

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.

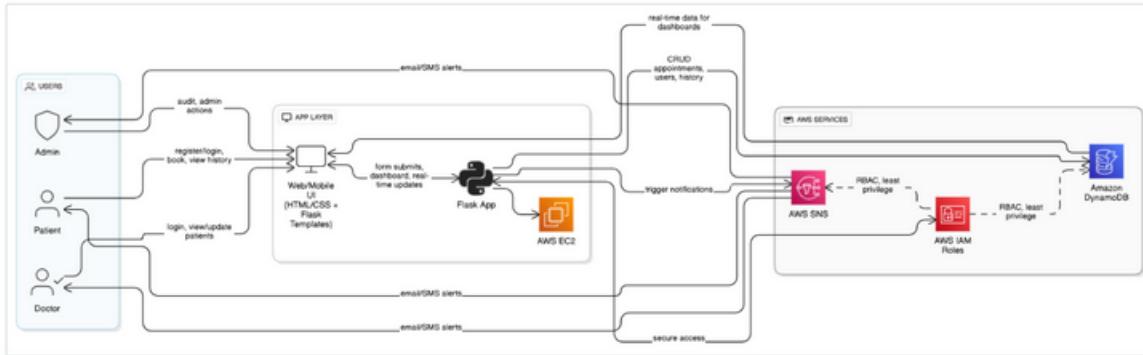
Scenario 2: Secure User Management with IAM

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.

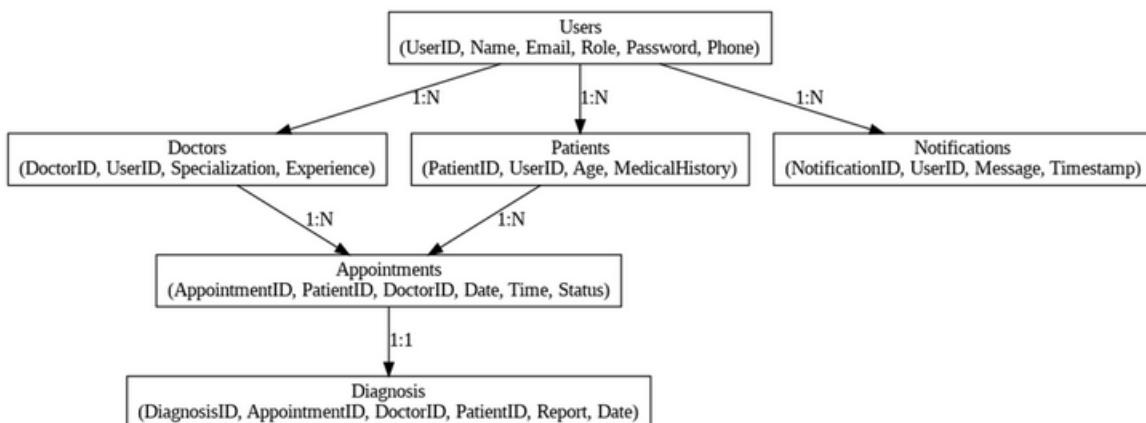
Scenario 3: Easy Access to Medical History and Resources

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-tJ6IgWtRbawWoA4W3PkxEFrSJZhKzULgLsjxo/edit?usp=sharing>

Important Instructions:

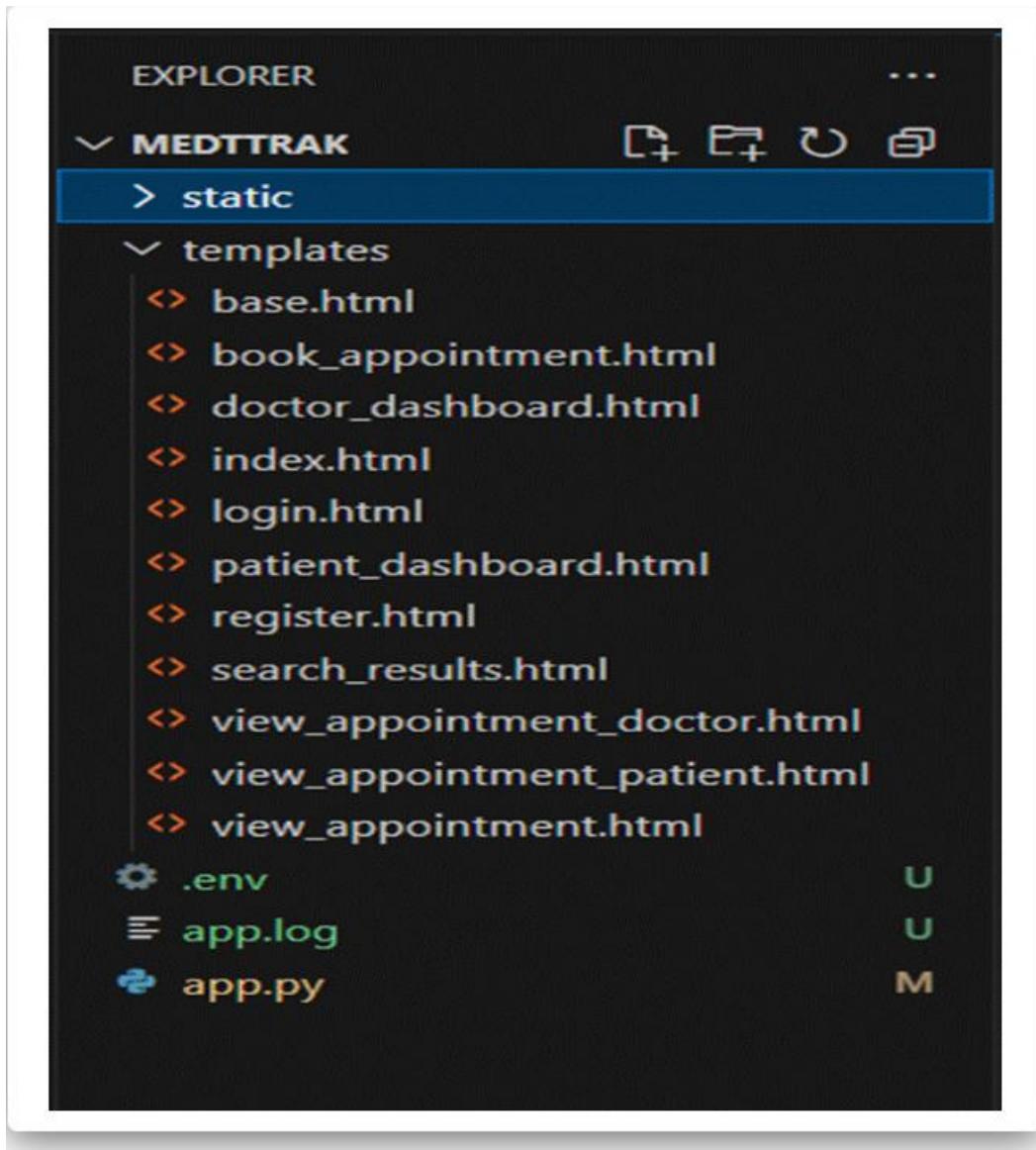
- Start by creating the necessary HTML pages and Flask routes (app.py) to build the core functionality of your application.
- During the initial development phase, store and retrieve data using Python dictionaries or lists locally. This will allow you to design, test, and validate your application logic 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

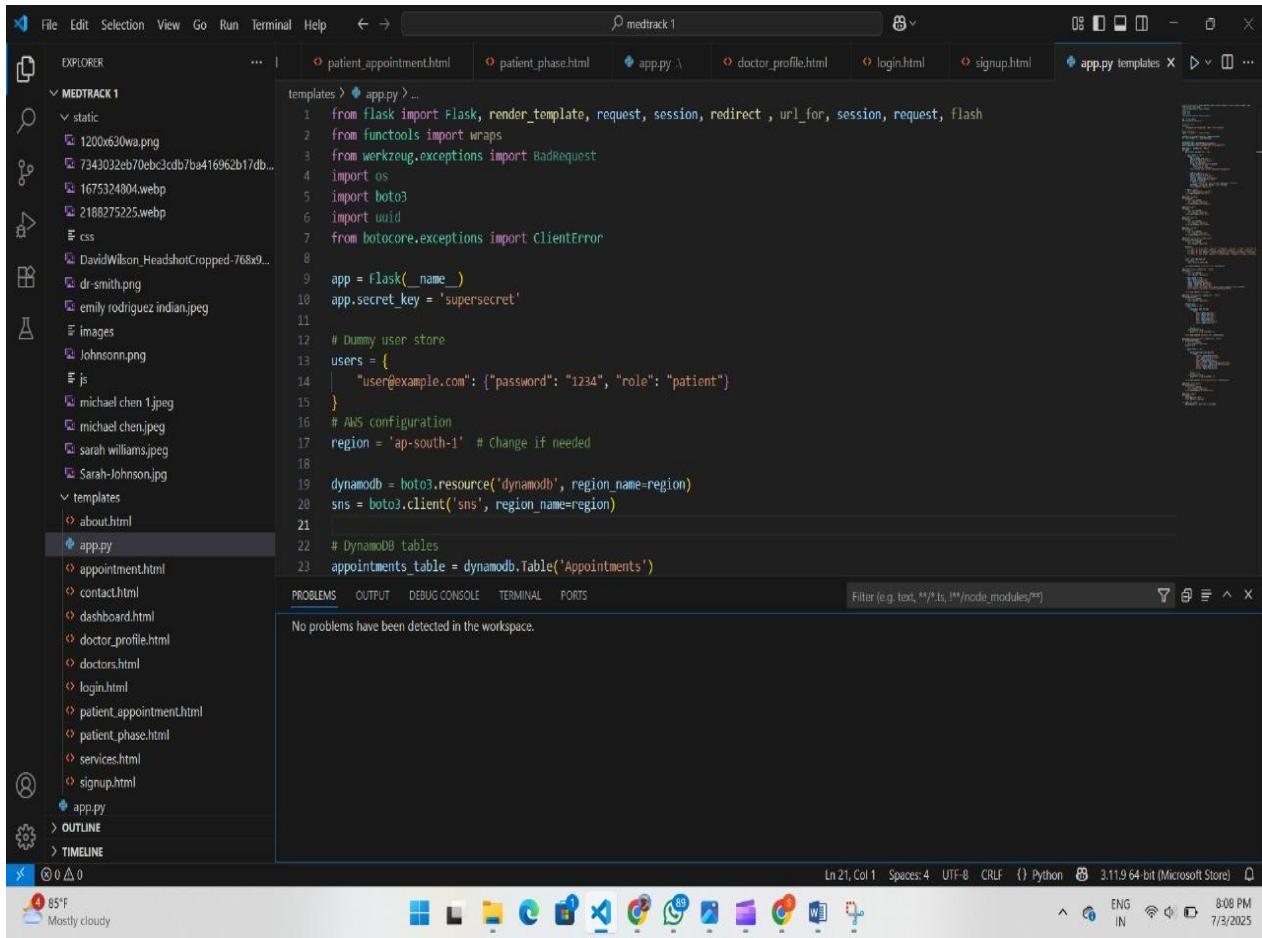


Description of the code :

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.



The screenshot shows the VS Code interface with the following details:

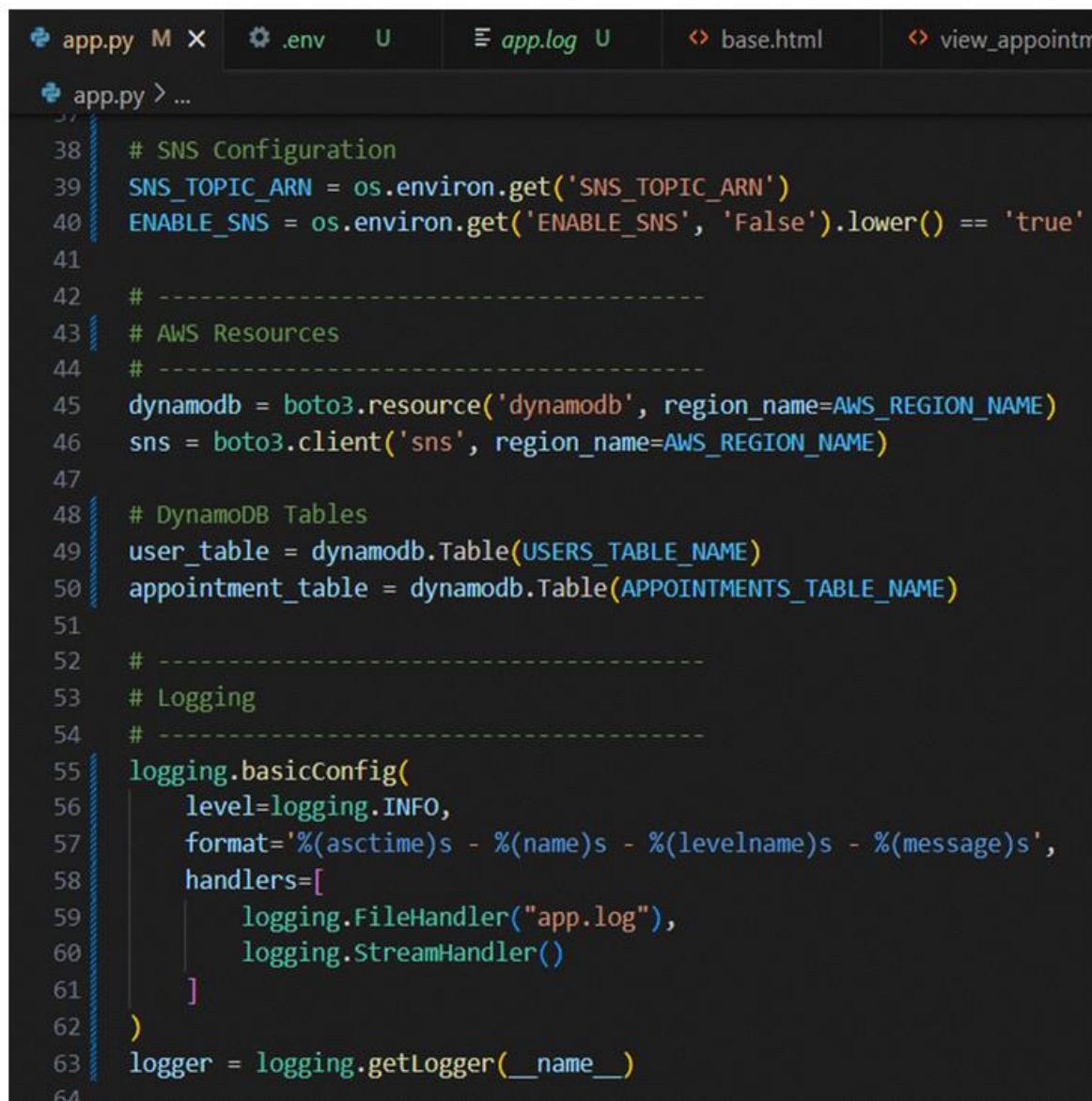
- File Explorer:** Shows the project structure under "MEDTRACK 1". It includes static files (images like 1200x630wa.png, 734032eb70ebc3db7ba416962b17db..., 1675324004.webp, 2188275225.webp), CSS files (css folder with DavidWilson_HeadshotCropped-768x9..., dr-smith.png, emily rodriguez indian.jpeg), JavaScript files (js folder with michael chen 1.jpeg, michael.chen.jpeg, sarah.williams.jpeg, Sarah-Johnson.jpg), and HTML files (templates folder with about.html, appointment.html, contact.html, dashboard.html, doctor.profile.html, doctors.html, login.html, patient.appointment.html, patient.phase.html, services.html, signup.html).
- Code Editor:** Displays the content of `app.py`:


```
templates > app.py > ...
1  from flask import Flask, render_template, request, session, redirect , url_for, session, request, flash
2  from functools import wraps
3  from werkzeug.exceptions import BadRequest
4  import os
5  import boto3
6  import uuid
7  from botocore.exceptions import ClientError
8
9  app = Flask(__name__)
10 app.secret_key = 'supersecret'
11
12 # Dummy user store
13 users = {
14     "user@example.com": {"password": "1234", "role": "patient"}
15 }
16 # AWS configuration
17 region = 'ap-south-1' # Change if needed
18
19 dynamodb = boto3.resource('dynamodb', region_name=region)
20 sns = boto3.client('sns', region_name=region)
21
22 # DynamoDB tables
23 appointments_table = dynamodb.Table('Appointments')
```
- Terminal:** Shows the command `Ln 21, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.11.9 64-bit (Microsoft Store)`.
- Bottom Status Bar:** Shows system information including weather (85°F Mostly cloudy), battery level, network status (ENG IN), and date/time (7/3/2025 8:08 PM).

