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.

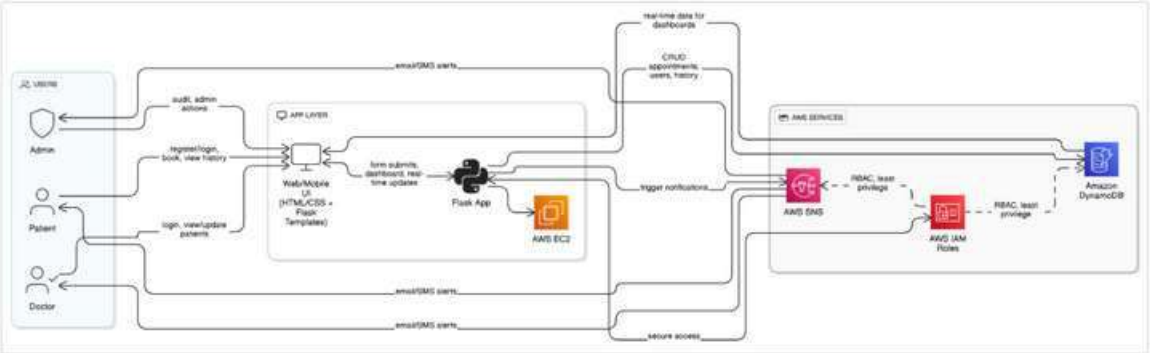
Scenario 2: Secure User Management with IAM

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.

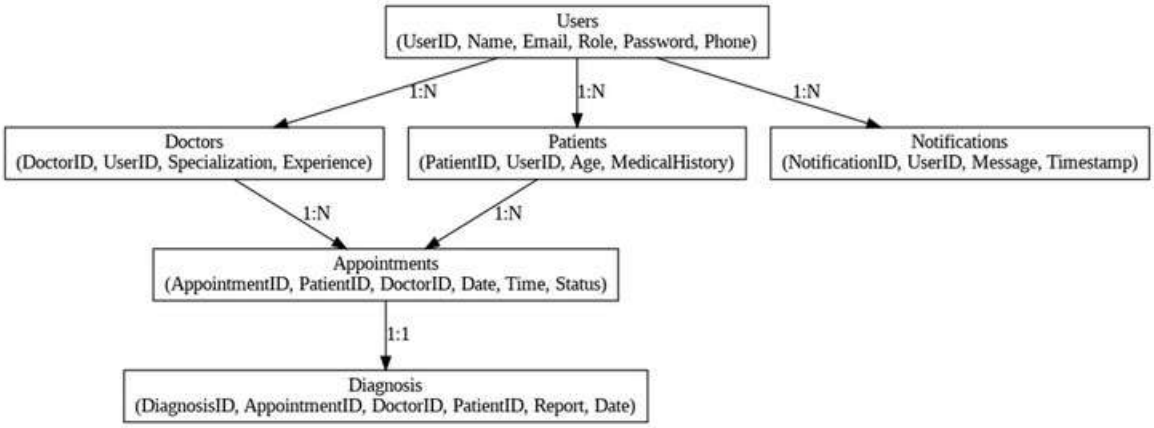
Scenario 3: Easy Access to Medical History and Resources

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:

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

Project WorkFlow:

Milestone 1. Web Application Development and Setup

Activity 1.1: Develop the Backend Using Flask**.**

Activity 1.2:Integrate AWS Services Using boto3**.**

Milestone 2. AWS Account Setup and Login

Activity 2.1: Set up an AWS account if not already done.

Activity 2.2:Login to AWS Management Console**.**

Milestone 3. DynamoDB Database Creation and Setup

Activity 3.1: Create a DynamoDB Table.

Activity 3.2: Configure Attributes for User Data and Book Requests.

Milestone 4. SNS Notification Setup

Activity 4.1:Create SNS topics for book request notifications**.**

Activity 4.2: Subscribe users and library staff to SNS email notifications**.**

Milestone 5. IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

Milestone 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.

Milestone 7. Deployment on EC2

Activity 7.1:Upload Flask Files

Activity 7.2: Run the Flask App

Milestone 8. Testing and Deployment

Activity 8.1: 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

Important Instructions:

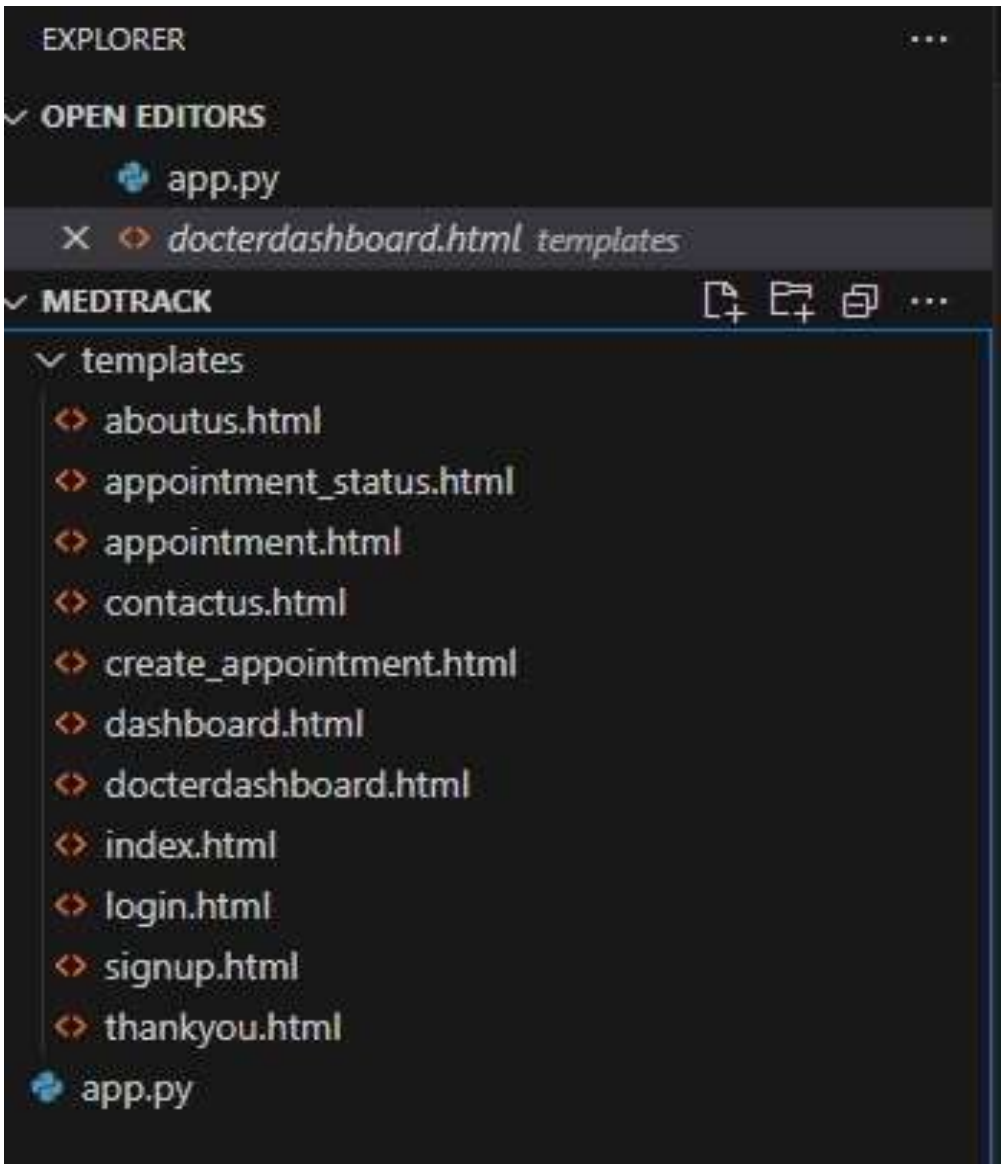
- Start by creating the necessary HTML pages and Flask routes (app.py) to build the core functionality of your application.
- During the initial development phase, store and retrieve data using Python dictionaries or lists locally. This will allow you to design, test, and validate your application logic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

Post Troven Access Activation:

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

FLASK DEPLOYMENT

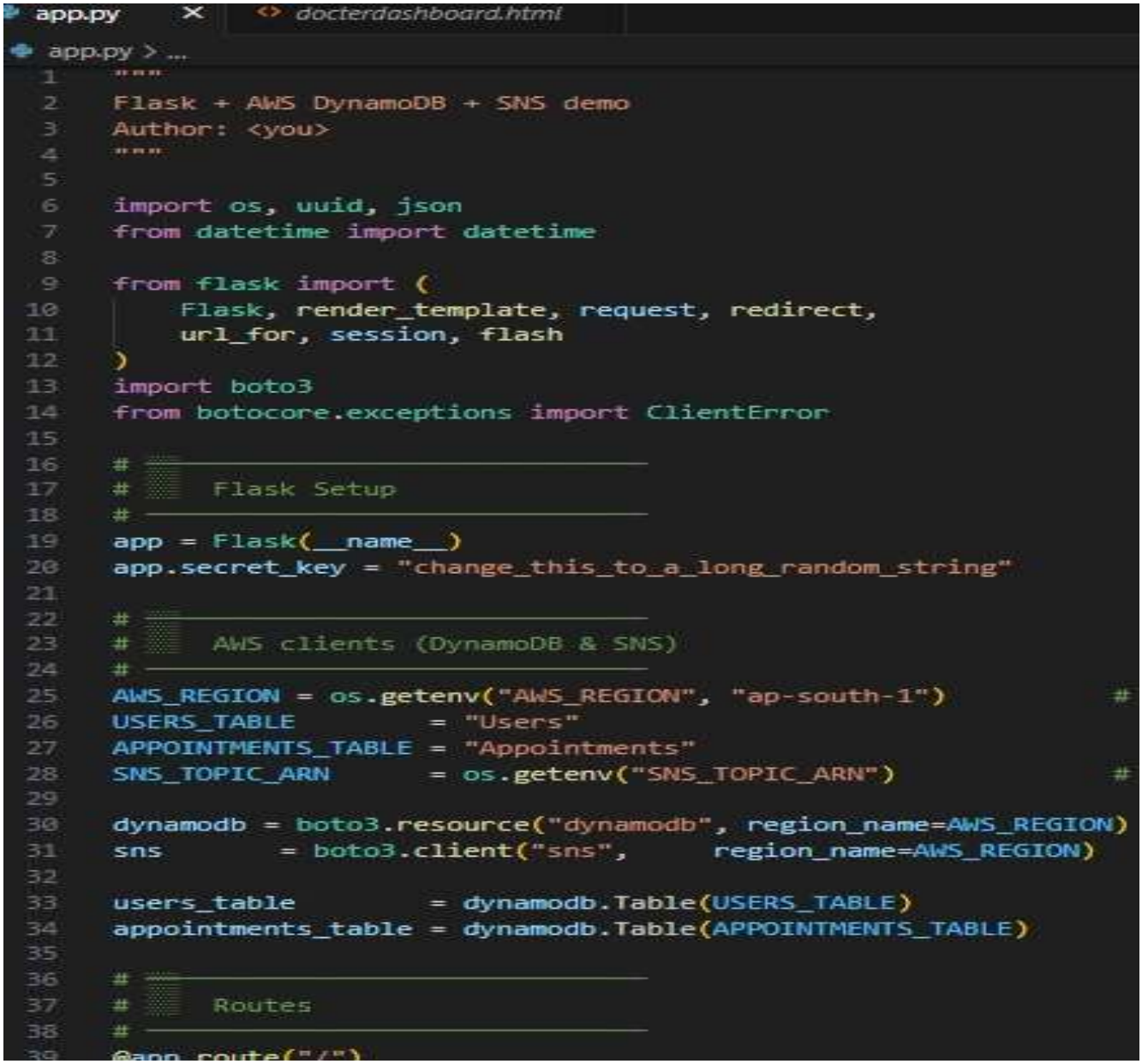
- File Explorer Structure



Description of the code :

Flask App Initialization:
In the MedTrack project, the Flask app is initialized to establish the backend infrastructure, enabling it to handle multiple user interactions such as patient registration, appointment

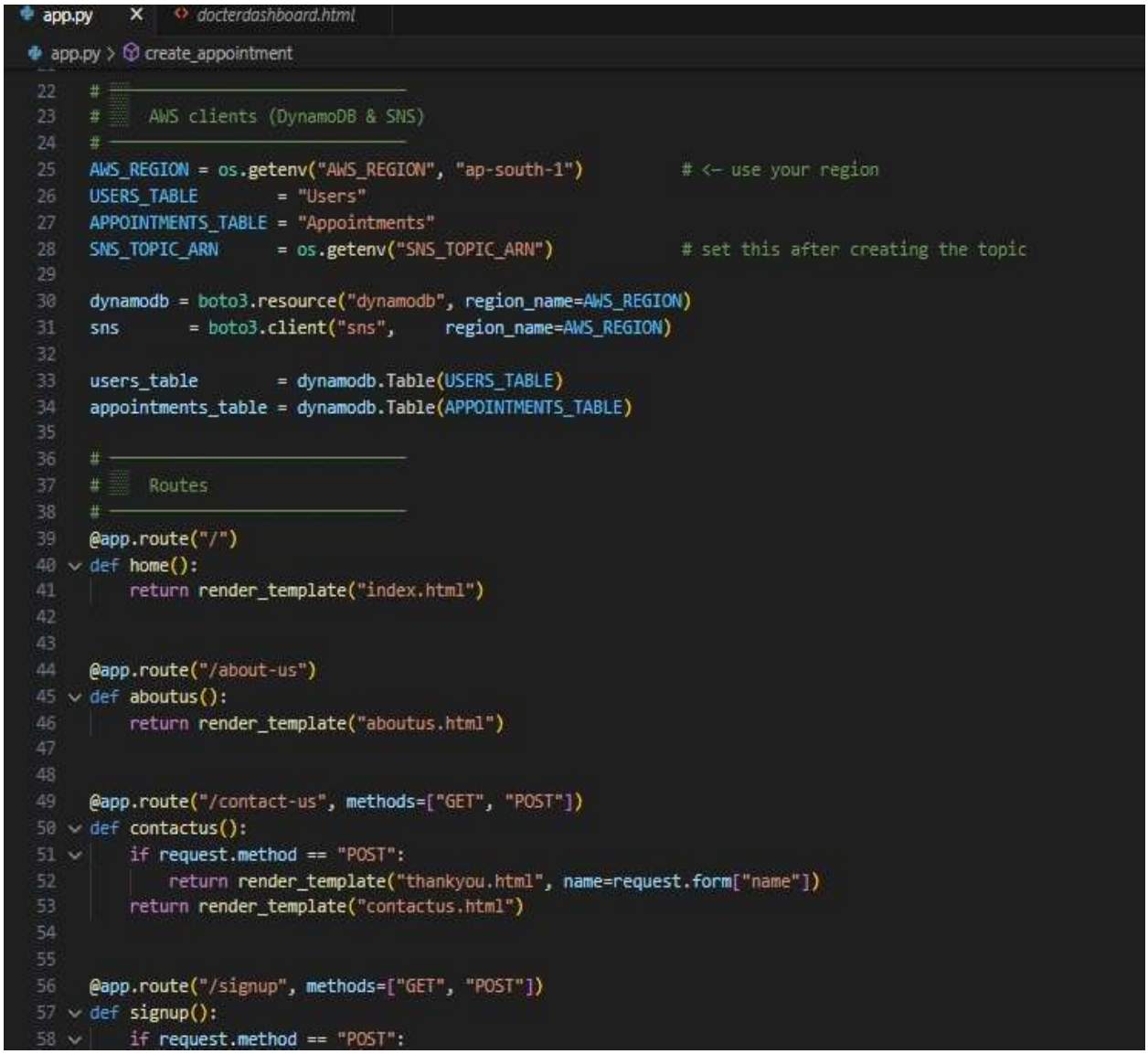
booking, and submission of medical reports. The Flask framework processes incoming requests, communicates with the DynamoDB database for storing user data, and integrates seamlessly with AWS services. Additionally, the routes and APIs are defined to manage different functionalities like secure login, appointment scheduling, and medical history retrieval. This initialization sets up the foundation for smooth, real-time communication between patients and doctors while ensuring the app is scalable and secure.



- Use boto3 to connect to DynamoDB for handling user registration, book requests database operations and also mention region_name where Dynamodb tables are created.

