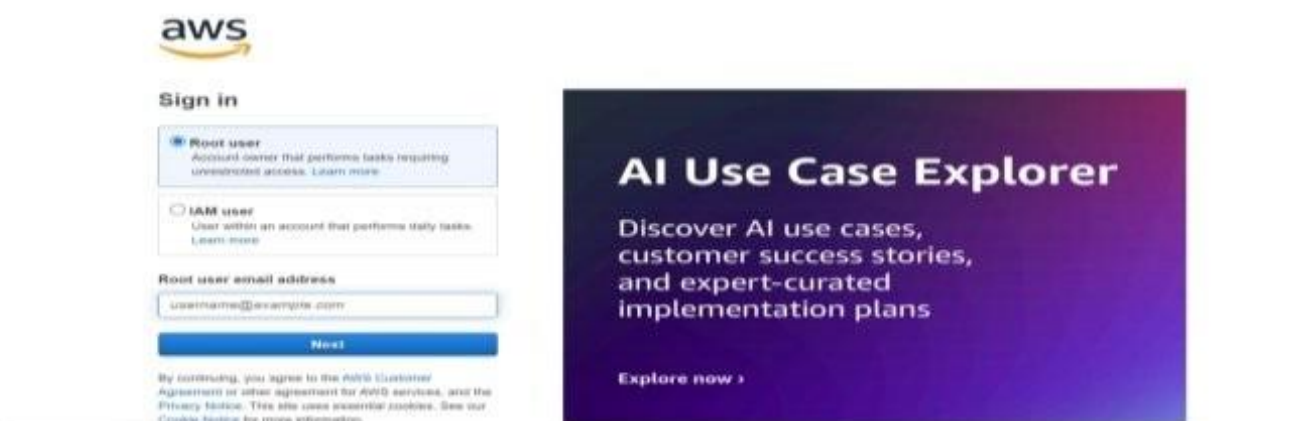


Milestone 1: AWS Account Setup and Login

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

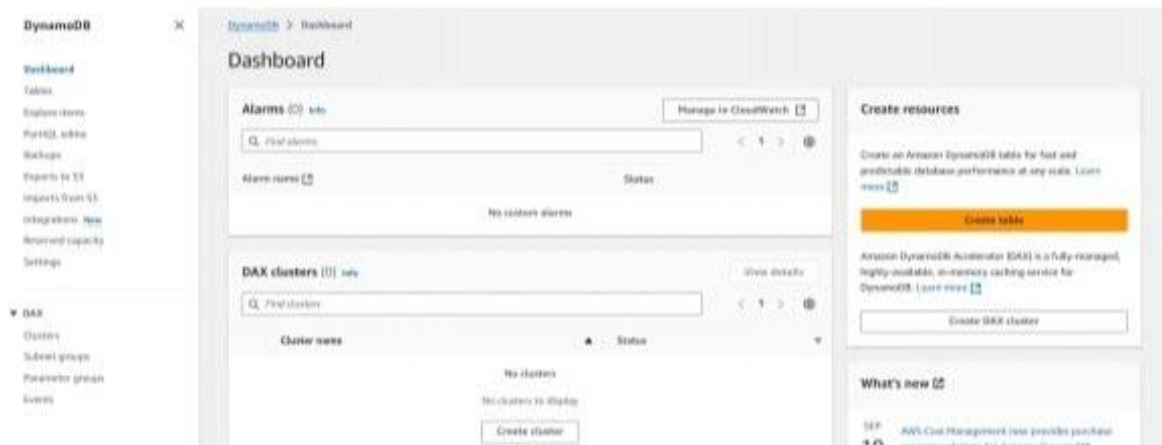
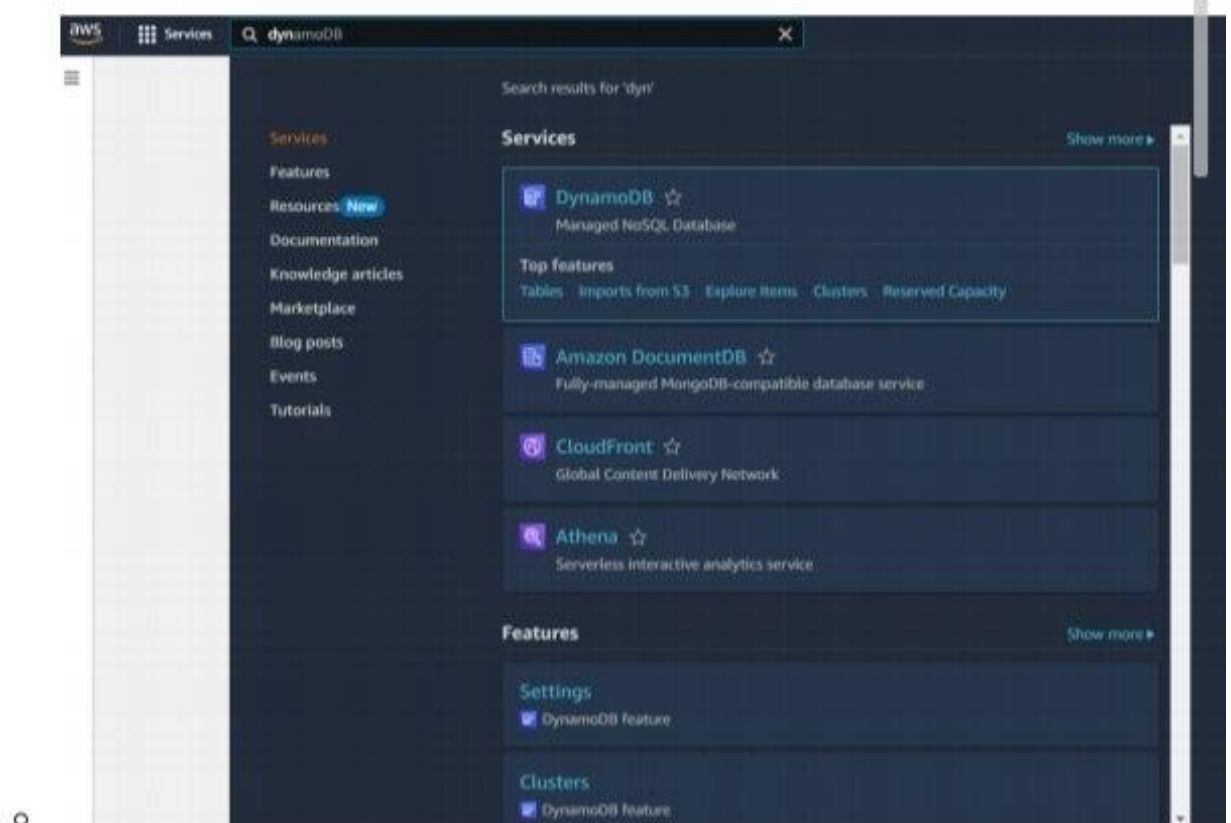