- 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.



```
app.py M X .env U app.log U base.html view_appointm
app.py > ...
38 # SNS Configuration
39 SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')
40 ENABLE_SNS = os.environ.get('ENABLE_SNS', 'False').lower() == 'true'
41
42 # -----
43 # AWS Resources
44 #
45 dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION_NAME)
46 sns = boto3.client('sns', region_name=AWS_REGION_NAME)
47
48 # DynamoDB Tables
49 user_table = dynamodb.Table(USER_TABLE_NAME)
50 appointment_table = dynamodb.Table(APPOINTMENTS_TABLE_NAME)
51
52 # -----
53 # Logging
54 #
55 logging.basicConfig(
56     level=logging.INFO,
57     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
58     handlers=[
59         logging.FileHandler("app.log"),
60         logging.StreamHandler()
61     ]
62 )
63 logger = logging.getLogger(__name__)
```

- **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 M X .env U app.log U base.html view_appointment_doctor.html

app.py > ...
71 def get_user_role(email):
72     ...
73     except Exception as e:
74         logger.error(f"Error fetching role: {e}")
75         return None
76
77
78 def send_email(to_email, subject, body):
79     if not ENABLE_EMAIL:
80         logger.info(f"[Email Skipped] Subject: {subject} to {to_email}")
81         return
82
83     try:
84         msg = MIMEText(body)
85         msg['From'] = SENDER_EMAIL
86         msg['To'] = to_email
87         msg['Subject'] = subject
88         msg.attach(MIMEText(body, 'plain'))
89
90         server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
91         server.starttls()
92         server.login(SENDER_EMAIL, SENDER_PASSWORD)
93         server.sendmail(SENDER_EMAIL, to_email, msg.as_string())
94         server.quit()
95
96         logger.info(f"Email sent to {to_email}")
97     except Exception as e:
98         logger.error(f"Email sending failed: {e}")
99
100
101 def publish_to_sns(message, subject="Salon Notification"):
102     if not ENABLE_SNS:
103         logger.info("[SNS Skipped] Message: {}".format(message))
104         return
105
106     try:
107         response = sns.publish(

```

- **Routes for Web Pages:**

Register Page

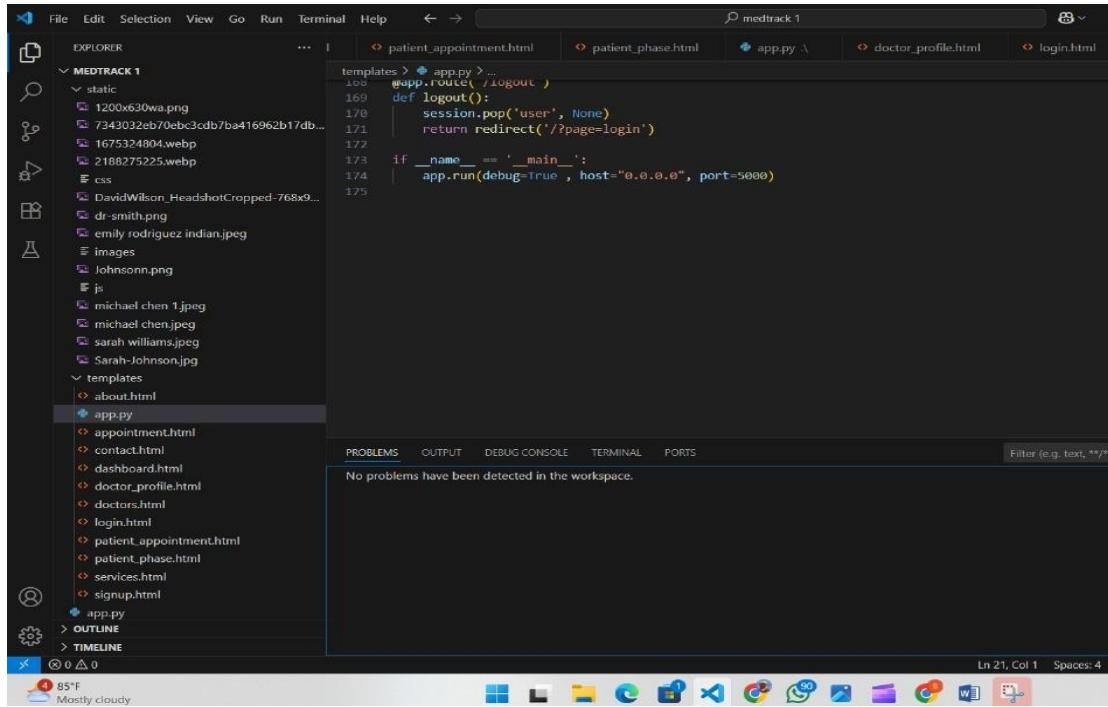
```

app.py      X  patient_details.html  doctor_dashboard.html  patient_dashboard.html  book_appointment

app.py > ...
10  @app.route('/')
11  def index():
12      return render_template('index.html')
13
14  @app.route('/signup', methods=['GET', 'POST'])
15  def signup():
16      if request.method == 'POST':
17          username = request.form['username']
18          email = request.form['email']
19          password = request.form['password']
20          confirm_password = request.form['confirm_password']
21
22          if username in users:
23              flash('Username already exists.', 'error')
24          elif password != confirm_password:
25              flash('Passwords do not match.', 'error')
26          else:
27              users[username] = {'email': email, 'password': password}
28              flash('Signup successful! Please log in.', 'success')
29              return redirect(url_for('login'))
30
31      return render_template('signup.html')
32
33  @app.route('/login', methods=['GET', 'POST'])
34  def login():
35      if request.method == 'POST':
36          username = request.form['username']
37          password = request.form['password']
38
39          user = users.get(username)
40          if user and user['password'] == password:
41              session['username'] = username
42              flash('Login successful.', 'success')
43              return redirect(url_for('home'))
44          else:
45              flash('Invalid credentials.', 'error')
46

```

- 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.



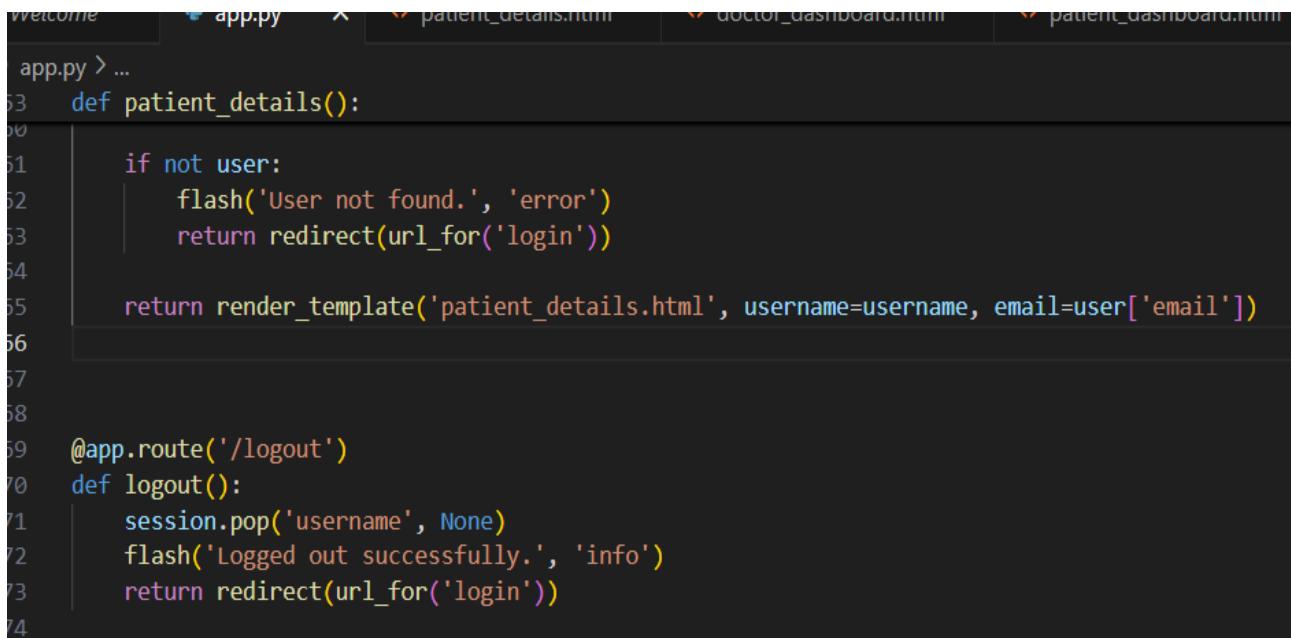
```

EXPLORER          patient_appointment.html      patient_phase.html      app.py      doctor_profile.html      login.html
MEDTRACK 1
static
1200x630wa.png
734302eb70ebc3db7ba416962b17db...
167532404.webp
216827525.webp
css
DavidWilson_HeadshotCropped-768x9...
dr-smith.png
emily rodriguez indian.jpeg
images
Johnsonn.png
js
michael chen 1.jpeg
michael chen.jpeg
sarah williams.jpeg
Sarah-Johnson.jpg
templates
about.html
app.py
appointment.html
contact.html
dashboard.html
doctor_profile.html
doctors.html
login.html
patient_appointment.html
patient_phase.html
services.html
signup.html
app.py
> OUTLINE
> TIMELINE
0 △ 0
85°F
Mostly cloudy

```

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.



```

app.py > ...
def patient_details():
    if not user:
        flash('User not found.', 'error')
        return redirect(url_for('login'))

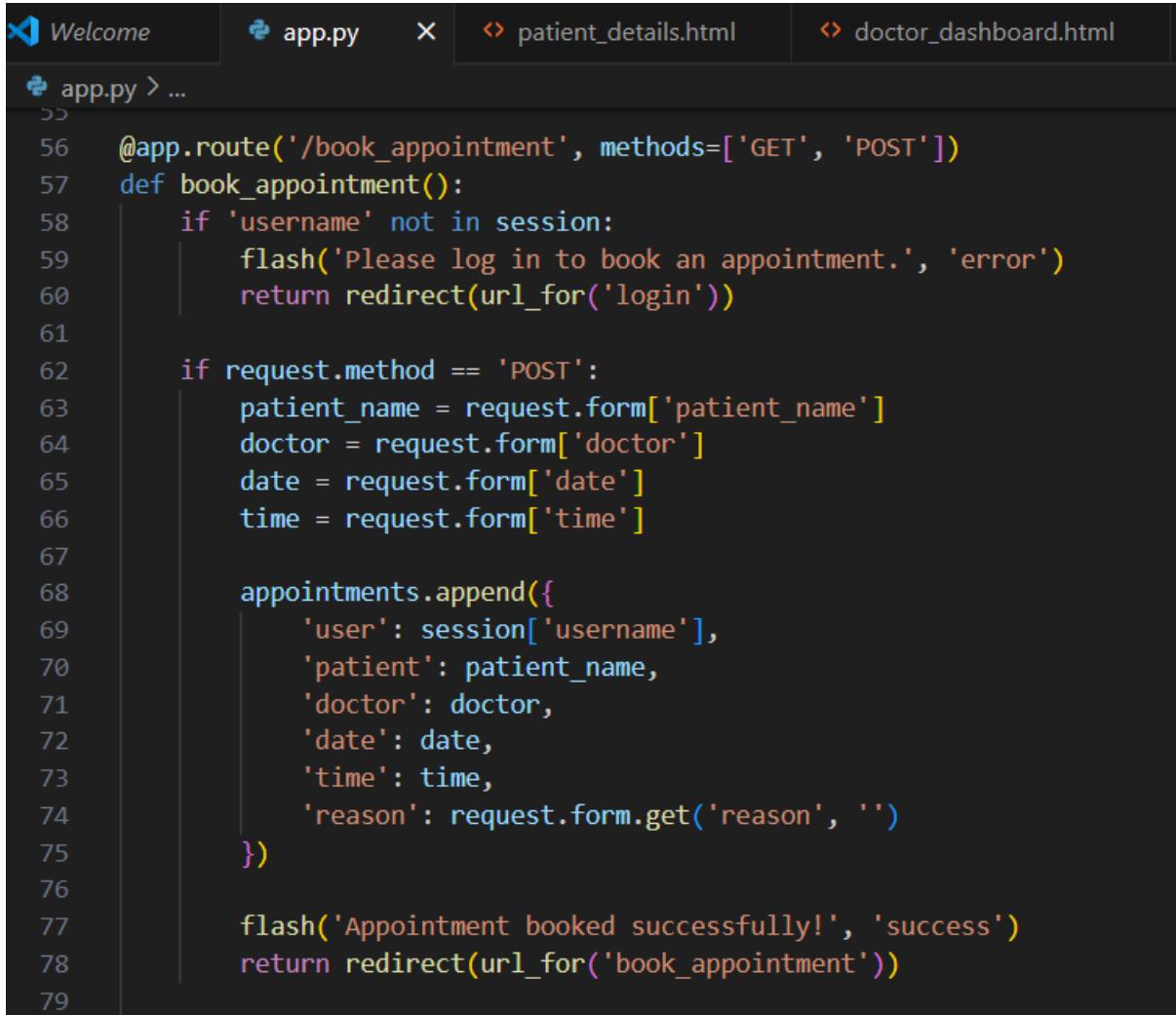
    return render_template('patient_details.html', username=username, email=user['email'])

@app.route('/logout')
def logout():
    session.pop('username', None)
    flash('Logged out successfully.', 'info')
    return redirect(url_for('login'))

```

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.



```

Welcome      app.py      patient_details.html      doctor_dashboard.html
app.py > ...
56 @app.route('/book_appointment', methods=['GET', 'POST'])
57 def book_appointment():
58     if 'username' not in session:
59         flash('Please log in to book an appointment.', 'error')
60         return redirect(url_for('login'))
61
62     if request.method == 'POST':
63         patient_name = request.form['patient_name']
64         doctor = request.form['doctor']
65         date = request.form['date']
66         time = request.form['time']
67
68         appointments.append({
69             'user': session['username'],
70             'patient': patient_name,
71             'doctor': doctor,
72             'date': date,
73             'time': time,
74             'reason': request.form.get('reason', '')
75         })
76
77         flash('Appointment booked successfully!', 'success')
78         return redirect(url_for('book_appointment'))
79

```

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.