SNS and Dynamodb initialization:

- In the MedTrack project, AWS SNS sends real-time notifications to patients and doctors about appointments and updates. DynamoDB stores user data, medical records, and appointments securely, offering fast, scalable access. Both services are integrated with Flask to ensure smooth communication and efficient data management.



```python
# ──────────────────────────────
#   AWS clients (DynamoDB & SNS)
# ──────────────────────────────
AWS_REGION = os.getenv("AWS_REGION", "ap-south-1")          # <- use your region
USERS_TABLE        = "Users"
APPOINTMENTS_TABLE = "Appointments"
SNS_TOPIC_ARN      = os.getenv("SNS_TOPIC_ARN")             # set this after creating the topic

dynamodb = boto3.resource("dynamodb", region_name=AWS_REGION)
sns      = boto3.client("sns",     region_name=AWS_REGION)

users_table        = dynamodb.Table(USERS_TABLE)
appointments_table = dynamodb.Table(APPOINTMENTS_TABLE)

# ──────────────────────────────
#   Routes
# ──────────────────────────────
@app.route("/")
def home():
    return render_template("index.html")


@app.route("/about-us")
def aboutus():
    return render_template("aboutus.html")


@app.route("/contact-us", methods=["GET", "POST"])
def contactus():
    if request.method == "POST":
        return render_template("thankyou.html", name=request.form["name"])
    return render_template("contactus.html")


@app.route("/signup", methods=["GET", "POST"])
def signup():
    if request.method == "POST":
```

- **SNS Connection**

Configure SNS to send notifications when a book request is submitted. Paste your stored ARN link in the sns_topic_arn space, along with the region_name where the SNS topic is created. Also, specify the chosen email service in SMTP_SERVER (e.g., Gmail, Yahoo, etc.) and enter the subscribed email in the SENDER_EMAIL section. Create an 'App password' for the email ID and store it in the SENDER_PASSWORD section.

app.py > create_appointment

```
120    def create_appointment():
149            # — 2.  Publish SNS confirmation
150            if SNS_TOPIC_ARN:
151                msg = {
152                    "appointment_id": appointment_id,
153                    "user": session["user"],
154                    "doctor": doctor,
155                    "date": date,
156                    "time": time
157                }
158                try:
159                    sns.publish(
160                        TopicArn=SNS_TOPIC_ARN,
161                        Subject="New Appointment Booked",
162                        Message=json.dumps(msg, indent=2)
163                    )
164                except ClientError as e:
165                    # don't block user; just log/flash if needed
166                    app.logger.error(f"SNS publish failed: {e}")
167
168            return render_template(
169                "appointment_status.html",
170                doctor=doctor, date=date, time=time, symptoms=symptoms
171            )
172
173        return render_template("appointment.html")
174
175
176    @app.route("/logout")
177    def logout():
178        session.clear()
179        flash("You have been logged out.")
180        return redirect(url_for("home"))
181
182
183    if __name__ == "__main__":
184        # debug=True for local dev only
185        app.run(debug=True, port=5000)
186
```

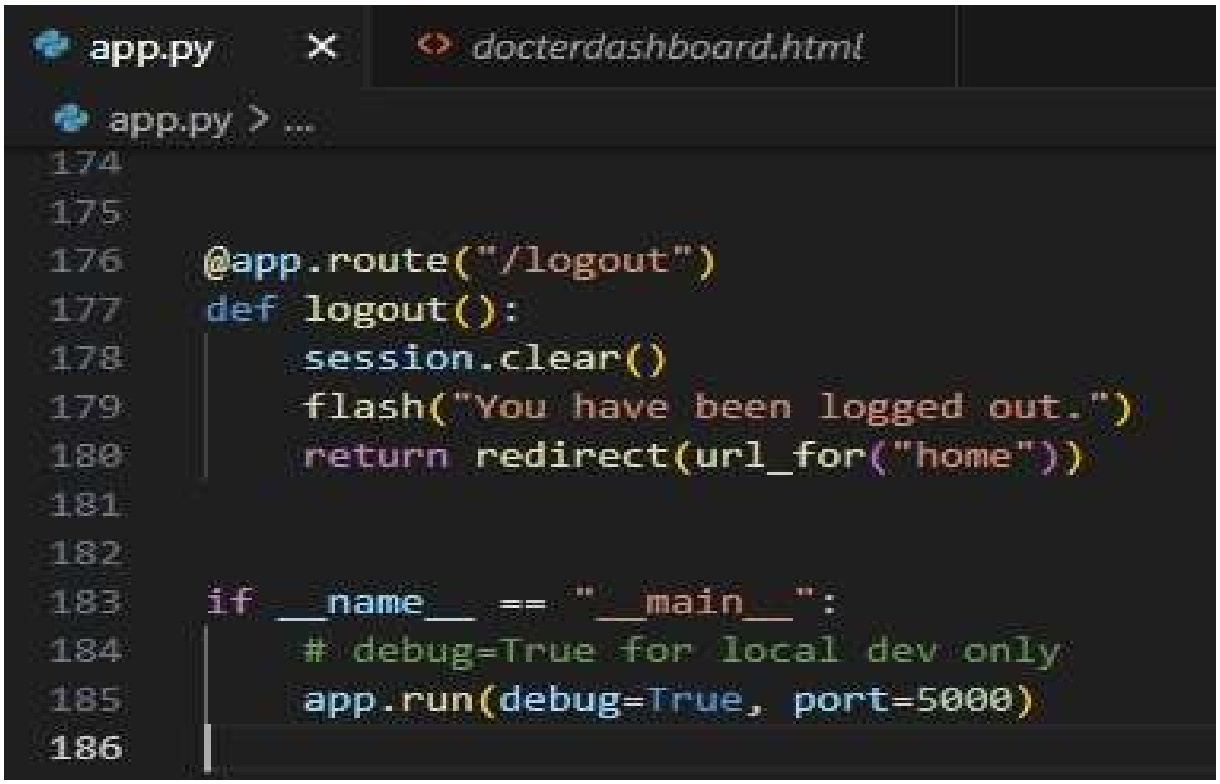
- **Routes for Web Pages:**
- **Register Page**

app.py > create_appointment

```python
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        email    = request.form.get("email")
        password = request.form.get("password")

        # fetch user
        try:
            resp = users_table.get_item(Key={"email": email})
            user = resp.get("Item")
        except ClientError as e:
            flash("Error accessing database."); return redirect(url_for("login"))

        if not user or user["password"] != password:
            flash("Invalid credentials."); return redirect(url_for("login"))

        session["user"] = user["fullname"]
        session["email"] = email
        return redirect(url_for("dashboard"))

    return render_template("login.html")


@app.route("/dashboard")
def dashboard():
    if "user" not in session:
        return redirect(url_for("login"))
    return render_template("dashboard.html", user=session["user"])
```

- The **login route** handles user authentication by verifying credentials stored in **DynamoDB**. Upon successful login, it increments the **login count** and redirects the user to their dashboard. This ensures secure access to the platform while maintaining user activity logs.

Logout Route:
The logout functionality allows users to securely end their session, clearing any session data and redirecting them to the login page. The dashboard provides users with an overview of their activities, such as upcoming appointments for patients or patient records for doctors, with relevant actions based on user roles.



Book Appointment Route:

The book appointment route allows users to select a date, time, and doctor for their appointment. Upon submission, the system stores the appointment details in DynamoDB and sends a confirmation notification via SNS. This ensures smooth scheduling and timely updates for both patients and doctors.

app.py doctordashboard.html

app.py > ...

```python

@app.route("/create-appointment", methods=["GET", "POST"])
def create_appointment():
    if "user" not in session:
        flash("Please log in to book an appointment.")
        return redirect(url_for("login"))

    if request.method == "POST":
        doctor   = request.form.get("doctor")
        date     = request.form.get("date")
        time     = request.form.get("time")
        symptoms = request.form.get("symptoms")

        # — 1.  Save appointment in DynamoDB
        appointment_id = str(uuid.uuid4())
        try:
            appointments_table.put_item(
                Item={
                    "appointment_id": appointment_id,
                    "email": session["email"],
                    "doctor": doctor,
                    "date": date,
                    "time": time,
                    "symptoms": symptoms,
                    "created_at": datetime.utcnow().isoformat()
                }
            )
        except ClientError as e:
            flash("DB error while booking appointment.")
            return redirect(url_for("create_appointment"))

```

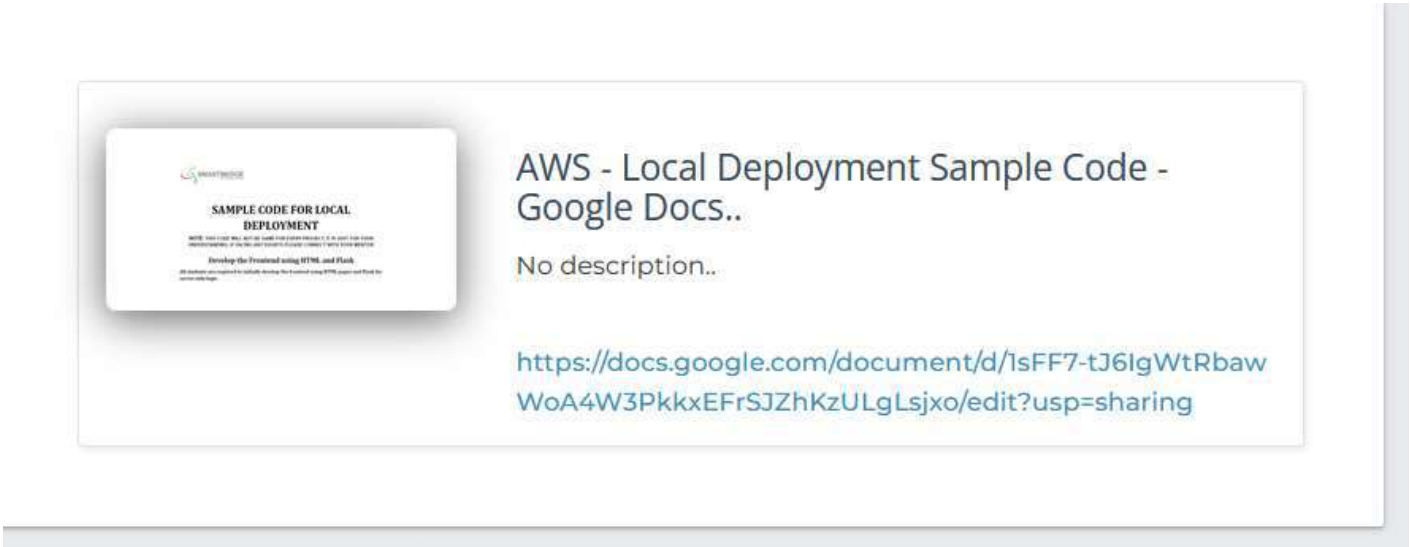
Deployment Code:

The health routing feature in the MedTrack project checks the system's status by sending a request to a specific endpoint, ensuring the backend services are functioning properly. The __name__ == '__main__' block is used in the Flask app to ensure that the application runs only if the script is executed directly, not when imported as a module, enabling local development or deployment on a server. This setup ensures that the app runs smoothly and is self-contained during execution.

app.py × docterdashboard.html

app.py > create_appointment

```
110
111
112 @app.route("/dashboard")
113 def dashboard():
114     if "user" not in session:
115         return redirect(url_for("login"))
116     return render_template("dashboard.html", user=session["user"])
117
118
119 @app.route("/create-appointment", methods=["GET", "POST"])
120 def create_appointment():
121     if "user" not in session:
122         flash("Please log in to book an appointment.")
123         return redirect(url_for("login"))
124
125     if request.method == "POST":
126         doctor   = request.form.get("doctor")
127         date     = request.form.get("date")
128         time     = request.form.get("time")
129         symptoms = request.form.get("symptoms")
130
131         # — 1.  Save appointment in DynamoDB
132         appointment_id = str(uuid.uuid4())
133         try:
134             appointments_table.put_item(
135                 Item={
136                     "appointment_id": appointment_id,
137                     "email": session["email"],
138                     "doctor": doctor,
139                     "date": date,
140                     "time": time,
```



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.

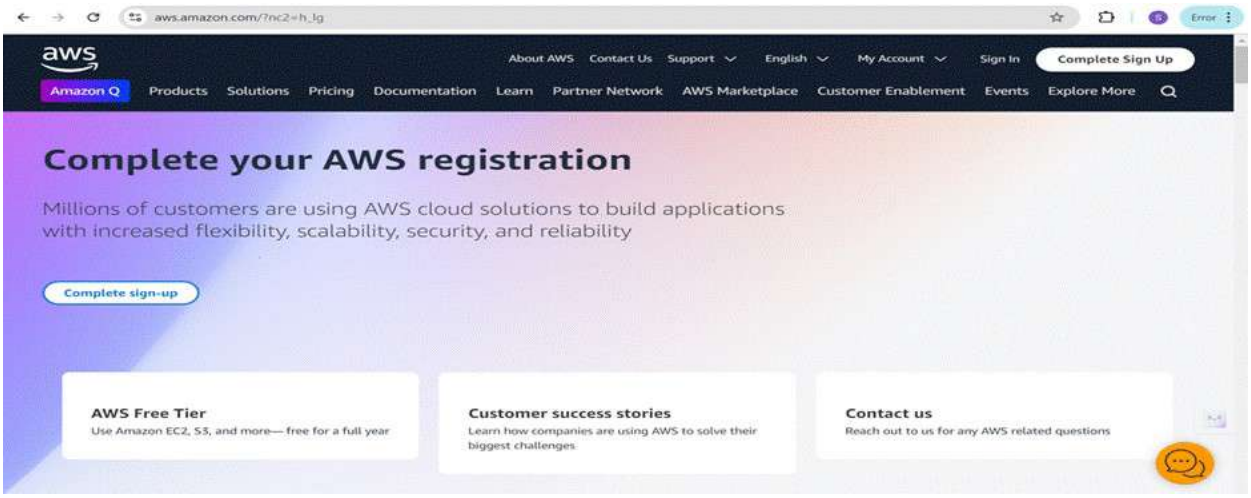
Reminder: You must complete the Web Development task before gaining access to Troven. Once accessed, the AWS Console via Troven is available for only 3 hours—please plan your work accordingly.
Please follow the provided guidelines and access AWS exclusively through Troven to avoid unnecessary issues.

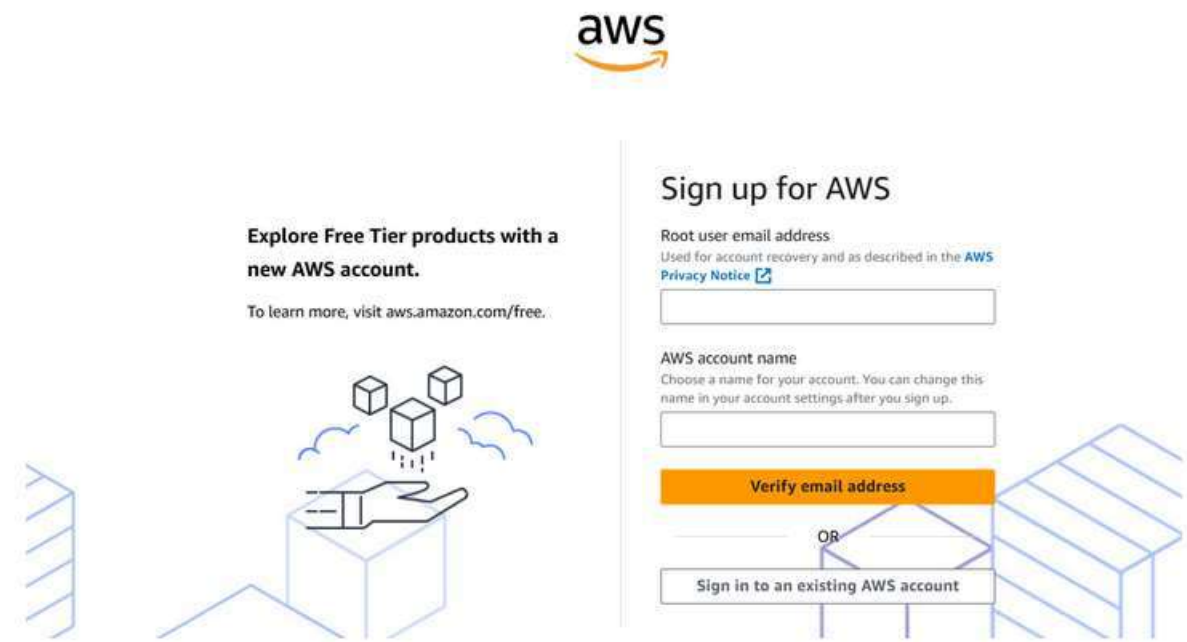
Please refer the below link -
https://drive.google.com/file/d/1HzWc7AMJ2BrxhV-uaw5s0vWtcd-28qgI/view?usp=sharing

AWS Account Setup and Login

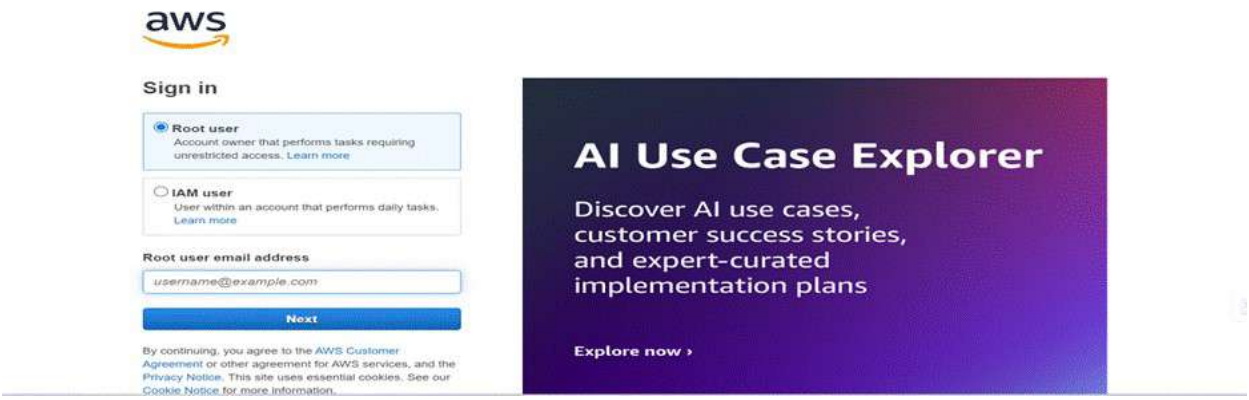
This is for your understanding only, please refrain from creating an AWS account. A temporary account will be provided via Troven.

- 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](#).

