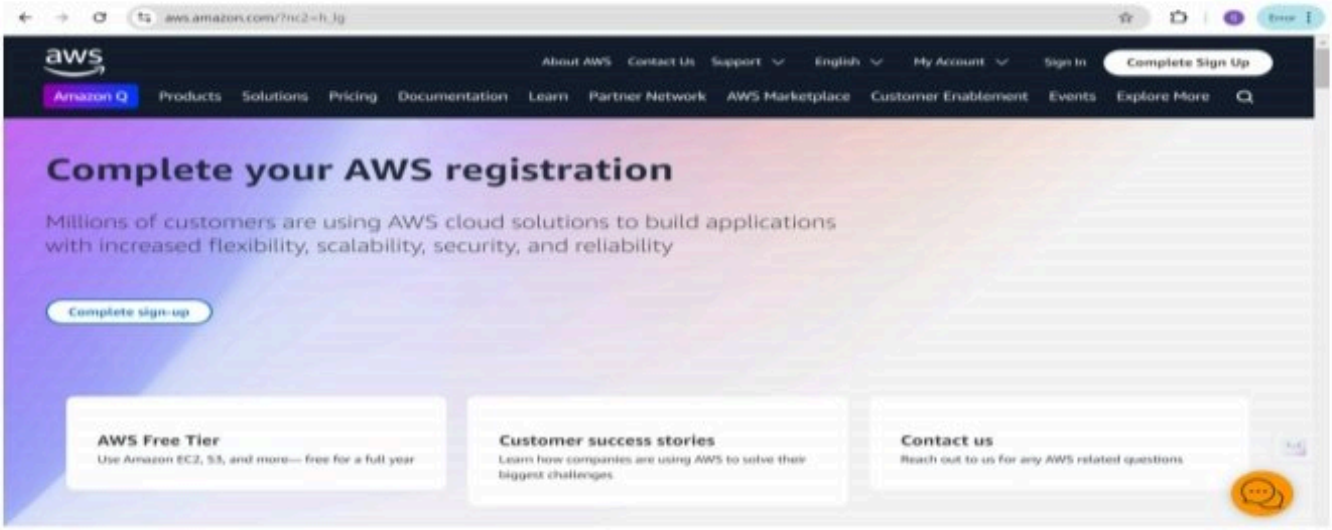
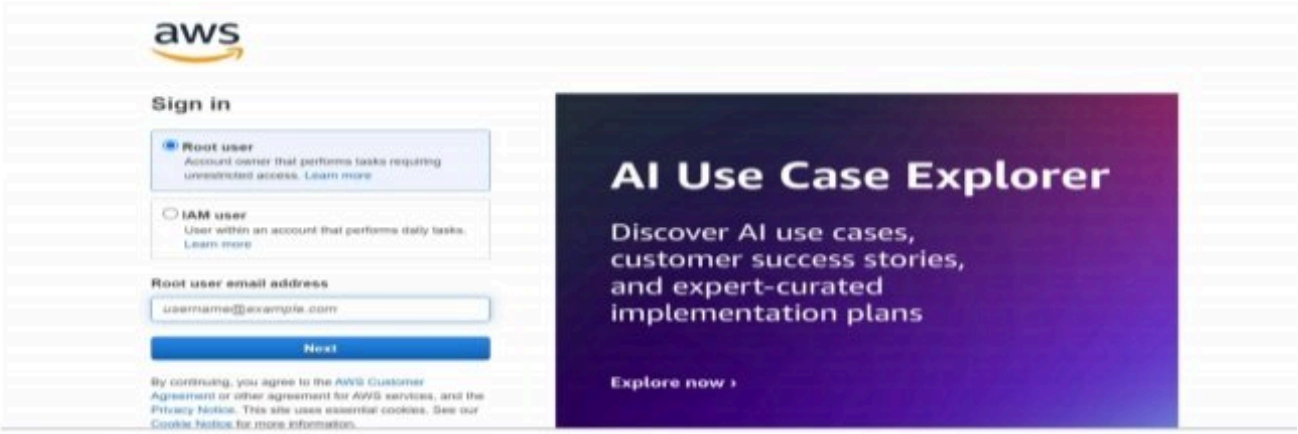
Milestone 1: AWS Account Setup and Login

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

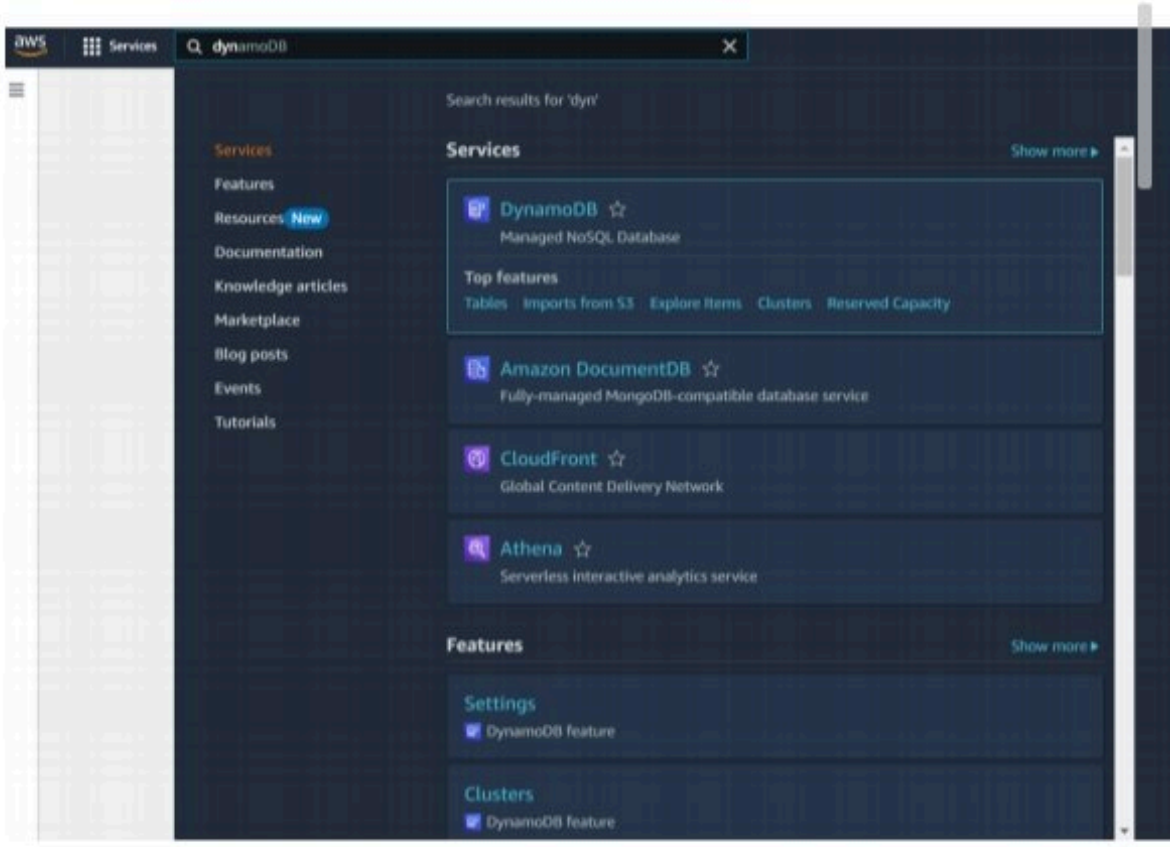


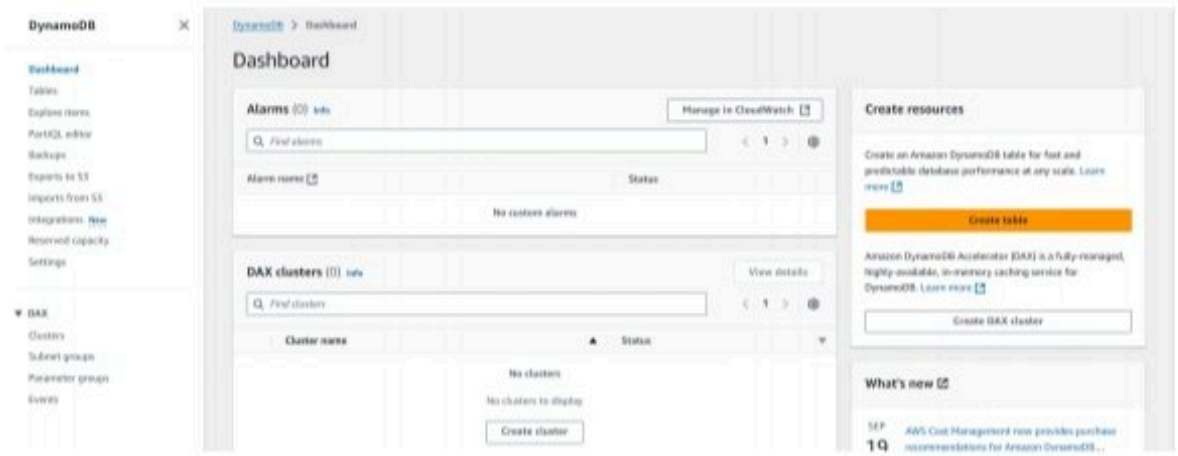
- **Activity 1.2: Log in to the AWS Management Console**
  - After setting up your account, log in to the AWS Management Console.



Milestone 2: DynamoDB Database Creation and Setup

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





- **Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**
    - Create Users table with partition key "Email" with type String and click on create tables.

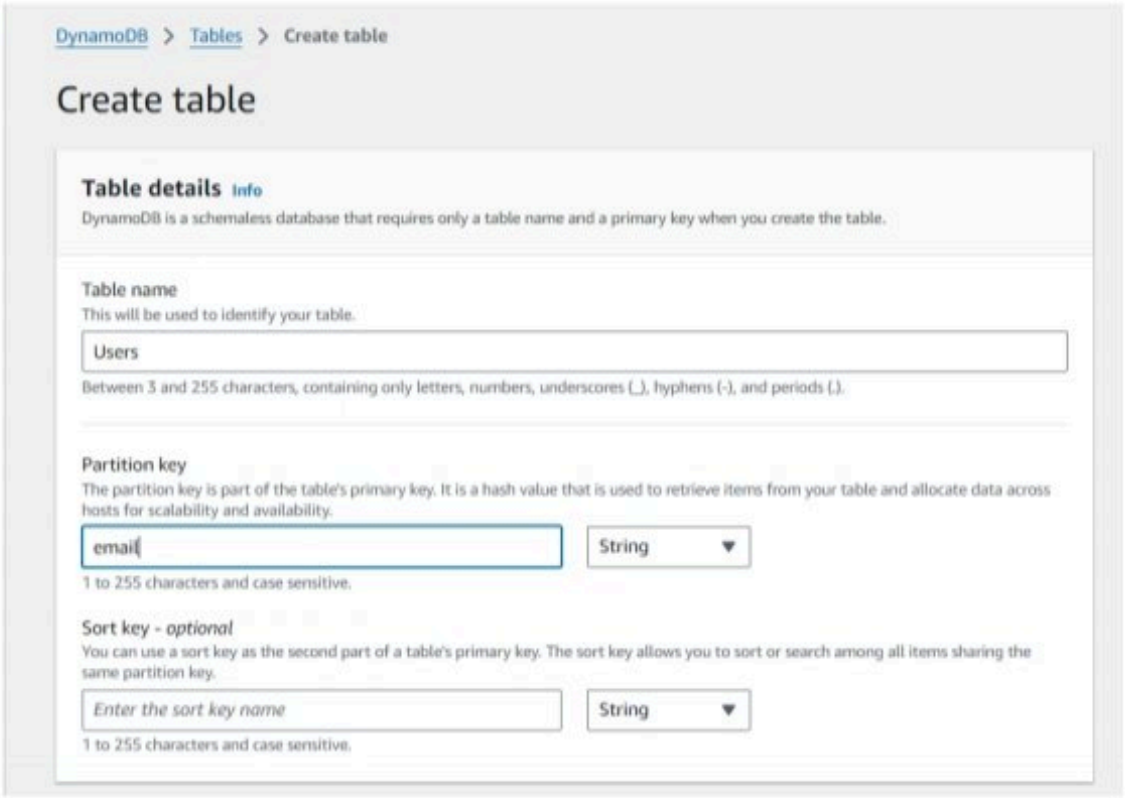


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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

Tags

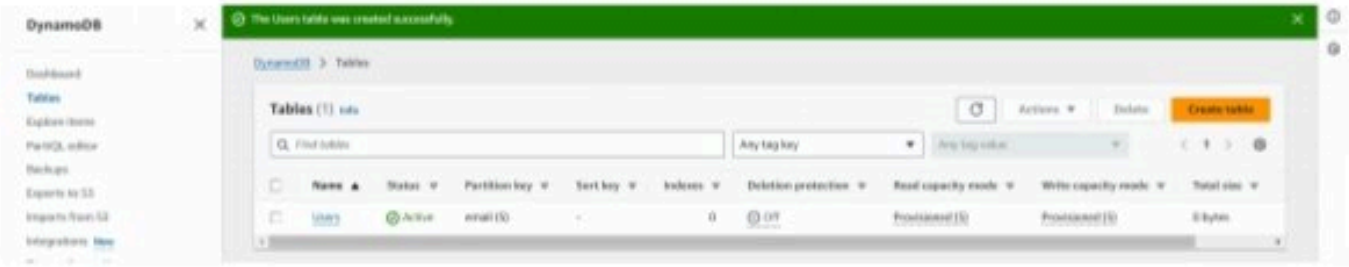
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