```
❖ Welcome    ❖ app.py    ❖ patient_details.html    ❖ doctor_dashboard.html    ❖ patient_dashboard.html
❖ app.py > ...
144 def patient_appointments():
145     flash('Please log in.', 'error')
146     return redirect(url_for('login'))
147
148
149     user_appts = [a for a in appointments if a['user'] == session['username']]
150     return render_template('patient_appointments.html', appointments=user_appts)
151
152 @app.route('/patient_details')
153 def patient_details():
154     if 'username' not in session:
155         flash('Please log in to view your details.', 'error')
156         return redirect(url_for('login'))
157
158     username = session['username']
159     user = users.get(username)
160
161     if not user:
162         flash('User not found.', 'error')
163         return redirect(url_for('login'))
164
165     return render_template('patient_details.html', username=username, email=user['email'])
166
167
168
169 @app.route('/logout')
170 def logout():
171     session.pop('username', None)
172     flash('Logged out successfully.', 'info')
173     return redirect(url_for('login'))
174
175 if __name__ == '__main__':
176     app.run(debug=True)
177
```



AWS - Local Deployment Sample Code - Google Docs..
No description..

<https://docs.google.com/document/d/1sFF7-tJ6IgWtRbawWoA4W3PkxEFrSJZhKzULgLsjxo/edit?usp=sharing>

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.

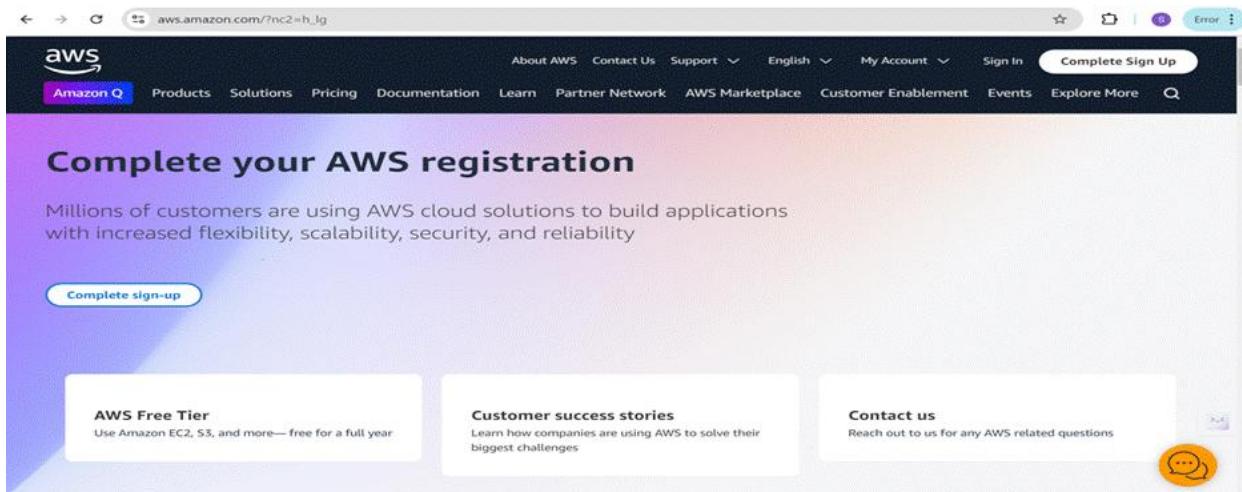
Please refer the below link -

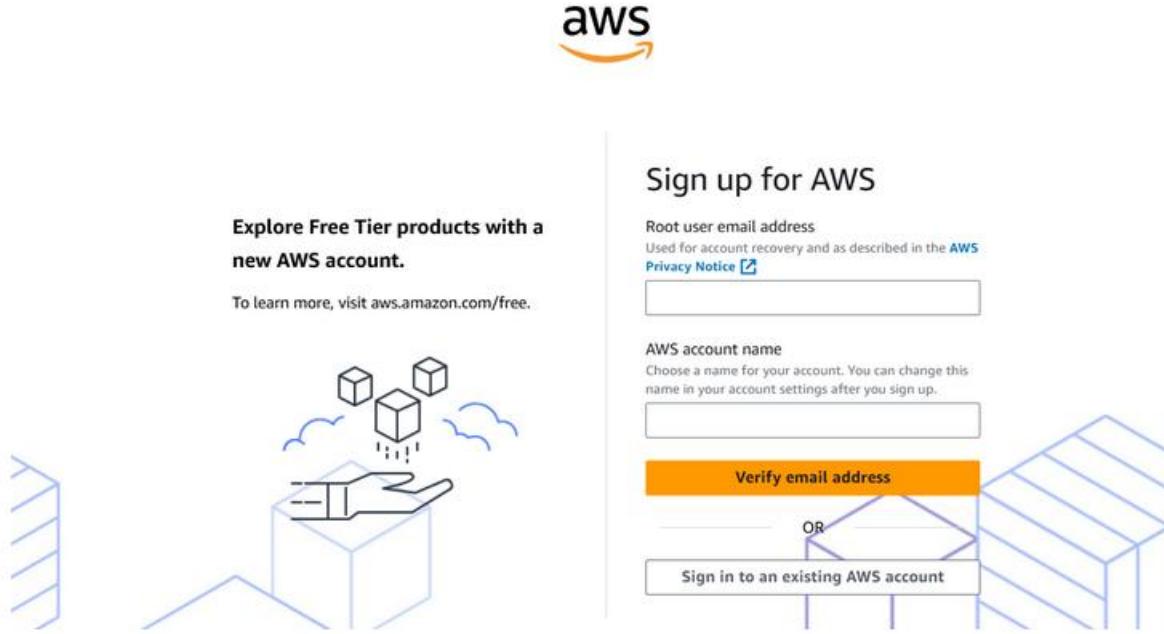
<https://drive.google.com/file/d/1HzWc7AMJ2BrxhV-uaw5s0vWtcd-28qgl/view?usp=sharing>

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.

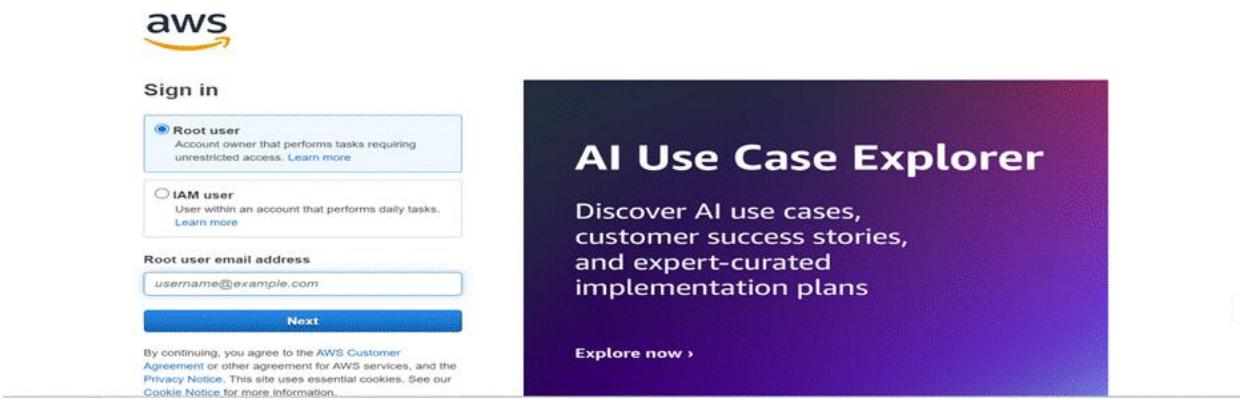
- Go to the AWS website (<https://aws.amazon.com/>).
- Click on the "Create an AWS Account" button.
- Follow the prompts to enter your email address and choose a password.
- Provide the required account information, including your name, address, and phone number.
- Enter your payment information. (Note: 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.





The image shows the AWS sign-up landing page. At the top is the AWS logo. Below it is a section titled "Explore Free Tier products with a new AWS account." with a sub-instruction "To learn more, visit aws.amazon.com/free". To the right is a form for "Sign up for AWS". It includes fields for "Root user email address" (with a note about account recovery), "AWS account name" (with a note about changing it later), and two buttons: "Verify email address" (highlighted in orange) and "Sign in to an existing AWS account". A large blue 3D cube graphic is visible on the right side of the page.

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



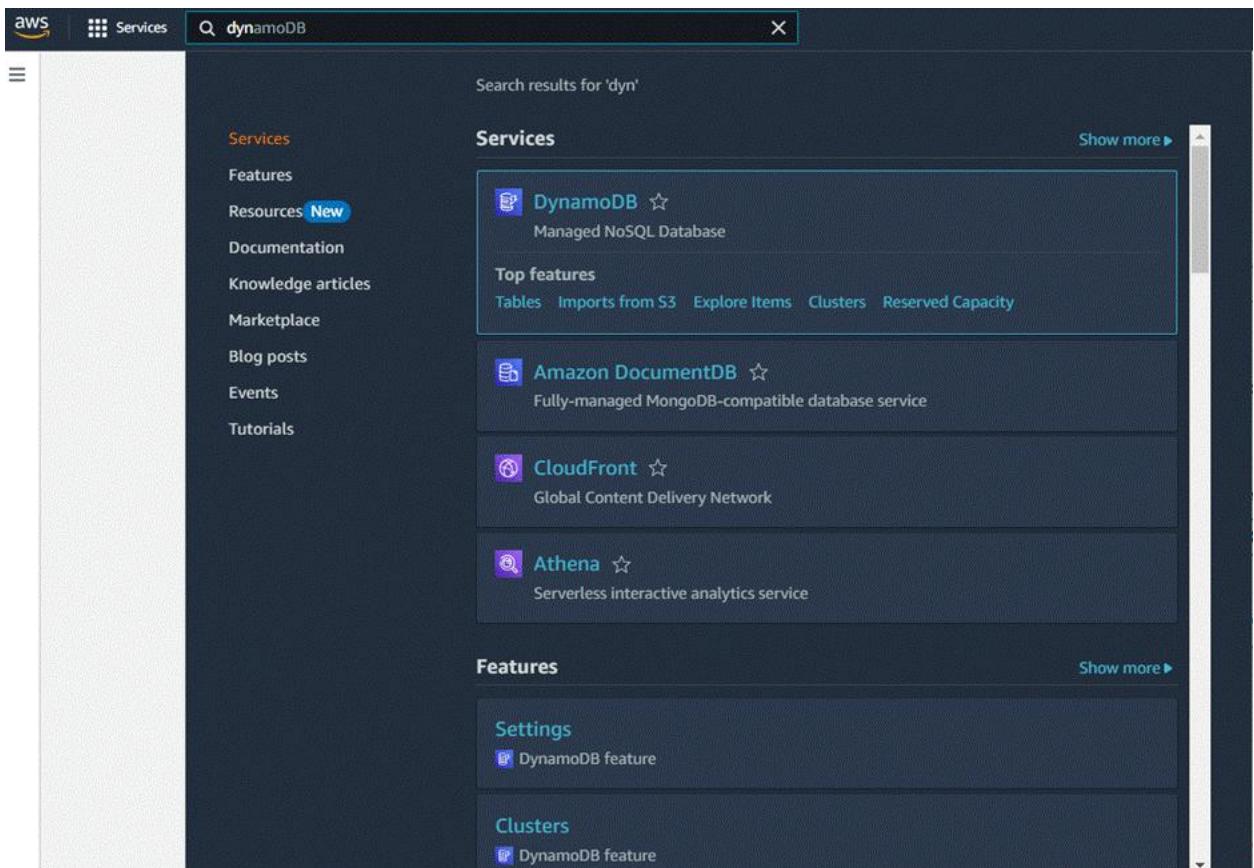
The image displays two adjacent screenshots. On the left is the AWS sign-in page, showing options for "Root user" or "IAM user", a "Root user email address" field containing "username@example.com", and a "Next" button. A small legal notice is at the bottom. On the right is a dark-themed promotional card for the "AI Use Case Explorer". It features the title "AI Use Case Explorer" in large white letters, followed by the text "Discover AI use cases, customer success stories, and expert-curated implementation plans", and a "Explore now >" button.

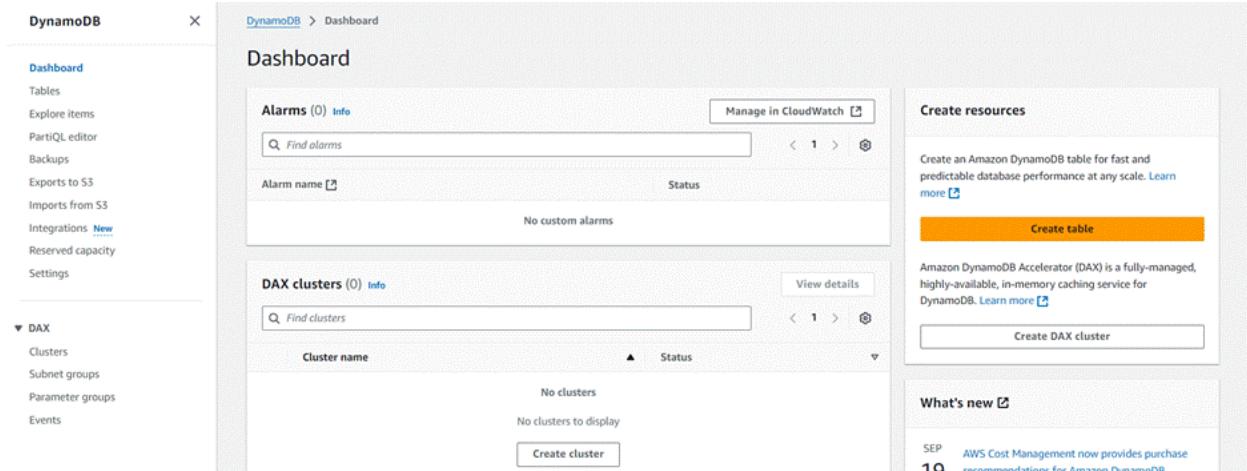
Milestone 3: DynamoDB Database Creation and Setup

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.





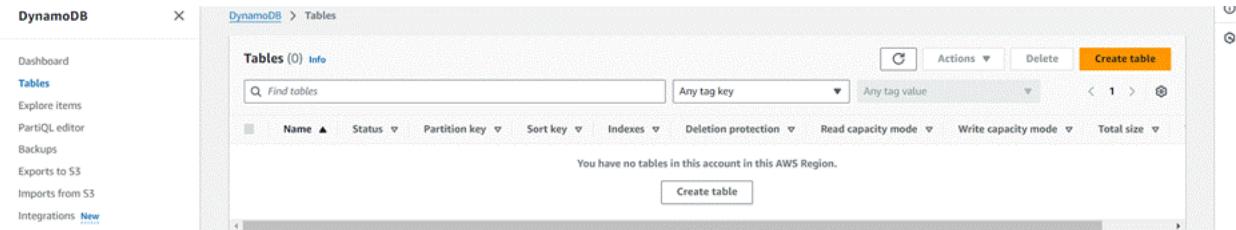
The screenshot shows the AWS DynamoDB Dashboard. On the left, there is a navigation sidebar with the following menu items:

- Dashboard**
- Tables
- Explore items
- PartiQL editor
- Backups
- Exports to S3
- Imports from S3
- Integrations New
- Reserved capacity
- Settings
- DAX**
 - Clusters
 - Subnet groups
 - Parameter groups
 - Events

The main content area displays two sections:

- Alarms (0) Info**: A table with one row: "No custom alarms". A "Manage in CloudWatch" button is available.
- DAX clusters (0) Info**: A table with one row: "No clusters". A "Create cluster" button is available.

On the right side, there is a "Create resources" section with a "Create table" button, a "Create DAX cluster" button, and a "What's new" section with a note about AWS Cost Management.



The screenshot shows the AWS DynamoDB Tables page. On the left, there is a navigation sidebar with the following menu items:

- Tables**
- Explore items
- PartiQL editor
- Backups
- Exports to S3
- Imports from S3
- Integrations New

The main content area displays a table titled "Tables (0) Info" with one row: "You have no tables in this account in this AWS Region." A "Create table" button is available.

Create a DynamoDB table for storing data

- Create Users table with partition key “Email” 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.

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

Partition key
 The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
 String
1 to 255 characters and case sensitive.

Sort key - optional
 You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 String
1 to 255 characters and case sensitive.

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.

Create table

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

aws [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.

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.

1 to 255 characters and case sensitive. String

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.

1 to 255 characters and case sensitive. String

Student - Skill Wallet | trover.in/students/labs/680331 | trover.in/students/labs/680331 | List tables | Amazon DynamoDB | +

us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables

aws [Alt+S]

DynamoDB > Tables

DynamoDB

- Dashboard
- Tables**
- Explore items
- PartQL Editor
- Backups
- Exports to S3
- Imports from S3
- Integrations New
- Reserved capacity
- Settings

DAX

- Clusters
- Subnet groups
- Parameter groups
- Events

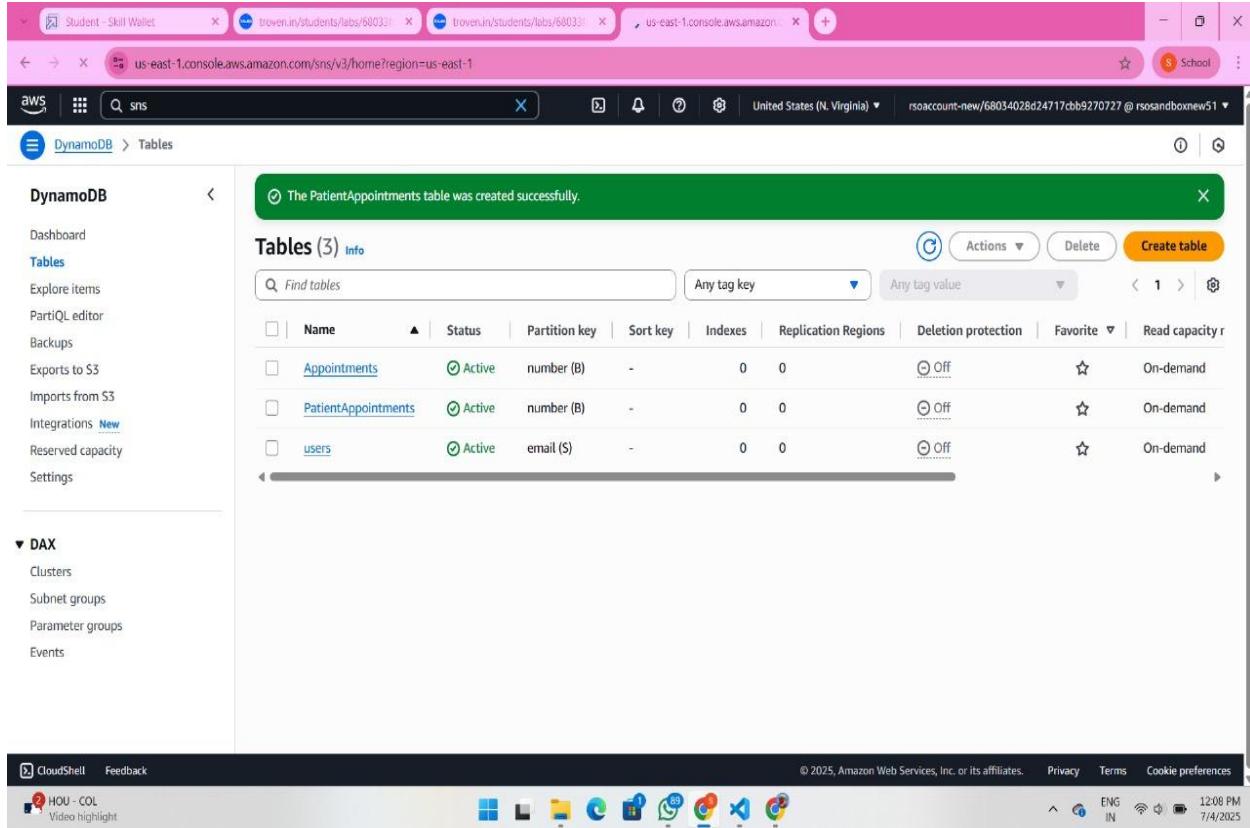
⌚ The PatientAppointments table was created successfully.

	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity r