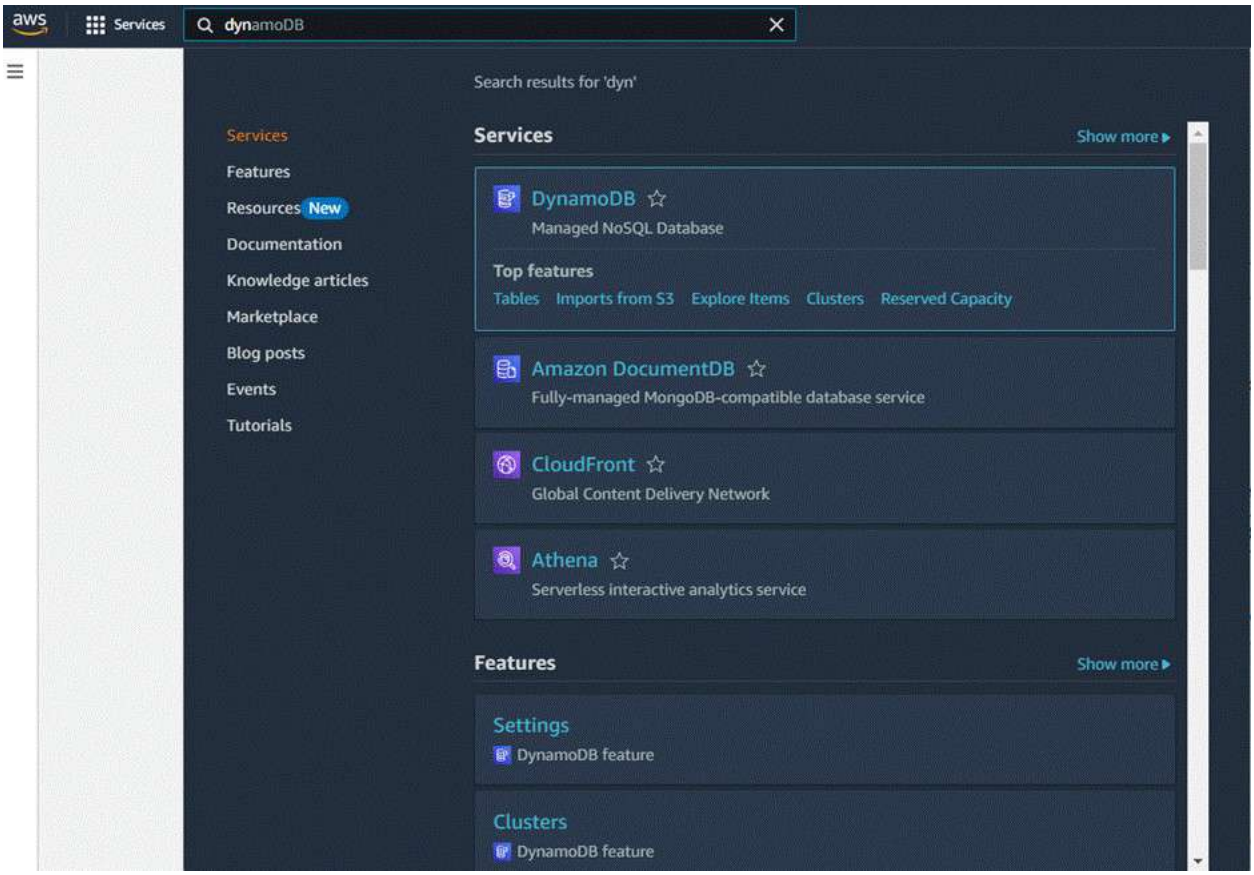


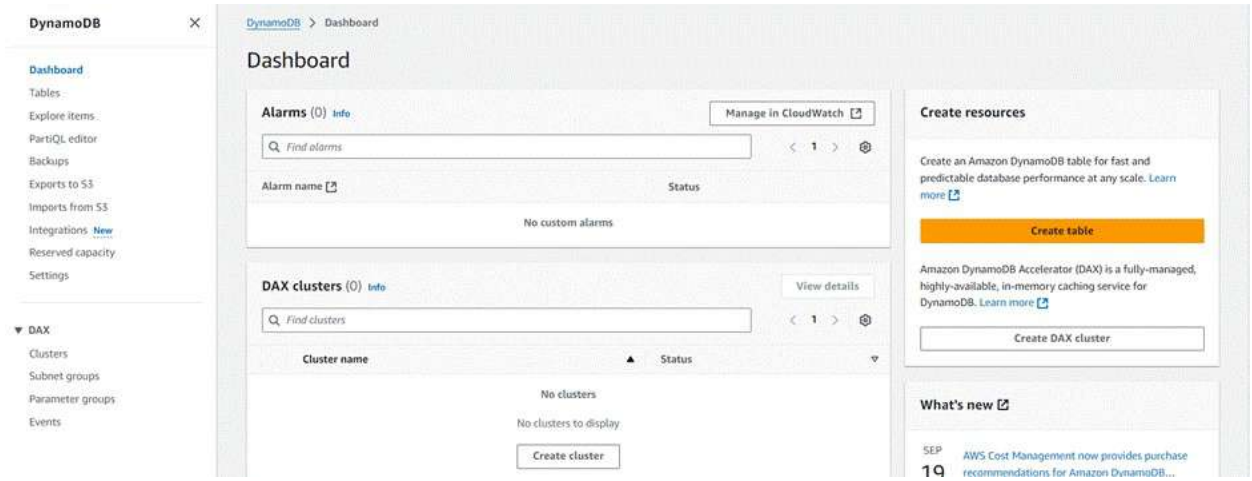
Milestone 3: DynamoDB Database Creation and Setup

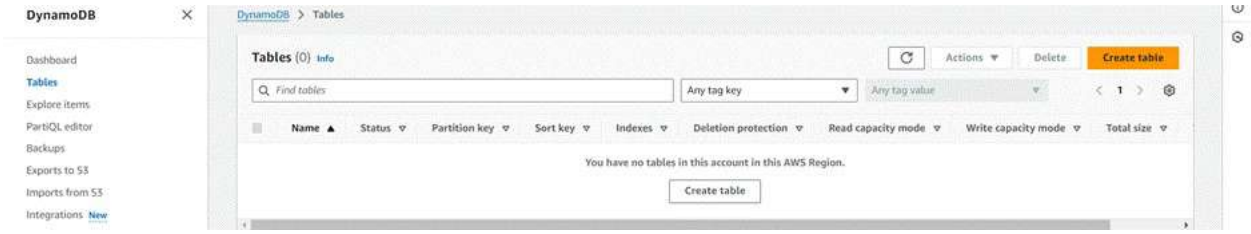
Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high-performance data storage for seamless backend operations.

Navigate to the DynamoDB

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

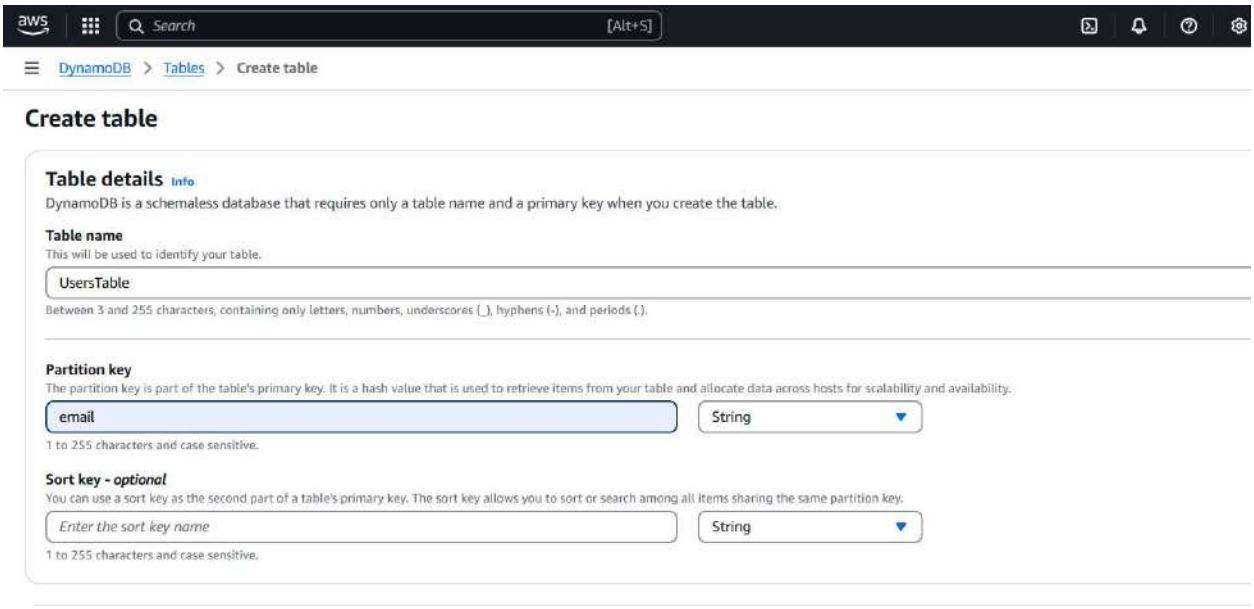


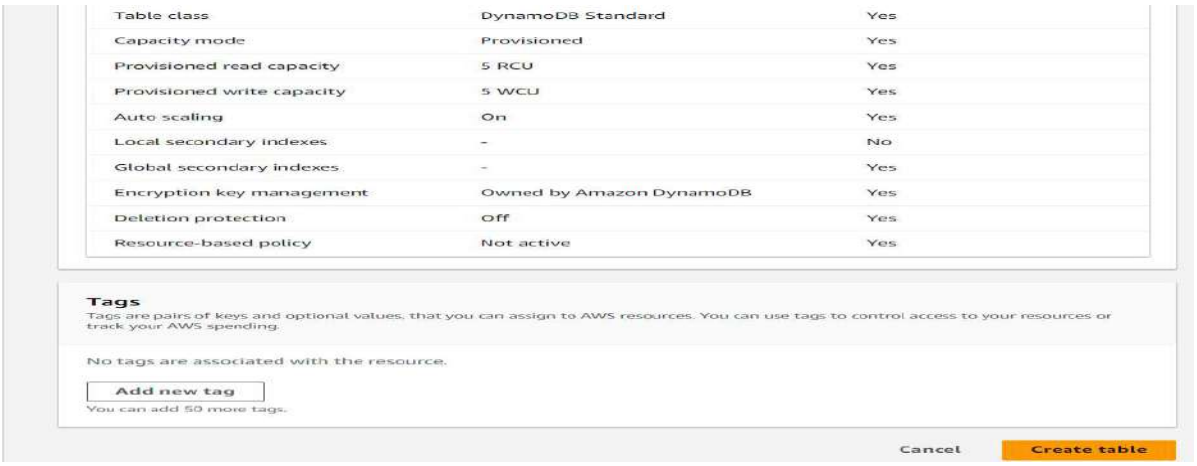




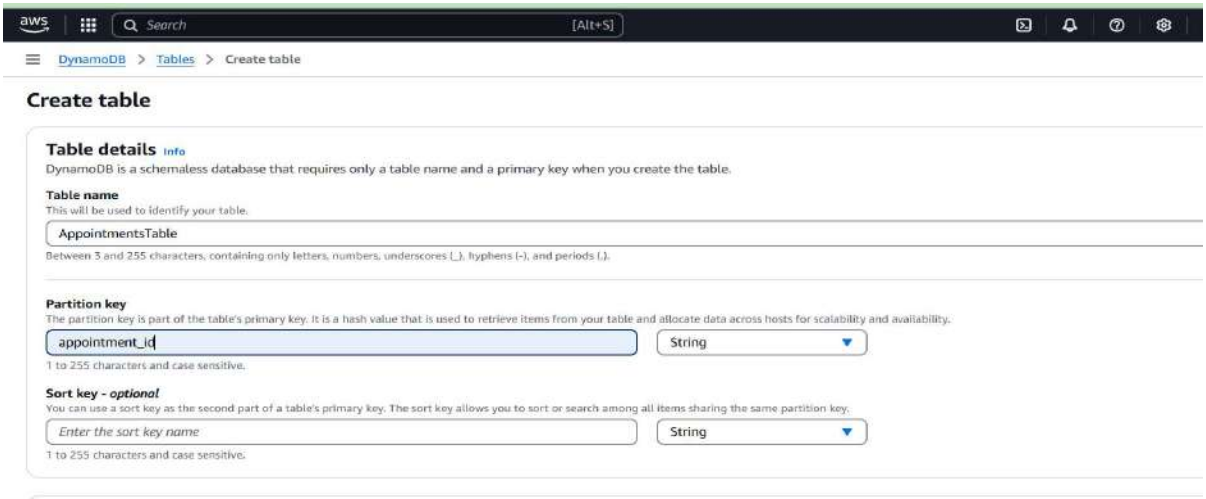
Create a DynamoDB table for storing data

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





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


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.

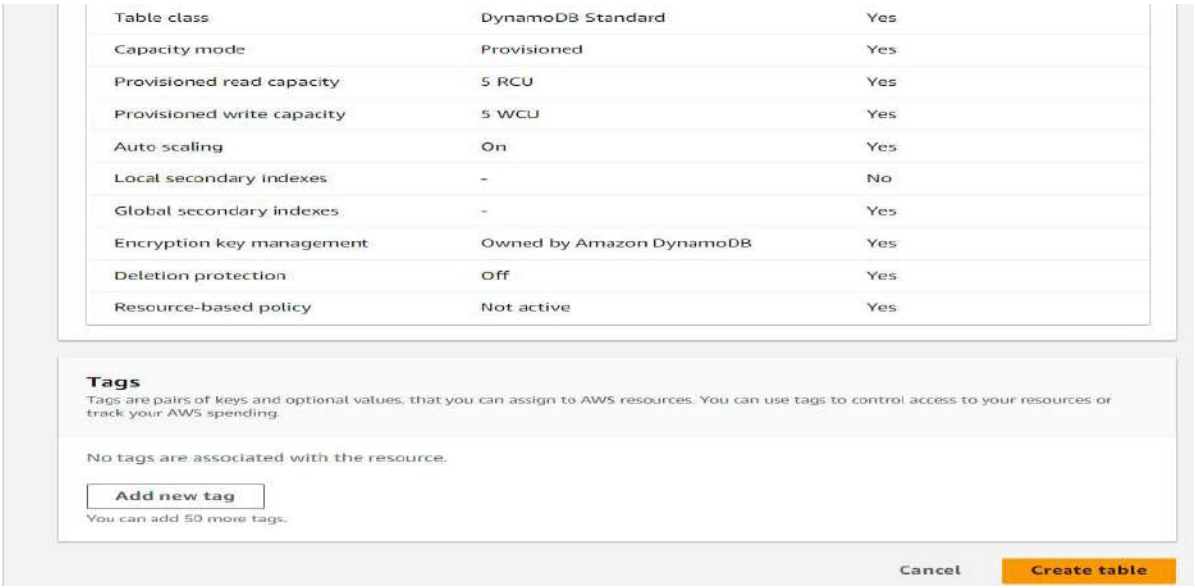
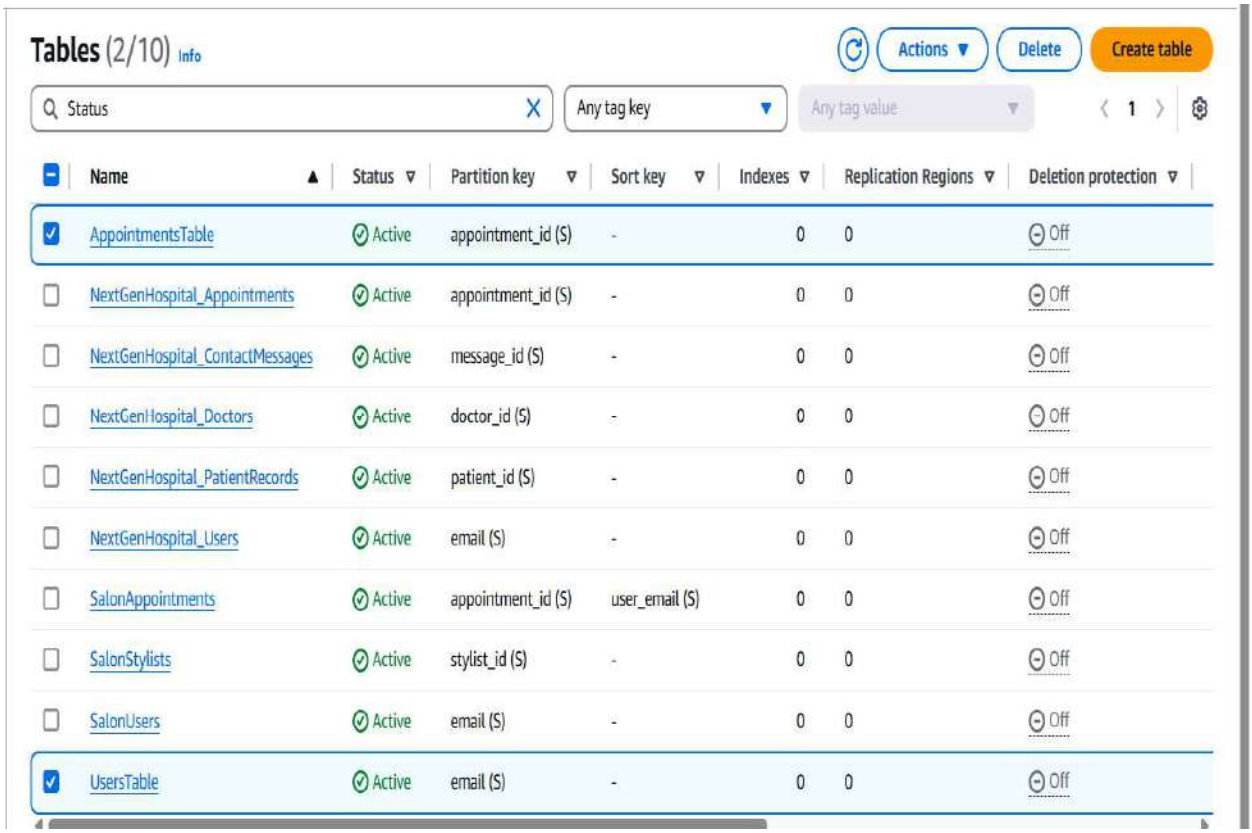


Table class
DynamoDB Standard
Yes
Capacity mode
Provisioned
Yes
Provisioned read capacity
5 RCU
Yes
Provisioned write capacity
5 WCU
Yes
Auto scaling
On
Yes
Local secondary indexes
-
No
Global secondary indexes
-
Yes
Encryption key management
Owned by Amazon DynamoDB
Yes
Deletion protection
Off
Yes
Resource-based 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