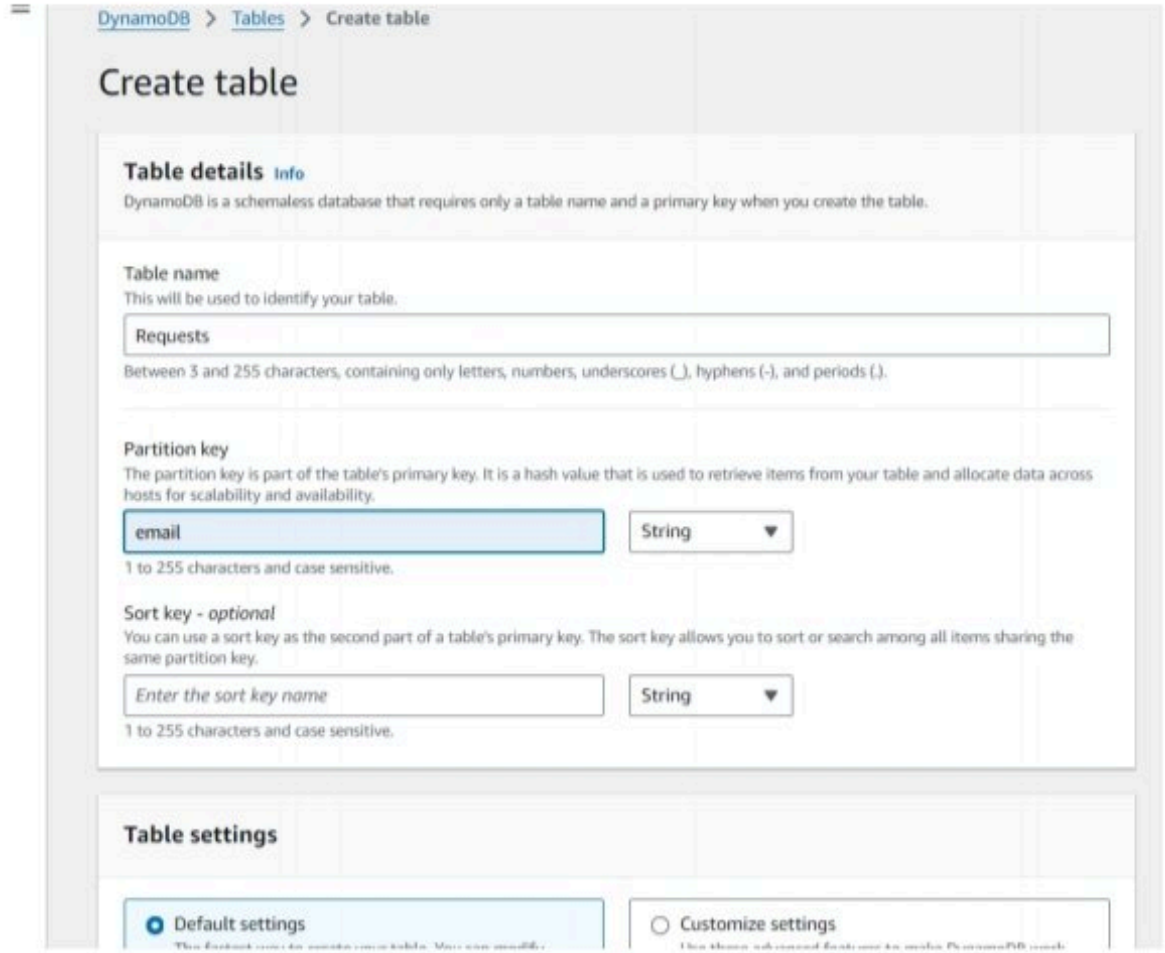
Add new tag

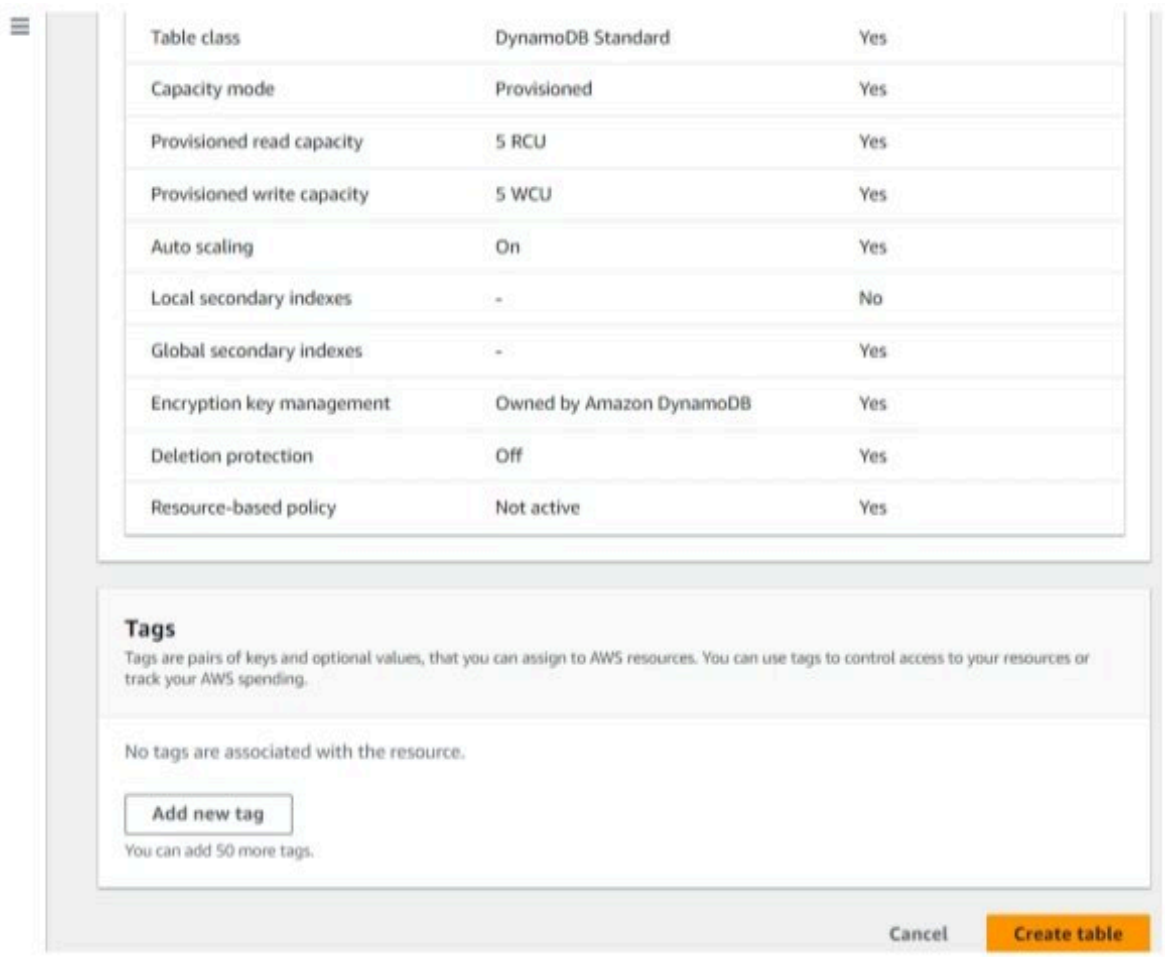
You can add 50 more tags.