<input type="checkbox"/>	Appointments	Active	number (B)	-	0	0	Off	★	On-demand
<input type="checkbox"/>	PatientAppointments	Active	number (B)	-	0	0	Off	★	On-demand
<input type="checkbox"/>	users	Active	email (\$)	-	0	0	Off	★	On-demand

C
[Actions](#) D
[Delete](#) Create table

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

76°F Cloudy ENG IN 11:58 AM 7/4/2025



The screenshot shows the AWS DynamoDB 'Tables' page. A success message at the top states: 'The PatientAppointments table was created successfully.' The table list shows three items:

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity
Appointments	Active	number (B)	-	0	0	Off	☆	On-demand
PatientAppointments	Active	number (B)	-	0	0	Off	☆	On-demand
users	Active	email (\$)	-	0	0	Off	☆	On-demand

Left sidebar navigation includes: Dashboard, Tables (selected), Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations (New), Reserved capacity, Settings, and DAX (Clusters, Subnet groups, Parameter groups, Events).

Bottom navigation bar includes: CloudShell, Feedback, © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, Cookie preferences, ENG IN, 12:08 PM, 7/4/2025.

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.

X

Search results for 'sns'

Services Show more ▶

- Features
- Resources New
- Documentation
- Knowledge articles
- Marketplace
- Blog posts
- Events
- Tutorials

Services

 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

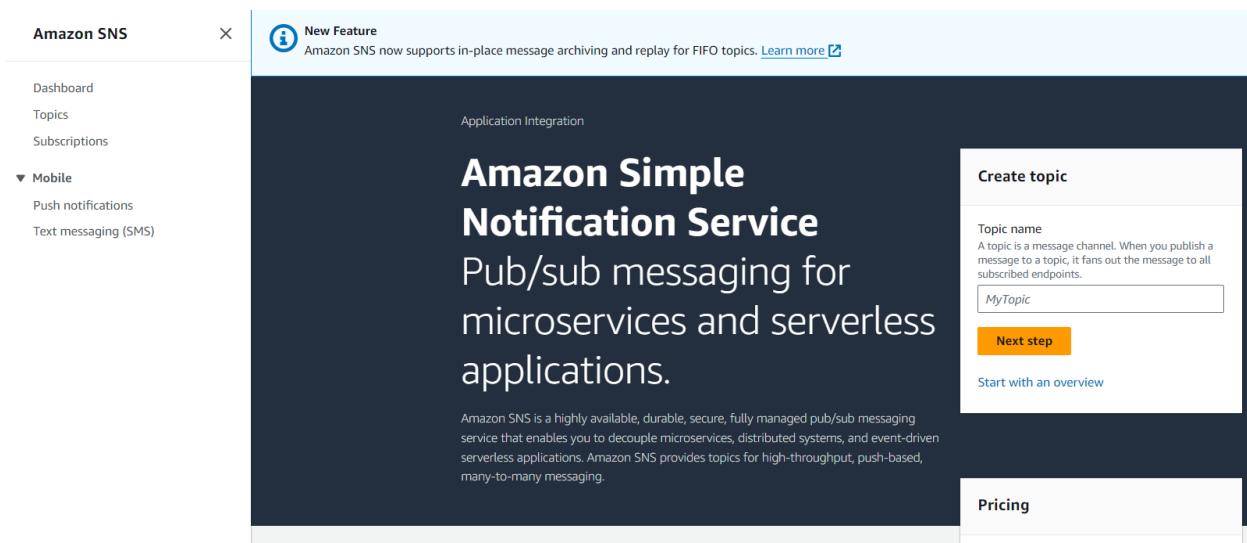
 ElastiCache feature

SMS

 AWS End User Messaging feature

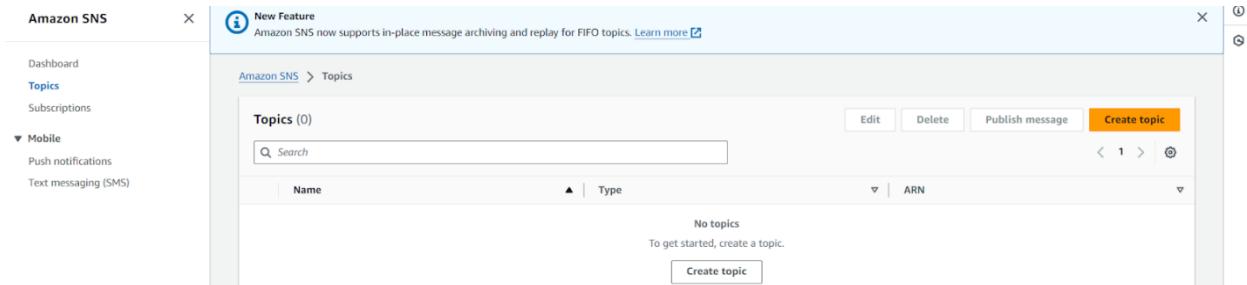
Hosted zones

 Route 53 feature



The screenshot shows the Amazon Simple Notification Service (SNS) "Create topic" page. The left sidebar has a "Topics" section selected. The main content area features a large heading "Amazon Simple Notification Service" and a sub-section "Pub/sub messaging for microservices and serverless applications." Below this is a brief description of the service. On the right, there is a "Create topic" form with a "Topic name" field containing "MyTopic". A "Next step" button is visible below the form, and a "Start with an overview" link is at the bottom.

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



The screenshot shows the Amazon SNS "Topics" page. The left sidebar has a "Topics" section selected. The main content area displays a table titled "Topics (0)" with columns for "Name", "Type", and "ARN". A search bar and pagination controls are at the top of the table. A "Create topic" button is located at the bottom of the table. The page also includes a note: "No topics To get started, create a topic."

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

i **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**

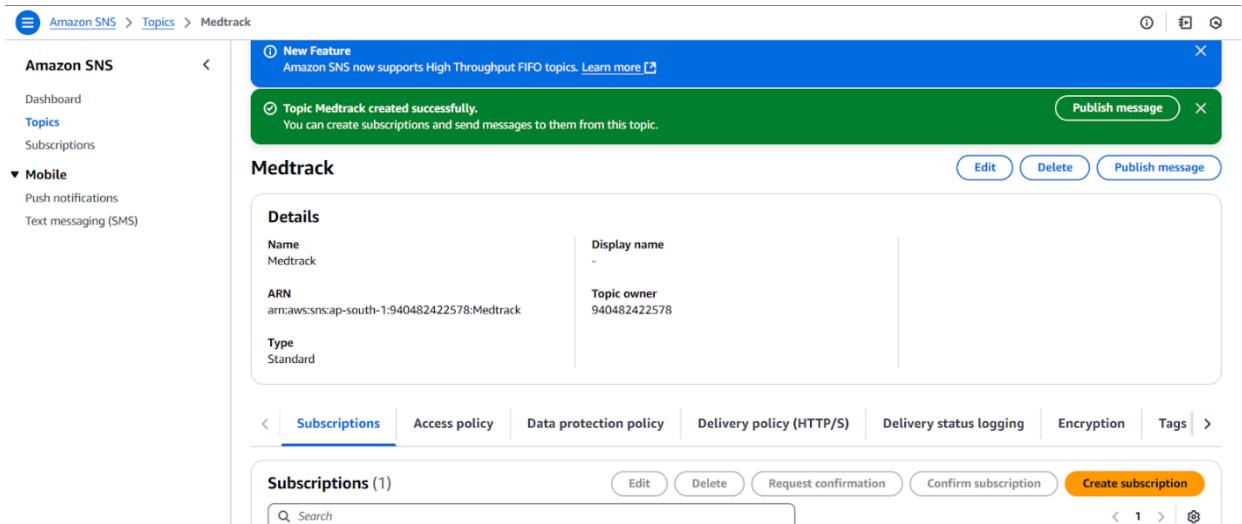
 A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)
▶ **Active tracing - optional** Info

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

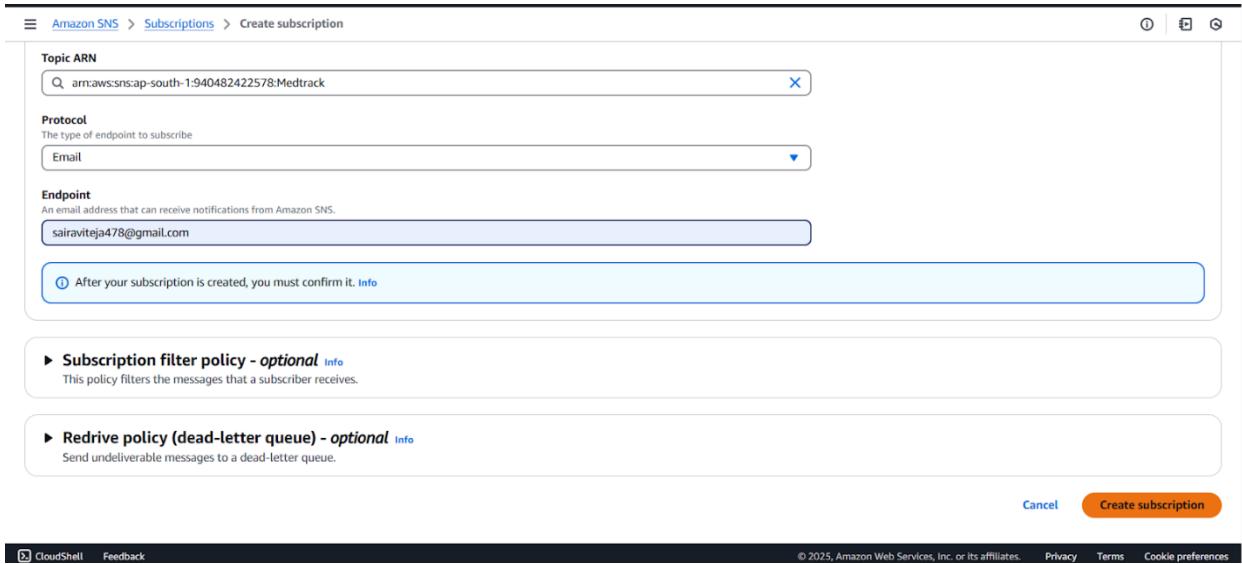
- Configure the SNS topic and note down the **Topic ARN**.



The screenshot shows the Amazon SNS Topics page. On the left, there's a sidebar with links for Dashboard, Topics, Subscriptions, and Mobile (Push notifications, Text messaging (SMS)). The main area shows a topic named "Medtrack". A green success message at the top says "Topic Medtrack created successfully. You can create subscriptions and send messages to them from this topic." Below the message, there are buttons for Edit, Delete, and Publish message. The "Details" section shows the Name (Medtrack), Display name (-), ARN (arn:aws:sns:ap-south-1:940482422578:Medtrack), and Topic owner (940482422578). The Type is listed as Standard. Below the details, there are tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags. The "Subscriptions" tab is selected, showing one subscription (1) with a "Create subscription" button. There's also a "Search" bar.

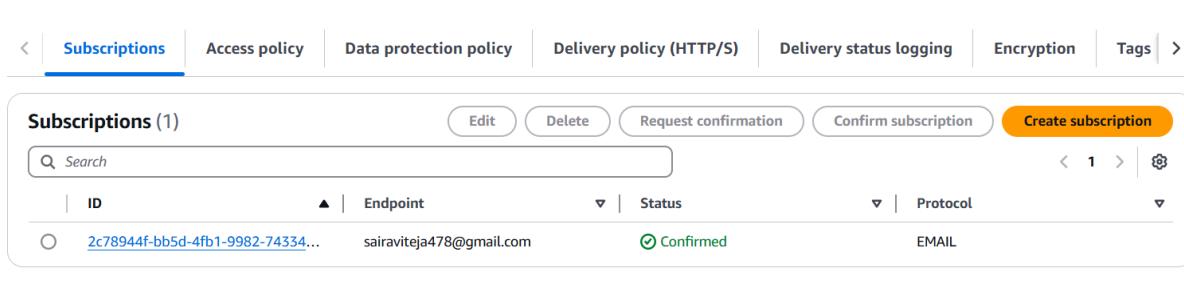
Subscribe users and Admin

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



The screenshot shows the "Create subscription" page for the "Medtrack" topic. At the top, it says "Topic ARN" and shows the ARN "arn:aws:sns:ap-south-1:940482422578:Medtrack". Below that, the "Protocol" is set to "Email". The "Endpoint" field contains the email address "sairaviteja478@gmail.com". A note below the endpoint says "After your subscription is created, you must confirm it." In the bottom left, there are sections for "Subscription filter policy - optional" (with a note about filtering messages) and "Redrive policy (dead-letter queue) - optional" (with a note about sending undeliverable messages to a dead-letter queue). At the bottom right, there are "Cancel" and "Create subscription" buttons.

- After subscription request for the mail confirmation



The screenshot shows the 'Subscriptions' tab selected in the navigation bar. Below it is a table with one row of data:

ID	Endpoint	Status	Protocol
2c78944f-bb5d-4fb1-9982-74334...	sairaviteja478@gmail.com	Confirmed	EMAIL

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

AWS Notification - Subscription Confirmation Inbox x

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

9

to me ▾

You have chosen to subscribe to the topic:

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

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

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

to me ▾

...

You have chosen to subscribe to the topic:

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

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)



Simple Notification Service

Subscription confirmed!

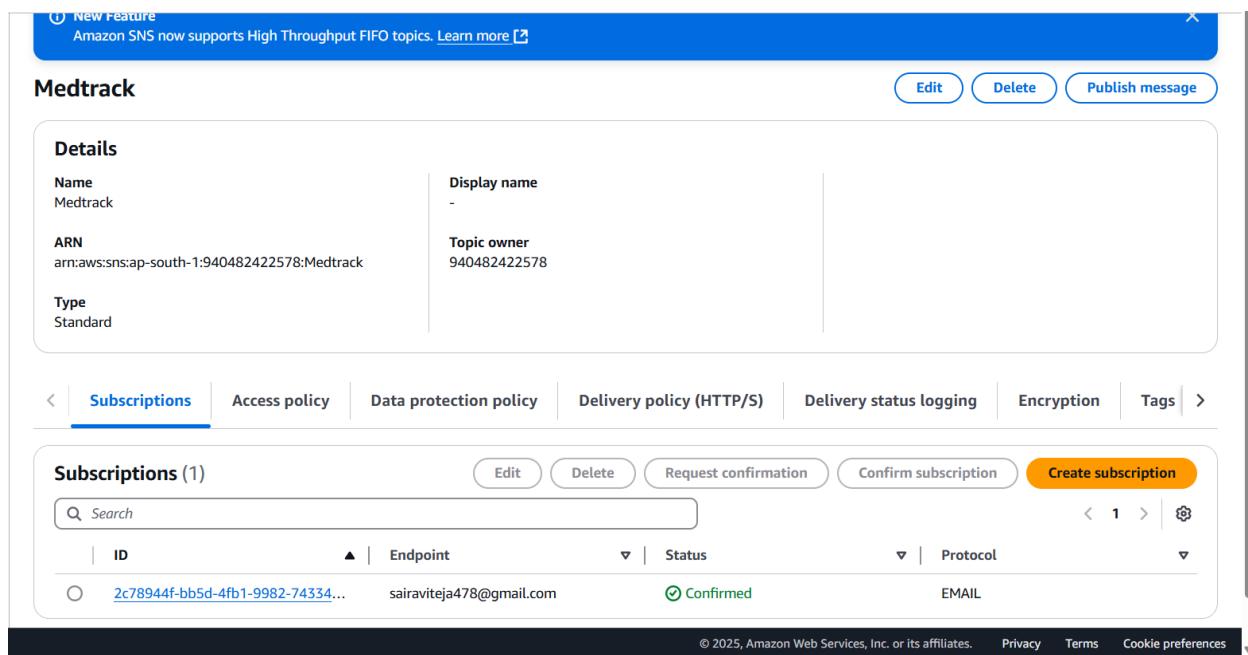
You have successfully subscribed.

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.



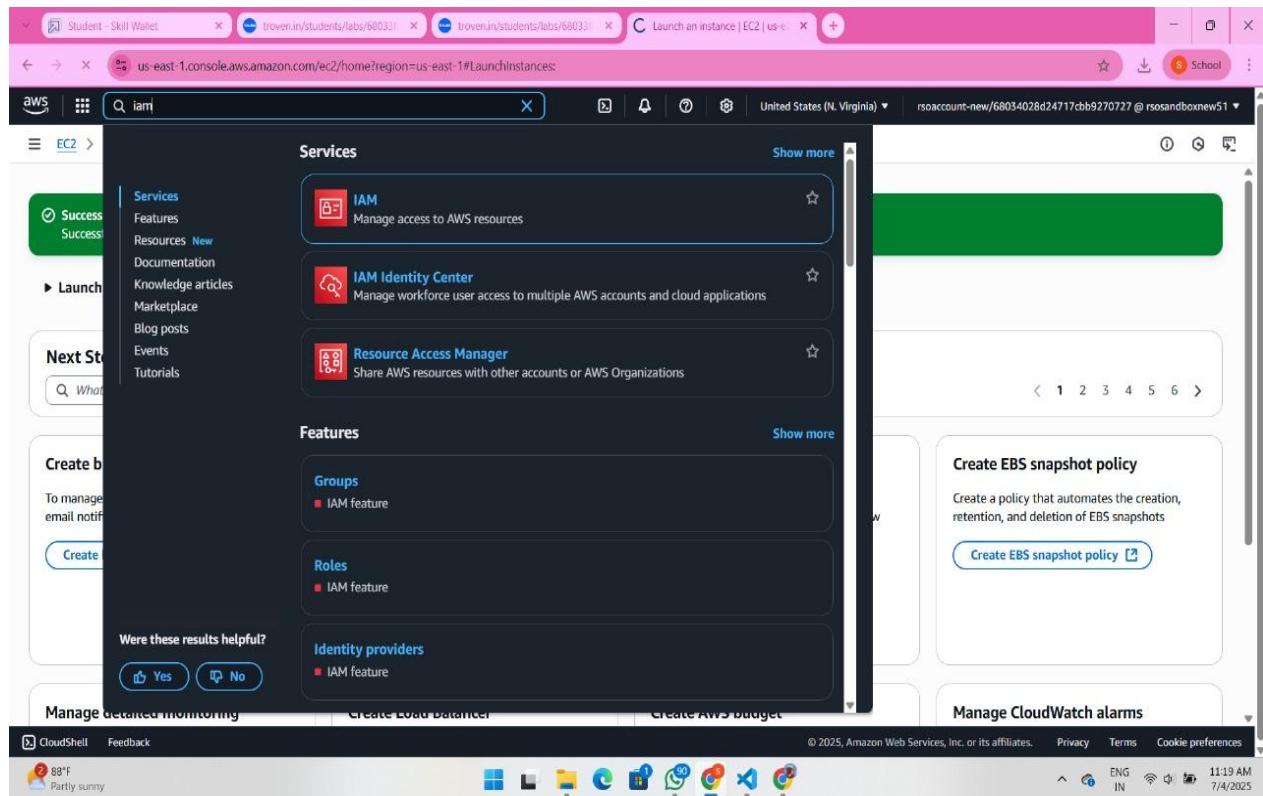
