

Homemade Pickles & Snacks: Taste The Best

Project Description:

This project is a user-friendly website designed to showcase and sell a variety of homemade pickles and Snacks. The *Homemade Pickles & Snacks: Taste The Best* website is a flavorful e-commerce platform designed to bring traditional, homemade delights to customers nationwide. It showcases a vibrant catalog of pickles, snacks, and preserves inspired by authentic regional recipes. The project aims to connect modern audiences with the nostalgic taste of home-cooked treats while supporting small-batch production methods. Built with a user-friendly interface, the website ensures smooth navigation and a personalized shopping experience. Products are categorized by flavor profile, ingredients, and dietary preferences, allowing users to find their perfect match effortlessly. An integrated Flask backend handles order processing and dynamic product listings. Secure payment gateways and user authentication boost customer trust and transaction safety. AWS integration enables scalable hosting, real-time inventory updates, and responsive content delivery. The design prioritizes accessibility and mobile optimization for broader reach. A feedback and review section encourages community engagement and builds brand transparency. The blog section features recipes, tips, and stories behind the products to deepen customer connection. Admin dashboards allow easy product management and order tracking. SEO optimization and social media linking help improve visibility and drive organic traffic. The project reflects a harmonious blend of tradition and technology to promote homemade excellence. Overall, it delivers a warm, trustworthy, and appetizing user journey from homepage to checkout.

Scenario 1: Efficient Order Placement by Customers

In the "Flavour Cart" homemade pickles and snacks website, AWS EC2 powers a reliable and scalable infrastructure, allowing multiple customers to browse and order products simultaneously. For example, a user visits the site, explores the pickles section, and places an order for a mango pickle jar. Flask handles the backend workflow, processing user selections, updating inventory, and confirming orders in real time. The cloud-based architecture ensures a seamless and fast shopping experience, even during peak demand like festivals or seasonal sales.

Scenario 2: Seamless Order Notifications for Kitchen Staff and Customers

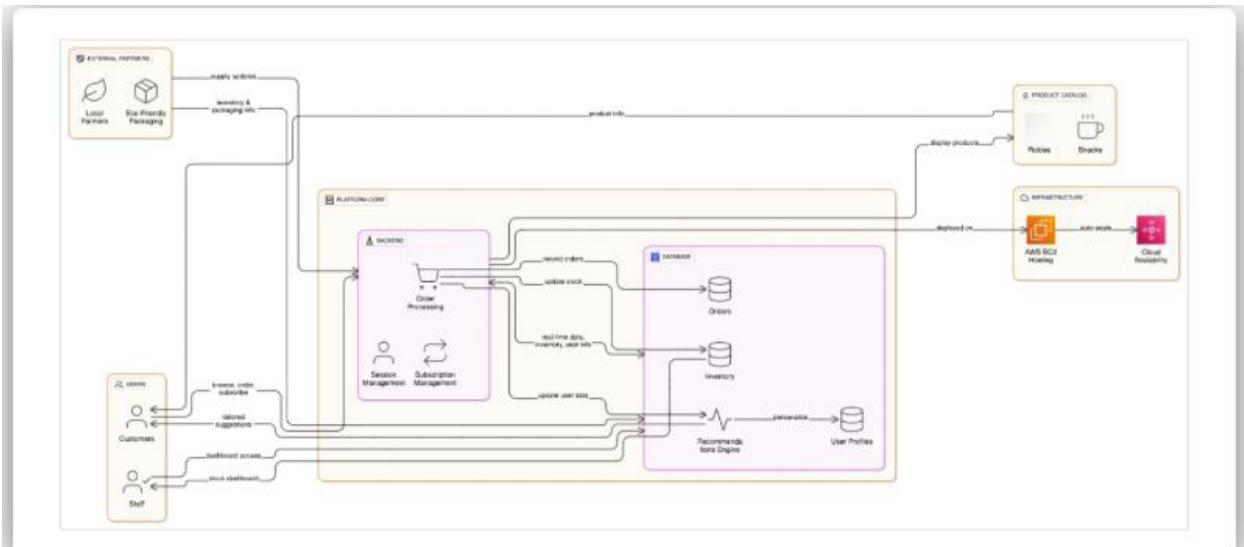
When an order is placed, the system uses AWS SNS to instantly notify both the customer and the in-house kitchen team. Flask captures the order details and triggers an SNS email or SMS alert:

the customer receives a confirmation with order tracking info, while the kitchen staff is alerted with the order details to initiate preparation. All order information is securely stored in AWS DynamoDB, ensuring timely fulfilment and status monitoring.