- **Activity 1.2: Log in to the AWS Management Console**
 - After setting up your account, log in to the [AWS Management Console](#).



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.

The screenshot shows the AWS Management Console's 'Create table' page for DynamoDB. The breadcrumb navigation at the top reads 'DynamoDB > Tables > Create table'. The main heading is 'Create table'. Below this is a section titled 'Table details' with an 'Info' link. A descriptive text states: 'DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.' The 'Table name' section includes a text box containing 'Users' and a note: 'This will be used to identify your table. Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).' The 'Partition key' section has a text box with 'email' and a dropdown menu set to 'String'. A note below states: '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.' The 'Sort key - optional' section has a text box with the placeholder 'Enter the sort key name' and a dropdown menu set to 'String'. A note below states: '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.'

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.

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.

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.

Table settings

☒ Default settings

This feature set is the default for new tables. More information.

☐ Customize settings

Use these advanced features to create ProvisionedDB mode.

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

DynamoDB

Dashboard

Tables

Explore items

Put/get/delete item

Backups

Export to S3

Import from S3

Integrations

Settings

The Request table was created successfully.

DynamoDB

Tables

Tables (2) only

Find tables

Any tag key

Any tag value

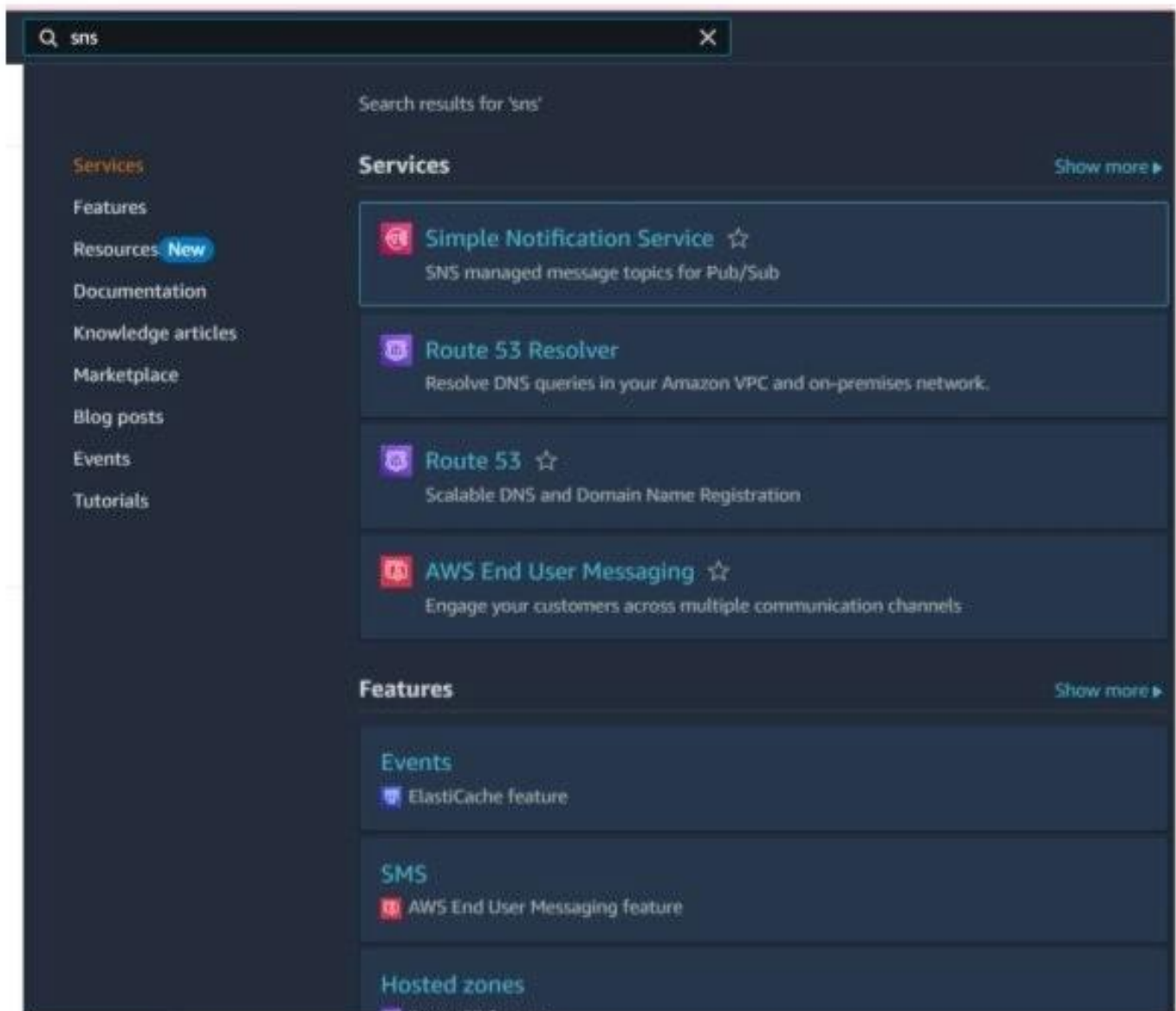
Create table

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size
<input type="checkbox"/>	Request	Active	email (S)	-	0	Off	Provisioned (5)	Provisioned (5)	0 bytes
<input type="checkbox"/>	Users	Active	email (S)	-	0	Off	Provisioned (5)	Provisioned (5)	0 bytes

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**.