The screenshot shows the AWS SNS console for the 'Medtrack' topic. At the top, there is a blue banner with a 'New Feature' message about High Throughput FIFO topics. Below the banner, the topic name 'Medtrack' is displayed, along with its ARN (arn:aws:sns:ap-south-1:940482422578:Medtrack) and Type (Standard). The 'Details' section shows the display name as '-' and the topic owner as 940482422578. On the right side of the topic card, there are three buttons: 'Edit', 'Delete', and 'Publish message'. Below the topic card, there is a navigation bar with tabs: Subscriptions (which is selected), Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags. Under the 'Subscriptions' tab, there is a table titled 'Subscriptions (1)'. The table has columns: ID, Endpoint, Status, and Protocol. One row is shown, with the ID '2c78944f-bb5d-4fb1-9982-74334...', the Endpoint 'sairaviteja478@gmail.com', the Status 'Confirmed' (indicated by a green circle icon), and the Protocol 'EMAIL'. There are also buttons for 'Edit', 'Delete', 'Request confirmation', 'Confirm subscription', and 'Create subscription'.

Milestone 5: IAM Role Setup

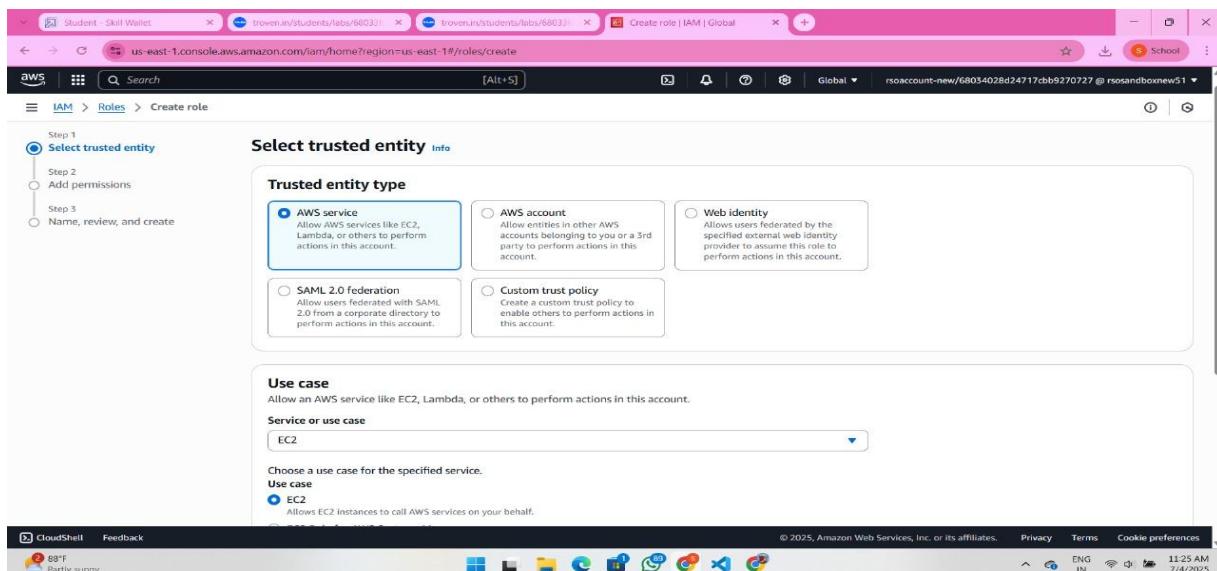
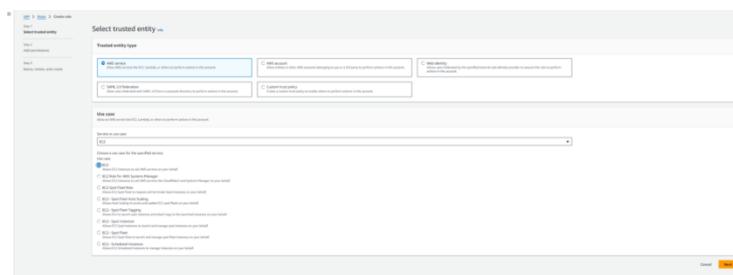
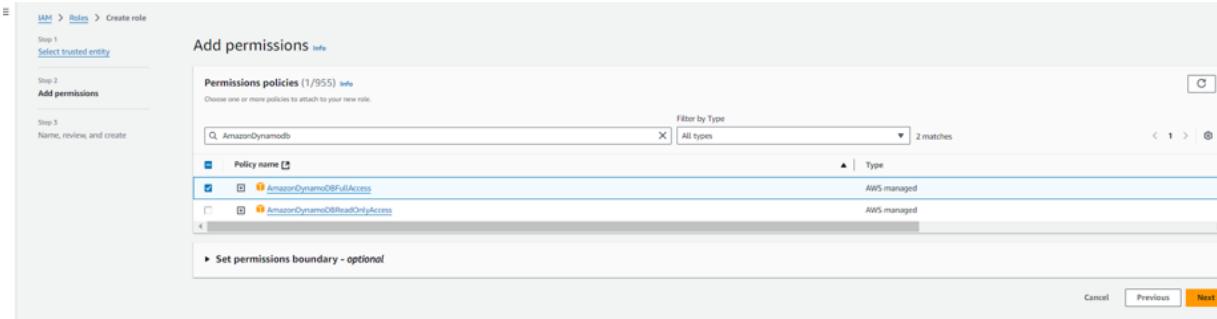
IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows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



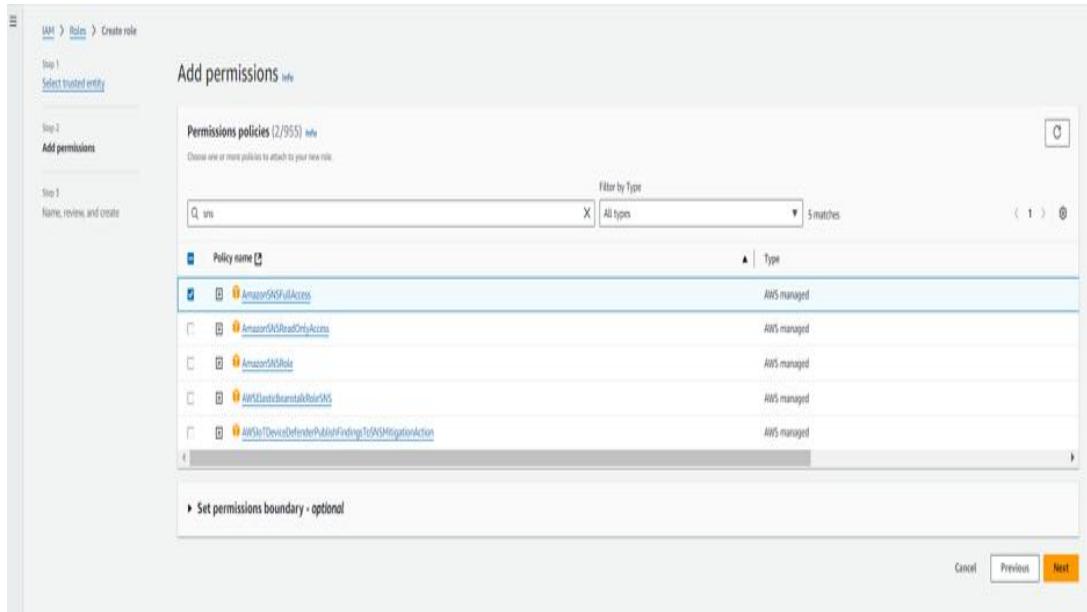
To create and select DynamoDBFullAccess and SNSFullAccess, go to the AWS IAM console, create a new role, and assign the respective policies. DynamoDBFullAccess allows full access to DynamoDB resources, while SNSFullAccess enables sending notifications via SNS. Attach the role to the relevant services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



The screenshot shows the 'Add permissions' step of the IAM role creation wizard. It lists several AWS managed policies under 'Permissions policies'. The 'AmazonSNSFullAccess' policy is selected and highlighted in blue. Other policies listed are: AmazonS3ReadOnlyAccess, AmazonVPCFullAccess, AmazonCloudWatchLogsRolePolicy, and AWSLambdaDeveloperPreviewFindingsToSNSIntegrationAction. The interface includes a search bar, a filter dropdown, and navigation buttons for 'Cancel', 'Previous', and 'Next'.

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.

FlaskDynamoSNSRole [Info](#)

Allows EC2 instances to call AWS services on your behalf.

[Delete](#) [Edit](#)

Summary	
Creation date	ARN
April 16, 2025, 22:10 (UTC+05:30)	arn:aws:iam::940482422578:role/FlaskDynamoSNSRole
Last activity	Instance profile ARN
5 hours ago	arn:aws:iam::940482422578:instance-profile/FlaskDynamoSNSRole
Maximum session duration	1 hour

[Permissions](#) [Trust relationships](#) [Tags](#) [Last Accessed](#) [Revoke sessions](#)

Permissions policies (2) [Info](#)

You can attach up to 10 managed policies.

[Simulate](#) [Remove](#) [Add permissions ▾](#)

Filter by Type		
<input type="text" value="Search"/>	All types	
<input type="checkbox"/> Policy name [edit]	▲ Type	▼ Attached entities
<input type="checkbox"/> AmazonDynamoDBFullAccess	AWS managed	3
<input type="checkbox"/> AmazonSNSFullAccess	AWS managed	3

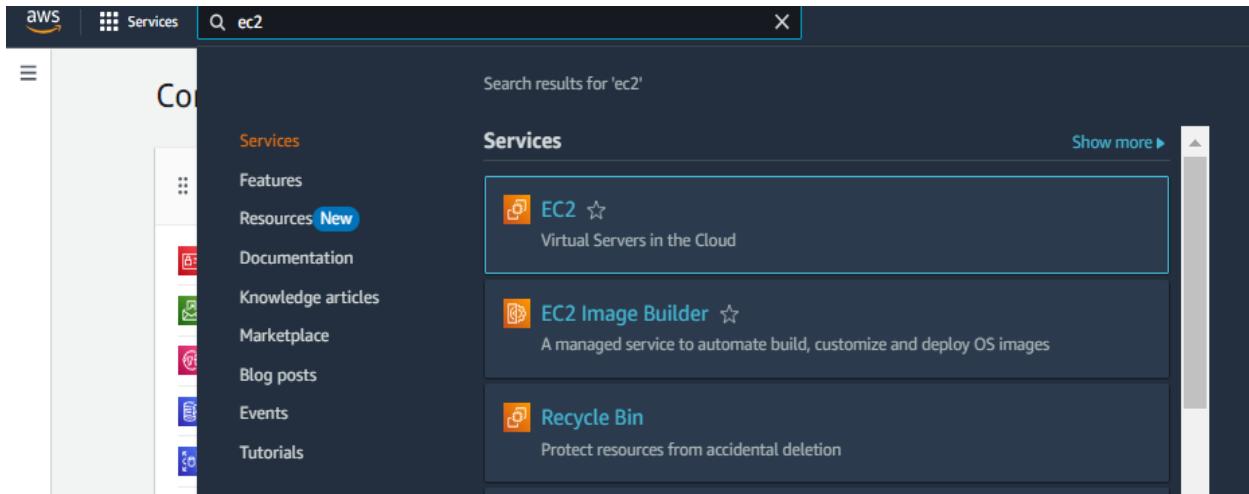
Milestone 6: EC2 Instance setup

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**

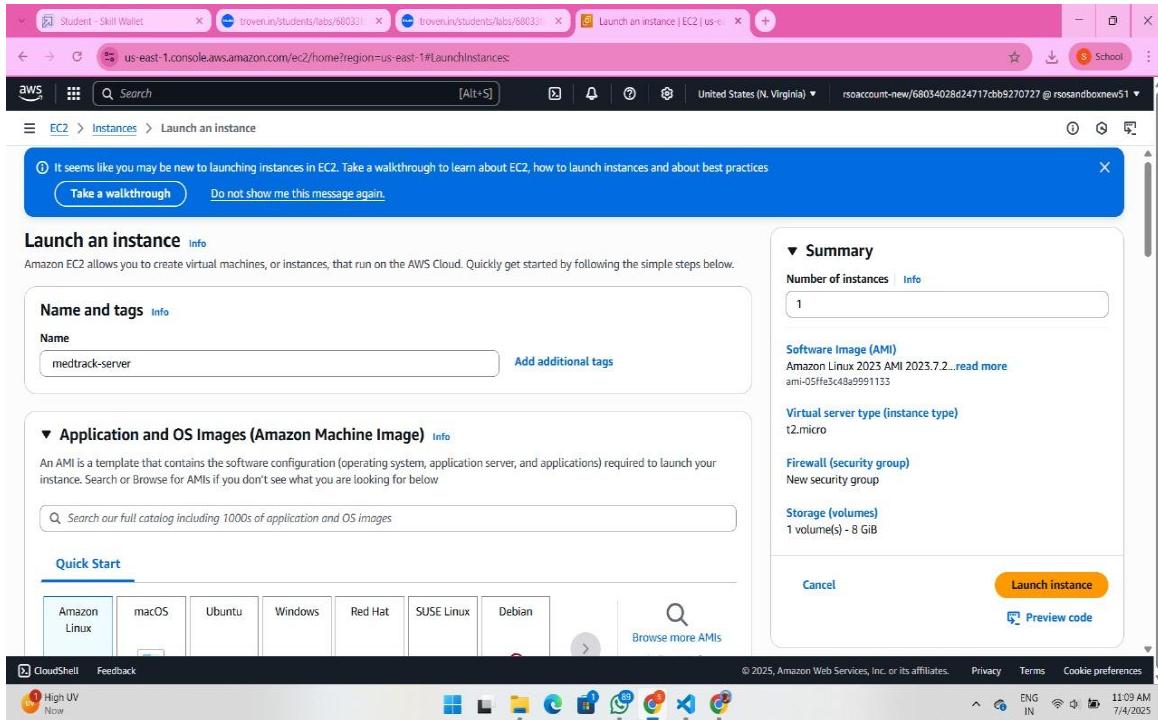
- In the AWS Console, navigate to EC2 and launch a new instance.



The screenshot shows the AWS CloudSearch interface with a search bar at the top containing 'ec2'. Below the search bar, there is a sidebar with various links: Services, Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main content area displays search results for 'ec2' under the 'Services' heading. The first result is 'EC2' with the subtext 'Virtual Servers in the Cloud'. The second result is 'EC2 Image Builder' with the subtext 'A managed service to automate build, customize and deploy OS images'. The third result is 'Recycle Bin' with the subtext 'Protect resources from accidental deletion'. There is also a 'Show more ▾' link.

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.



The screenshot shows the 'Launch an instance' wizard on the AWS EC2 dashboard. At the top, there is a message: 'It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices'. Below this are two buttons: 'Take a walkthrough' and 'Do not show me this message again.'.

The main form has the following sections:

- Name and tags**: A field where 'medtrack-server' is entered, with a link to 'Info' and a button to 'Add additional tags'.
- Application and OS Images (Amazon Machine Image)**: A section with a search bar and a list of OS options: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, Debian. A link to 'Browse more AMIs' is also present.
- Quick Start**: A row of buttons for different operating systems: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, Debian.
- Summary**: A summary table with the following data:

Number of instances	Info
1	

 Other details listed include:
 - Software Image (AMI)**: Amazon Linux 2023.7.2... [read more](#)
 - Virtual server type (instance type)**: t2.micro
 - Firewall (security group)**: New security group
 - Storage (volumes)**: 1 volume(s) - 8 GiB
- Buttons**: 'Cancel', 'Launch instance' (highlighted in orange), and 'Preview code'.

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.



Amazon Linux



macOS



Ubuntu



Windows



Red Hat



Search

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI Free tier eligible ▾

ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

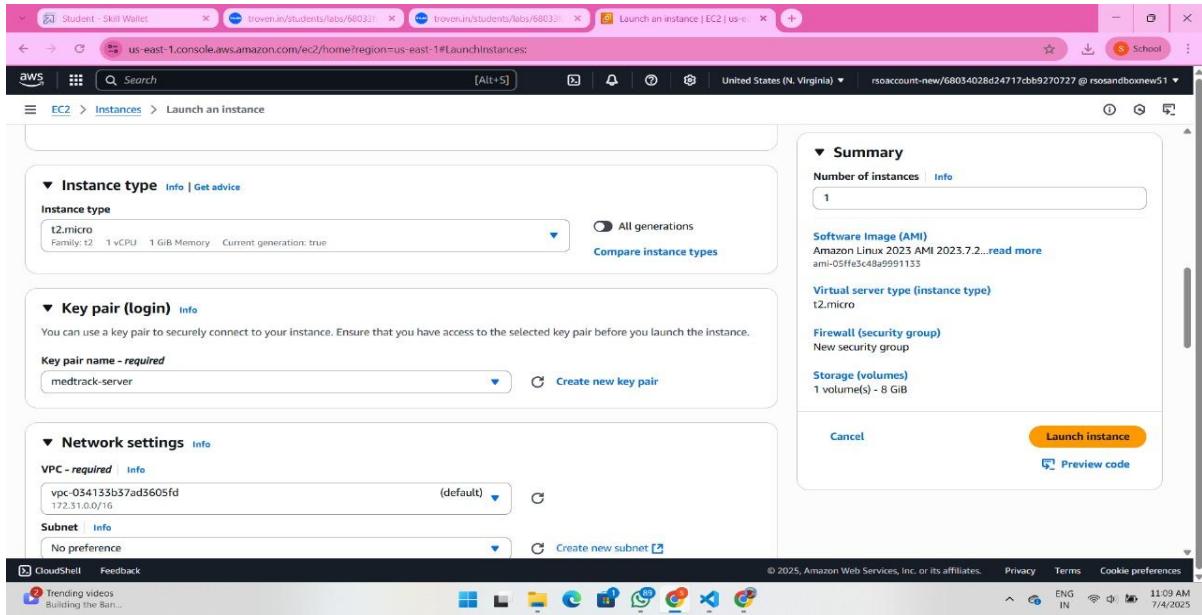
Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture	Boot mode	AMI ID
64-bit (x86) ▾	uefi-preferred	ami-02b49a24cfb95941c

Verified provider

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.



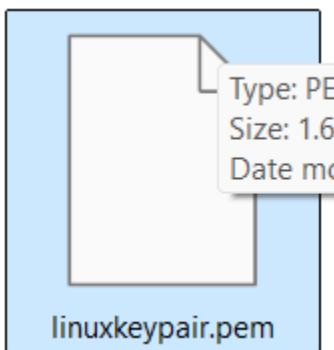
The screenshot shows the AWS EC2 'Launch an instance' wizard. The configuration includes:

- Instance type:** t2.micro (Family: t2, 1 vCPU, 1 GiB Memory, Current generation: true)
- Key pair (login):** medtrack-server
- Network settings:** VPC: vpc-034135b357ad3605fd (default), Subnet: No preference
- Summary:** Number of instances: 1, Software Image (AMI): Amazon Linux 2023 AMI 2023.7.2..., Virtual server type (instance type): t2.micro, Firewall (security group): New security group, Storage (volumes): 1 volume(s) - 8 GiB.
- Buttons:** Cancel, Launch instance (highlighted in orange), Preview code.

▼ 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.