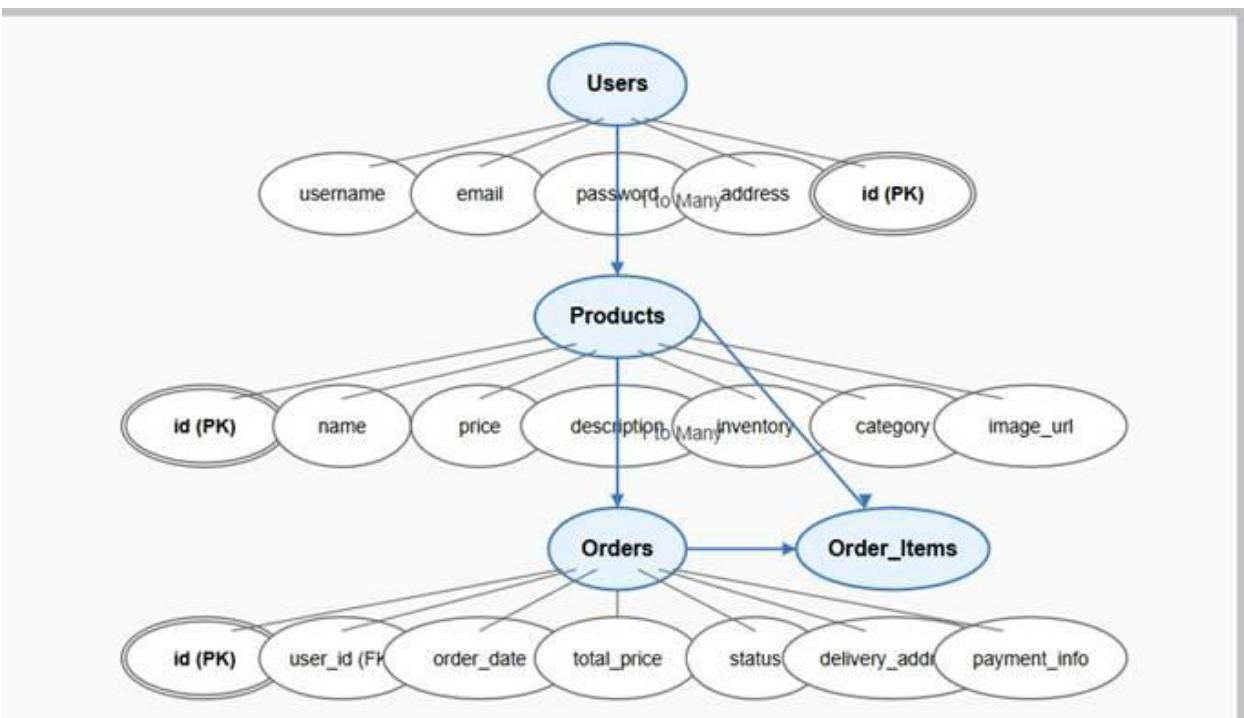
Scenario 3: Real-Time Inventory and Product Availability

The website allows users to explore the entire product range—spicy pickles, sweet treats, crunchy snacks—with up-to-date availability status. Flask fetches real-time data from DynamoDB, showing users whether an item is in stock or on backorder. Thanks to EC2, the site delivers a consistently smooth and responsive experience, even when accessed by many users across mobile or desktop platforms, ensuring customers always have access to the latest listings.

AWS ARCHITECTURE :



Entity Relationship (ER) Diagram :



Pre-requisites:

- 1.AWS Account Setup:** AWS Account Setup
- 2.Understanding IAM:** IAM Overview
- 3.Amazon EC2 Basics:** EC2 Tutorial
- 4.DynamoDB Basics:** DynamoDB Introduction
- 5.VS Code Installation:** VISUALSTUDIO.COM
- 6.Git Version Control:** Git Documentation

Project Work Flow:

1. AWS Account Setup and Login

Activity 1.1: Set up an AWS account if not already done.

Activity 1.2: Log in to the AWS Management Console.

2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.

Activity 2.2: Configure Attributes for User Data and Book Requests

3. SNS Notification Setup

Activity 3.1: Create SNS topics for book request notifications.

Activity 3.2: Subscribe users and library staff to SNS email notifications.

4. Backend Development and Application Setup

Activity 4.1: Develop the Backend Using Flask..

Activity 4.2: Integrate AWS Services Using boto3.

5. IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

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.