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
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

☒ Standard

- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

Name

BookRequestNotifications

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**.

Amazon SNS

Home

Topics

Subscriptions

Messages

Topic configurations

Red messaging (beta)

Topic Feedback
Amazon SNS now supports in-place message archiving and routing for FRS topics. [Learn more](#)

Topic AmazonRequestNotifications created successfully
You can create other topics and send messages to them from this topic.

Amazon SNS > Topics > AmazonRequestNotifications

AmazonRequestNotifications Edit Reset Publish message

Details [info](#)

Name	AmazonRequestNotifications	Display name	
ARN	arn:aws:sns:us-east-1:441619123456:AmazonRequestNotifications	Topic owner	441619123456
Topic	Standard		

[Subscription](#) [Access policy](#) [Data protection policy](#) [Delivery policy \(HTTP/S\)](#) [Delivery status logging](#) [Encryption](#) [Tags](#) [Integrations](#)

Subscriptions (0) Edit Reset Request confirmation Confirm subscription Create subscription

Search

ID	Endpoint	Status	Platform
No subscriptions found. You don't need this option right now. Learn more			

[Create subscription](#)

- **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.

Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN

Protocol

The type of endpoint to subscribe

Endpoint

An email address that can receive notifications from Amazon SNS

ⓘ After your subscription is created, you must confirm it. [info](#)

► **Subscription filter policy - optional** [info](#)

This policy filters the messages that a subscriber receives.

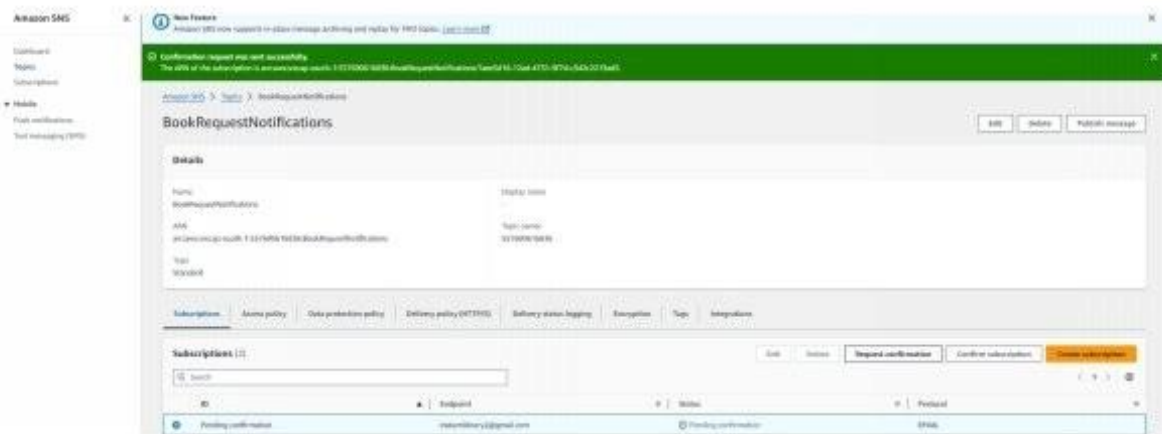
► **Redrive policy (dead-letter queue) - optional** [info](#)

Send undeliverable messages to a dead-letter queue.

Cancel **Create subscription**



- 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.

AWS Notification - Subscription Confirmation Inbox

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

9

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 ▾

1

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!

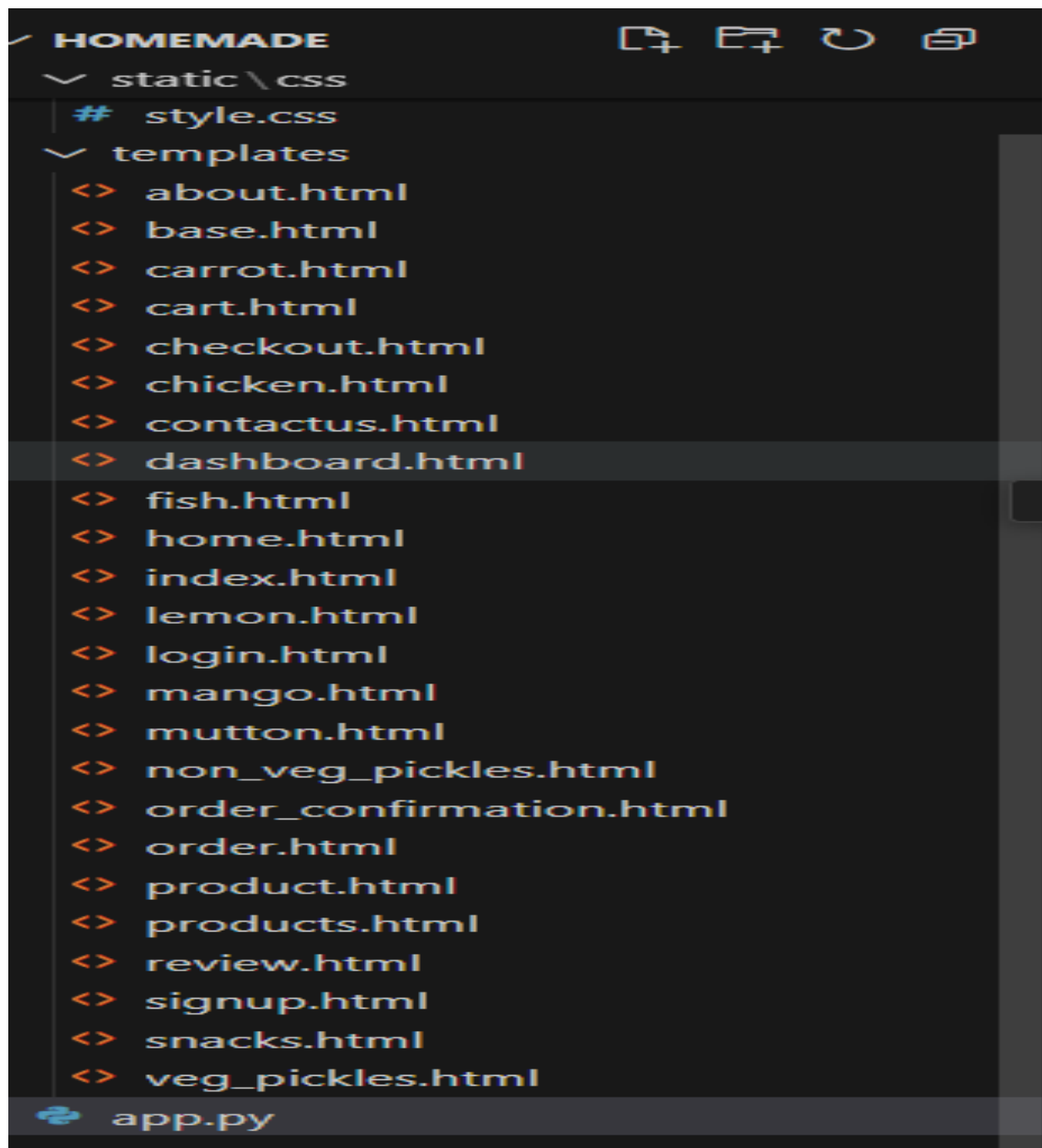
You have successfully subscribed.