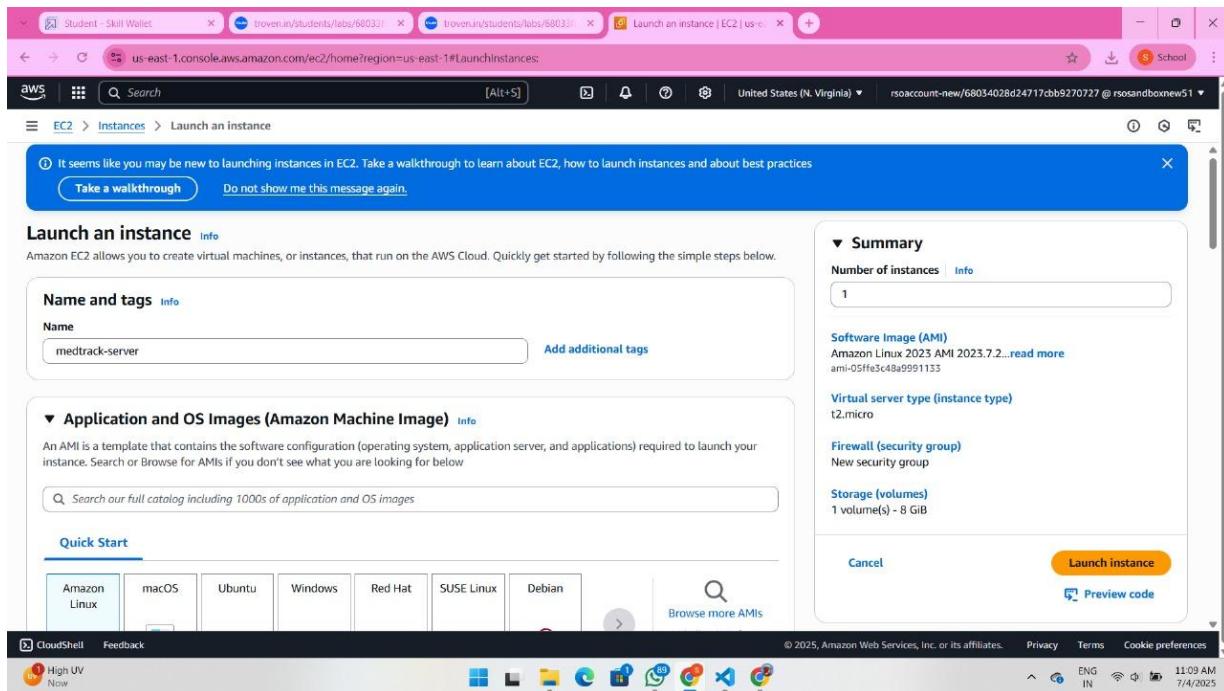
Key pair name - required

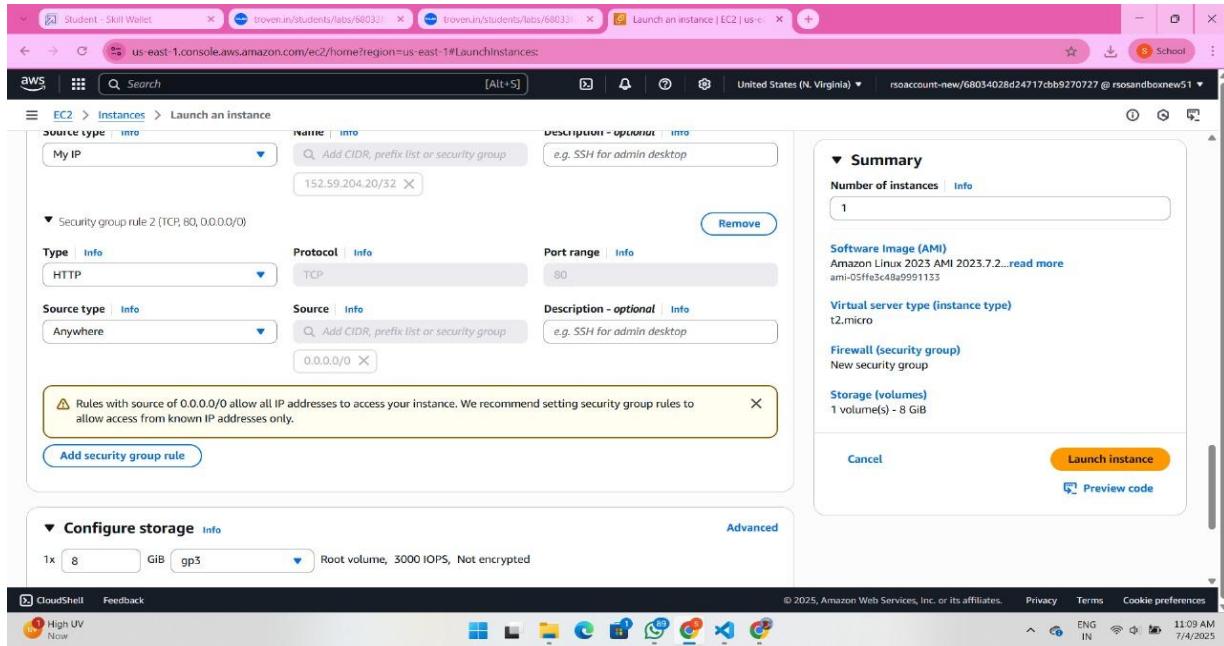
[Create new key pair](#)


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.





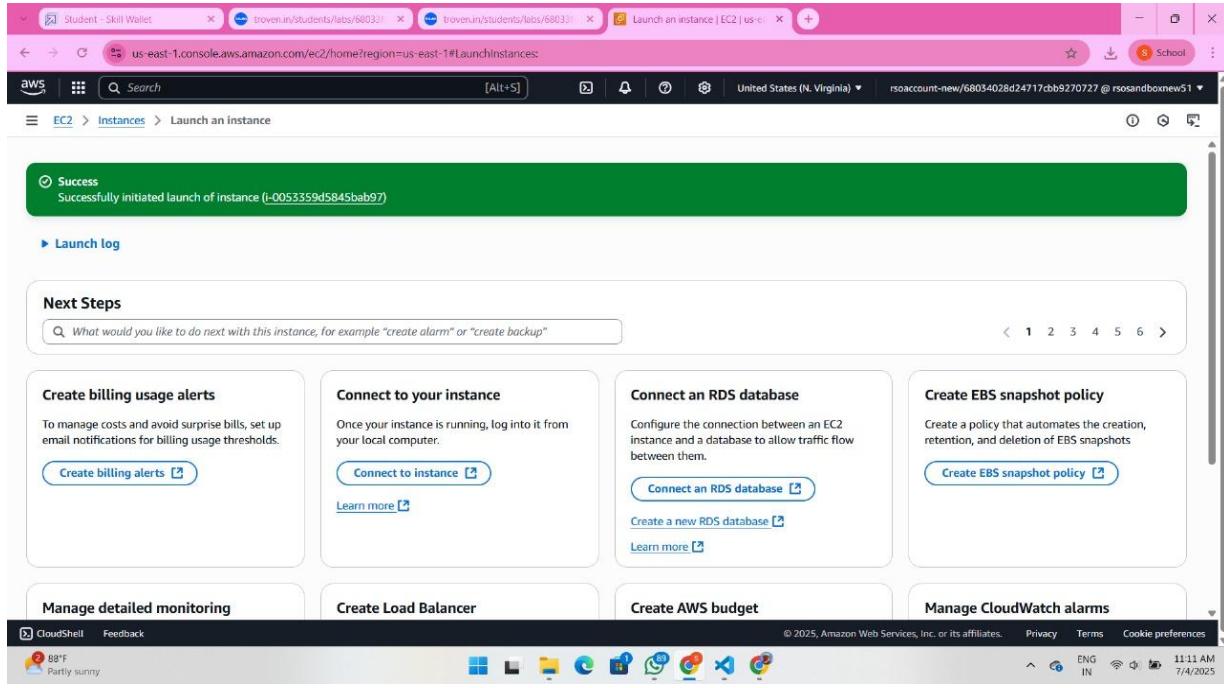
The screenshot shows the AWS EC2 'Launch an instance' wizard. In the 'Security group rule 2 (TCP, 80, 0.0.0.0/0)' section, there is a warning message: '⚠ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.' Below this, there is a 'Add security group rule' button.

In the 'Configure storage' section, it shows 1x 8 GiB gp3 Root volume, 3000 IOPS, Not encrypted.

On the right side, the 'Summary' panel shows:

- Number of instances: 1
- Software Image (AMI): Amazon Linux 2023 AMI 2023.7.2... (read more)
- Virtual server type (instance type): t2.micro
- Firewall (security group): New security group
- Storage (volumes): 1 volume(s) - 8 GiB

At the bottom right are 'Cancel', 'Launch instance', and 'Preview code' buttons.



The screenshot shows the AWS EC2 instance launch success page. It displays a green success message: 'Success Successfully initiated launch of instance (i-0053359d5845bab97)'. Below this, there is a 'Launch log' button.

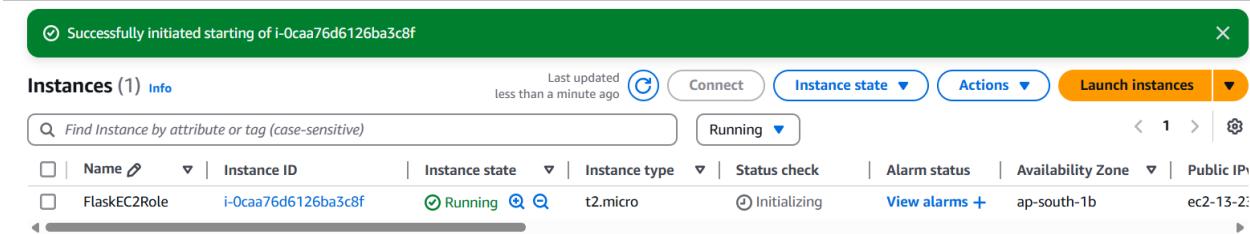
The 'Next Steps' section contains several buttons:

- Create billing usage alerts
- Connect to your instance
- Connect an RDS database
- Create EBS snapshot policy
- Manage detailed monitoring
- Create Load Balancer
- Create AWS budget
- Manage CloudWatch alarms

At the bottom, it shows the AWS navigation bar with CloudShell, Feedback, and various icons. The weather is listed as 88°F Partly sunny.

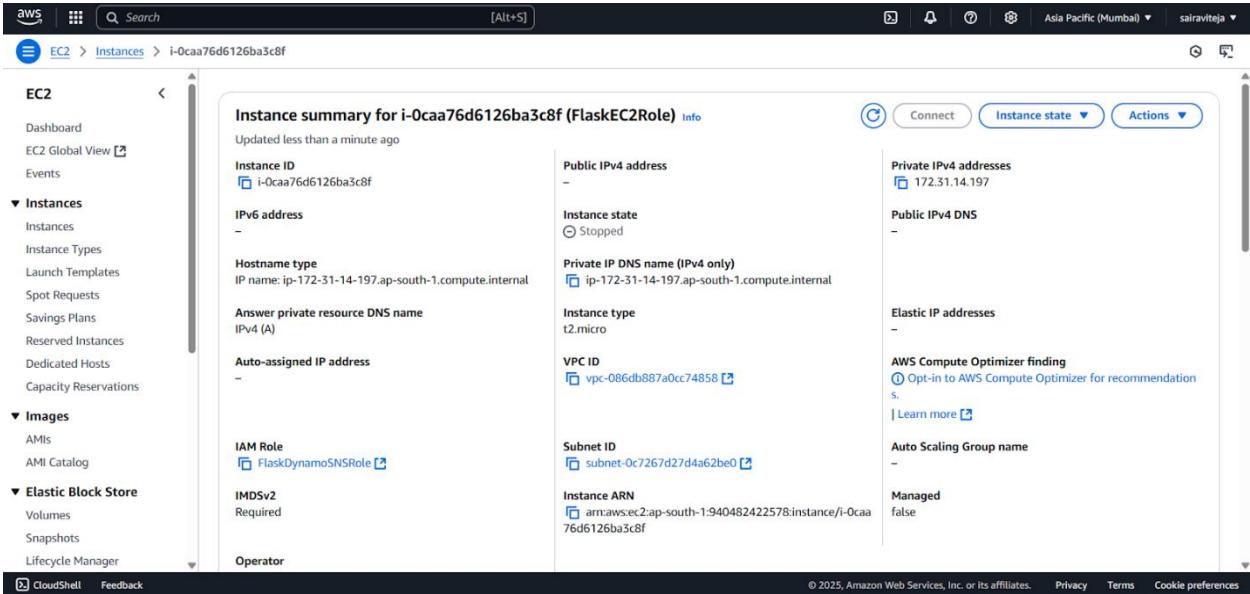
- To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the

- 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 screenshot shows the AWS EC2 Instances page. A green banner at the top indicates "Successfully initiated starting of i-0caa76d6126ba3c8f". Below the banner, there is a search bar and filters for "Name", "Instance ID", "Instance state", "Instance type", "Status check", "Alarm status", "Availability Zone", and "Public IP". A single instance is listed: "FlaskEC2Role" with Instance ID "i-0caa76d6126ba3c8f", which is "Running" (t2.micro, Initializing). The "Actions" dropdown menu is visible.

- 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.



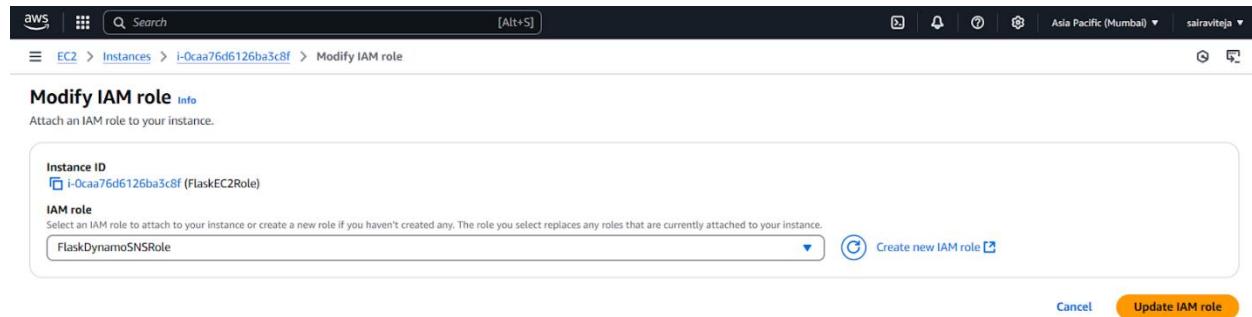
The screenshot shows the "Instance summary for i-0caa76d6126ba3c8f (FlaskEC2Role)" page. The left sidebar shows navigation options like Dashboard, EC2 Global View, Events, Instances, Images, Elastic Block Store, CloudShell, and Feedback. The main content area displays various instance details:

- Instance ID:** i-0caa76d6126ba3c8f
- IPv6 address:** -
- Hostname type:** IP name: ip-172-31-14-197.ap-south-1.compute.internal
- Answer private resource DNS name:** IPv4 (A)
- Auto-assigned IP address:** -
- IAM Role:** FlaskDynamoSNSRole
- IMDSv2:** Required
- Operator:** -
- Public IPv4 address:** -
- Instance state:** Stopped
- Private IP DNS name (IPv4 only):** ip-172-31-14-197.ap-south-1.compute.internal
- Instance type:** t2.micro
- VPC ID:** vpc-086db887a0cc74858
- Subnet ID:** subnet-0c7267d27d4a62be0
- Instance ARN:** arn:aws:ec2:ap-south-1:940482422578:instance/i-0caa76d6126ba3c8f
- Private IPv4 addresses:** 172.31.14.197
- Public IPv4 DNS:** -
- Elastic IP addresses:** -
- AWS Compute Optimizer finding:** Opt-in to AWS Compute Optimizer for recommendations. Learn more
- Auto Scaling Group name:** -
- Managed:** false

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

- Go to the **AWS IAM Console**, select **Roles**, and find the **flaskec2role**.
- Click **Attach Policies**, then choose the required policies (e.g., **DynamoDBFullAccess**, **SNSFullAccess**) and click **Attach Policy**.

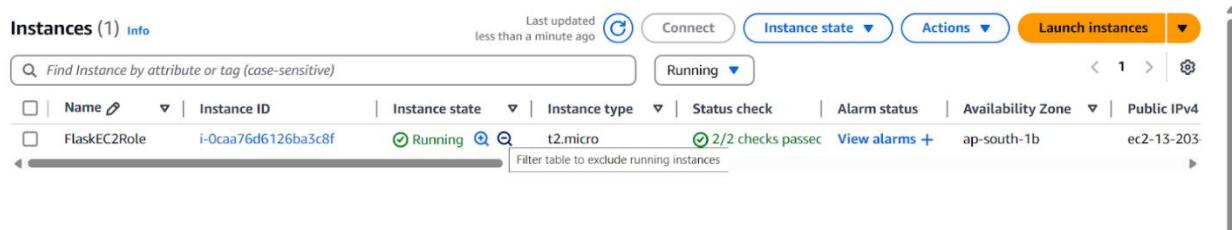
3. If needed, update the instance to use this modified role by selecting the EC2 instance, clicking **Actions**, then **Security**, and **Modify IAM role** to select the updated role.



The screenshot shows the 'Modify IAM role' dialog for an EC2 instance. At the top, it says 'Instance ID: i-0caa76d6126ba3c8f (FlaskEC2Role)'. Below that, under 'IAM role', there is a dropdown menu showing 'FlaskDynamoSNSRole' and a button to 'Create new IAM role'. At the bottom right are 'Cancel' and 'Update IAM role' buttons.

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.



The screenshot shows the EC2 Instances dashboard with one instance listed. The instance details are as follows:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
FlaskEC2Role	i-0caa76d6126ba3c8f	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1b	ec2-13-203

- Now connect the EC2 with the files

Student - Skill Wallet troven.in/students/labs/68033? troven.in/students/labs/68033? Instances | EC2 | us-east-1 +

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:

aws Search [Alt+S] United States (N. Virginia) rsosaccount-new/68034028d24717ccb9270727 @ rsosandboxnew51

EC2 Instances

Instances (1/1) info Last updated less than a minute ago

Find Instance by attribute or tag (case-sensitive)

All states

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
medtrack-server	i-0053359d5845bab97	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1d	ec2-184-

i-0053359d5845bab97 (medtrack-server)

Details Status and alarms Monitoring Security Networking Storage Tags

Instance summary

Instance ID: i-0053359d5845bab97 Public IPv4 address: 184.72.139.136 [open address]

IPv6 address: Instance state: Running Private IPv4 addresses: 172.31.24.105

Public DNS: ec2-184-72-139-136.compute-1.amazonaws.com [open address]

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 11:44 AM 7/4/2025

Student - Skill Wallet troven.in/students/labs/68033? troven.in/students/labs/68033? Instances | EC2 | us-east-1 EC2 Instance Connect | us-east-1 EC2 Instance Connect | us-east-1 +

us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?region=us-east-1&connType=standard&instanceId=i-0053359d5845bab97&osUser=ec2-user&sshPort=22&address...

aws Search [Alt+S] United States (N. Virginia) rsosaccount-new/68034028d24717ccb9270727 @ rsosandboxnew51