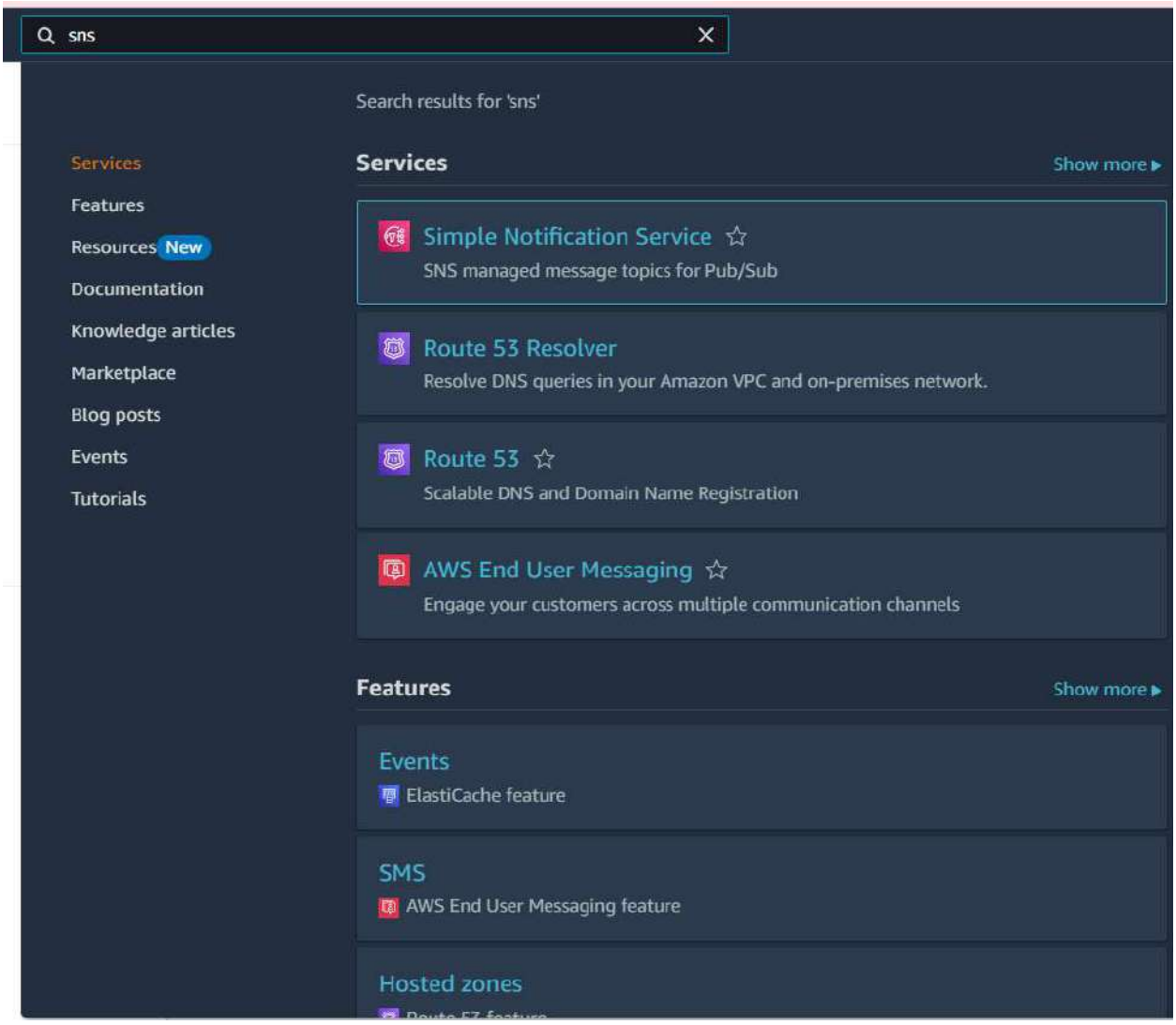


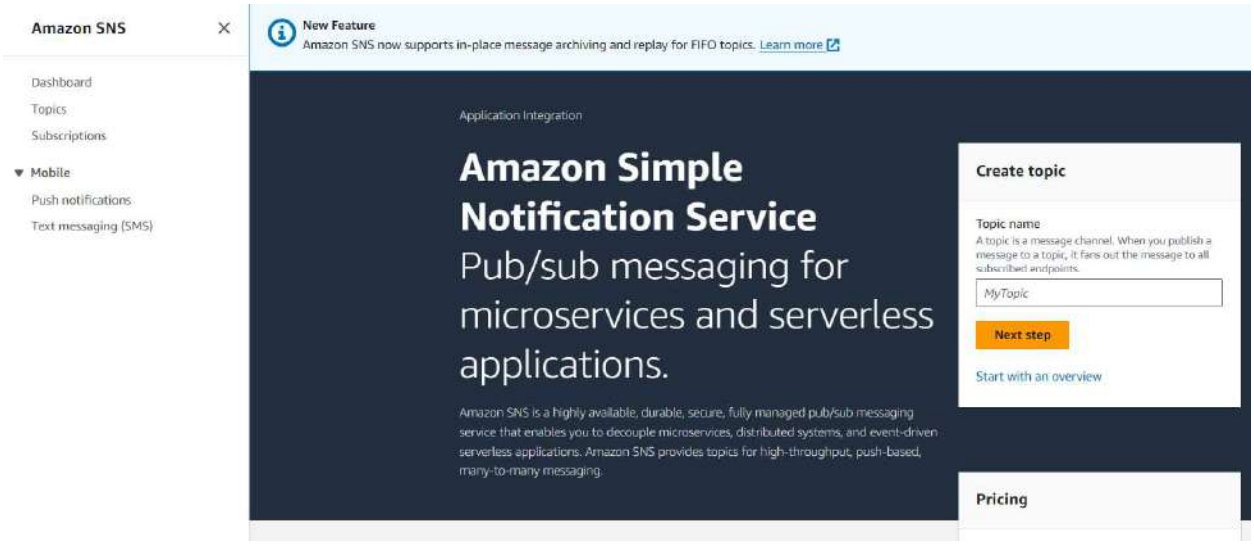
Milestone 4 : SNS Notification Setup

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.

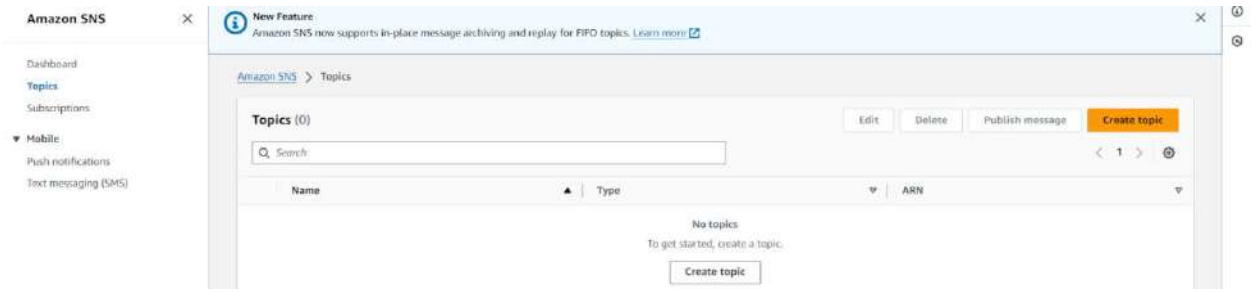
SNS topics for email notifications

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

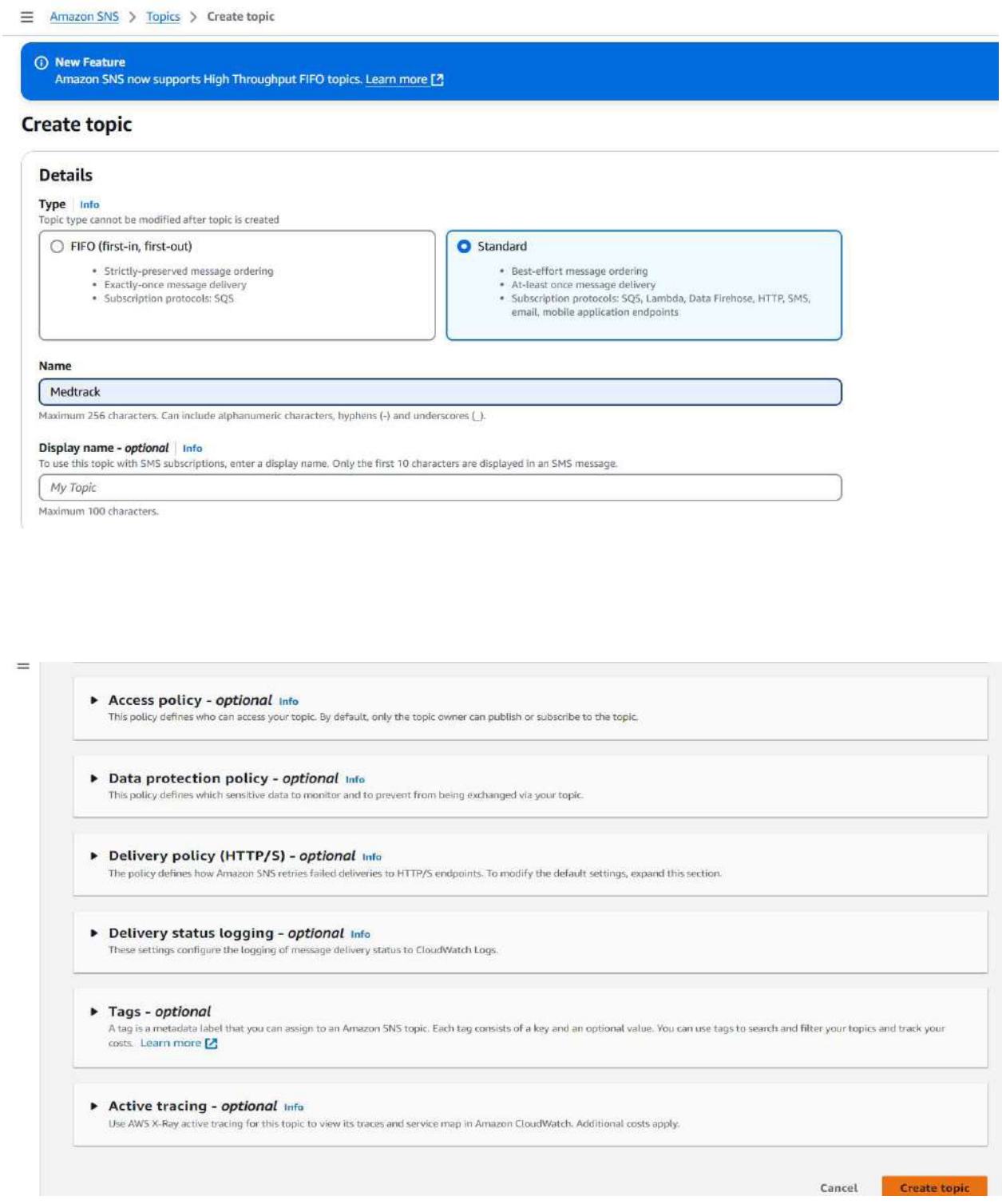

sns
Search results for 'sns'
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Services
Show more
Simple Notification Service
SNS managed message topics for Pub/Sub
Route 53 Resolver
Resolve DNS queries in your Amazon VPC and on-premises network.
Route 53
Scalable DNS and Domain Name Registration
AWS End User Messaging
Engage your customers across multiple communication channels
Features
Show more
Events
ElastiCache feature
SMS
AWS End User Messaging feature
Hosted zones



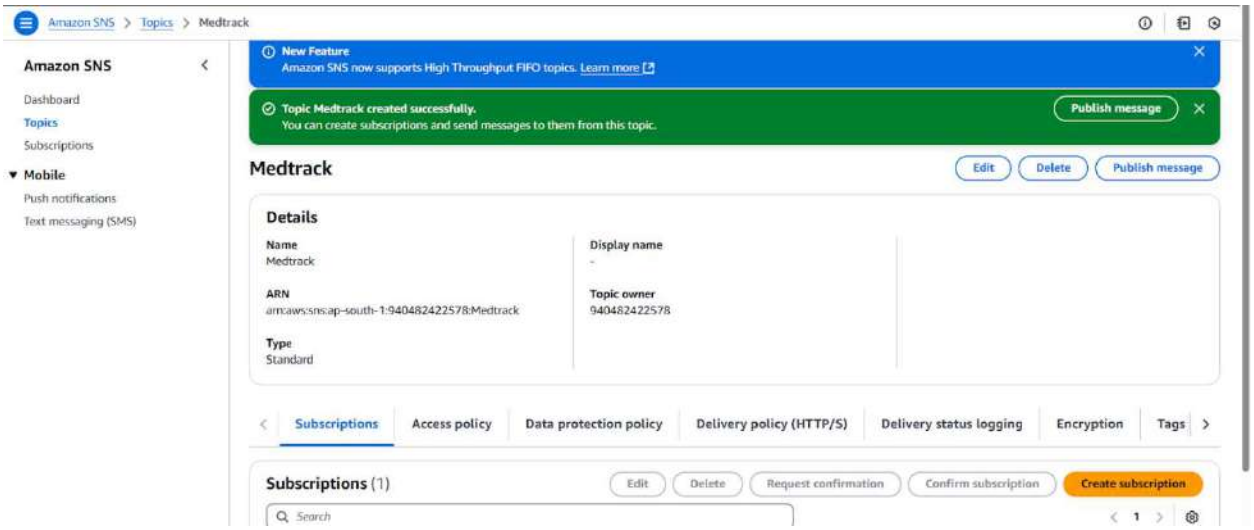
- Click on **Create Topic** and choose a name for the topic.



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

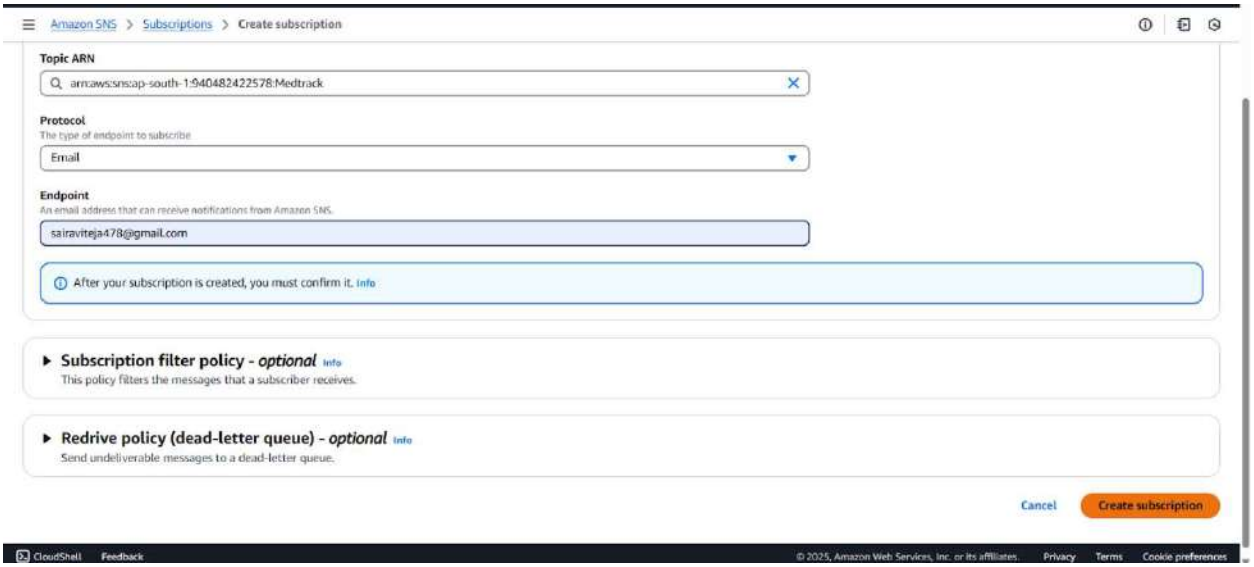

Amazon SNS > Topics > Create topic
New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more
Create topic
Details
Type Info
Topic type cannot be modified after topic is created
FIFO (first-in, first-out)
Strictly-preserved message ordering
Exactly-once message delivery
Subscription protocols: SQS
Standard
Best-effort message ordering
At-least once message delivery
Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints
Name
Medtrack
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).
Display name - optional Info
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.
My Topic
Maximum 100 characters.
Access policy - optional Info
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.
Data protection policy - optional Info
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.
Delivery policy (HTTP/S) - optional Info
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.
Delivery status logging - optional Info
These settings configure the logging of message delivery status to CloudWatch Logs.
Tags - optional
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. Learn more
Active tracing - optional Info
Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.
Cancel
Create topic