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](#).

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



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'
table = dynamodb.Table('Users')
```


- 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')
users_table = dynamodb.Table('Users')
orders_table = dynamodb.Table('Orders')
```

```
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': 'Kadu Chikki', 'weights': {'250': 250, '500': 500, '1000': 750}}
```

```

@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        try:
            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']
            if check_password_hash(user['password'], password):
                session['logged_in'] = True
                session['username'] = username
                session.setdefault('home', [])
                return redirect(url_for('home')) # ✅ Add this to redirect after login
            else:
                return render_template('login.html', error="Incorrect password")
        except Exception as e:
            return render_template('login.html', error=f"An error occurred: {str(e)}")

    # ✅ This was missing
    return render_template('login.html')

```

```

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

```

Web

Page routing and login

```

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

            hashed_password = generate_password_hash(password)

            users_table.put_item(
                Item={
                    'username': username,
                    'email': email,
                    'password': hashed_password,
                }
            )

            return redirect(url_for('login'))

        except Exception as e:
            app.logger.error(f"Signup error: {str(e)}")
            return render_template('signup.html', error='Registration failed. Please try again.')
    return render_template('signup.html')

@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))

@app.route('/home')
def home():

```

Ln 2, Col 1

Homepage Routing: home page contains different routings veg_pickles,non_veg pickles,snacks,cart etc

pp.py > {} Flask

```
@app.route('/non_vegpickles')
def non_vegpickles():
    return render_template('non_vegpickles.html', products=products ['non_vegpickles'])

@app.route('/veg_pickles')
def veg_pickles():
    # Simply pass all products without filtering
    return render_template('veg_pickles.html', products=products ['veg_pickles'])

@app.route('/snacks')
def snacks():
    return render_template('snacks.html', products=products['snacks'])

@app.route('/check_out', methods=['GET', 'POST'])
def check_out():
    if not session.get('logged_in'):
        return redirect(url_for('login'))
    error_message = None # Variable to hold error messages
    if request.method == 'POST':
        try:
            name = request.form.get('name', '').strip()
            address = request.form.get('address', '').strip()
            phone = request.form.get('phone', '').strip()
            payment_method = request.form.get('payment', '').strip()
            #Validate inputs
            if not all([name, address, phone, payment_method]):
                return render_template('check_out.html', error="All fields are required.")

            if not phone.isdigit() or len(phone) != 10:
                return render_template('check_out.html', error="Phone number must be exactly 10 digits.")
            #Get cart data from hidden inputs
            cart_data = request.form.get('cart_data', '[]')
            total_amount = request.form.get('total_amount', '0')
            try:
                cart_items = json.loads(cart_data)
                total_amount = float(total_amount)
```

templates > <> checkout.html > ...

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Checkout</title>
6 </head>
7 <body>
8     <h1>Checkout</h1>
9     <p>Enter your shipping and payment details to complete the purchase.</p>
10 </body>
11 </html>
```

```

pp.py
except (json.JSONDecodeError, ValueError):
    return render_template('check_out.html', error="Invalid cart data format.")
# Ensure cart is not empty
if not cart_items:
    return render_template('check_out.html', error="Your cart is empty.")
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('check_out.html', error="Failed to save order. Please try again later.")
# Redirect to success page with success message
return redirect(url_for('success', 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('check_out.html')

@app.route('/cart', methods=['GET', 'POST'])
def cart():
    if request.method == 'POST':
        if 'cart' not in session:
            session['cart'] = []

    # Fetch form data
    product_id = request.form.get('product_id')

```

Ln 2, Col 1 Spaces: 4 UTF-8

```

pp.py
    session.modified = True

    return render_template('cart.html', cart=session.get('cart', []))

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

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

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

        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
        server.send_message(msg)
        server.quit()
    except Exception as e:
        print(f"Failed to send email: {e}")