```

Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

Last login: Fri Jul  4 07:22:47 2025 from 18.206.107.28
[ec2-user@ip-172-31-24-105 ~]$ sudo su
[root@ip-172-31-24-105 ec2-user]# sudo su
[root@ip-172-31-24-105 ec2-user]# yum install python3
Last metadata expiration check: 0:13:29 ago on Fri Jul  4 07:25:33 2025.
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@ip-172-31-24-105 ec2-user]# yum install git -y
Last metadata expiration check: 0:13:44 ago on Fri Jul  4 07:25:33 2025.
Package git-2.47.1-1.amzn2023.0.3.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@ip-172-31-24-105 ec2-user]# yum install git -y
Last metadata expiration check: 0:14:08 ago on Fri Jul  4 07:25:33 2025.
Package git-2.47.1-1.amzn2023.0.3.x86_64 is already installed.

i-0053359d5845bab97 ( medtrack-server )
PublicIPs: 184.72.139.136 PrivateIPs: 172.31.24.105
```

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 1:30 PM 7/4/2025

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/paletilikhitha/Medtrack.git'

- This will download your project to the EC2 instance.

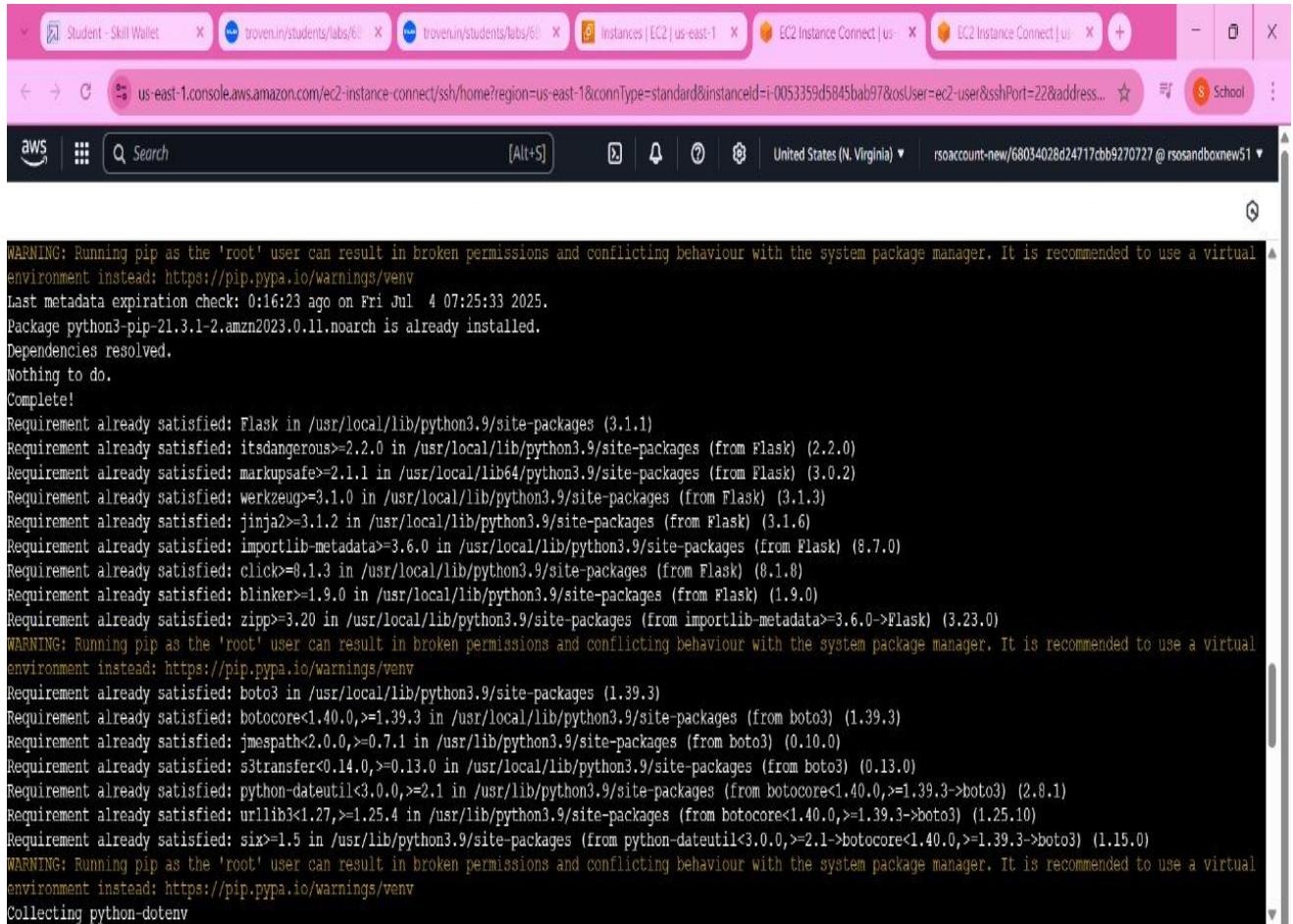
To navigate to the project directory, run the following command:

```
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
```



```

WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Last metadata expiration check: 0:16:23 ago on Fri Jul 4 07:25:33 2025.
Package python3-pip-21.3.1-2.amzn2023.0.11.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
Requirement already satisfied: Flask in /usr/local/lib/python3.9/site-packages (3.1.1)
Requirement already satisfied: itsdangerous>=2.2.0 in /usr/local/lib/python3.9/site-packages (from Flask) (2.2.0)
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib64/python3.9/site-packages (from Flask) (3.0.2)
Requirement already satisfied: werkzeug>=3.1.0 in /usr/local/lib/python3.9/site-packages (from Flask) (3.1.3)
Requirement already satisfied: jinja2>=3.1.2 in /usr/local/lib/python3.9/site-packages (from Flask) (3.1.6)
Requirement already satisfied: importlib-metadata>=3.6.0 in /usr/local/lib/python3.9/site-packages (from Flask) (3.7.0)
Requirement already satisfied: click>=0.1.3 in /usr/local/lib/python3.9/site-packages (from Flask) (0.1.8)
Requirement already satisfied: blinker>=1.9.0 in /usr/local/lib/python3.9/site-packages (from Flask) (1.9.0)
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.9/site-packages (from importlib-metadata>=3.6.0->Flask) (3.23.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Requirement already satisfied: boto3 in /usr/local/lib/python3.9/site-packages (1.39.3)
Requirement already satisfied: botocore<1.40.0,>=1.39.3 in /usr/local/lib/python3.9/site-packages (from boto3) (1.39.3)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Requirement already satisfied: s3transfer<0.14.0,>=0.13.0 in /usr/local/lib/python3.9/site-packages (from boto3) (0.13.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3->boto3) (2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3->boto3) (1.25.10)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.3->boto3) (1.15.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Collecting python-dotenv

```

i-0053359d5845bab97 (medtrack-server)

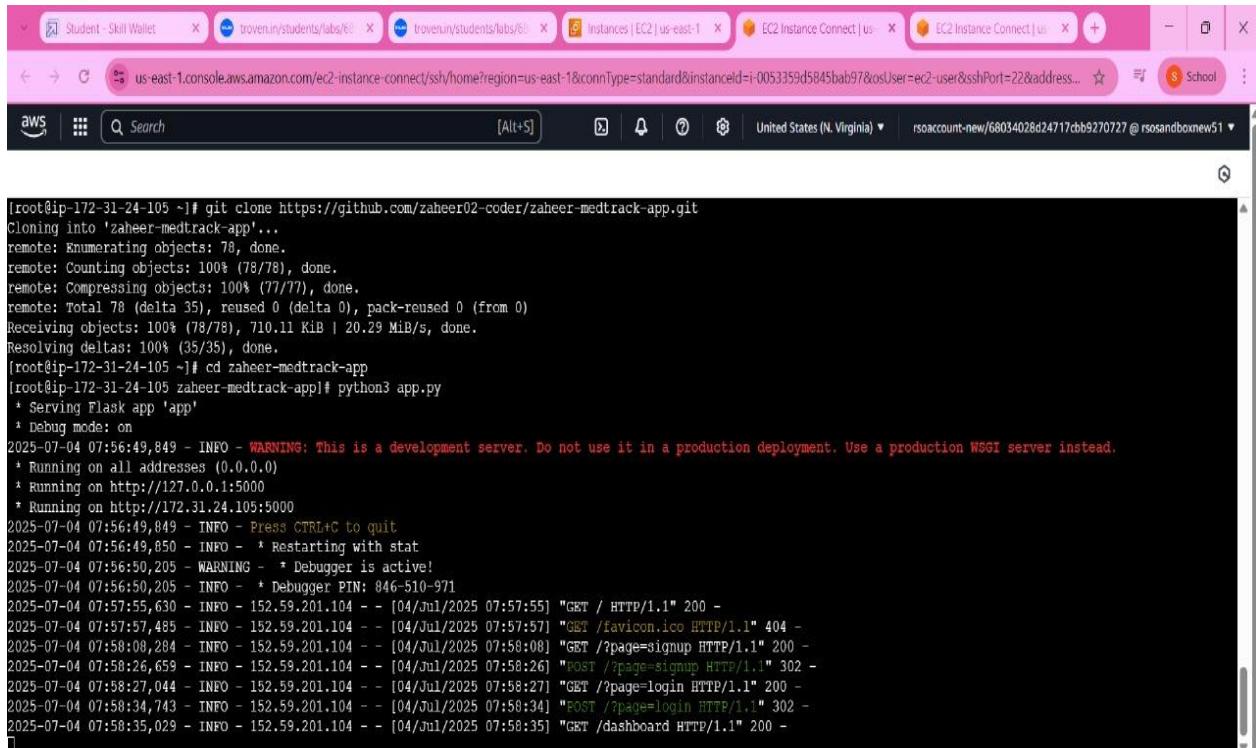
Public IPs: 184.72.139.136 Private IPs: 172.31.24.105



Verify the Flask app is running:

<http://your-ec2-public-ip>

- o Run the Flask app on the EC2 instance