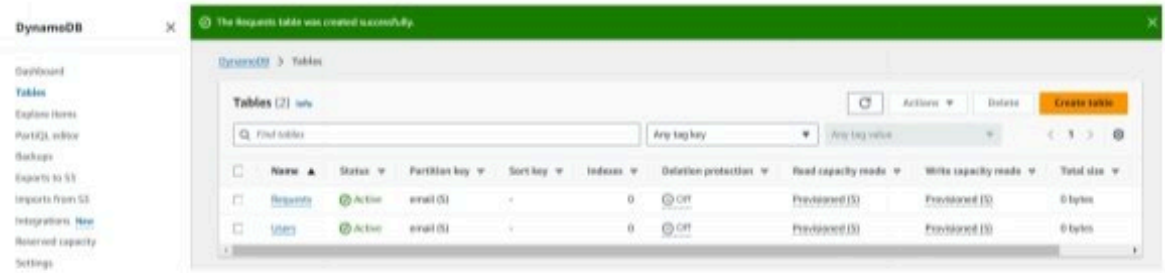
Cancel Create table



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



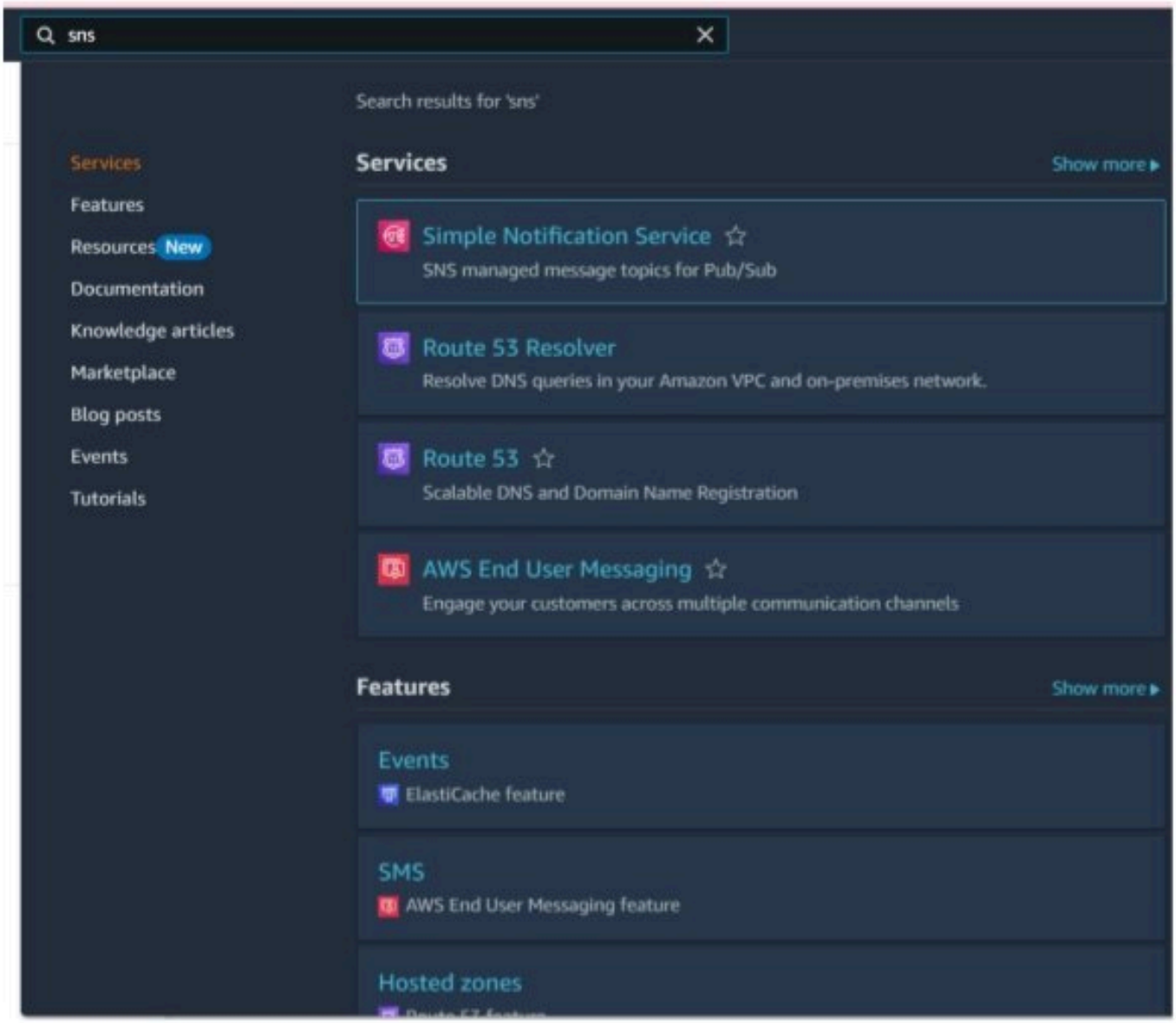


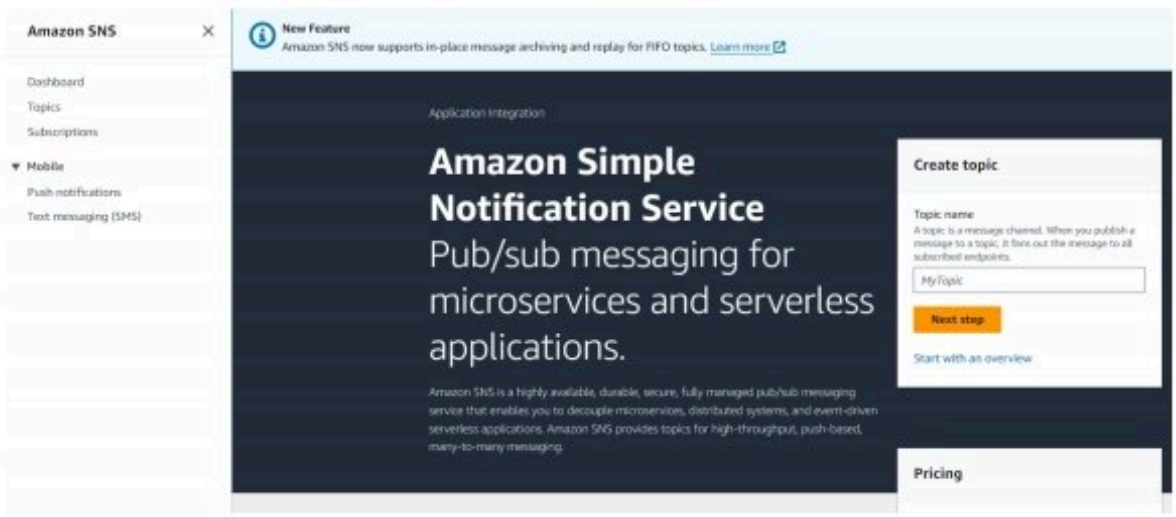


Milestone 3: SNS Notification Setup

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

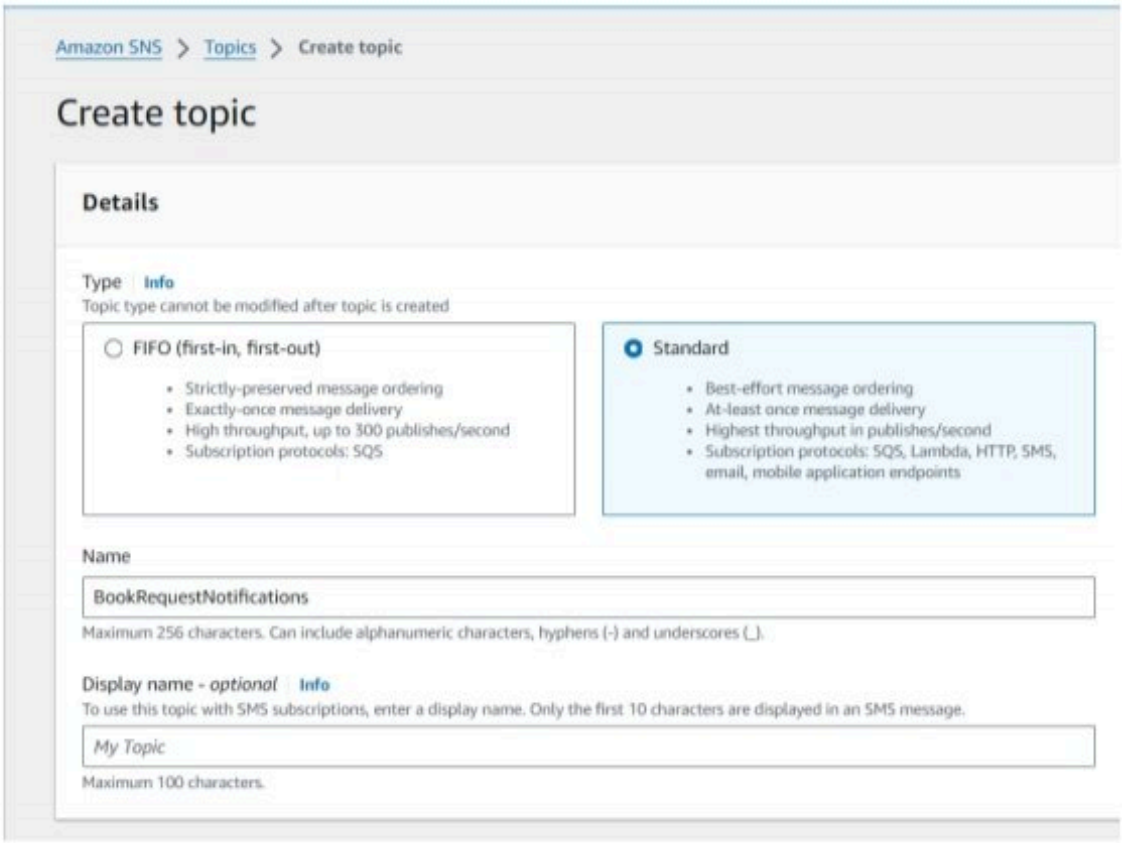
- In the AWS Console, search for SNS and navigate to the SNS Dashboard.

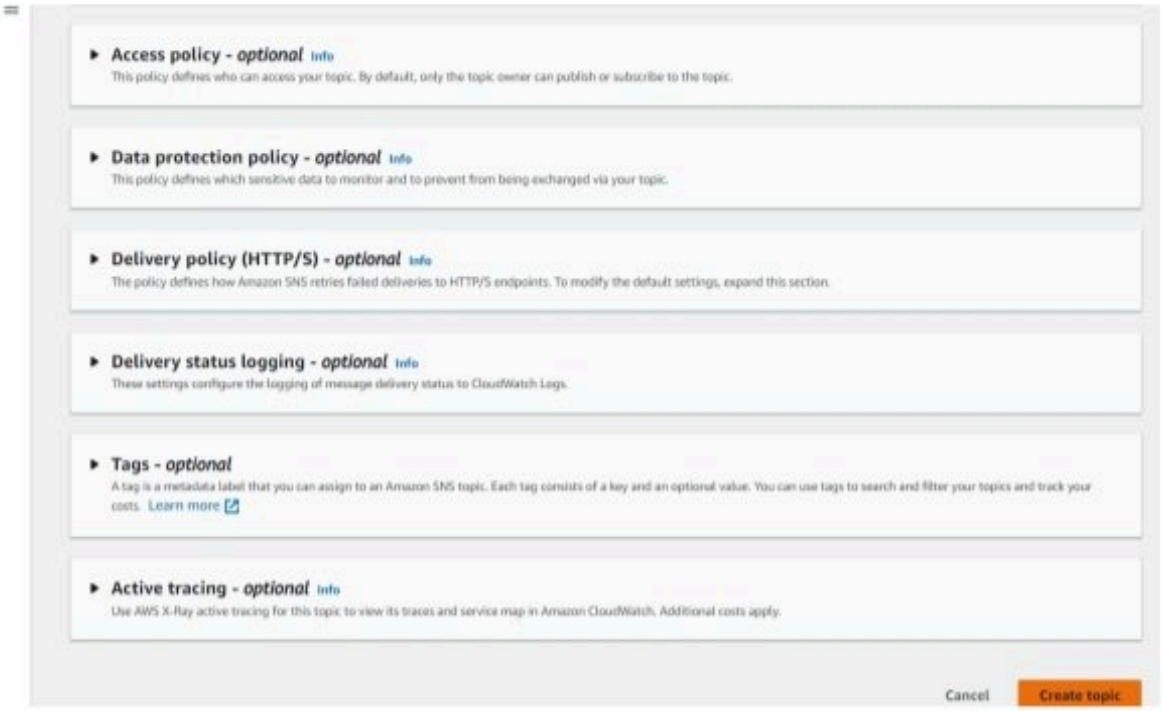




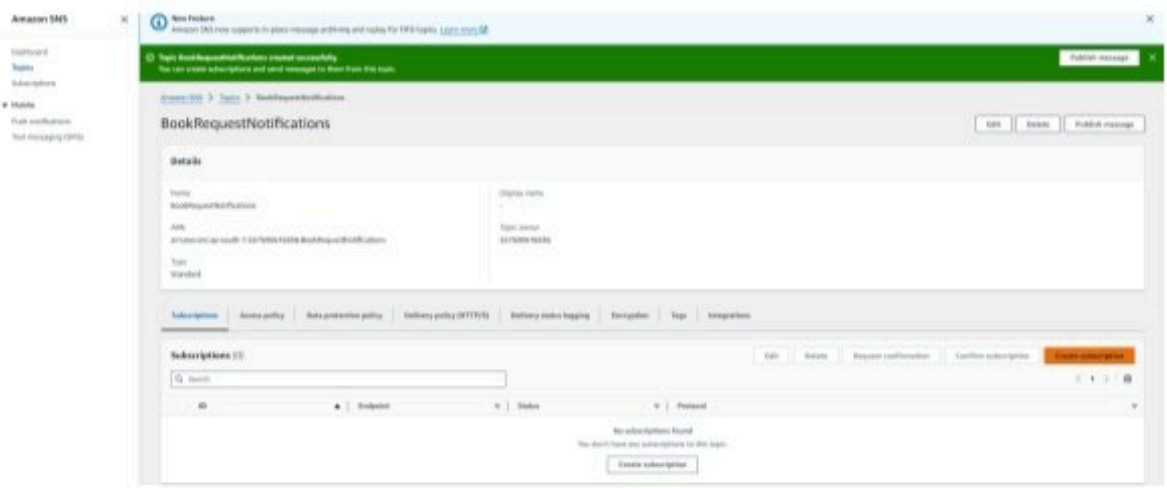
- Click on **Create Topic** and choose a name for the topic.

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



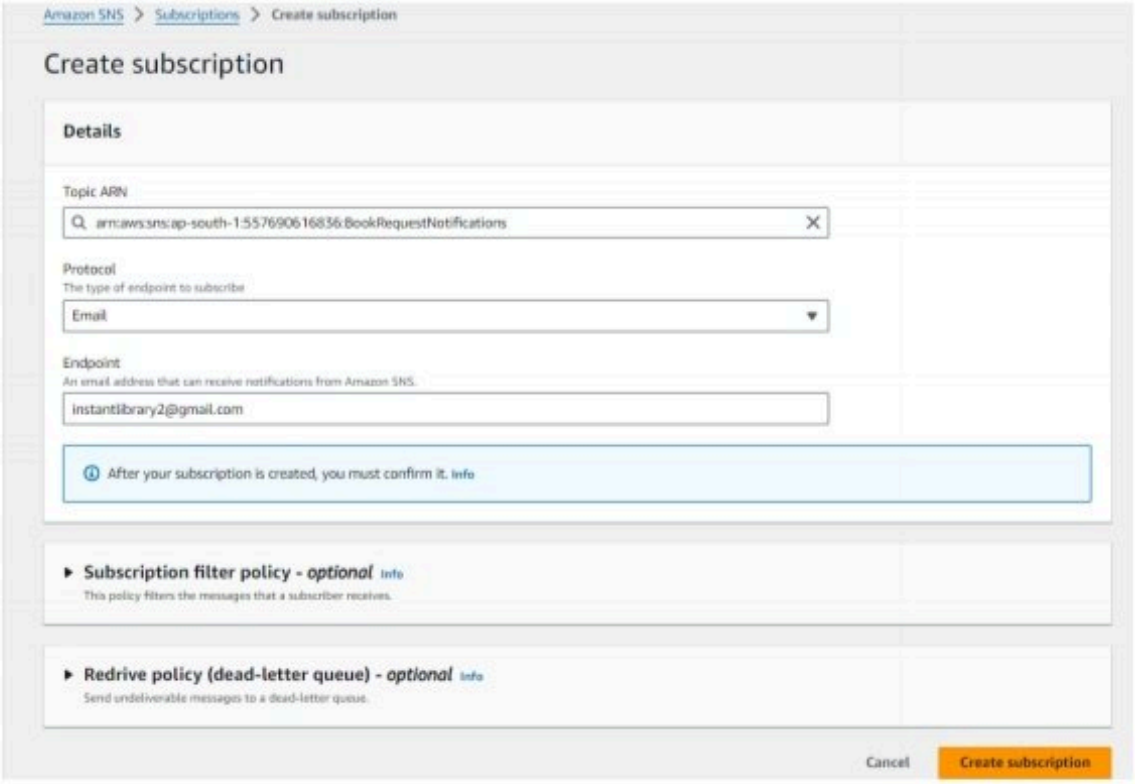


- Configure the SNS topic and note down the **Topic ARN**.