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.**

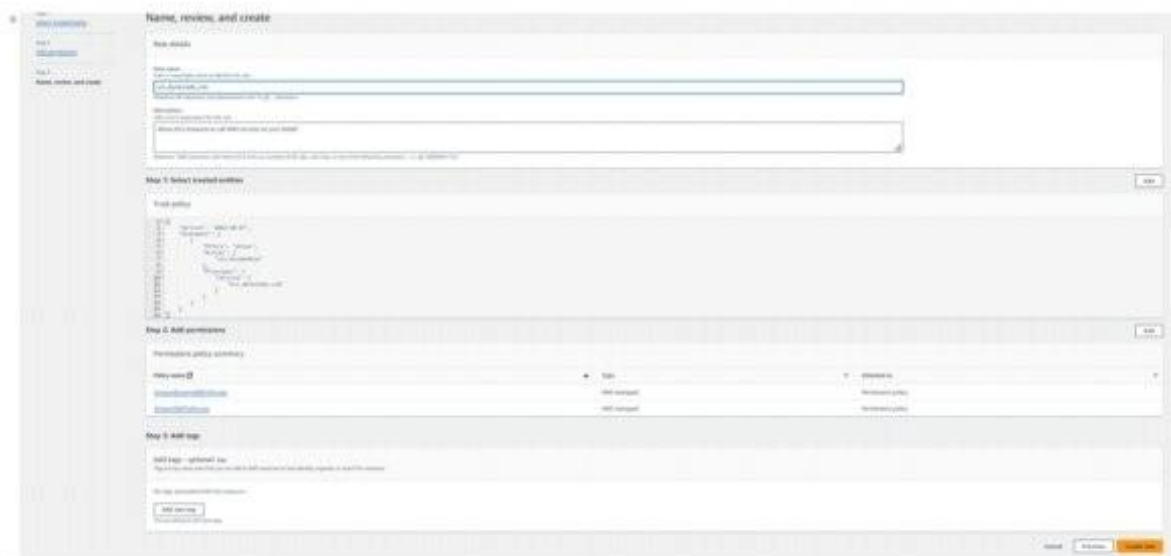
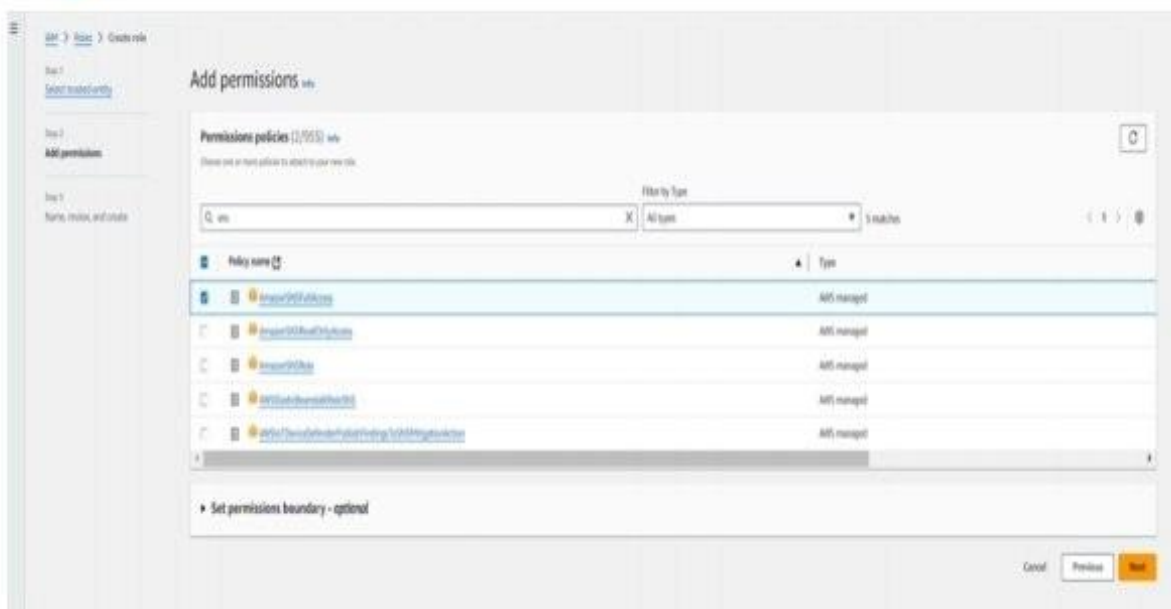
- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.

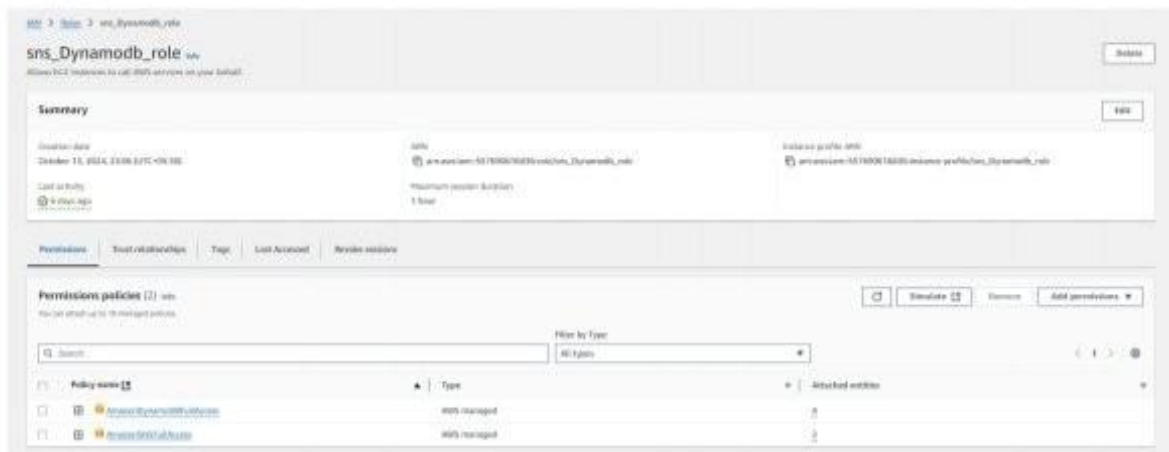


The screenshot shows the 'Add permissions' dialog box in Windows 10. The dialog is titled 'Add permissions' and has a search bar at the top. Below the search bar, there is a section titled 'Permissions policies (1/1/1/1)' with a sub-header 'Select one or more policies to add to your role'. A search filter 'AccessControl' is applied, showing 2 matches. The results are displayed in a table with columns 'Policy name' and 'Type'. The table lists two policies: 'AccessControl' and 'AccessControl', both with 'Self managed' types. Below the table, there is a section titled 'Set permissions boundary - optional'.