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

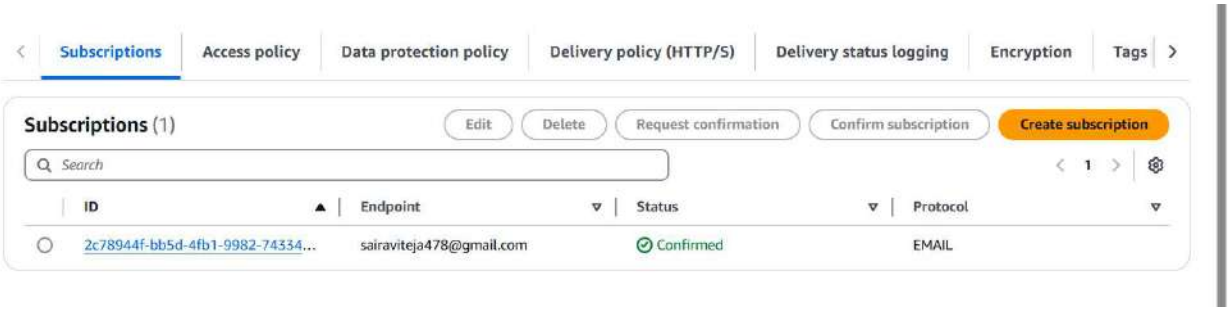


Subscribe users and Admin

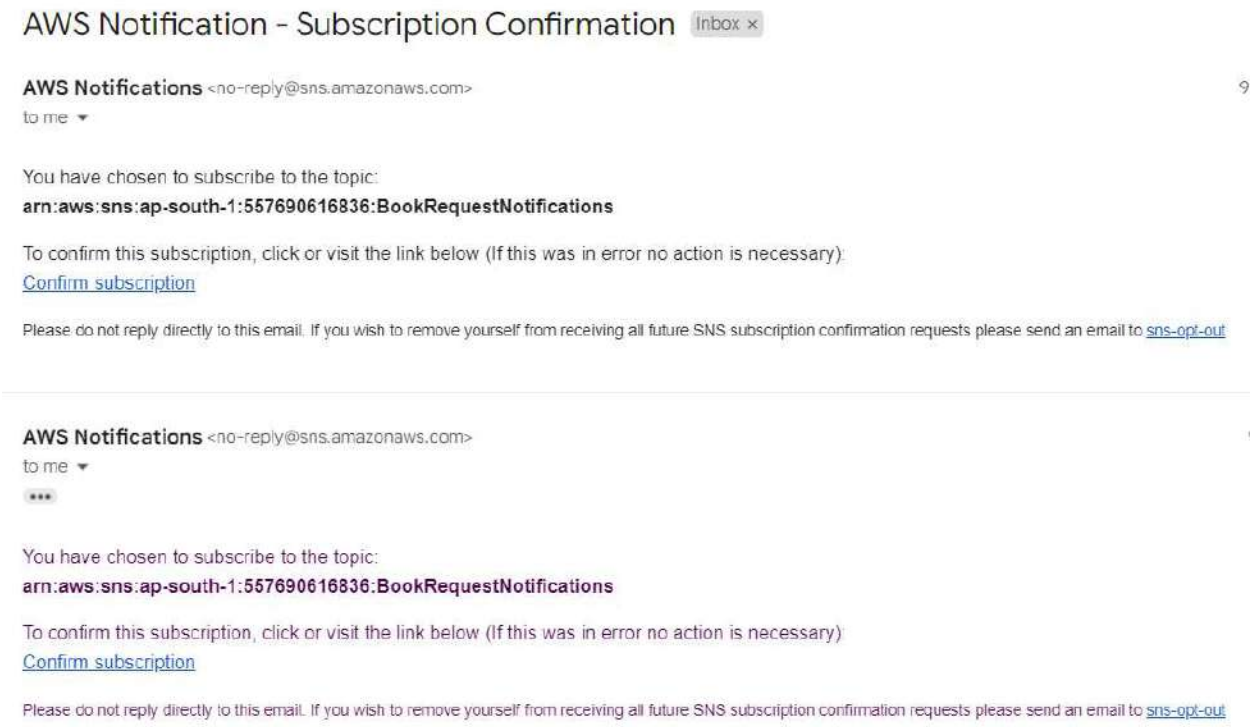
- Subscribe users (or admin staff) to this topic via email. When a book request is made, notifications will be sent to the subscribed emails**.**



- After subscription request for the mail configuration



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



aws
Simple Notification Service

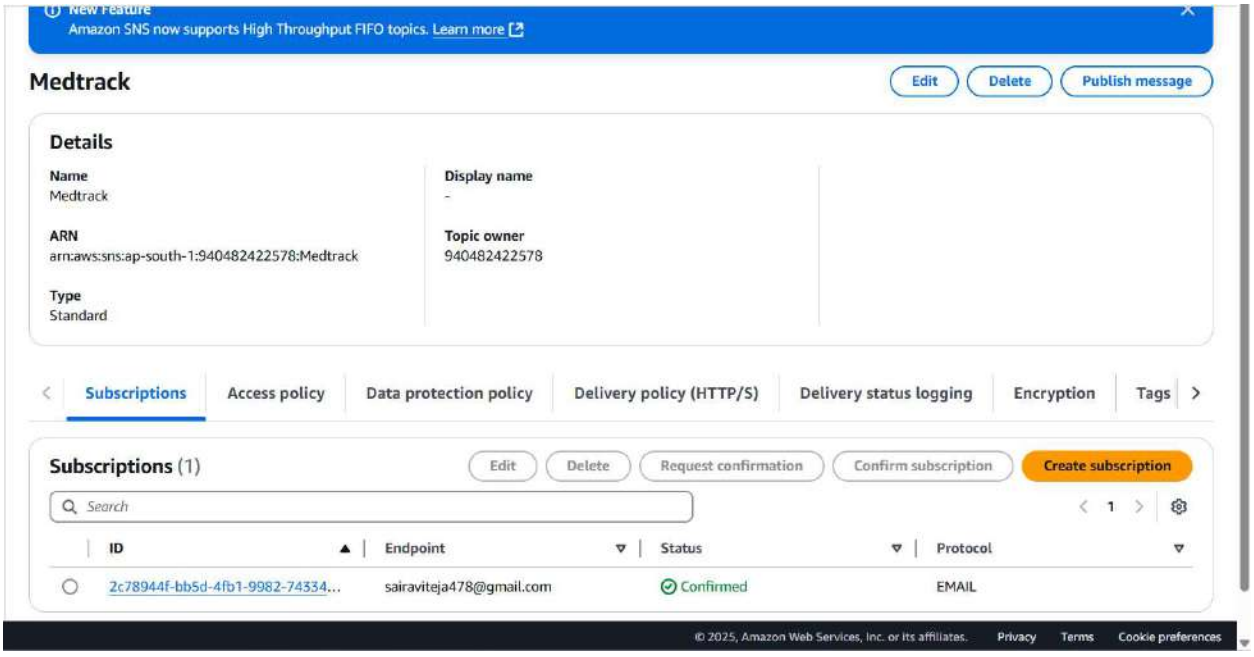
Subscription confirmed!

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.

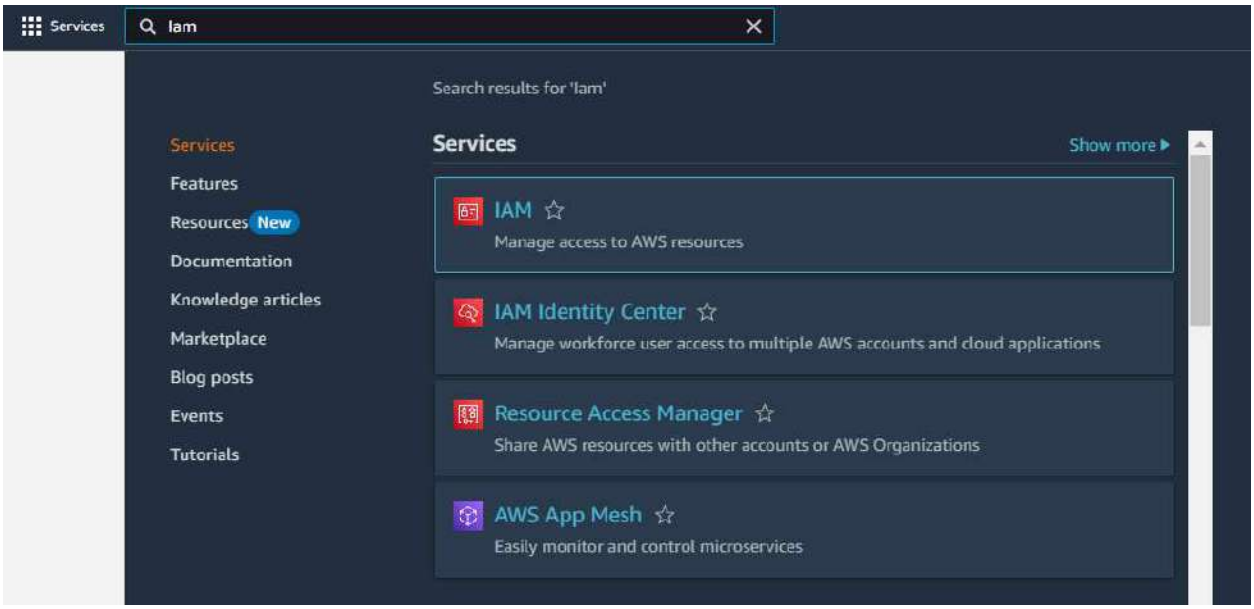


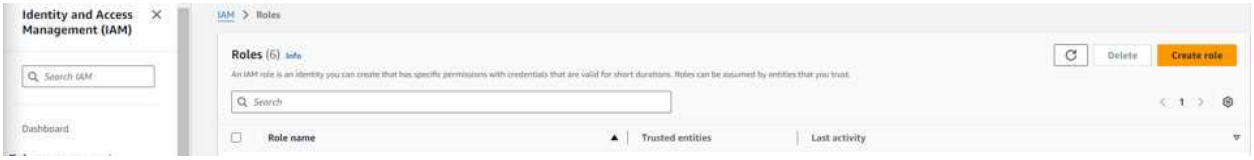
Milestone 5: IAM Role Setup

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.

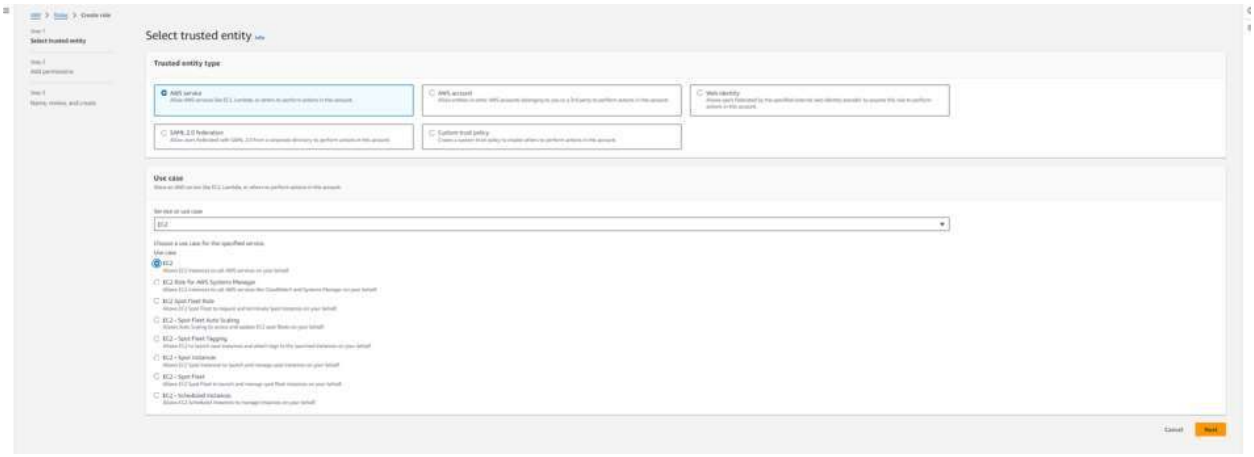
Create IAM Role.

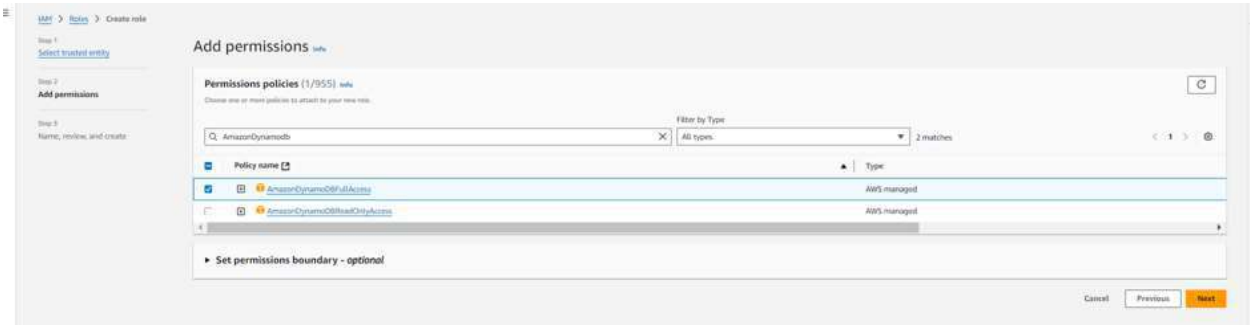
- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.





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.

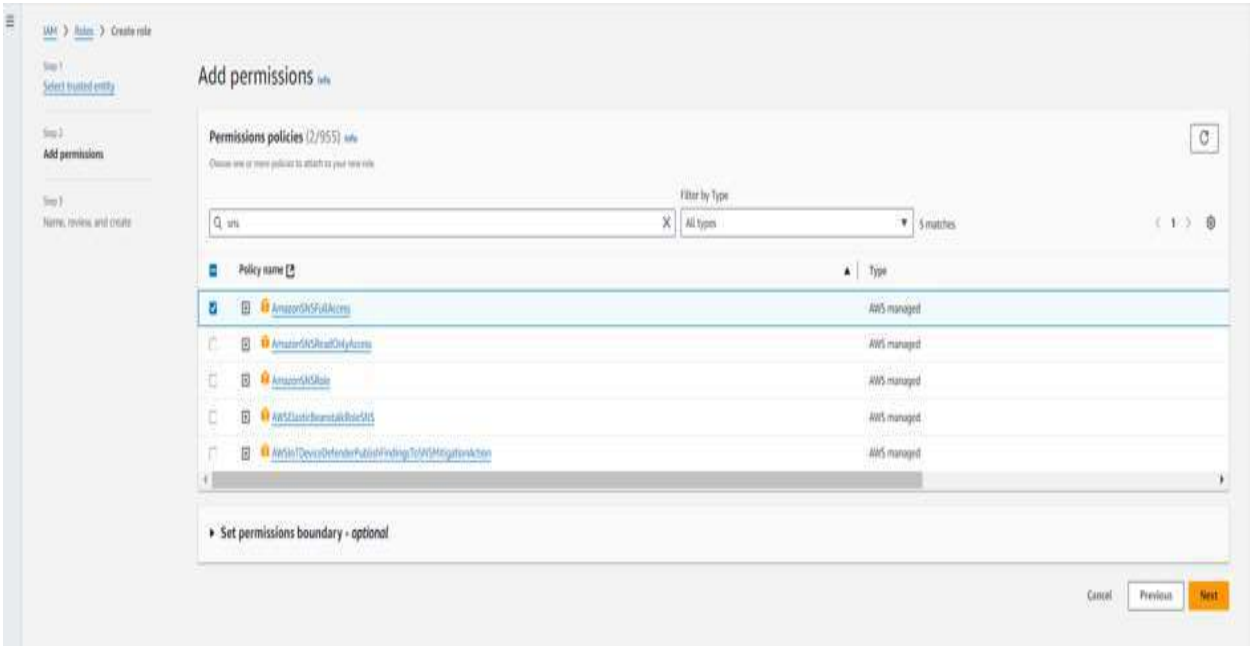




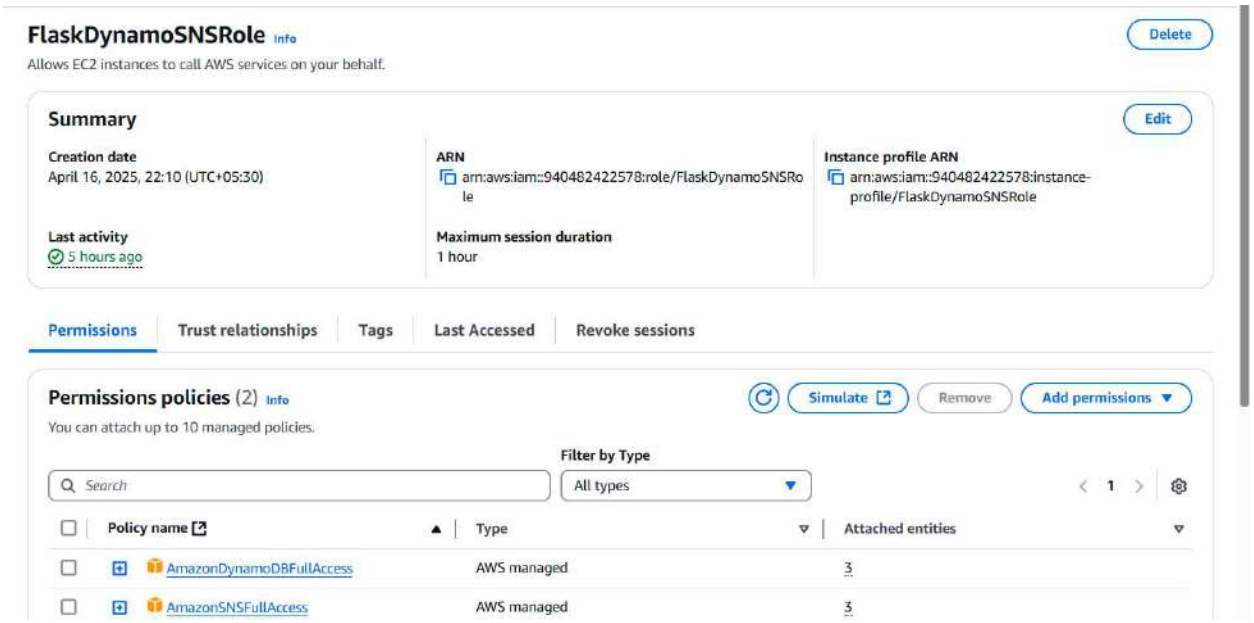
Attach Policies.

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 **flask dynamodb sns,** go to the AWS IAM console, create a new role, and assign DynamoDB Full Access and SNS FullAccess 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.