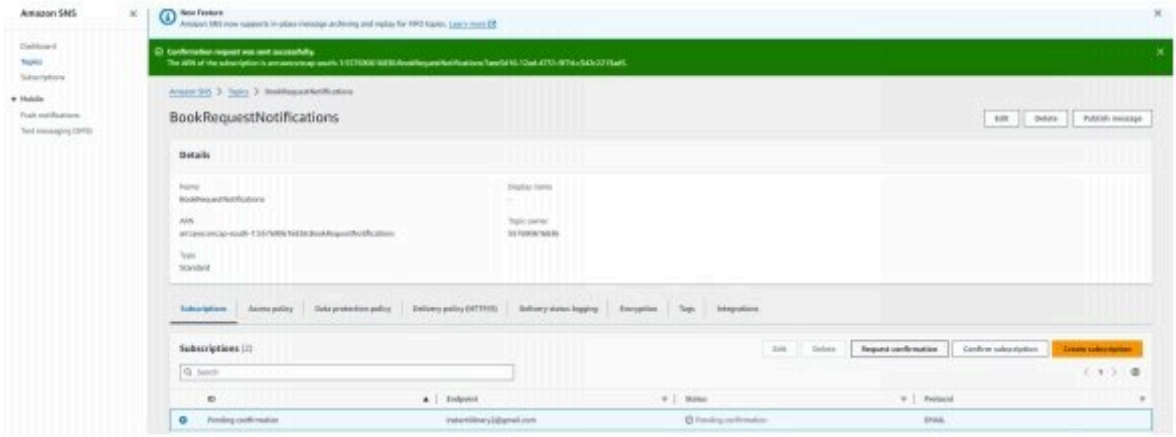


- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

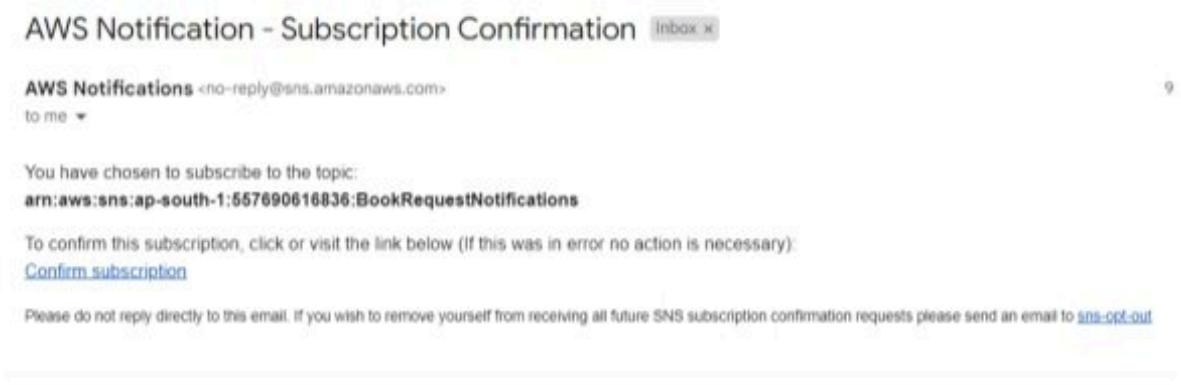
    - Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.

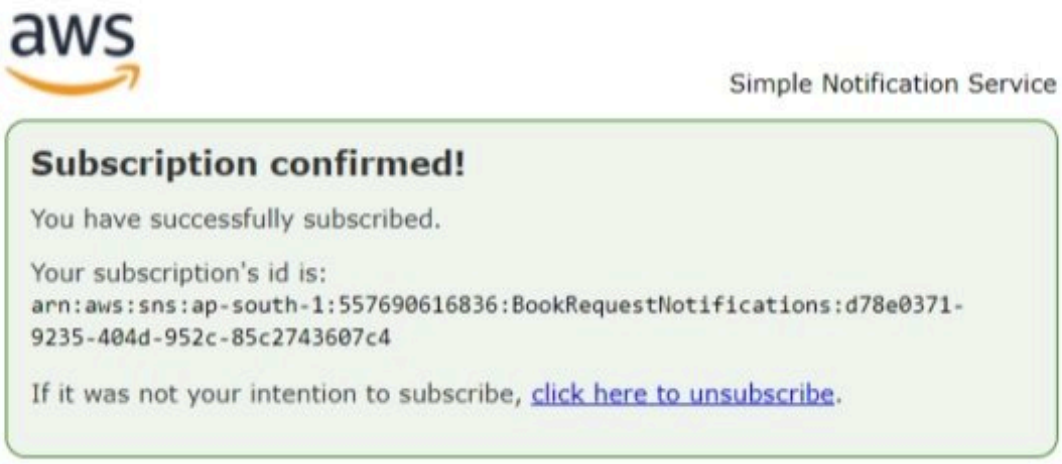


- After subscription request for the mail confirmation



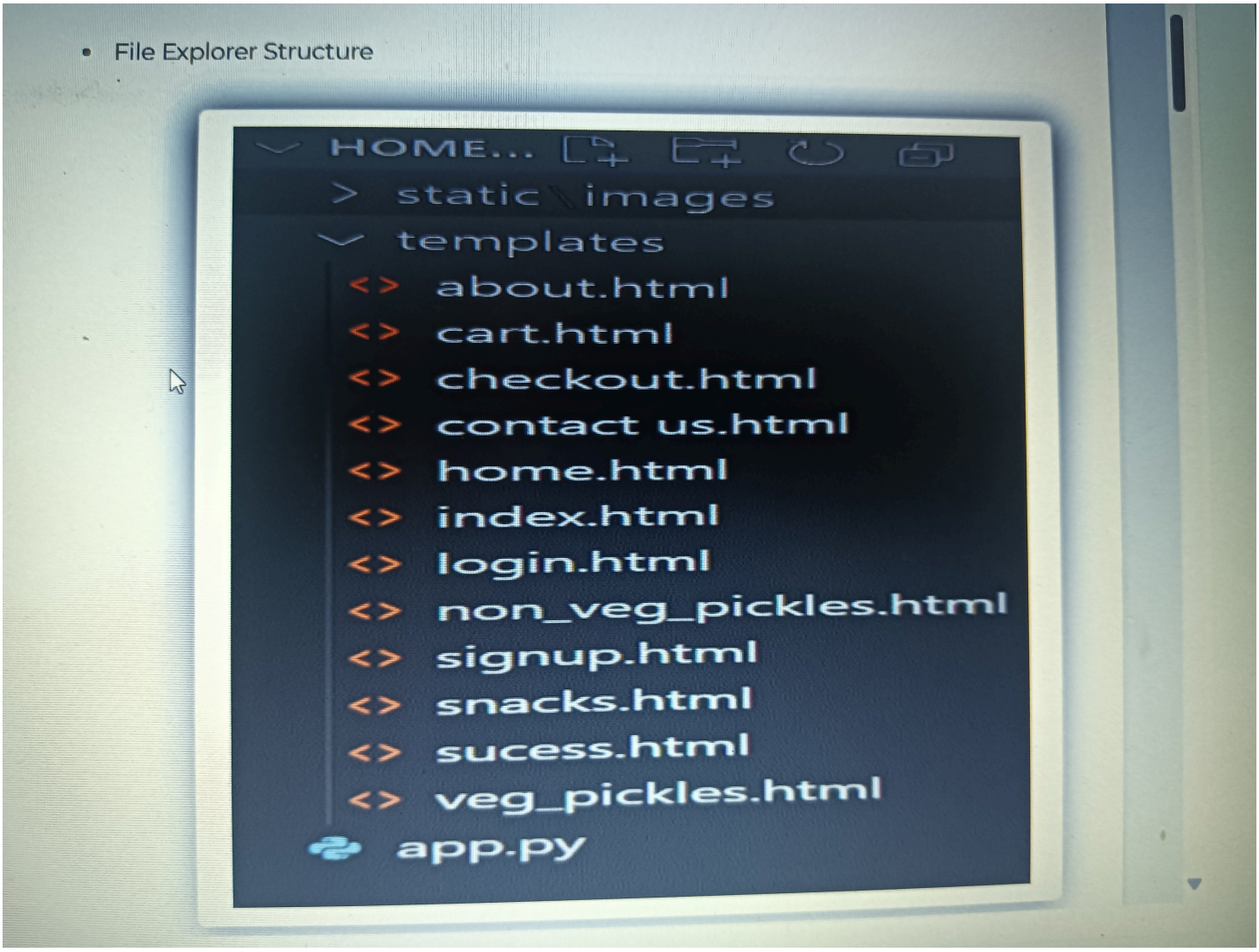
- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.





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

- File Explorer Structure



Description of the code :

? Flask App Initialization

```
from flask import Flask, render_template, request, redirect, url_for, session
from werkzeug.security import generate_password_hash, check_password_hash
import boto3
from datetime import datetime
import json,uuid
```

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

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

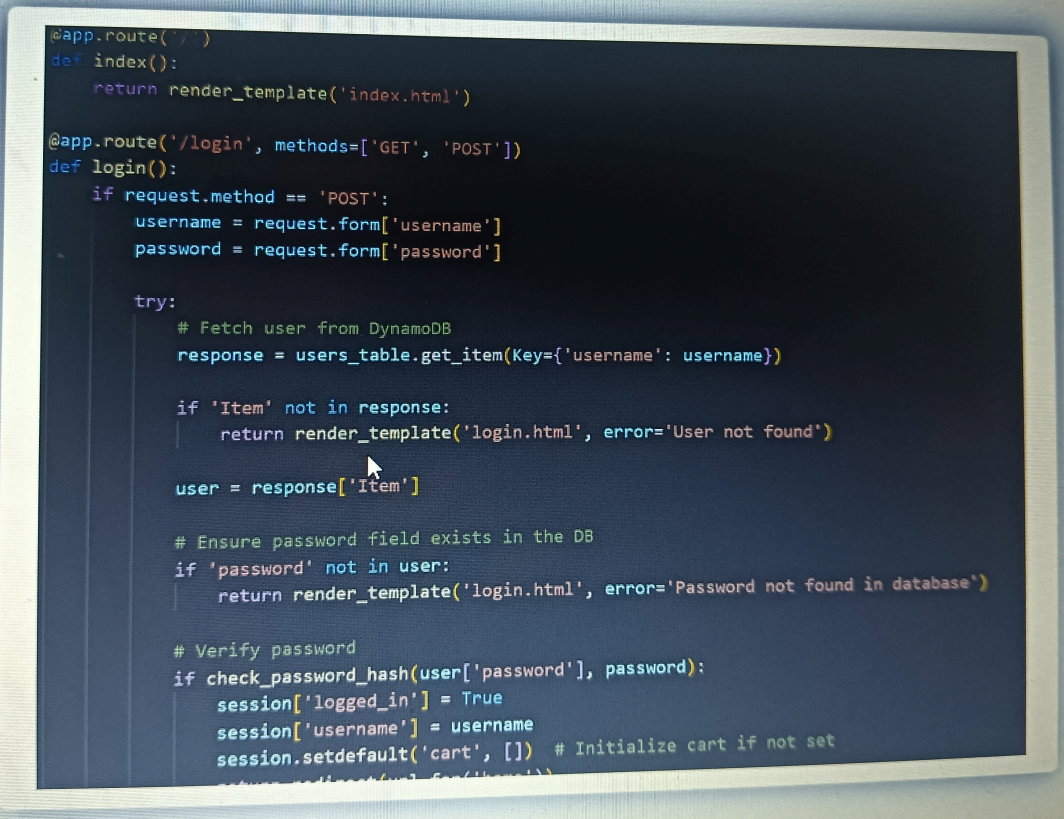
```
dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')  # e.g., 'us-east-1'
```

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

```python
dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')
users_table = dynamodb.Table('Users')
orders_table = dynamodb.Table('Orders')
```

```python
products = {
    'non_veg_pickles': [
        {'id': 1, 'name': 'Chicken Pickle', 'weights': {'250': 600, '500': 1200, '1000': 1800}},
        {'id': 2, 'name': 'Fish Pickle', 'weights': {'250': 200, '500': 400, '1000': 800}},
        {'id': 3, 'name': 'Gongura Mutton', 'weights': {'250': 400, '500': 800, '1000': 1600}},
        {'id': 4, 'name': 'Mutton Pickle', 'weights': {'250': 400, '500': 800, '1000': 1600}},
        {'id': 5, 'name': 'Gongura Prawns', 'weights': {'250': 600, '500': 1200, '1000': 1800}},
        {'id': 6, 'name': 'Chicken Pickle (Gongura)', 'weights': {'250': 350, '500': 700, '1000': 1050}}
    ],
    'veg_pickles': [
        {'id': 7, 'name': 'Traditional Mango Pickle', 'weights': {'250': 150, '500': 280, '1000': 500}},
        {'id': 8, 'name': 'Zesty Lemon Pickle', 'weights': {'250': 120, '500': 220, '1000': 400}},
        {'id': 9, 'name': 'Tomato Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 10, 'name': 'Kakarakaya Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 11, 'name': 'Chintakaya Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 12, 'name': 'Spicy Pandu Mirchi', 'weights': {'250': 130, '500': 240, '1000': 450}}
    ],  # Add your veg pickle products here
    'snacks': [
        {'id': 7, 'name': 'Banana Chips', 'weights': {'250': 300, '500': 600, '1000': 800}},
        {'id': 8, 'name': 'Crispy Aam-Papad', 'weights': {'250': 150, '500': 300, '1000': 600}},
        {'id': 9, 'name': 'Crispy Chekka Pakodi', 'weights': {'250': 50, '500': 100, '1000': 200}},
        {'id': 10, 'name': 'Boondhi Acchu', 'weights': {'250': 300, '500': 600, '1000': 900}},
        {'id': 11, 'name': 'Chekkalu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 12, 'name': 'Ragi Laddu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 13, 'name': 'Dry Fruit Laddu', 'weights': {'250': 500, '500': 1000, '1000': 1500}},
        {'id': 14, 'name': 'Kara Boondi', 'weights': {'250': 250, '500': 500, '1000': 750}},
        {'id': 15, 'name': 'Gavvalu', 'weights': {'250': 250, '500': 500, '1000': 750}},
        {'id': 16, 'name': 'Kaju Chikki', 'weights': {'250': 250, '500': 500, '1000': 750}},
```