7.Deployment on EC2

Activity 7.1: Upload Flask Files

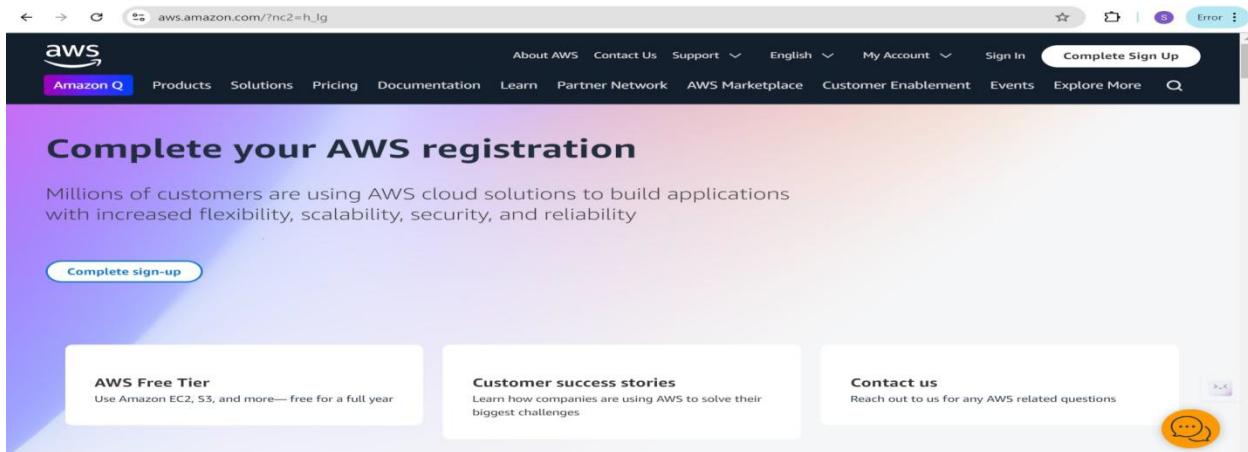
Activity 7.2: Run the Flask App

8. Testing and Deployment

Activity 8.1:Conduct functional testing to verify user signup, login, orders and notifications.

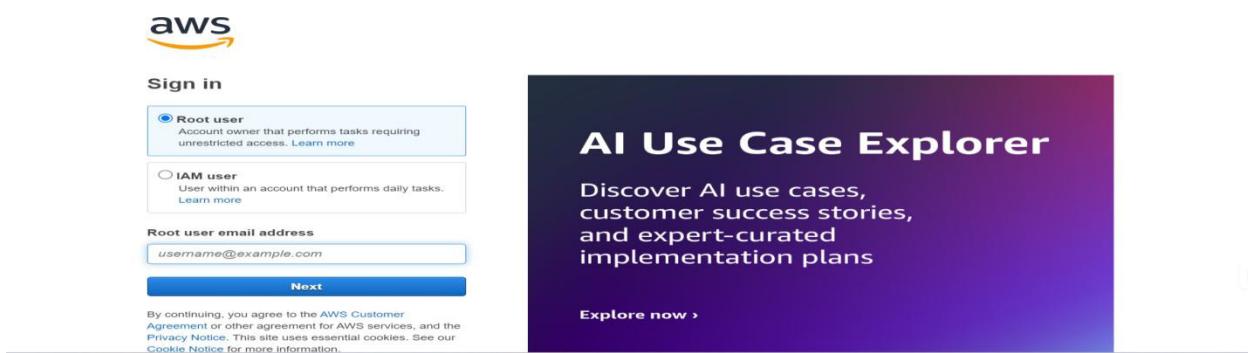
Milestone 1: AWS Account Setup and Login

- **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.

AWS Services Search: dynamoDB

Search results for 'dyn'

Services

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

DynamoDB ☆
Managed NoSQL Database

Top features

Tables Imports from S3 Explore Items Clusters Reserved Capacity

Amazon DocumentDB ☆
Fully-managed MongoDB-compatible database service

CloudFront ☆
Global Content Delivery Network

Athena ☆
Serverless interactive analytics service

Features

Show more ▶

Settings

DynamoDB feature

Clusters

DynamoDB feature

This screenshot shows the AWS search interface for the term 'dynamodb'. On the left, there's a sidebar with links to various AWS services and features. The main area displays a list of services, each with a thumbnail, name, and a star icon. Below the services, there are sections for 'Features' and 'Clusters', each containing a single item.

DynamoDB Dashboard

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

DynamoDB Tables

DynamoDB > Tables

Tables (0) Info

Find tables

Name Status Partition key Sort key Indexes Deletion protection Read capacity mode Write capacity mode Total size

You have no tables in this account in this AWS Region.