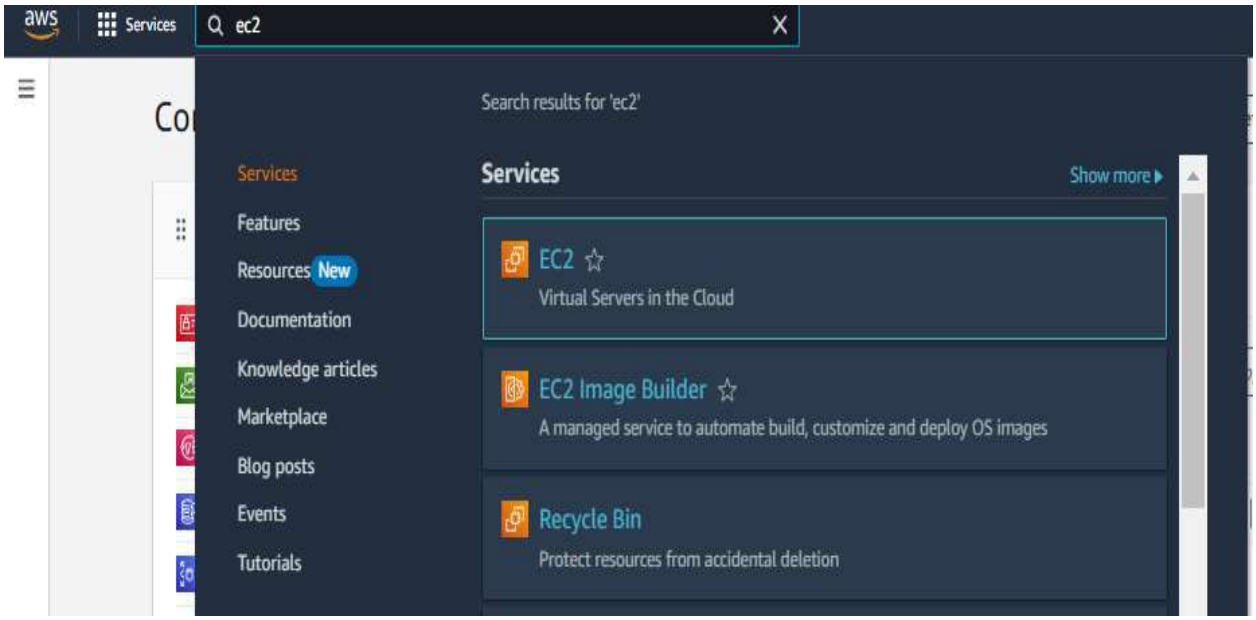


Milestone 6: EC2 Instance setup

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.

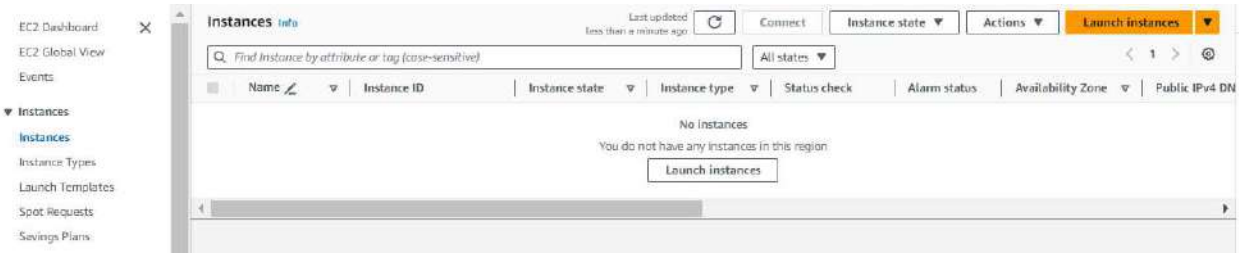
- **Launch EC2 Instance**
    - In the AWS Console, navigate to EC2 and launch a new instance.

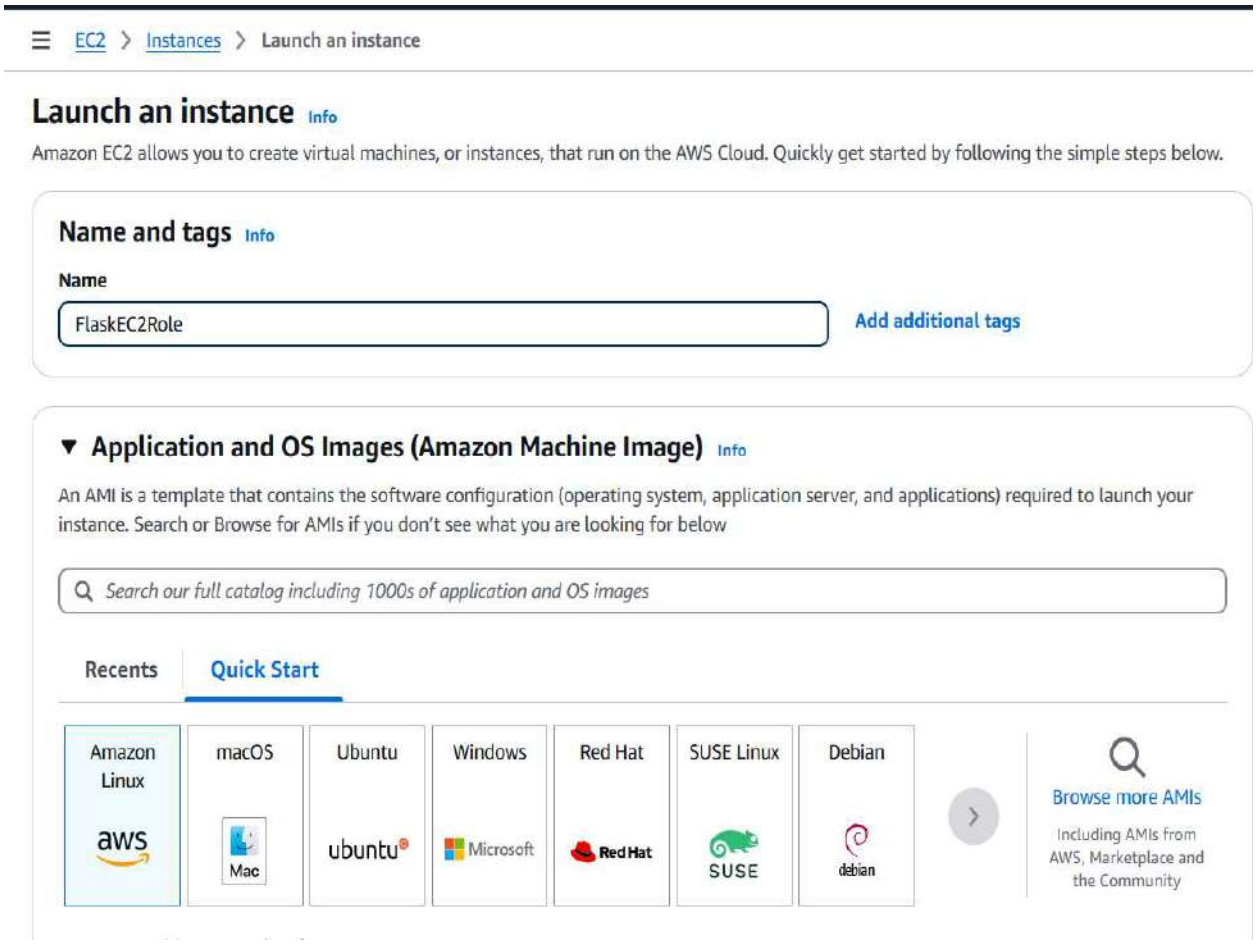


To launch an EC2 instance with the name **flaskec2role**, follow these steps:

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.

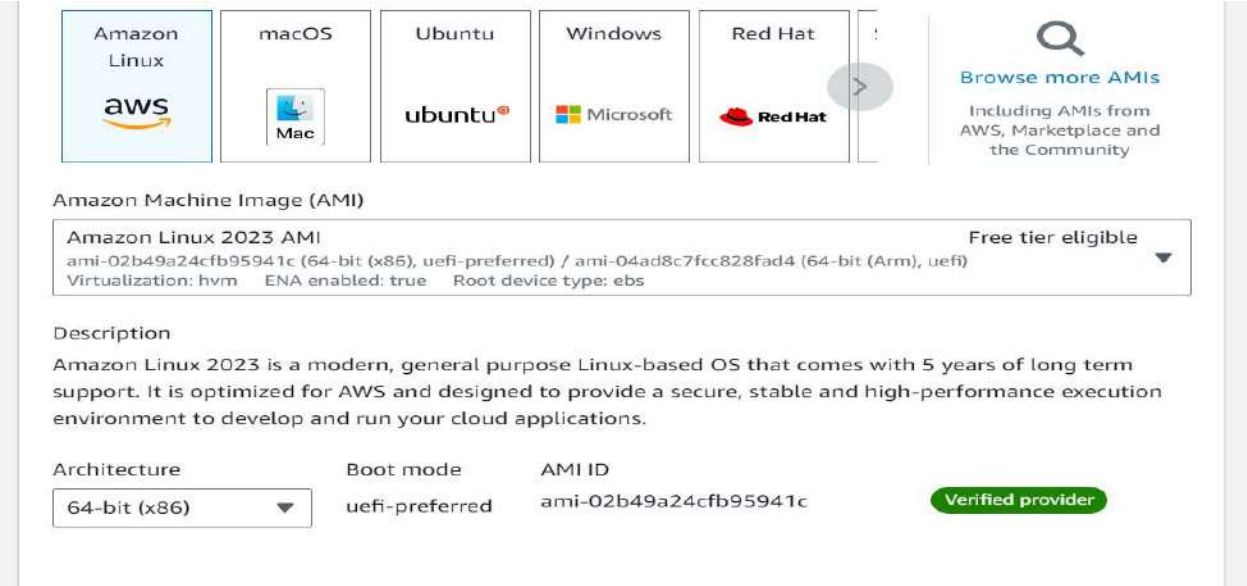




To launch an EC2 instance with **Amazon Linux 2** or **Ubuntu** as the AMI and **t2.micro** as the instance type (free-tier eligible):

1. In the **Launch Instance** wizard, choose **Amazon Linux 2** or **Ubuntu** from the available AMIs.

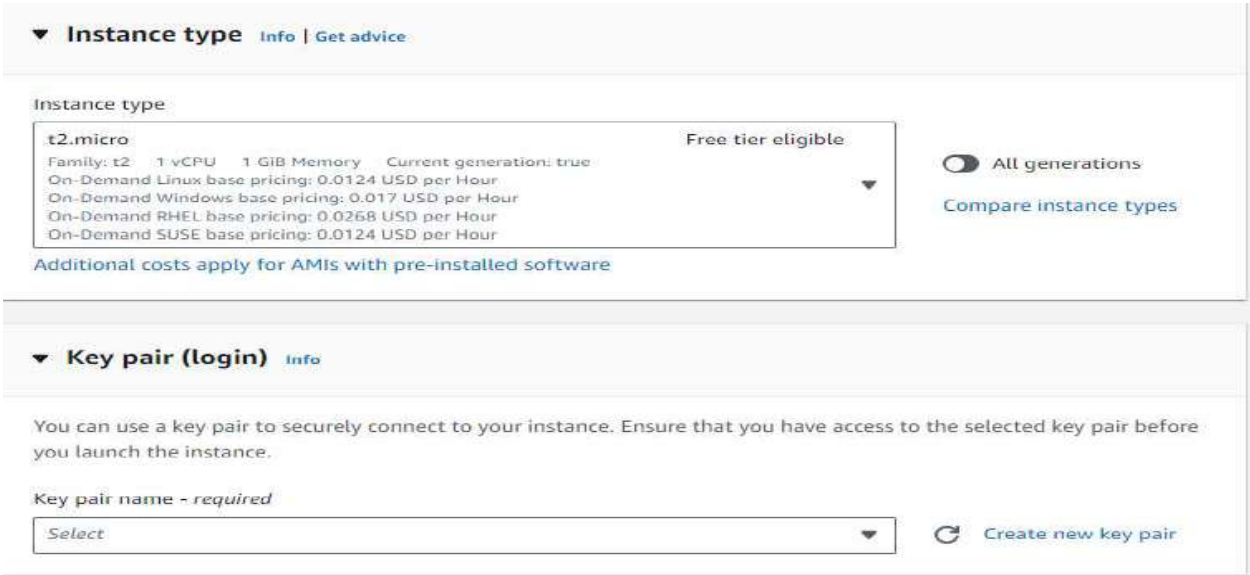
2. Select **t2.micro** as the instance type, which is free-tier eligible, and continue with the configuration and launch steps.

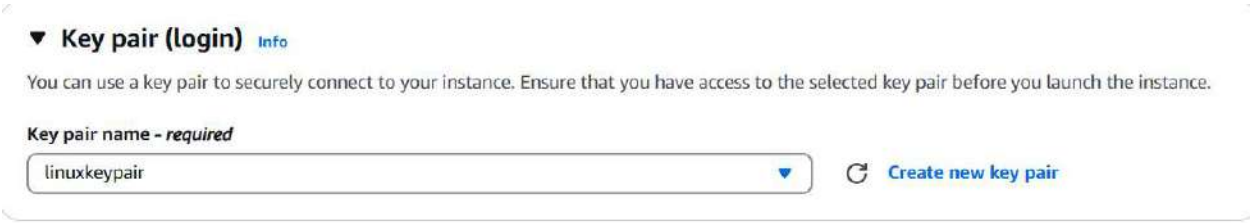


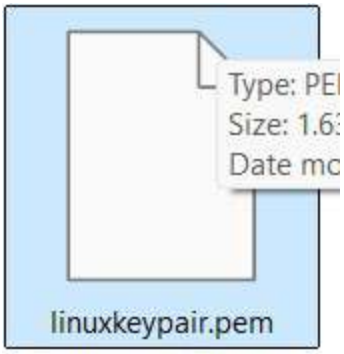
To create and download the key pair for server access:

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.







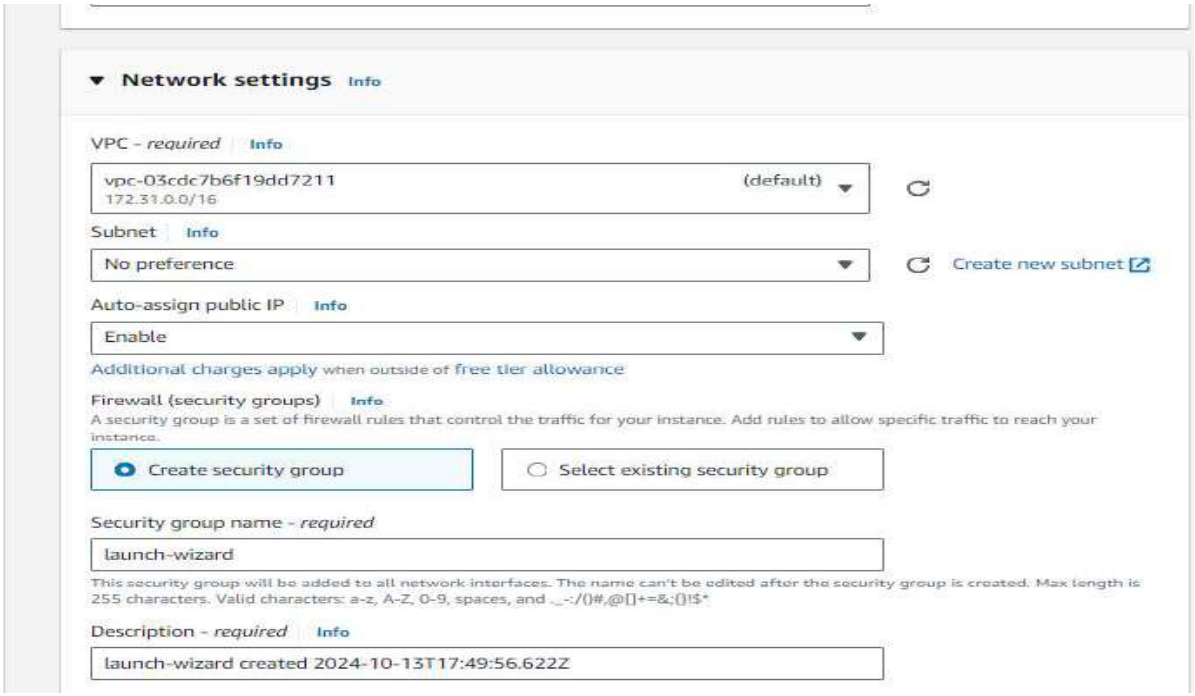
Configure security groups for HTTP, and SSH access.