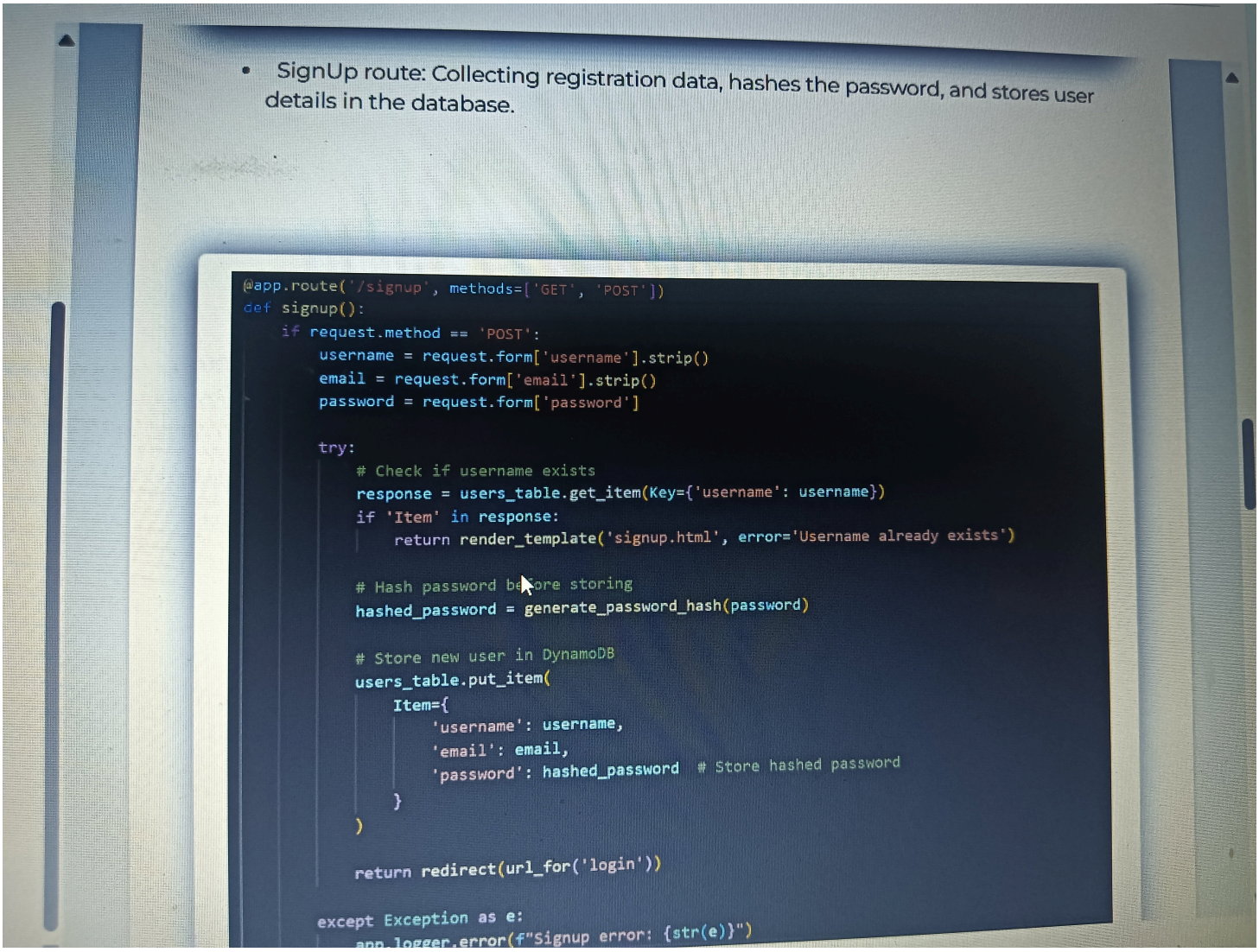
- Routes for Web Pages
- Login Route (GET/POST):Verifies user credentials, increments login count, and redirects to the dashboard on success.



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

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        try:
            # Fetch user from DynamoDB
            response = users_table.get_item(Key={'username': username})

            if 'Item' not in response:
                return render_template('login.html', error='User not found')

            user = response['Item']

            # Ensure password field exists in the DB
            if 'password' not in user:
                return render_template('login.html', error='Password not found in database')

            # Verify password
            if check_password_hash(user['password'], password):
                session['logged_in'] = True
                session['username'] = username
                session.setdefault('cart', [])  # Initialize cart if not set
```

...registration data, hashes the password, and stores user

- SignUp route: Collecting registration data, hashes the password, and stores user details in the database.



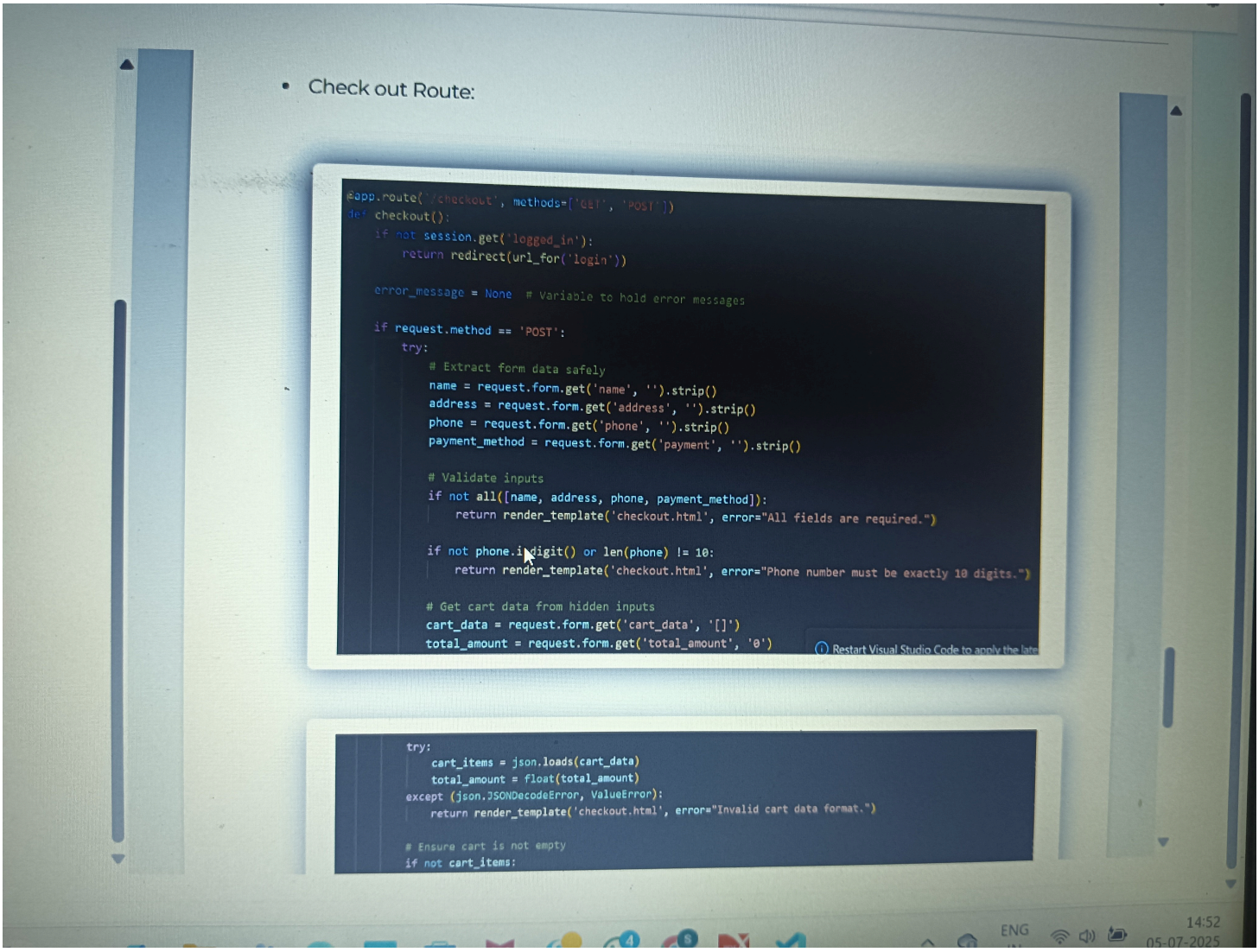
```python
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username'].strip()
        email = request.form['email'].strip()
        password = request.form['password']

        try:
            # Check if username exists
            response = users_table.get_item(Key={'username': username})
            if 'Item' in response:
                return render_template('signup.html', error='Username already exists')

            # Hash password before storing
            hashed_password = generate_password_hash(password)

            # Store new user in DynamoDB
            users_table.put_item(
                Item={
                    'username': username,
                    'email': email,
                    'password': hashed_password  # Store hashed password
                }
            )

            return redirect(url_for('login'))

        except Exception as e:
            app.logger.error(f"Signup error: {str(e)}")
```

- Home Route: Home page contains the routing for different categories which are Veg_pickles,Non_Veg_pickles,Snacks.

```python
@app.route('/home')
def home():
    if not session.get('logged_in'):
        return redirect(url_for('login'))
    return render_template('home.html')

@app.route('/non_veg_pickles')
def non_veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    return render_template('non_veg_pickles.html', products=products['non_veg_pickles'])

@app.route('/veg_pickles')
def veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    # Simply pass all products without filtering
    return render_template('veg_pickles.html', products=products['veg_pickles'])

@app.route('/snacks')
def snacks():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    return render_template('snacks.html', products=products['snacks'])
```

Restart Visual Studio

- Check out Route:



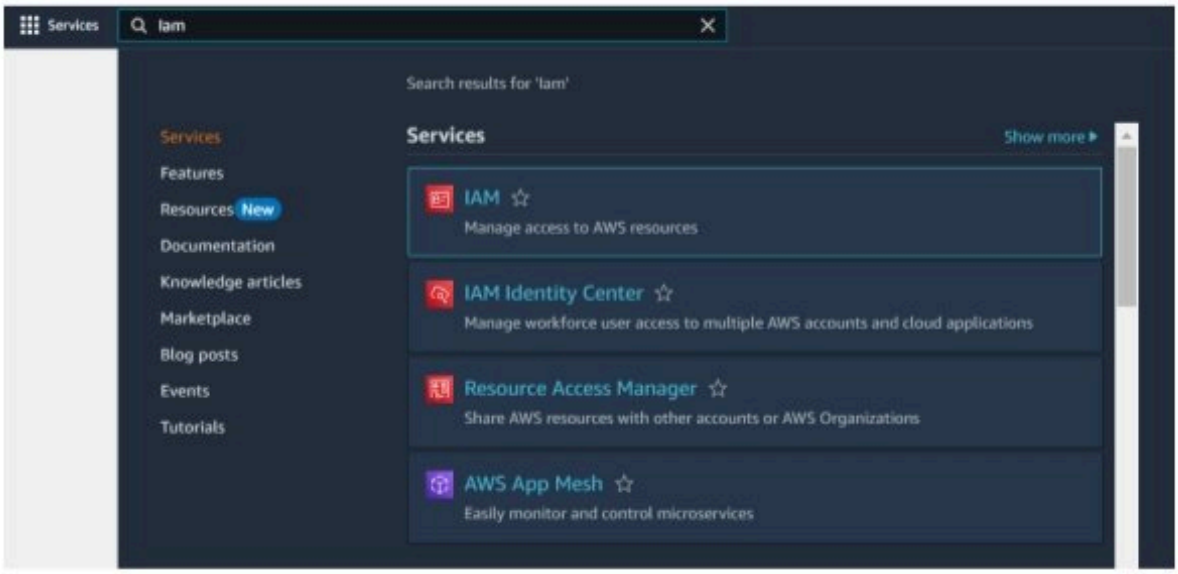
```
        cart_items = json.loads(cart_data)
        total_amount = float(total_amount)
    except (json.JSONDecodeError, ValueError):
        return render_template('checkout.html', error="Invalid cart data format.")