Policy name	Type
AccessControl	Self managed
AccessControl	Self managed

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





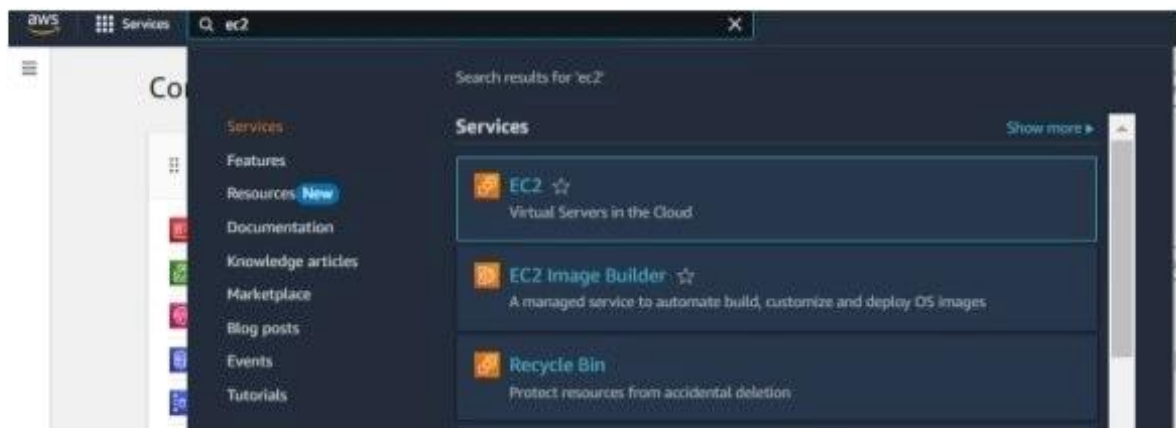
Milestone 6: EC2 Instance Setup

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

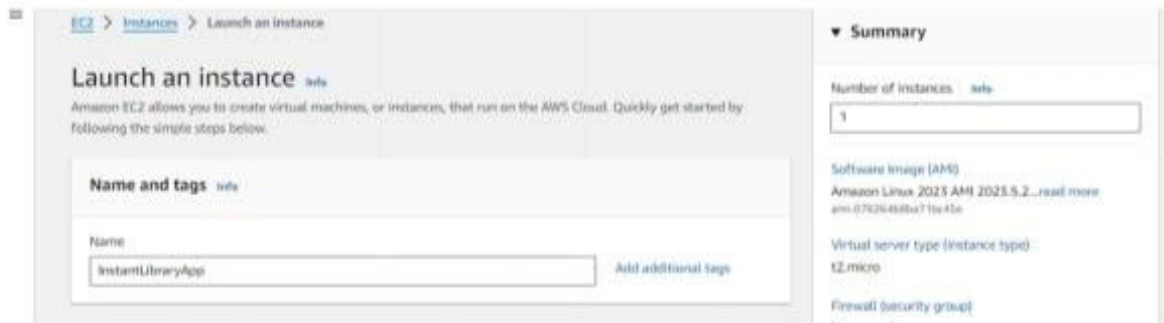
 static	Initial commit
 templates	Update statistics.html
 app.py	Update app.py



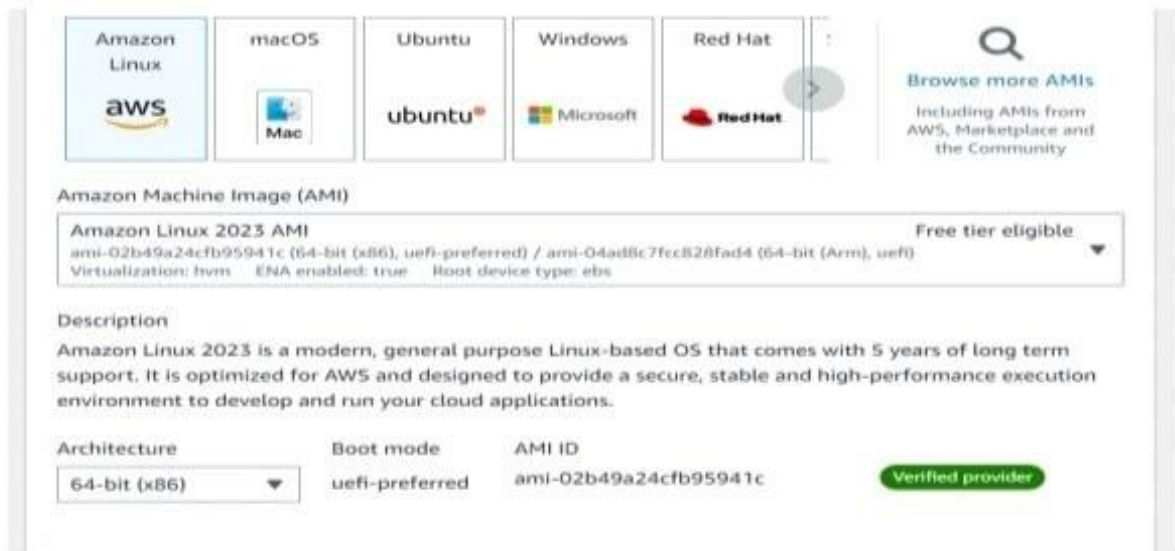
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**
 - **Launch EC2 Instance**
 - In the AWS Console, navigate to EC2 and launch a new instance.



- 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.

▼ Instance type

Info | Get advice

Instance type

t2.micro

Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0124 USD per Hour
On-Demand Windows base pricing: 0.017 USD per Hour
On-Demand RHEL base pricing: 0.0268 USD per Hour
On-Demand SUSE base pricing: 0.0124 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login)

Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select

Create new key pair

Create key pair

Key pair name

Key pairs allow you to connect to your instance securely.