```
[root@ip-172-31-24-105 ~]# git clone https://github.com/zaheer02-coder/zaheer-medtrack-app.git
Cloning into 'zaheer-medtrack-app'...
remote: Enumerating objects: 78, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (77/77), done.
remote: Total 78 (delta 35), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (78/78), 710.11 KiB | 20.29 MiB/s, done.
Resolving deltas: 100% (35/35), done.
[root@ip-172-31-24-105 ~]# cd zaheer-medtrack-app
[root@ip-172-31-24-105 zaheer-medtrack-app]# python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
2025-07-04 07:56:49,849 - INFO - 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://127.0.0.1:5000
 * Running on http://172.31.24.105:5000
2025-07-04 07:56:49,849 - INFO - Press CTRL+C to quit
2025-07-04 07:56:49,850 - INFO - * Restarting with stat
2025-07-04 07:56:50,205 - WARNING - * Debugger is active!
2025-07-04 07:56:50,205 - INFO - * Debugger PIN: 846-510-971
2025-07-04 07:57:55,630 - INFO - 152.59.201.104 - - [04/Jul/2025 07:57:55] "GET / HTTP/1.1" 200 -
2025-07-04 07:57:57,485 - INFO - 152.59.201.104 - - [04/Jul/2025 07:57:57] "GET /favicon.ico HTTP/1.1" 404 -
2025-07-04 07:58:08,284 - INFO - 152.59.201.104 - - [04/Jul/2025 07:58:08] "GET /?page=signup HTTP/1.1" 200 -
2025-07-04 07:58:26,659 - INFO - 152.59.201.104 - - [04/Jul/2025 07:58:26] "POST /?page=signup HTTP/1.1" 302 -
2025-07-04 07:58:27,044 - INFO - 152.59.201.104 - - [04/Jul/2025 07:58:27] "GET /?page=login HTTP/1.1" 200 -
2025-07-04 07:58:34,743 - INFO - 152.59.201.104 - - [04/Jul/2025 07:58:34] "POST /?page=login HTTP/1.1" 302 -
2025-07-04 07:58:35,029 - INFO - 152.59.201.104 - - [04/Jul/2025 07:58:35] "GET /dashboard HTTP/1.1" 200 -

```

i-0053359d5845bab97 (medtrack-server)

PublicIPs: 184.72.139.136 PrivateIPs: 172.31.24.105



Access the website through:

Public IPs: <http://172.18.160.204:5000>

Milestone 8: Testing and Deployment

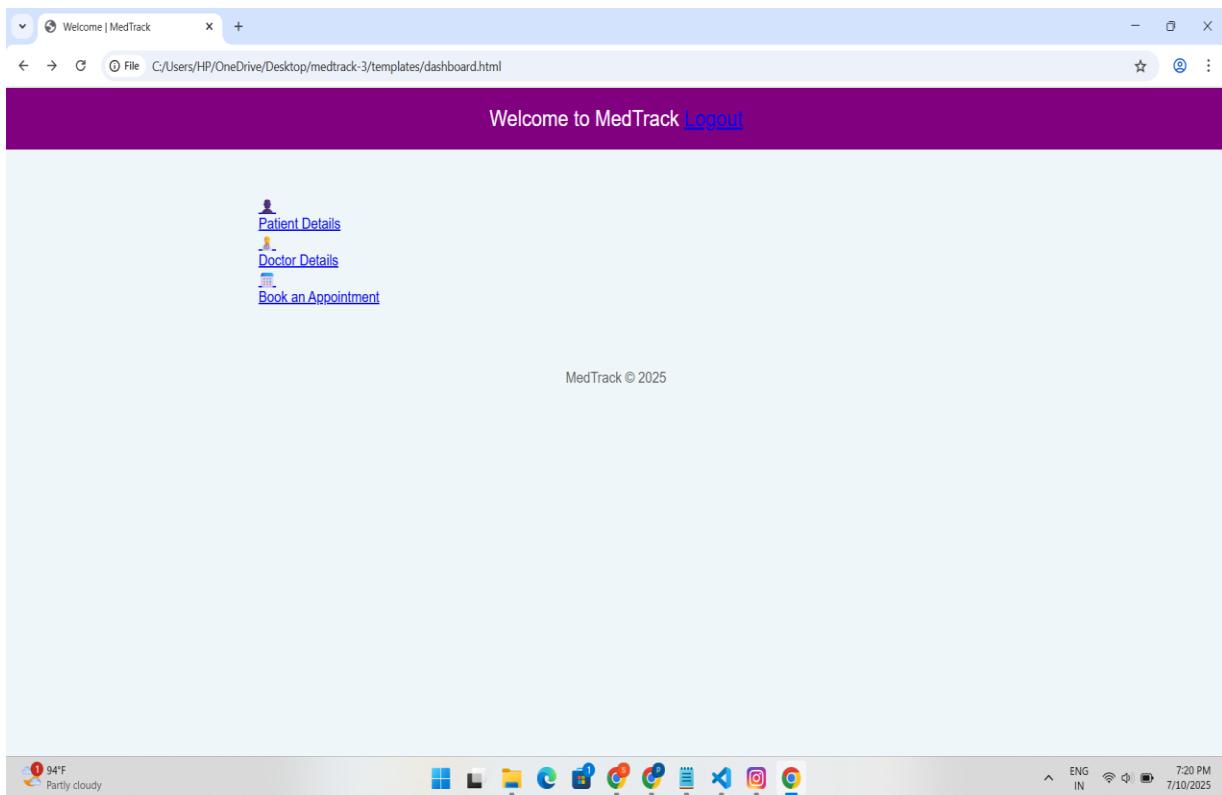
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and 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.

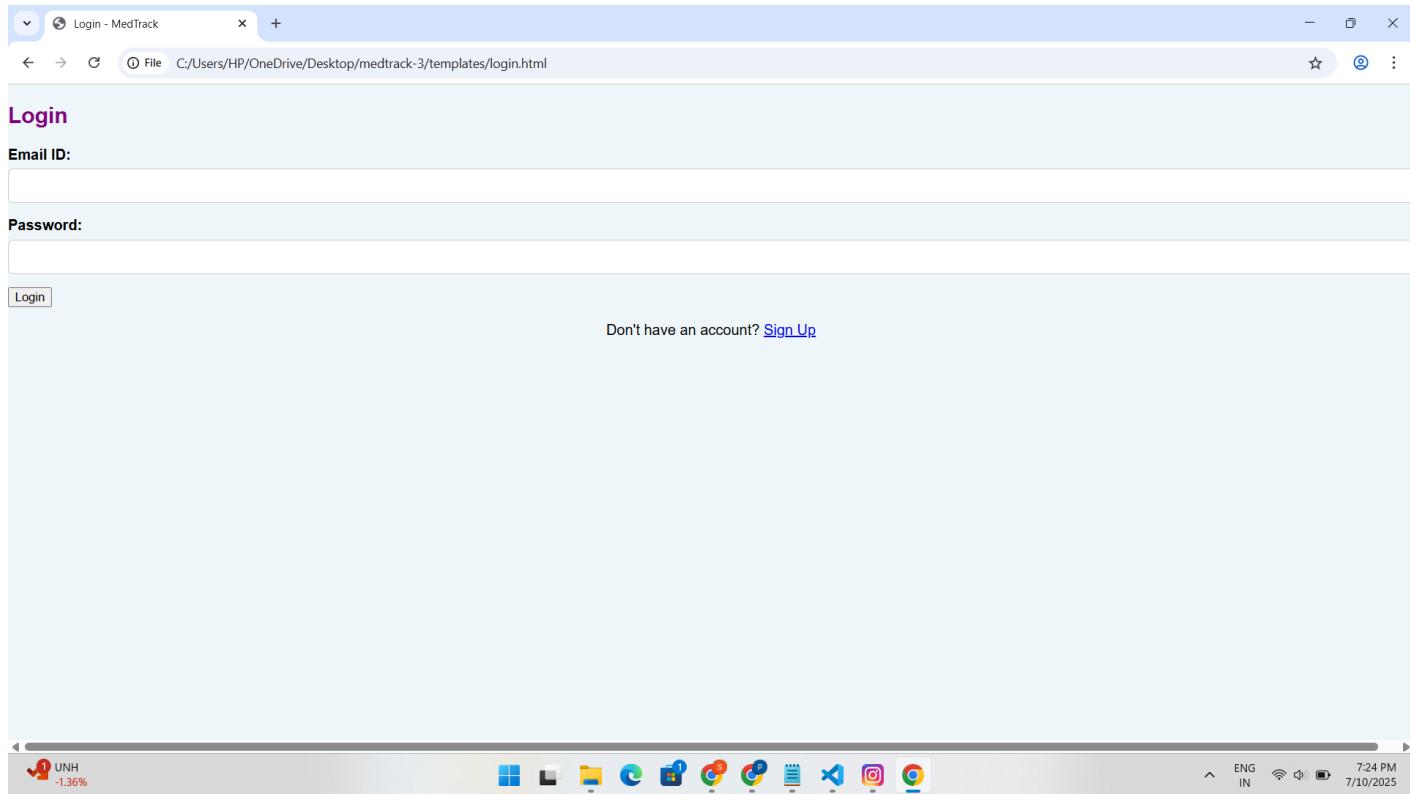


PATIENT LOGIN PAGE:

The Patient Login Page allows 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.

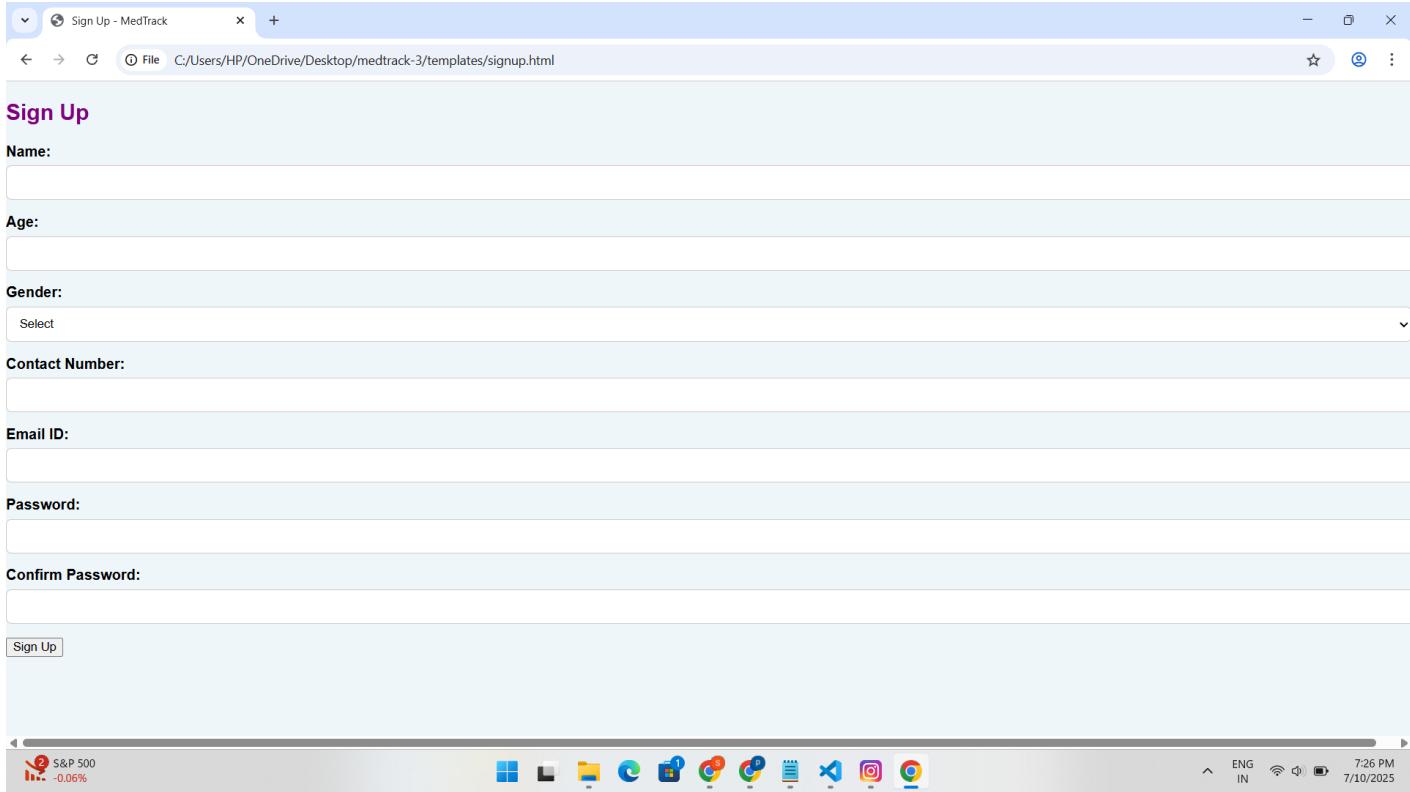
Once logged in, patients are redirected to their respective dashboards to manage appointments, medical records, and other relevant tasks.



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.



The screenshot shows a web browser window with the title "Sign Up - MedTrack". The address bar displays the file path "C:/Users/HP/OneDrive/Desktop/medtrack-3/templates/signup.html". The main content area contains a "Sign Up" form with the following fields:

- Name: [Text input field]
- Age: [Text input field]
- Gender: [Select dropdown menu] (options: Select, Male, Female)
- Contact Number: [Text input field]
- Email ID: [Text input field]
- Password: [Text input field]
- Confirm Password: [Text input field]

A "Sign Up" button is located at the bottom left of the form. The browser interface includes standard navigation buttons (back, forward, search), a toolbar, and a status bar at the bottom showing system information like battery level, signal strength, and the date/time (7/10/2025).

Patient Details Form

File C:/Users/HP/OneDrive/Desktop/medtrack-3/templates/patientform.html

Patient Details Form

Age

Gender

Contact Number

Address

Blood Group

Medical History

Submit

DynamoDB Database updati

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.

Table: UsersTable - Items returned (4)

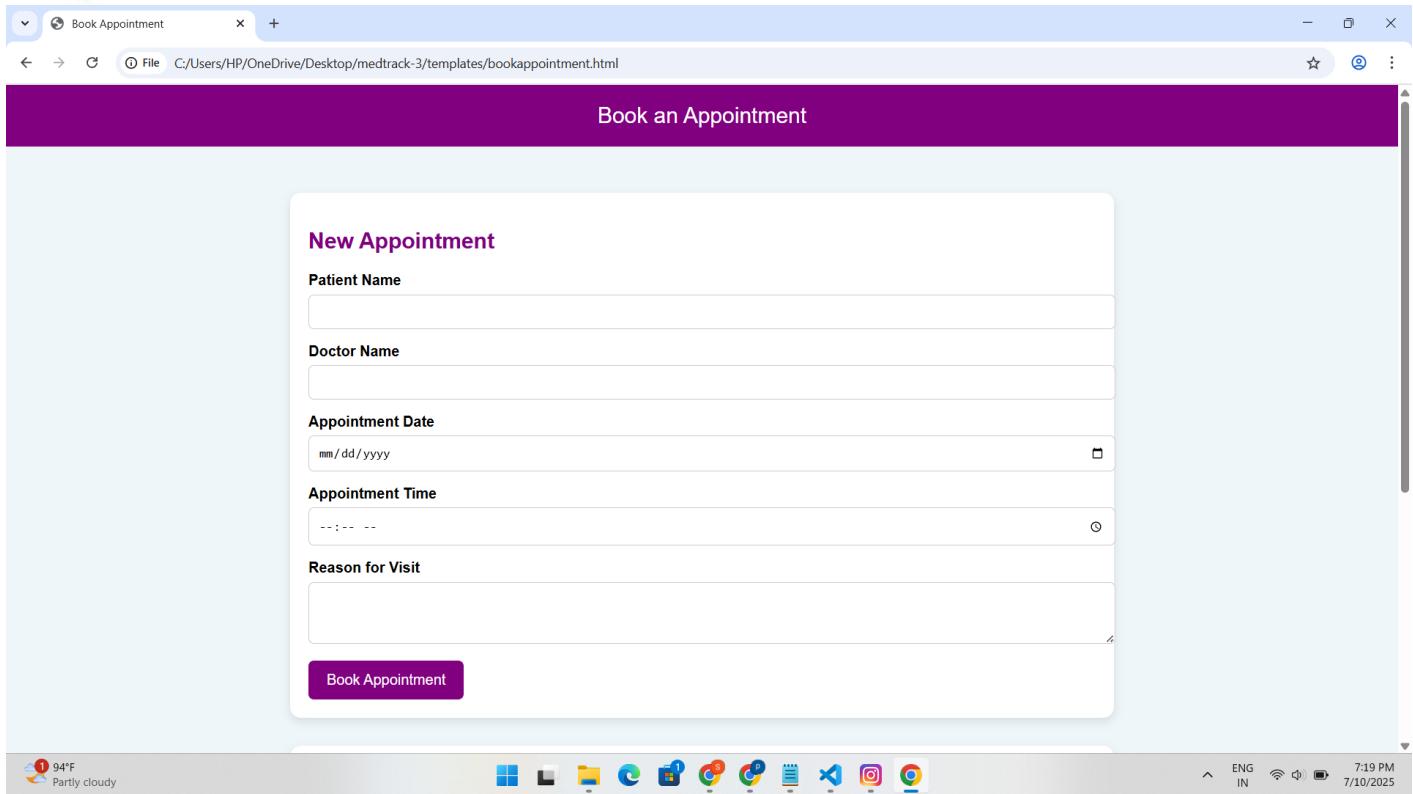
Scan started on April 18, 2025, 19:27:34

 Actions ▾ Create item

< 1 > | 

<input type="checkbox"/>	email (String)	age	created_at	gender	login_count	name	
<input type="checkbox"/>	sunder@thesmartbrid...	25	2025-04-18...	male	2	sunder	
<input type="checkbox"/>	gtharunasri19@gmail...	21	2025-04-18...	female	1	 	tharuna
<input type="checkbox"/>	sairaviteja478@gmail....	21	2025-04-18...	male	2		Kambhan
<input type="checkbox"/>	siri@thesmartbridge.c...	24	2025-04-18...	female	1		siri

Appointment confirmation:



The screenshot shows a web browser window with the title "Book Appointment". The URL in the address bar is "C:/Users/HP/Desktop/medtrack-3/templates/bookappointment.html". The main content is a form titled "New Appointment" with the following fields:

- Patient Name: An input field.
- Doctor Name: An input field.
- Appointment Date: A date input field with placeholder "mm / dd / yyyy".
- Appointment Time: A time input field with placeholder "... : -- : --".
- Reason for Visit: A large text area for notes.

At the bottom of the form is a purple "Book Appointment" button.

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.