# Check if cart is empty
if not cart_items:
    return render_template('checkout.html', error="Your cart is empty.")

# Store order in DynamoDB
try:
    orders_table.put_item(
        Item={
            'order_id': str(uuid.uuid4()),
            'username': session.get('username', 'Guest'),
            'name': name,
            'address': address,
            'phone': phone,
            'items': cart_items,
            'total_amount': total_amount,
            'payment_method': payment_method,
            'timestamp': datetime.now().isoformat()
        }
    )
except Exception as db_error:
    print(f"DynamoDB Error: {db_error}")
    return render_template('checkout.html', error="Failed to save order. Please try again later.")
```

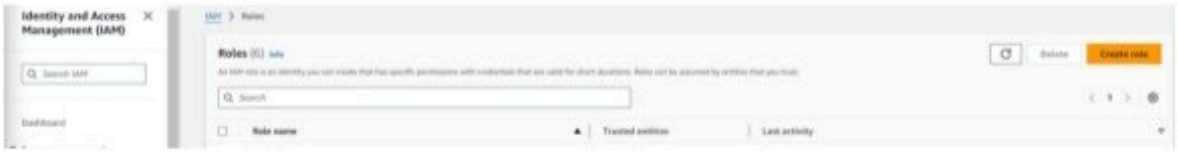
```
        # Redirect to success page with success message
        return redirect(url_for('sucess', message="Your order has been placed successfully!"))

    except Exception as e:
        print(f"Checkout error: {str(e)}")
        return render_template('checkout.html', error="An unexpected error occurred. Please try again.")

return render_template('checkout.html')  # Render checkout page for GET request
```

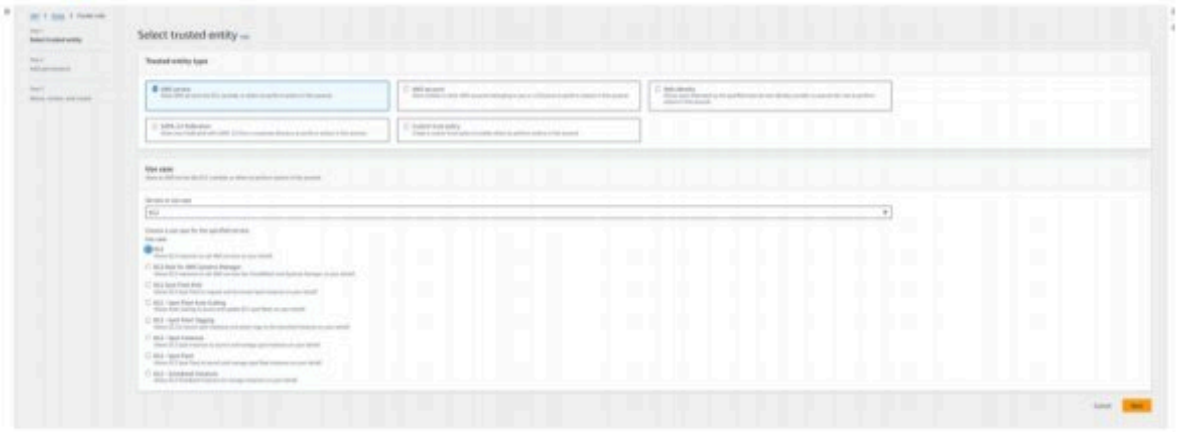
```
            return redirect(url_for('sucess', message="Your order has been placed successfully."))

        except Exception as e:
            print(f"Checkout error: {str(e)}")
            return render_template('checkout.html', error="An unexpected error occurred. Please try again.")