InstantLibrary

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

☒ RSA

RSA encrypted private and public key pair

☐ ED25519

ED25519 encrypted private and public key pair

Private key file format

☒ .pem

For use with OpenSSH

☐ .ppk

For use with PuTTY

⚠ When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#)

Cancel

Create key pair



InstantLibrary.pem

Description

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

Architecture

64-bit x86

Boot mode

uefi-preferred

AMI ID

ami-07826468a71b-c45e

Username

ec2-user

Verified provider

▼ Instance type

info | Get advice

Instance type

t2.micro

Free tier eligible

Family 1: 1 vCPU, 1 GB Memory - Current generation, free
On-Demand Linux base pricing: 0.0124 USD per Hour
On-Demand Windows base pricing: 0.0117 USD per Hour
On-Demand RHEL base pricing: 0.0268 USD per Hour
On-Demand SUSE base pricing: 0.0124 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login)

info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

InstantLibrary

Create new key pair

▼ Summary

Number of instances

info

1

Software image (AMI)

Amazon Linux 2023 AMI 2023.5.2 - read more
ami-07826468a71b-c45e

Virtual server type (instance type)

t2.micro

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 8 GiB

Free tier

In your first year includes:
750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GiB of snapshots, and 100 GiB of bandwidth to the internet.

Can cancel

Preview code

Launch instance

Created in Photo-to-PDF One Click Converter. Download here: <https://firehawk.ai/phototopdf/>

▼ Network settings [Info](#)

VPC - *required* [Info](#)

vpc-03cdc7b6f19dd7211 (default) [Refresh](#)

172.31.0.0/16

Subnet [Info](#)

No preference [▼](#) [Refresh](#) [Create new subnet](#)

Auto-assign public IP [Info](#)

Enable [▼](#)

Additional charges apply when outside of free tier allowance

Firewall (security groups) [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group ☐ Select existing security group

Security group name - *required*

launch-wizard

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, space, and _-./()@[]+=&,'!*

Description - *required* [Info](#)

launch-wizard created 2024-10-15T17:49:56.522Z

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0) [Remove](#)

Type [Info](#)

ssh [▼](#)

Source type [Info](#)

Anywhere [▼](#)

Protocol [Info](#)

TCP

Port range [Info](#)

22

Source [Info](#)

[Add CIDR, prefix list or security...](#)
0.0.0.0/0 [✕](#)

Description - optional [Info](#)

e.g. SSH for admin desktop

▼ Security group rule 2 (TCP, 80, 0.0.0.0/0) [Remove](#)

Type [Info](#)

HTTP [▼](#)

Source type [Info](#)

Custom [▼](#)

Protocol [Info](#)

TCP

Port range [Info](#)

80

Source [Info](#)

[Add CIDR, prefix list or security...](#)
0.0.0.0/0 [✕](#)

Description - optional [Info](#)

e.g. SSH for admin desktop

▼ Security group rule 3 (TCP, 5000, 0.0.0.0/0) [Remove](#)

Type [Info](#)

Custom TCP [▼](#)

Source type [Info](#)

Custom [▼](#)

Protocol [Info](#)

TCP

Port range [Info](#)

5000

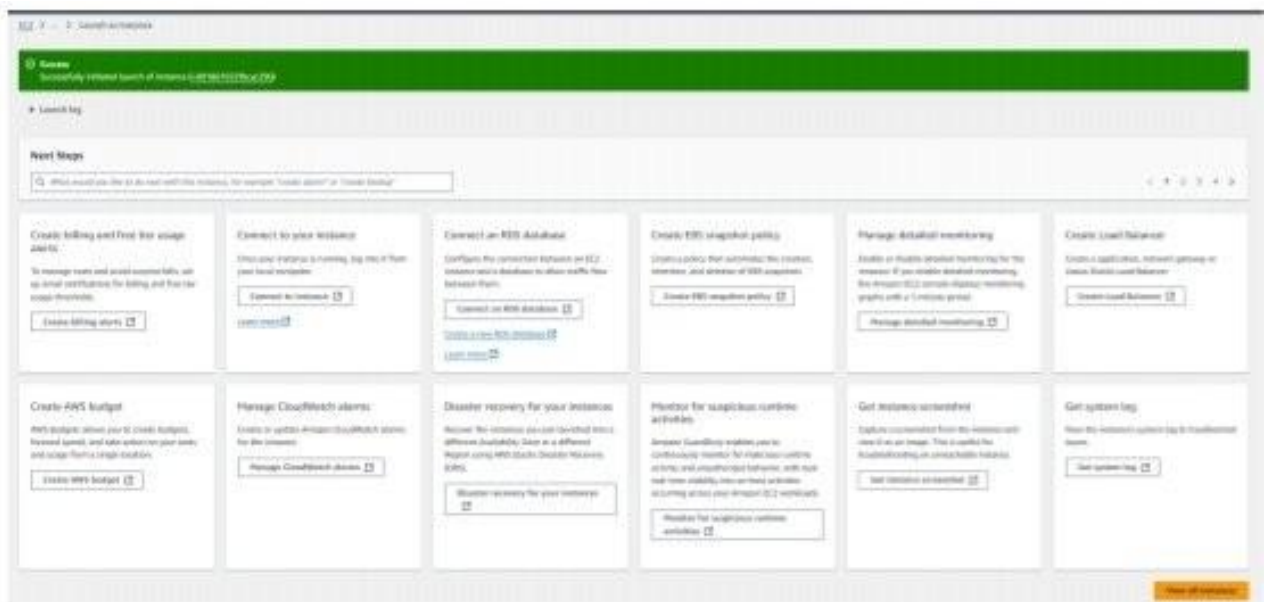
Source [Info](#)

[Add CIDR, prefix list or security...](#)
0.0.0.0/0 [✕](#)