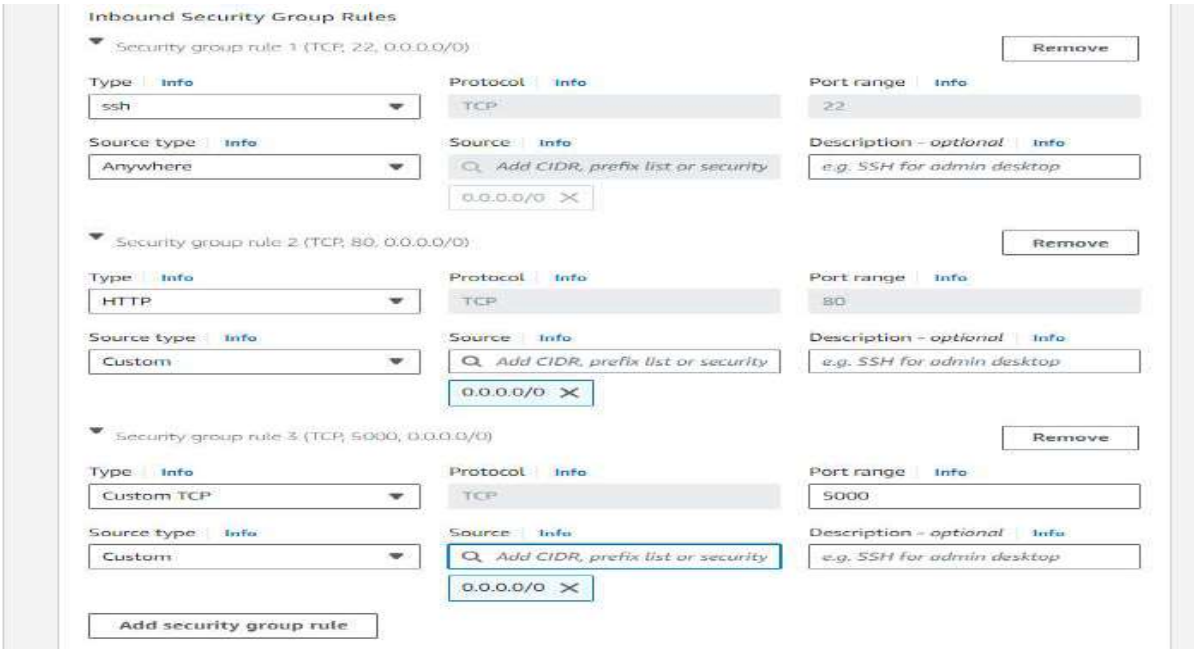
For network settings during EC2 instance launch:

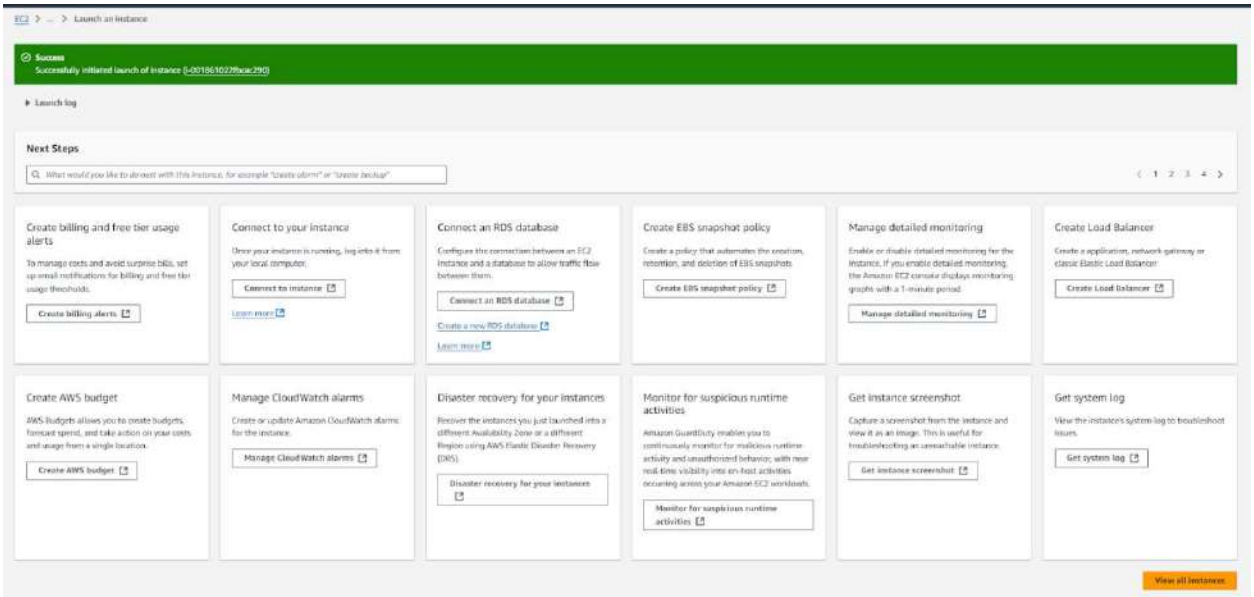
1. In the **Network Settings** section, select the **VPC** and **Subnet** you wish to use (if unsure, the default VPC and subnet should work).

2. Ensure **Auto-assign Public IP** is enabled so your instance can be accessed from the internet.

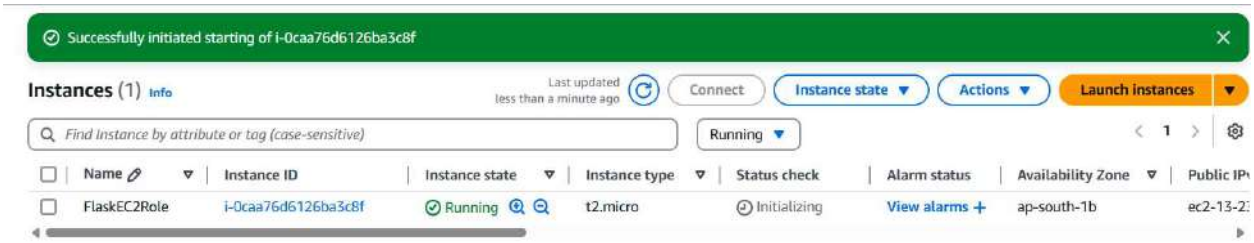
3. In **Security Group**, either select an existing one or create a new one that allows SSH (port 22) access to your EC2 instance for remote login.


Network settings Info
VPC - required Info
vpc-03cdc7b6f19dd7211 (default)
172.31.0.0/16
Subnet Info
No preference
Create new subnet
Auto-assign public IP Info
Enable
Additional charges apply when outside of free tier allowance
Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group
Select existing security group
Security group name - required
launch-wizard
This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and ._-:/()#,@[]+=&;{}!$*
Description - required Info
launch-wizard created 2024-10-13T17:49:56.622Z

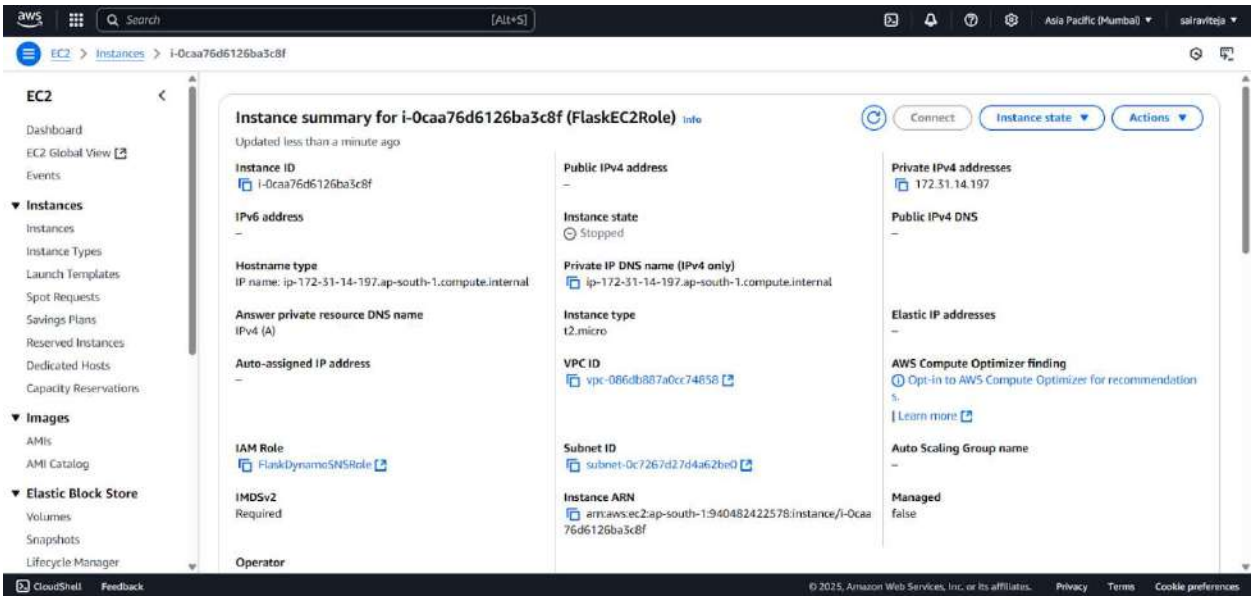

Inbound Security Group Rules
Security group rule 1 (TCP, 22, 0.0.0.0/0)
Remove
Type Info
ssh
Protocol Info
TCP
Port range Info
22
Source type Info
Anywhere
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Security group rule 2 (TCP, 80, 0.0.0.0/0)
Remove
Type Info
HTTP
Protocol Info
TCP
Port range Info
80
Source type Info
Custom
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Security group rule 3 (TCP, 5000, 0.0.0.0/0)
Remove
Type Info
Custom TCP
Protocol Info
TCP
Port range Info
5000
Source type Info
Custom
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Add security group rule



- 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.

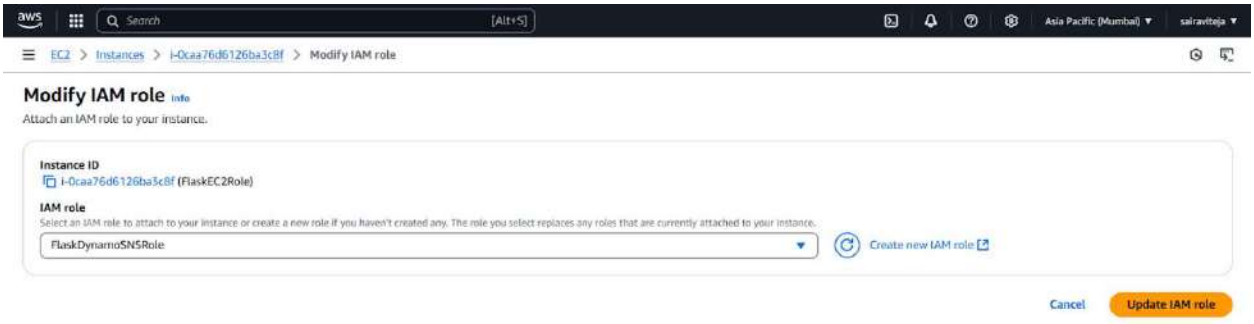


To modify the **IAM role** for your EC2 instance:

1. Go to the **AWS IAM Console**, select **Roles**, and find the **flaskec2role**.

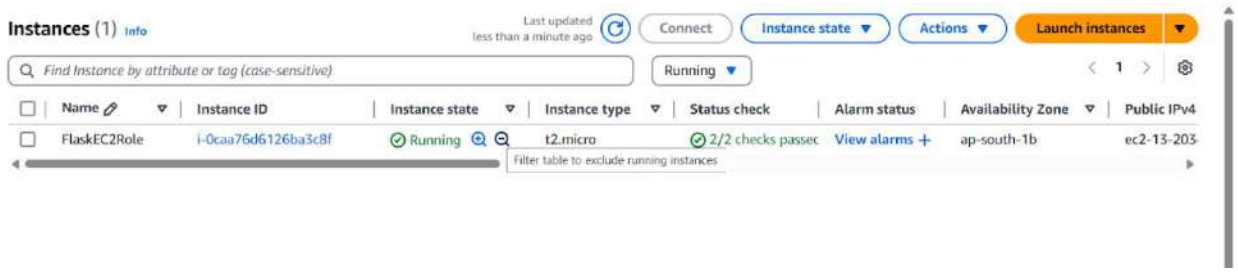
2. Click **Attach Policies**, then choose the required policies (e.g., **DynamoDBFullAccess**, **SNSFullAccess**) and click **Attach Policy**.

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.

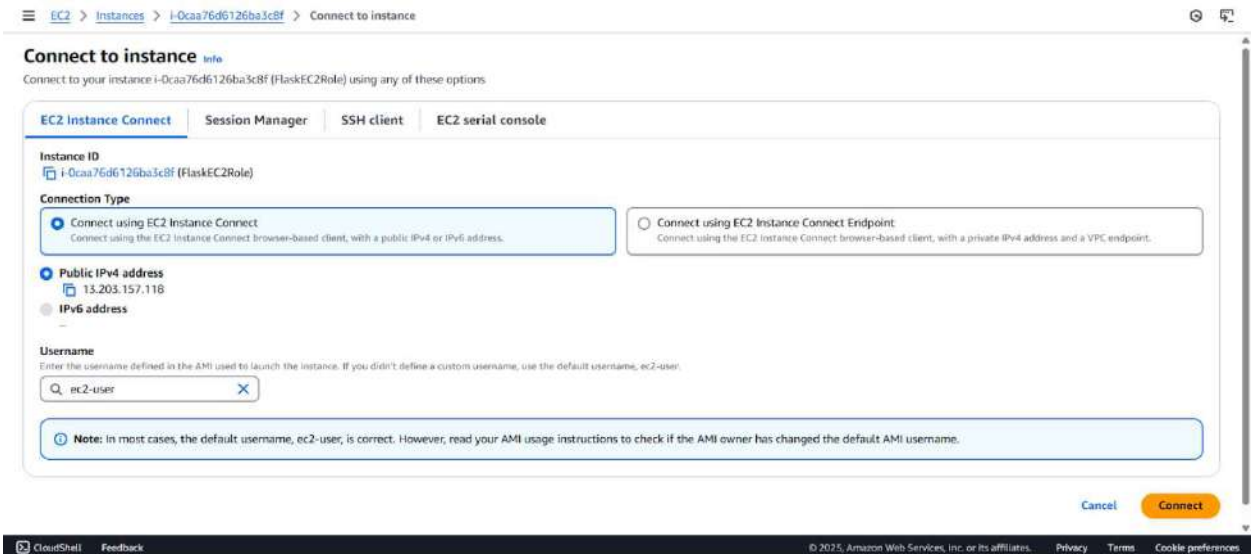


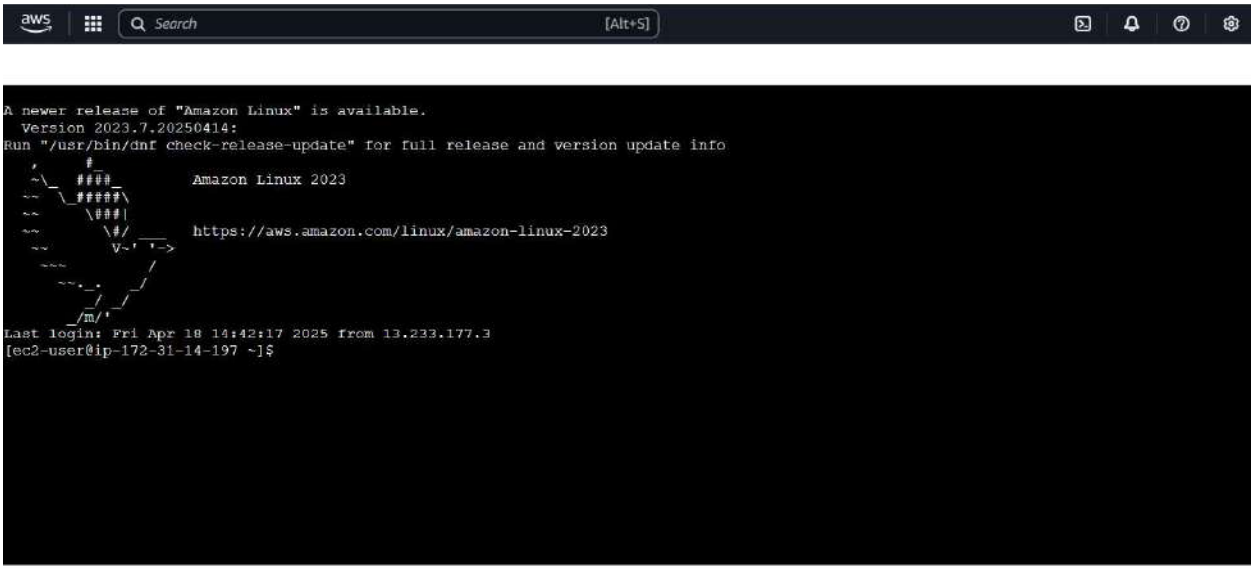
To connect to your EC2 instance:

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 -iflaskkeypair.pem ec2-user@<public-ip>) to access your instance using the downloaded .pem key.



- Now connect the EC2 with the files





i-0caa76d6126ba3c8f (FlaskEC2Role)

PublicIPs: 13.203.157.118 PrivateIPs: 172.31.14.197

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/divyasree-2003/medtrack.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

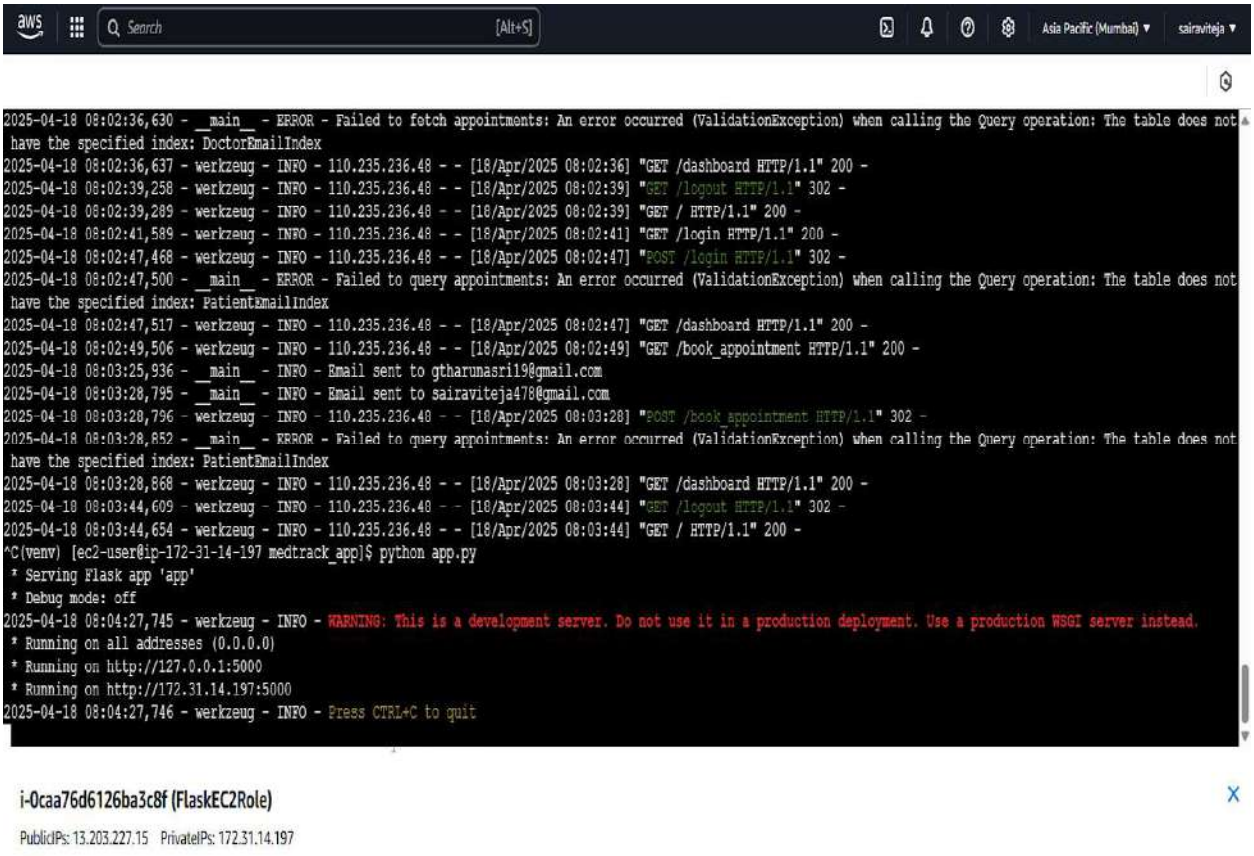
- This will download your project to the EC2 instance.