    return render_template('checkout.html')  # Render checkout page for GET request


@app.route('/sucess')
def success():
    return render_template('sucess.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)  # Add debug=True temporarily
```

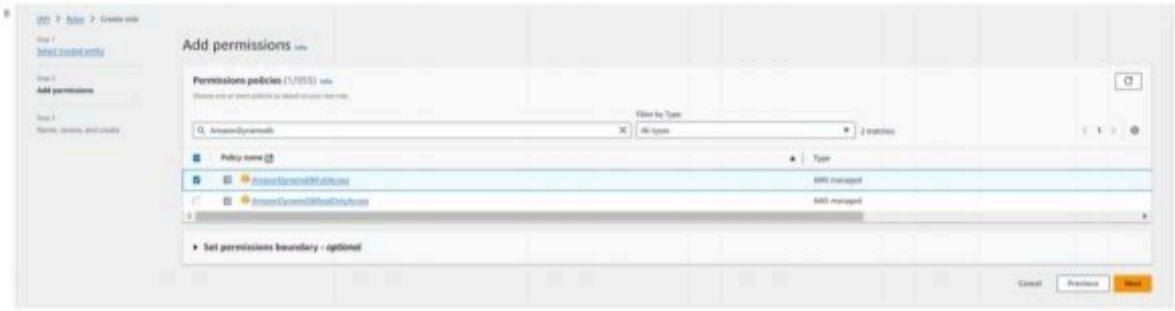
Milestone 5: IAM Role Setup

- **Activity 5.1:Create IAM Role.**
    - In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.





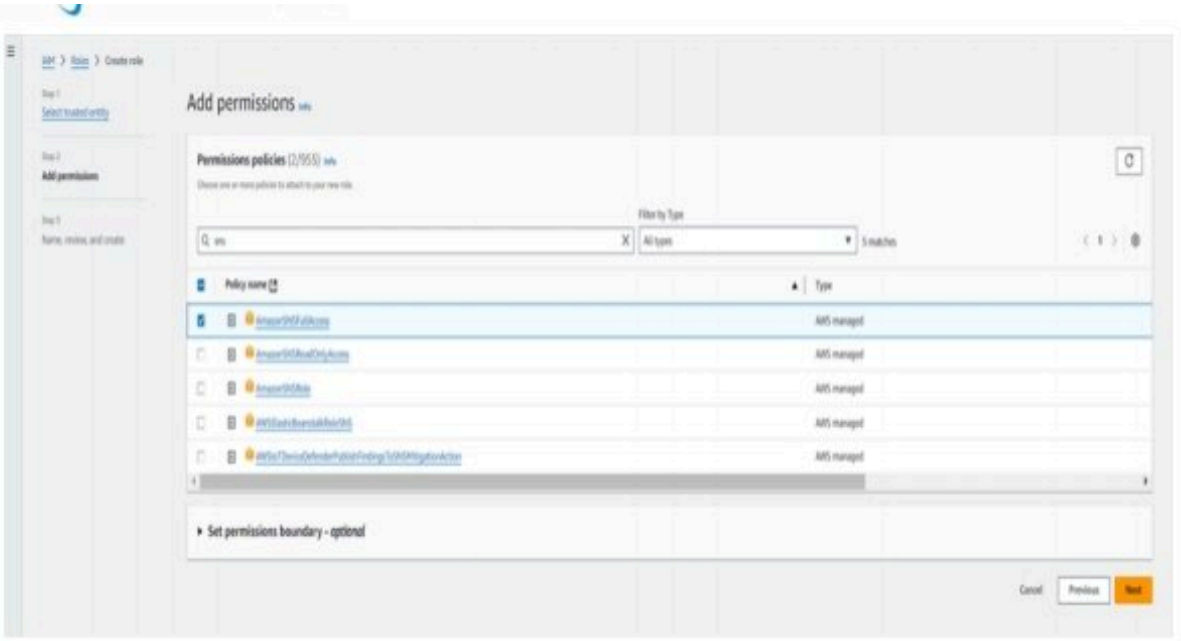


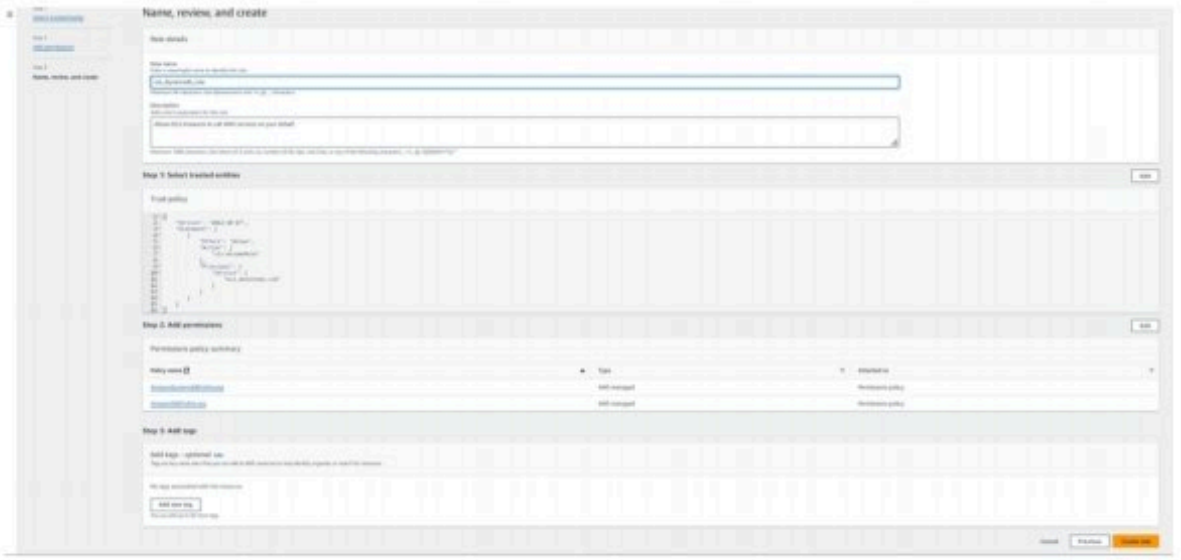


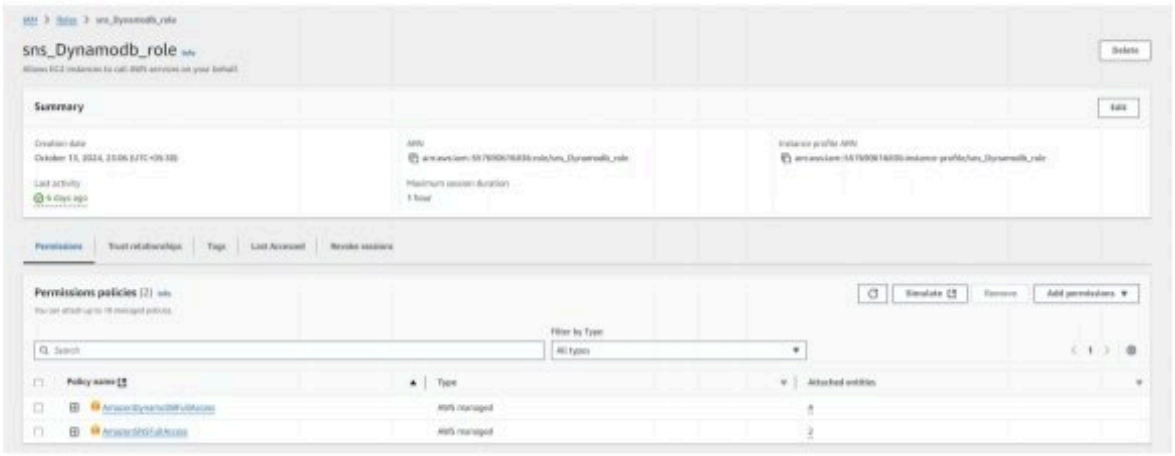
- **Activity 5.2: Attach Policies.**

  Attach the following policies to the role:

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



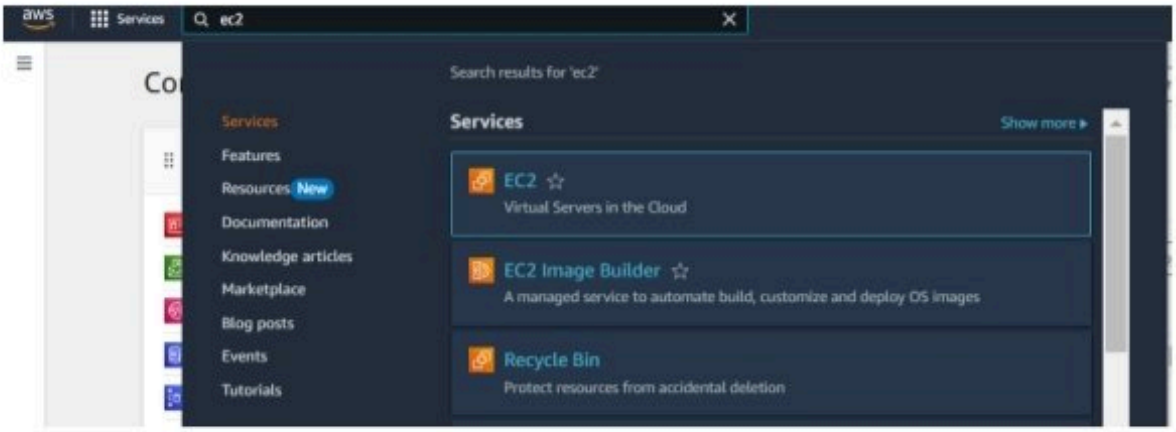




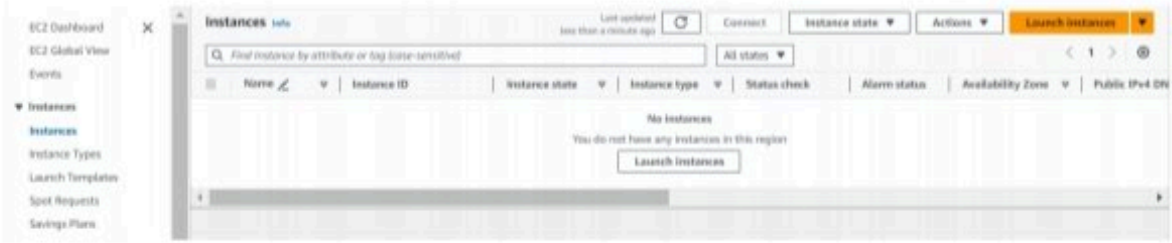
Milestone 6: EC2 Instance Setup

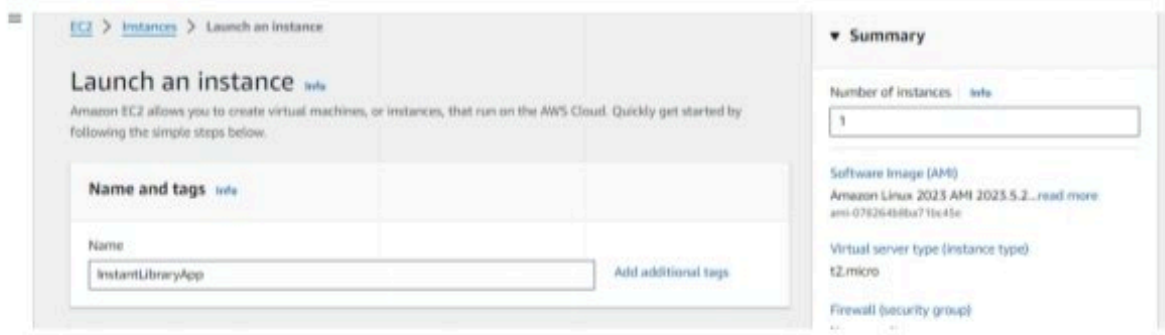
- **Note: Load your Flask app and Html files into GitHub repository.**

- **Activity 6.1: Launch an EC2 instance to host the Flask application.**
    - **Launch EC2 Instance**
