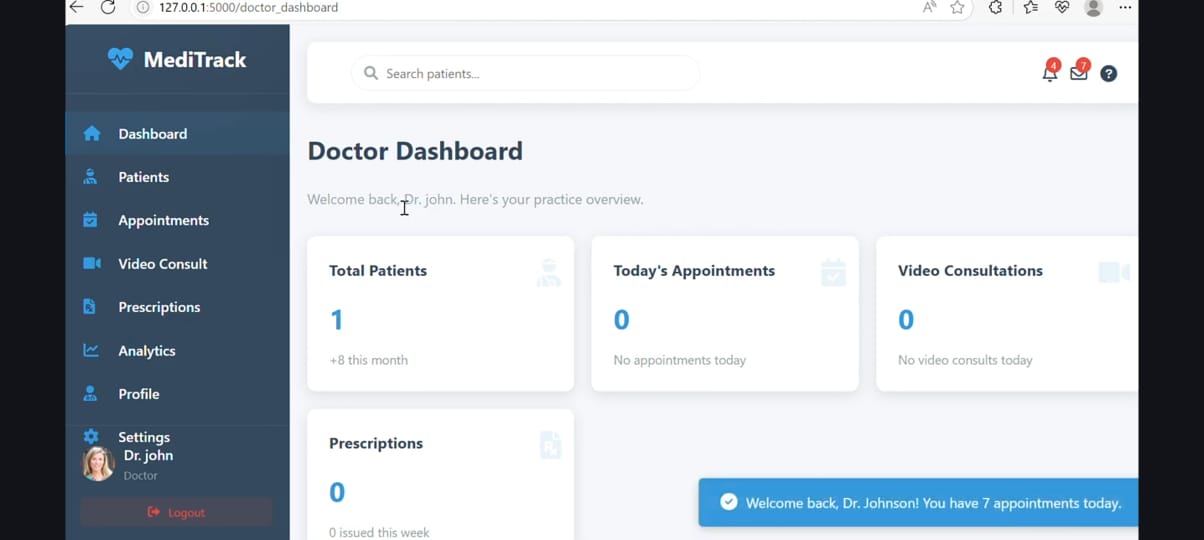
  **Doctor Dashboard:**

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

1. **Upcoming Appointments List: A table or list showing patient names, appointment times, and appointment statuses (e.g., confirmed, pending).**
2. **Patient Details: Quick access to each patient's medical history, contact information, and previous visit records.**
3. **Appointment Actions: Options to view, confirm, reschedule, or cancel appointments, ensuring efficient management.**

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

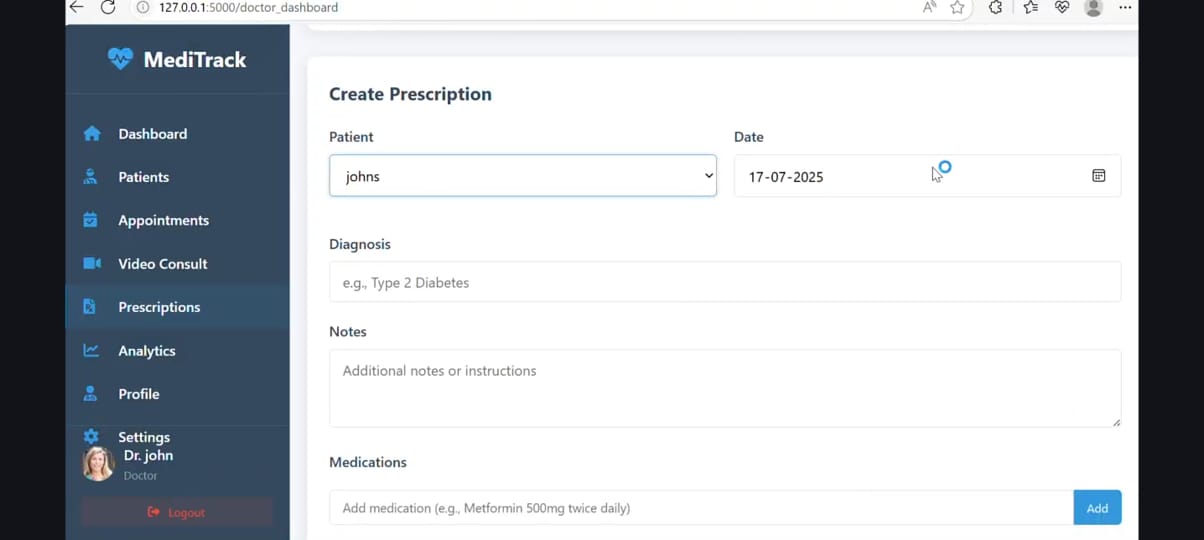


 **Search Appointment:**

**The Search Appointment feature in the Doctor Dashboard allows doctors to quickly find and manage specific patient appointments. It typically includes:**

1. **Search Bar: A field where doctors can enter a patient's name, appointment date, or other relevant details to filter appointments.**
2. **Appointment List: Displays the search results, including patient names, appointment times, and statuses, for easy access.**
3. **Action Options: Options to view, update, or cancel appointments directly from the search results.**

**This feature streamlines appointment management, making it easier for doctors to find and handle patient requests.**

 **Conclusion:**

The **MedTrack application** has been successfully developed and deployed using a robust cloud-based architecture tailored for modern healthcare environments. Leveraging AWS services such as EC2 for hosting, DynamoDB for secure and scalable patient data management, and SNS for real-time alerts, the platform ensures reliable and efficient access to essential medical tracking services. This system addresses critical challenges in healthcare such as managing patient records, monitoring medication schedules, and ensuring timely communication between healthcare providers and patients.

The cloud-native approach enables seamless scalability, allowing MedTrack to support increasing numbers of users and data without compromising performance or reliability. The integration of Flask with AWS ensures smooth backend operations, including patient registration, medication reminders, and health updates. Thorough testing has validated that all features—from user onboarding to alert notifications—function reliably and securely.

In conclusion, the MedTrack application delivers a smart, efficient solution for modernizing healthcare management, improving patient care, and streamlining communication between medical staff and patients. This project highlights the transformative power of cloud-based technologies in solving real-world challenges in the healthcare sector.