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

```
[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:

Public IPs: http://127.0.0.1:5000

Milestone 8: Testing and Deployment

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

Functional testing to verify the project

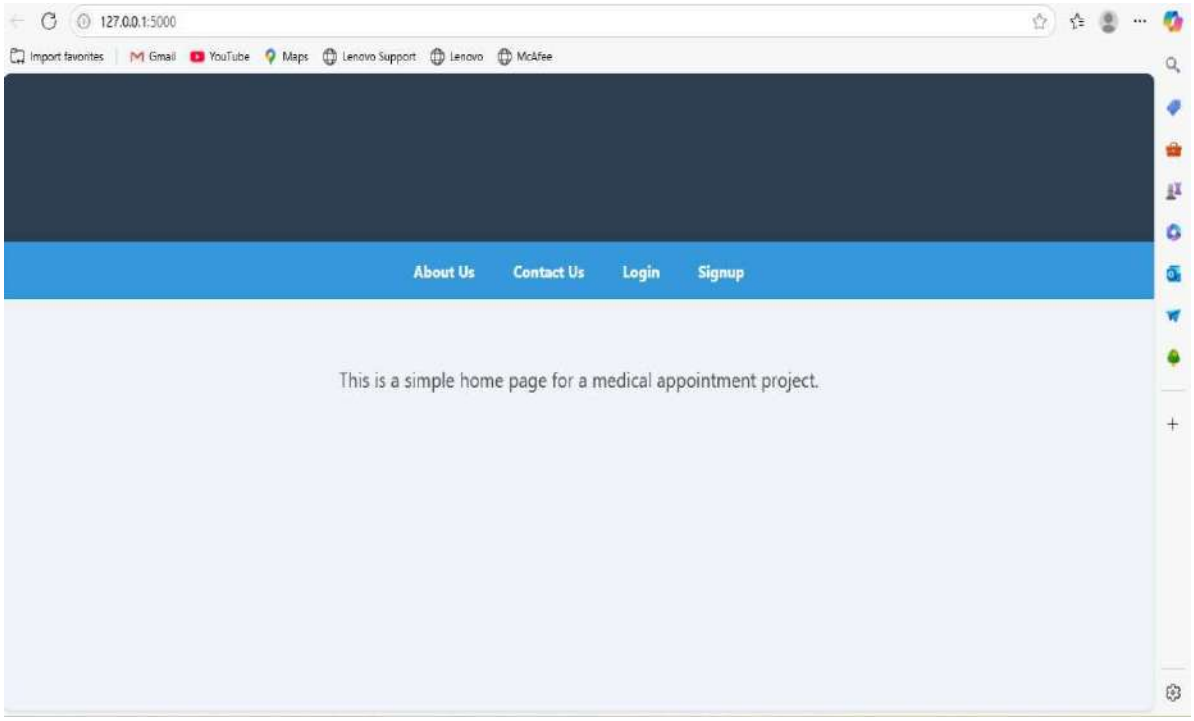
Home Page:

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

- Input Fields: For users to enter basic information like appointment requests, diagnosis submissions, or service bookings.

- Navigation: Links to other sections such as the login page, dashboard, or service options. 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.



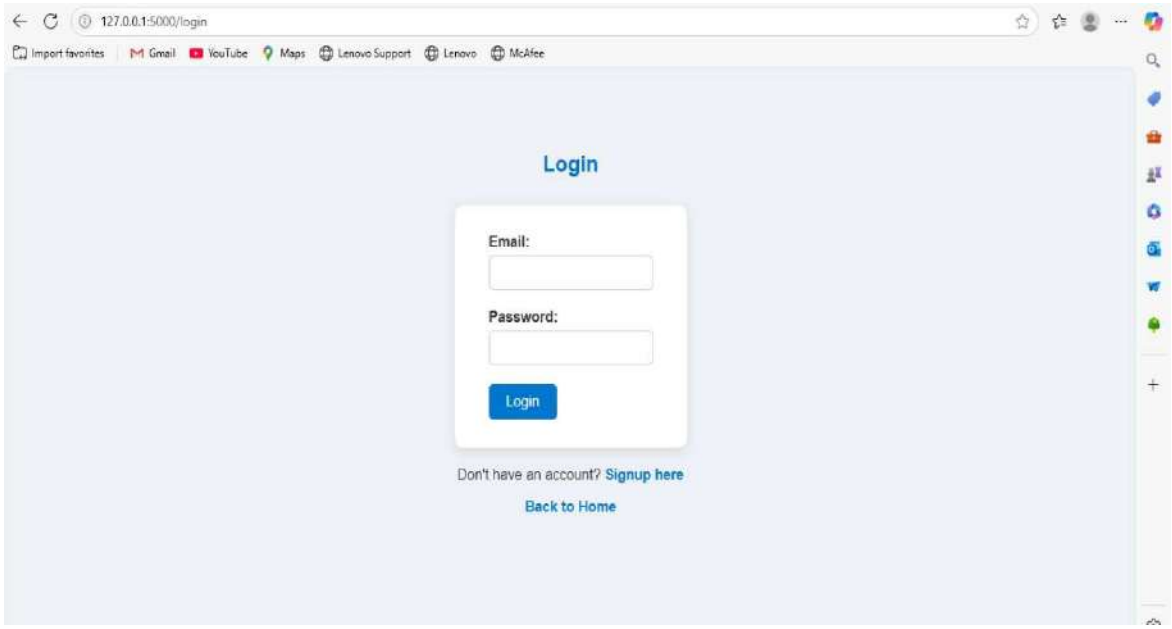
PATIENT LOGIN PAGE:

The Patient Login Page 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 redirected to their respective dashboards to manage appointments, medical records, and other relevant tasks.



User Dashboard:

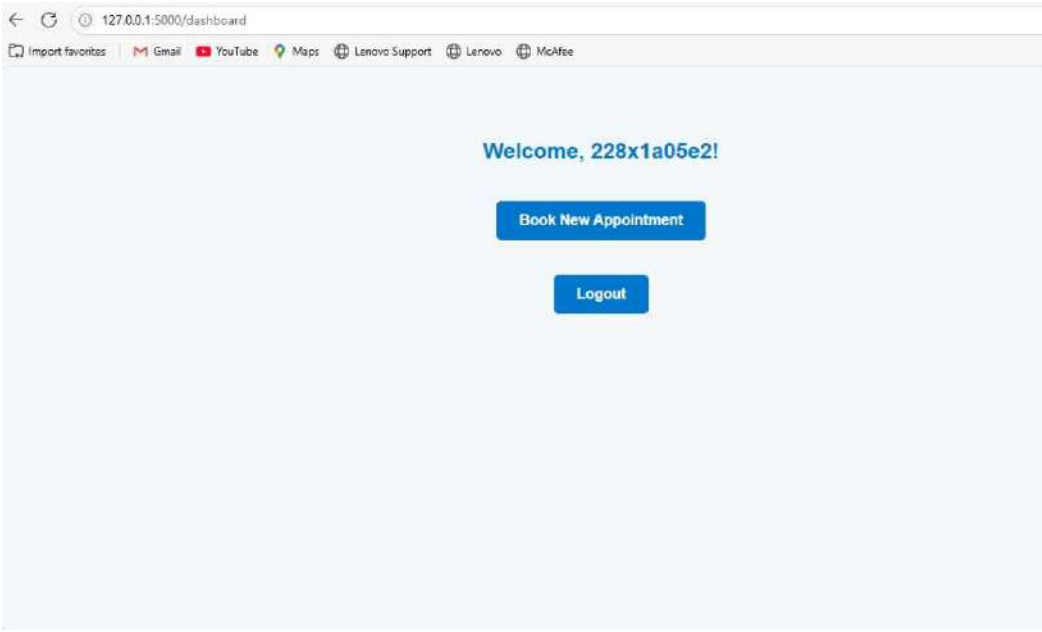
The User Dashboard (for patients) provides an easy interface to manage appointments and track their status. It typically includes:

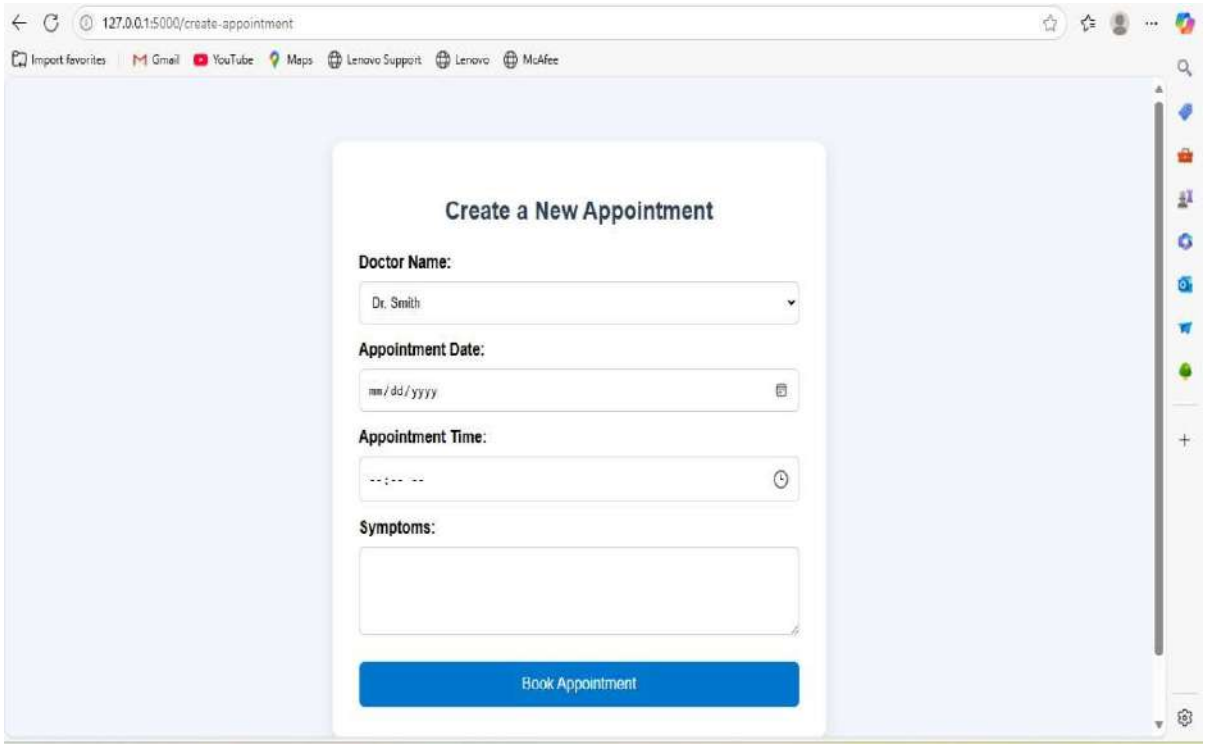
- Book Appointment Section: A form for selecting a doctor, choosing an appointment time, and submitting the request.

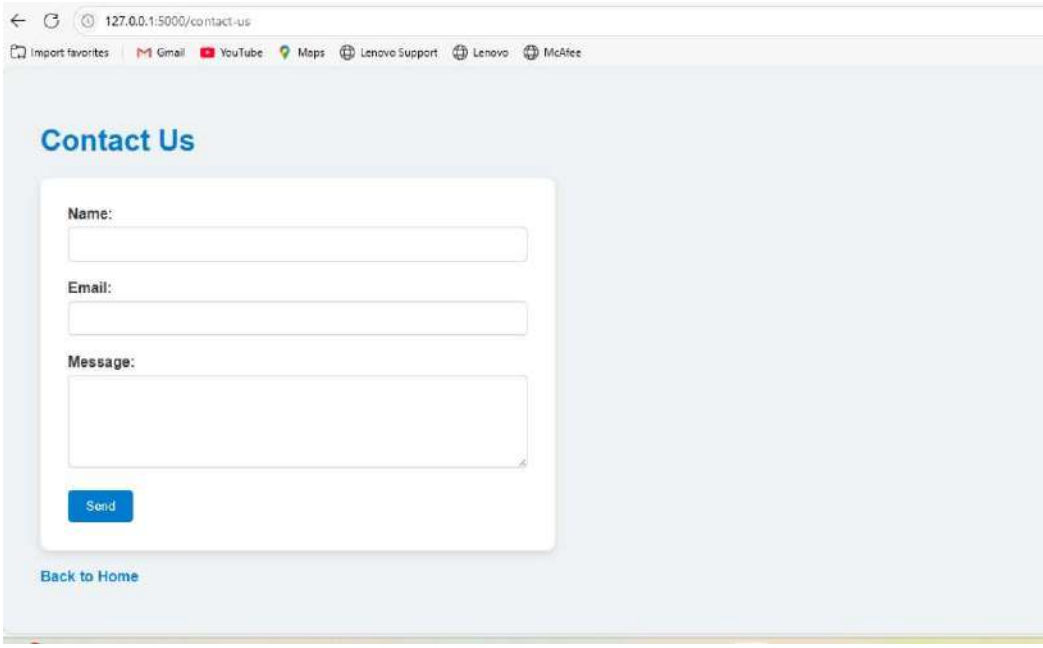
- Appointment Status: A section showing the current status of appointments (e.g., confirmed, pending, or completed) with options to view details or cancel.

- 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.


127.0.0.1:5000/dashboard
Import favorites
Gmail
YouTube
Maps
Lenovo Support
Lenovo
McAfee
Welcome, 228x1a05e2!
Book New Appointment
Logout


127.0.0.1:5000/create-appointment
Import favorites
Gmail
YouTube
Maps
Lenovo Support
Lenovo
McAfee
Create a New Appointment
Doctor Name:
Dr. Smith
Appointment Date:
mm/dd/yyyy
Appointment Time:
--:-- --
Symptoms:
Book Appointment



DynamoDB Database updations :

1. **Users table :**

In the Users Table of DynamoDB, the data structure is designed to store user-related information for both patients and doctors. Typical updates include:

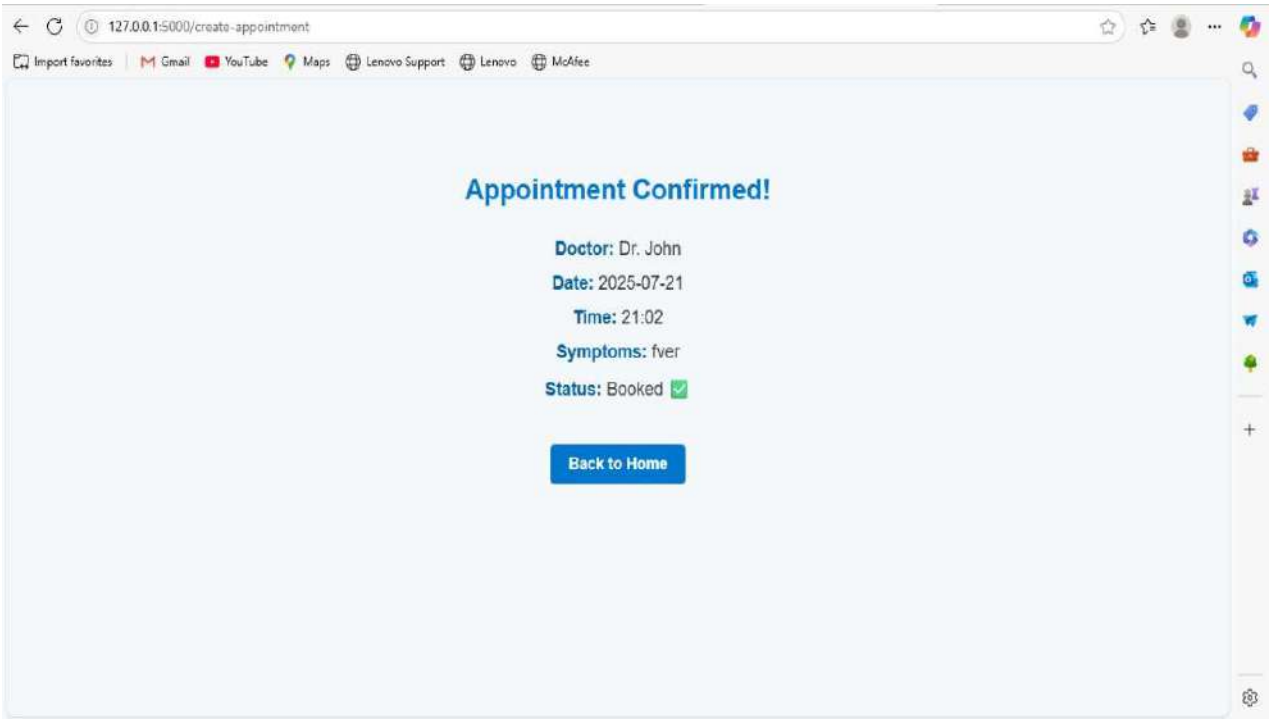
1. Add New Users: When a new patient or doctor registers, their details such as name, email, role (patient/doctor), contact info, and password hash are added to the table.

2. Update User Info: If a user updates their profile (e.g., changing contact details), the corresponding record in the table is modified.

3. Status Tracking: Track the status of user accounts (active, inactive) based on their activity or admin updates.

The Users Table serves as the central repository for all user data, enabling quick access and modification of details when necessary

Appointment confirmation:



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.