        - In the AWS Console, navigate to EC2 and launch a new instance.

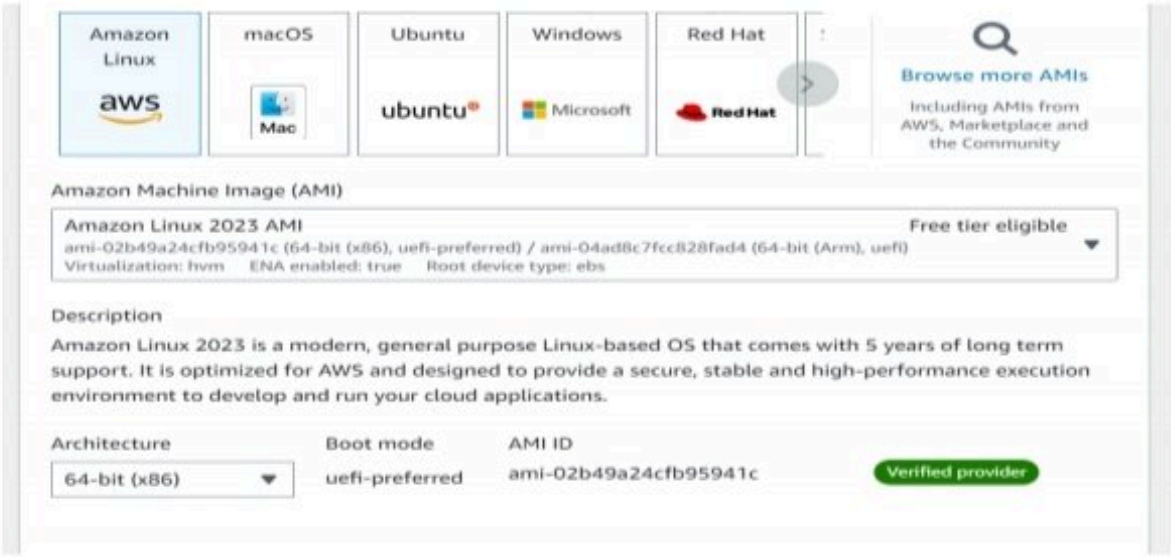


- Click on Launch instance to launch EC2 instance

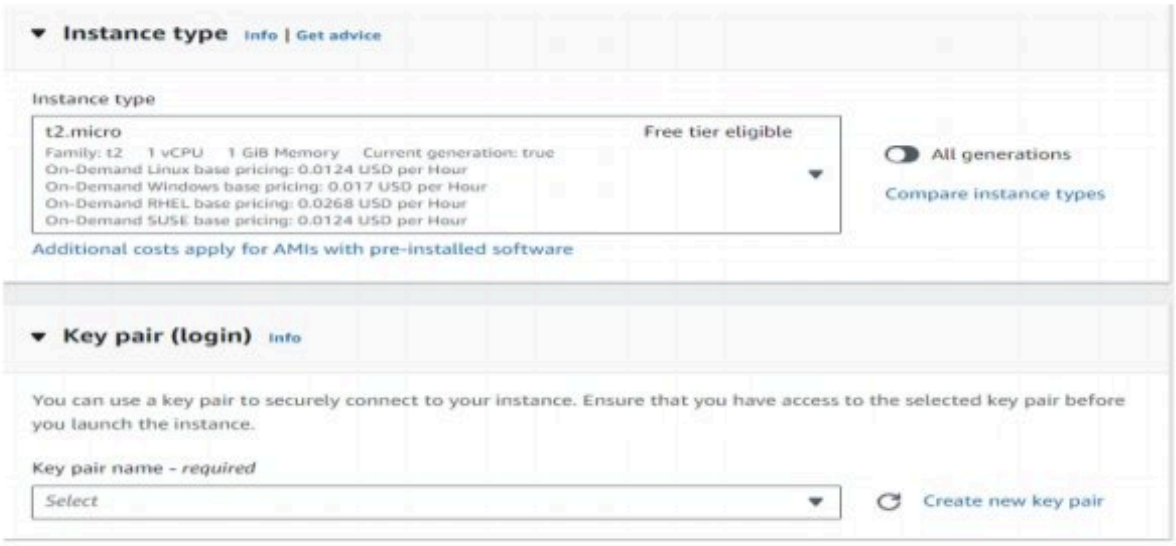


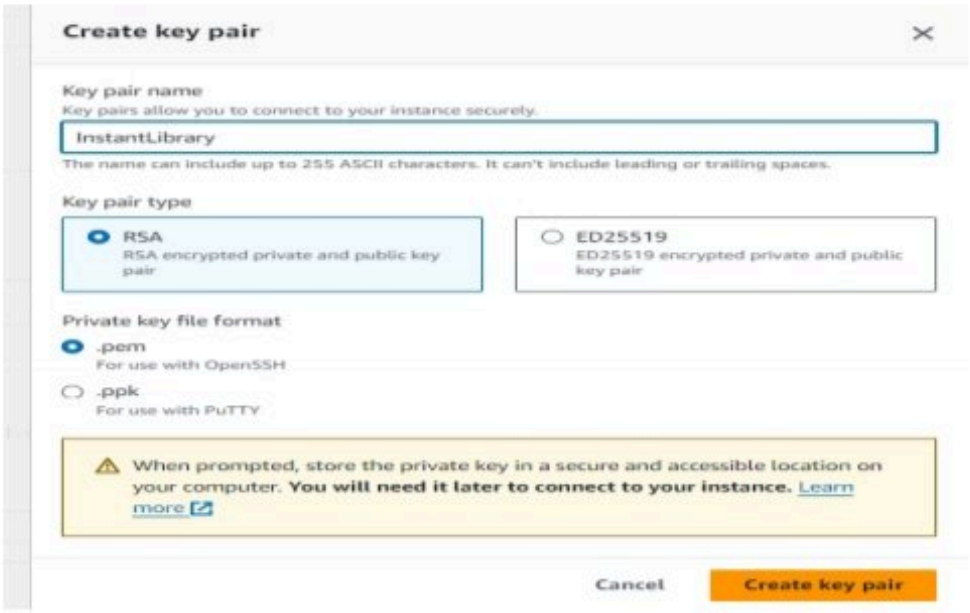


- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

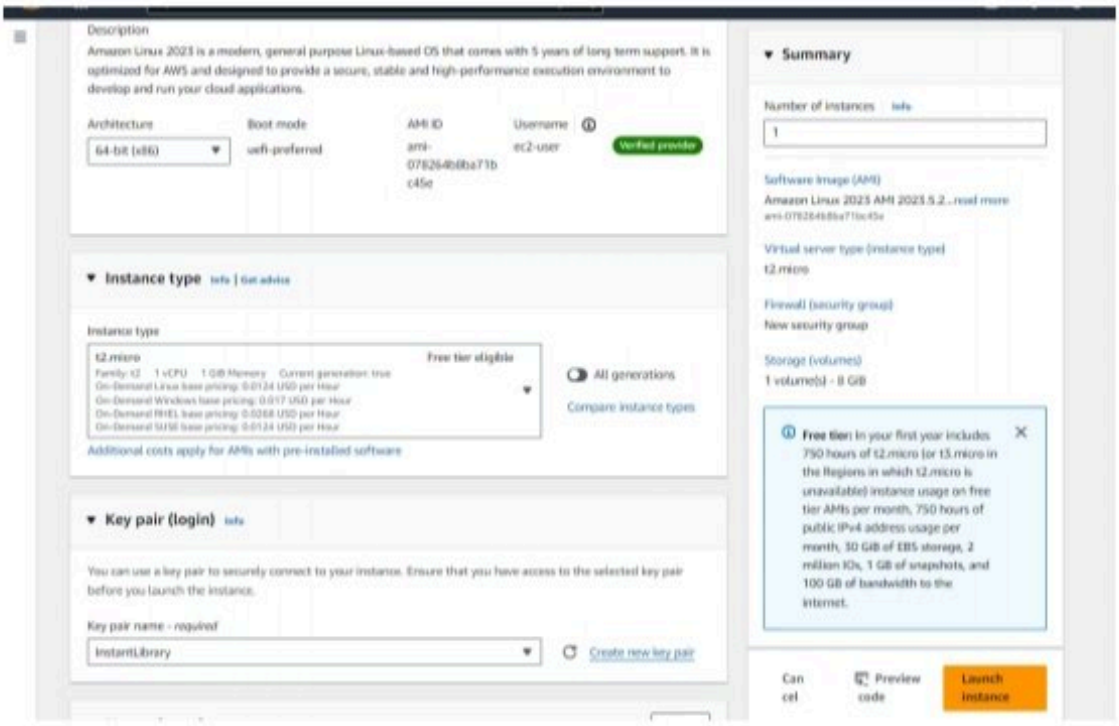


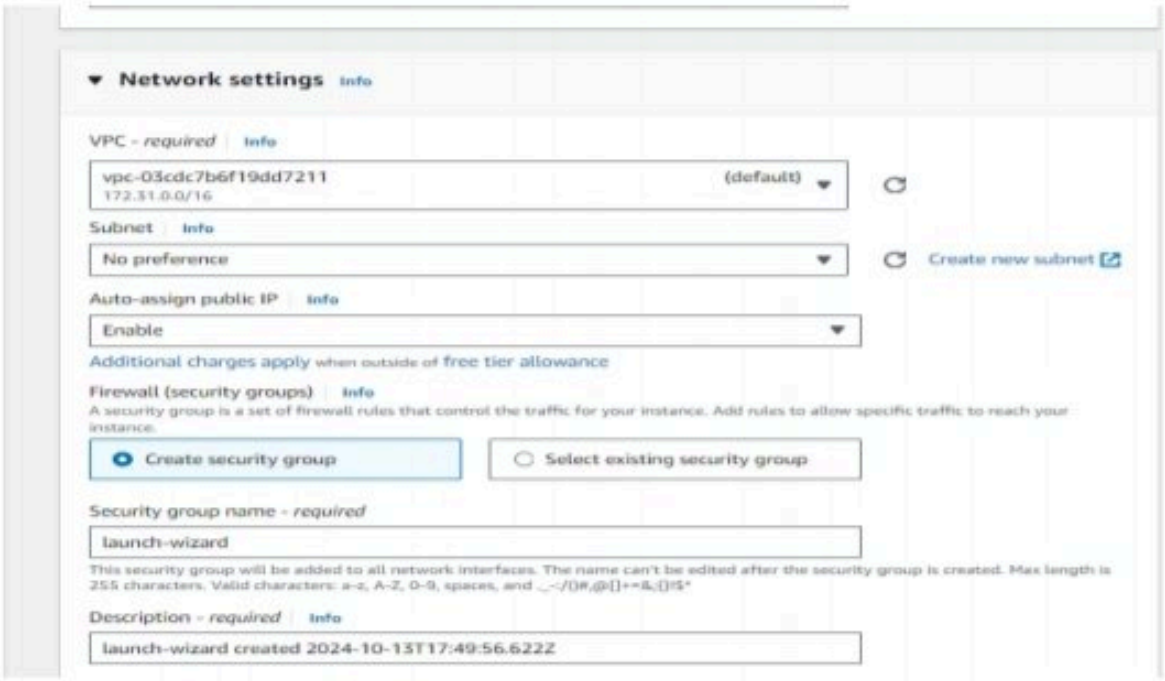
- Create and download the key pair for server access.

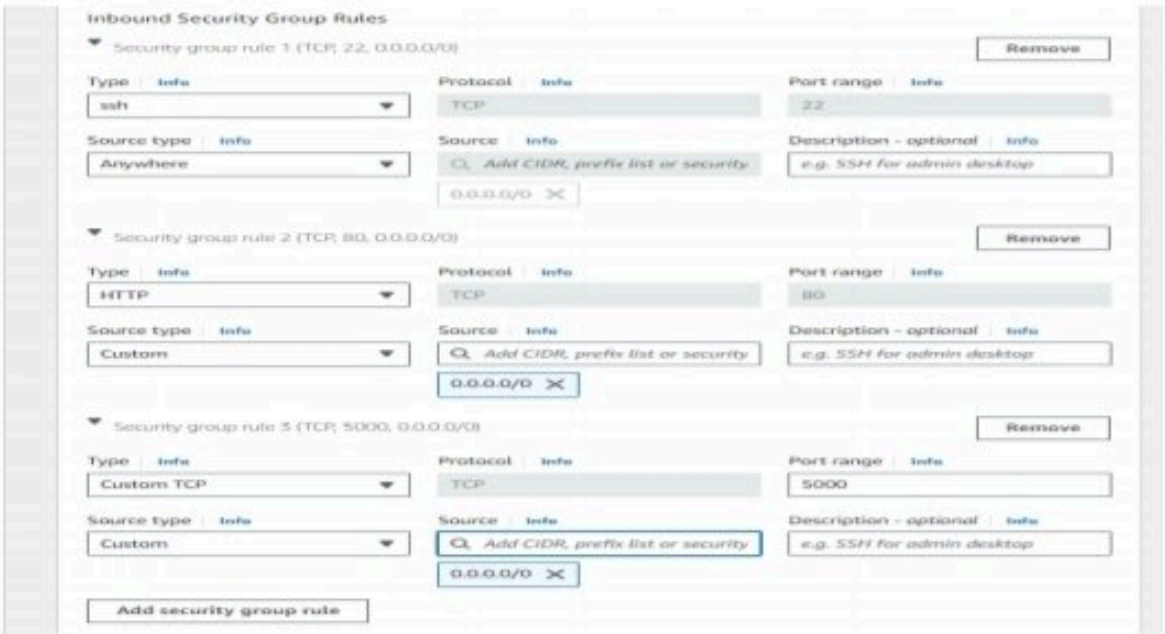


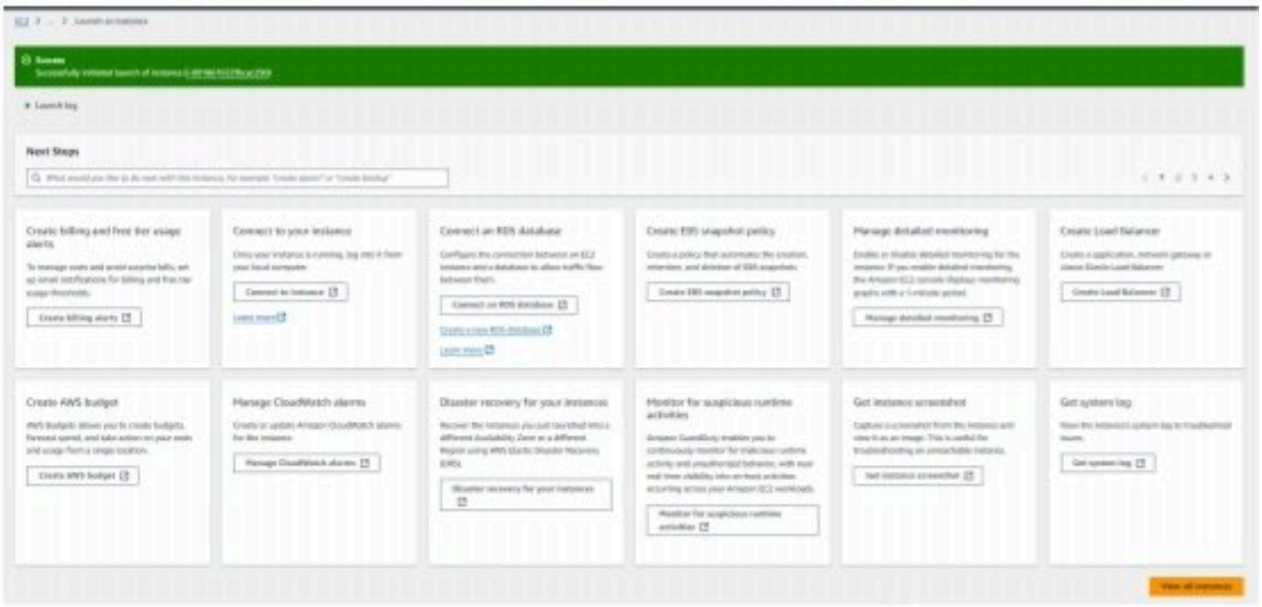


InstantLibrary.pem

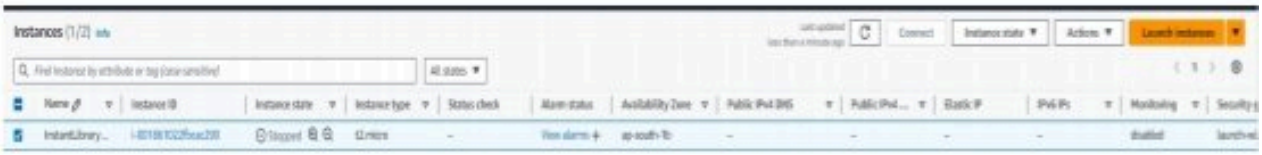


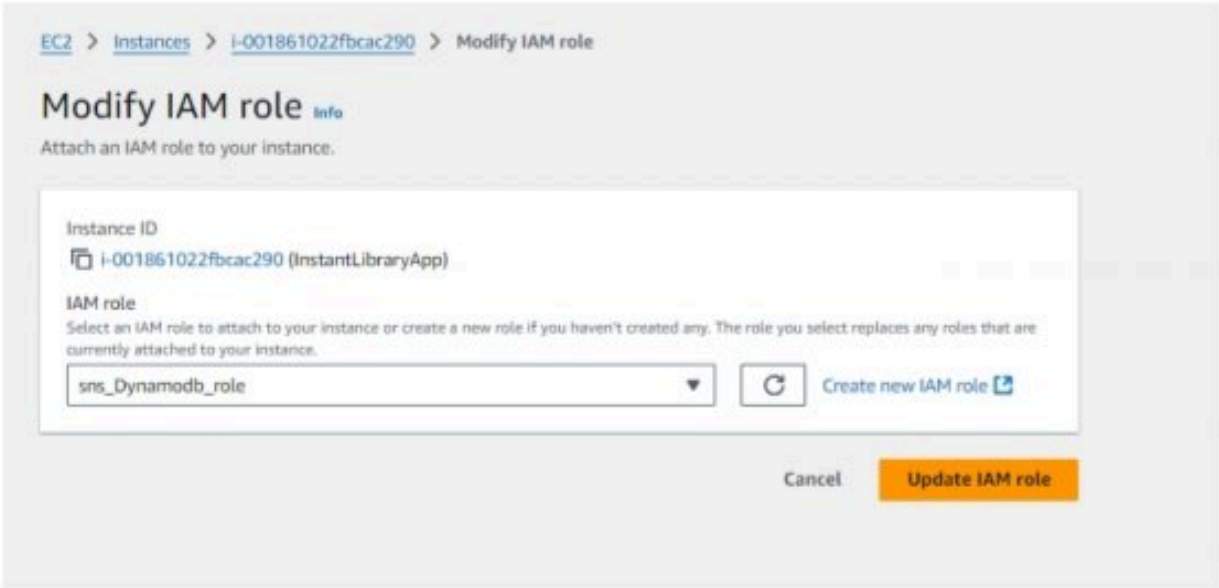




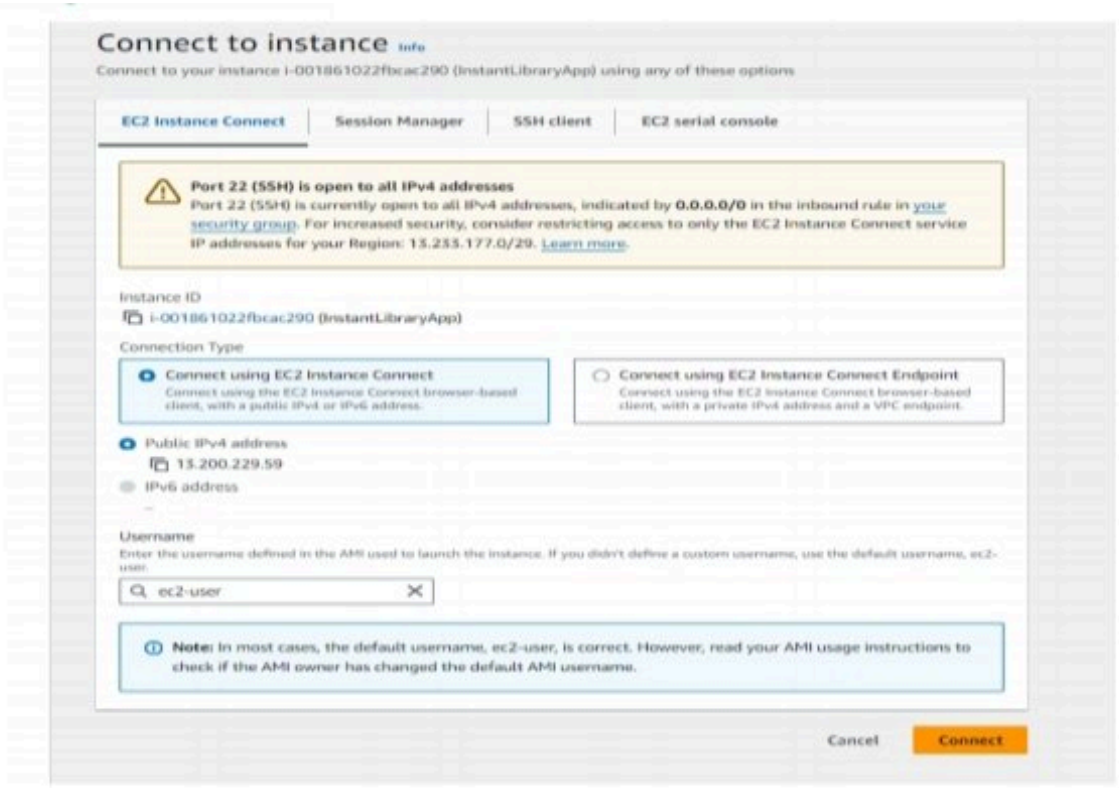


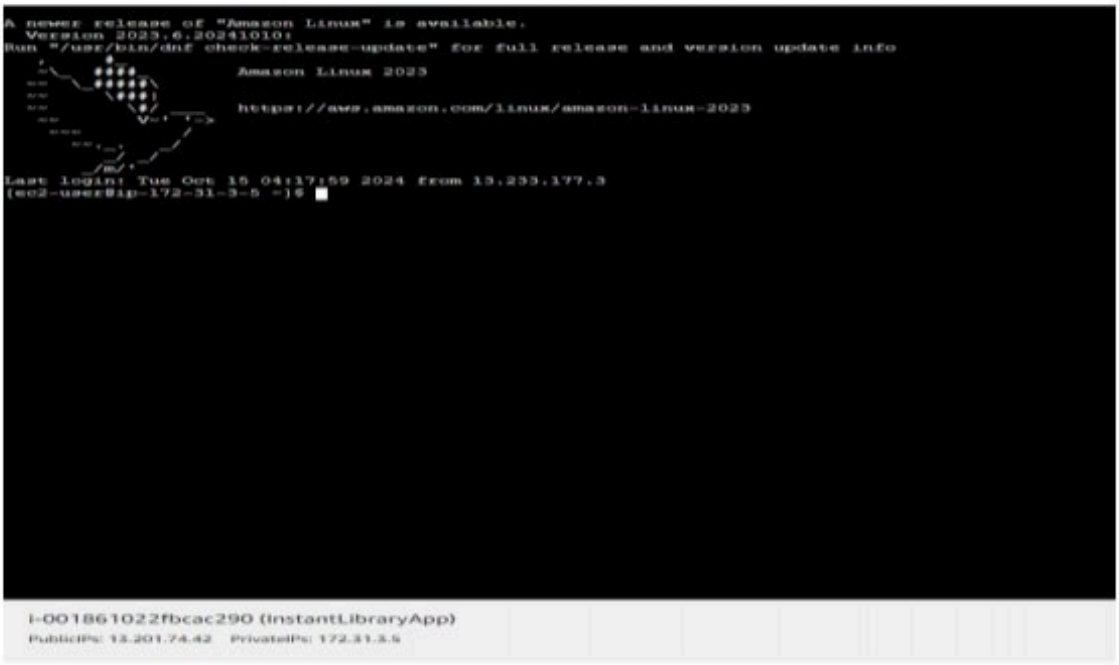
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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





- Now connect the EC2 with the files





Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

```
sudo yum update -y

sudo yum install python3 git

sudo pip3 install flask boto3
```