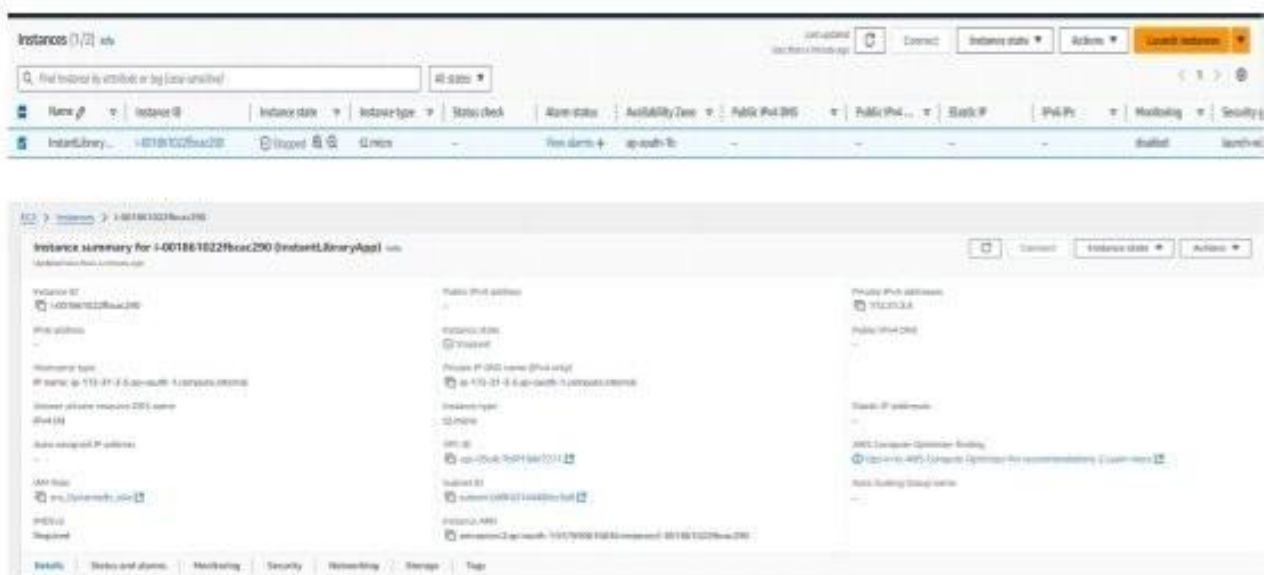
Description - optional [Info](#)

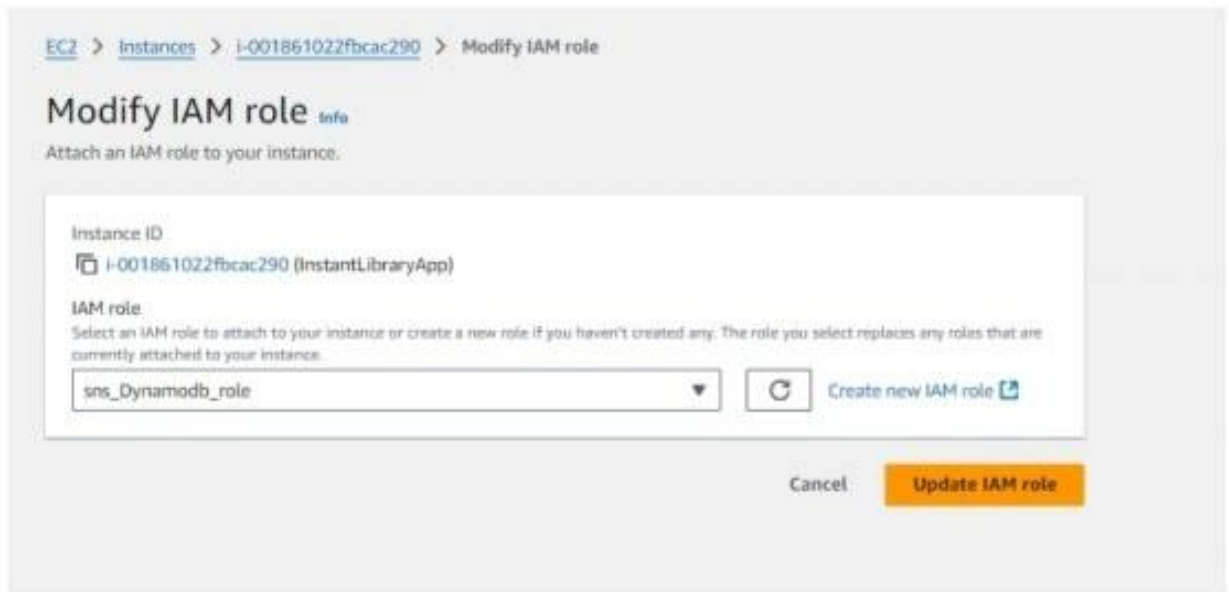
e.g. SSH for admin desktop

[Add security group rule](#)



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





- Created in Photo-to-PDF One Click Converter. Download here: <https://firehawk.ai/phototopdf/>

Connect to instance info


Connect to your instance i-001861022fbcac290 (InstantLibraryApp) using any of these options

EC2 Instance Connect

Session Manager

SSH client

EC2 serial console

**Port 22 (SSH) is open to all IPv4 addresses.**

Port 22 (SSH) is currently open to all IPv4 addresses, indicated by **0.0.0.0/0** in the inbound rule in your [security group](#). For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: [13.233.177.0/29](#). [Learn more](#).

Instance ID
i-001861022fbcac290 (InstantLibraryApp)

Connection Type


☒ **Connect using EC2 Instance Connect**
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.

☐ **Connect using EC2 Instance Connect Endpoint**
Connect using the EC2 Instance Connect browser-based client, with a private IPv6 address and a VPC endpoint.

☒ **Public IPv4 address**
13.200.229.59

☐ IPv6 address

Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, `ec2-user`.

**Note:** In most cases, the default username, `ec2-user`, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Cancel

Connect

```
A newer release of "Amazon Linux" is available.
Version 2023.6.20241010+
Run "/usr/bin/dnf check-releases-update" for full release and version update info

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

Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$
```

i-001861022fbcac290 (InstantLibraryApp)
PublicIP: 13.201.74.42 PrivateIP: 172.31.3.5

24

Created in Photo-to-PDF One Click Converter. Download here: <https://firehawk.ai/phototopdf/>

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
```