Create table

Actions Delete Create table

Create resources

Create table

Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. Learn more

Create DAX cluster

What's new

SEP 19 AWS Cost Management now provides purchase recommendations for Amazon DynamoDB...

This screenshot shows the AWS DynamoDB management console. It includes a top-level navigation bar, a sidebar with various management options, and two main content areas. The top area is the 'Dashboard' showing 'Alarms' and 'DAX clusters'. The bottom area is the 'Tables' section, which is currently empty. A 'Create resources' sidebar on the right provides links to create new tables or DAX clusters.

Activity 2.2:Create a DynamoDB table for storing Signup details and pickle requests.

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

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The top navigation bar shows 'DynamoDB > Tables > Create table'. The main title is 'Create table'. Under 'Table details', there is an 'Info' link. A note states: 'DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.' The 'Table name' field is set to 'Users'. Below it, a note says: 'This will be used to identify your table.' The 'Partition key' section contains a field with 'email' and a dropdown menu set to 'String'. A note below says: '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.' The 'Sort key - optional' section contains a field with 'Enter the sort key name' and a dropdown menu set to 'String'. A note below says: '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.' Both fields have a note: '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](#)

The screenshot shows the AWS DynamoDB 'Tables' page. A green header bar at the top indicates that the 'Users' table was created successfully. The main table view shows one item: 'Users' (Status: Active, Partition key: email (\$), Sort key: -, Indexes: 0, Read capacity mode: Off, Write capacity mode: Provisioned (5), Total size: 0 bytes). The table has columns for Name, Status, Partition key, Sort key, Indexes, Deletion protection, Read capacity mode, Write capacity mode, and Total size. There are also 'Actions' and 'Delete' buttons at the top of the table area.

Follow the same steps to create a requests table with Email as the primary key for pickles order, contact requests data.

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and primary key when you create the

Table name

This will be used to identify your table.

PicklesOrder

Between 3 and 255 characters, containing letters, numbers, and periods ..

Partition key

The partition key part of the table's primary key. It is a hash value that is used to retrieve items from your table hosts for scalability and availability.

order_id

String



1 to 255 characters and case sensitive.

Sort key - optional

You can use a sort key as the second part of table's primary key. Sort key allows you to sort or search among all the same partition key.

Enter the sort key name

String



1 to 255 characters and case sensitive.

Default settings

This option optimizes your table based on AWS

Customize settings

Use this option to specify or make DynamoDB

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](#)

The screenshot shows the AWS DynamoDB console interface. The left sidebar has 'DynamoDB' selected under 'Tables'. The main area shows a table with the following data:

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity mode
PickleContacts	Active	contact_id (\$)	-	0	0	Off	☆	On-demand
PickleOrders	Active	order_id (\$)	-	0	0	Off	☆	On-demand

A blue banner at the top right says 'The PickleContacts table was created successfully.' There are also 'Share feedback' and 'Create table' buttons.

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](#)

ⓘ Share your feedback on Amazon DynamoDB

Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.

[Share feedback](#)

ⓘ The PickleContacts table was created successfully.

Tables (3) [Info](#)



[Actions](#) ▾

[Delete](#)

[Create table](#)

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity mode
<input type="checkbox"/>	PickleContacts	Active	contact_id (\$)	-	0	0	Off	☆	On-demand
<input type="checkbox"/>	PickleOrders	Active	order_id (\$)	-	0	0	Off	☆	On-demand
<input type="checkbox"/>	PickleUsers	Active	email (\$)	-	0	0	Off	☆	On-demand

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

SNS

Search results for 'sns'

Services

Features

Resources **New**

Documentation

Knowledge articles

Marketplace

Blog posts

Events

Tutorials

Services

Show more ▶

 Simple Notification Service ☆
SNS managed message topics for Pub/Sub

 Route 53 Resolver
Resolve DNS queries in your Amazon VPC and on-premises network.

 Route 53 ☆
Scalable DNS and Domain Name Registration

 AWS End User Messaging ☆
Engage your customers across multiple communication channels

Features

Show more ▶

Events

 ElastiCache feature

SMS

 AWS End User Messaging feature

Hosted zones

 Route 53 feature

Amazon SNS

Dashboard

Topics

Subscriptions

▼ Mobile

Push notifications

Text messaging (SMS)

 New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Application Integration

Amazon Simple Notification Service

Pub/sub messaging for microservices and serverless applications.

Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.

Create topic

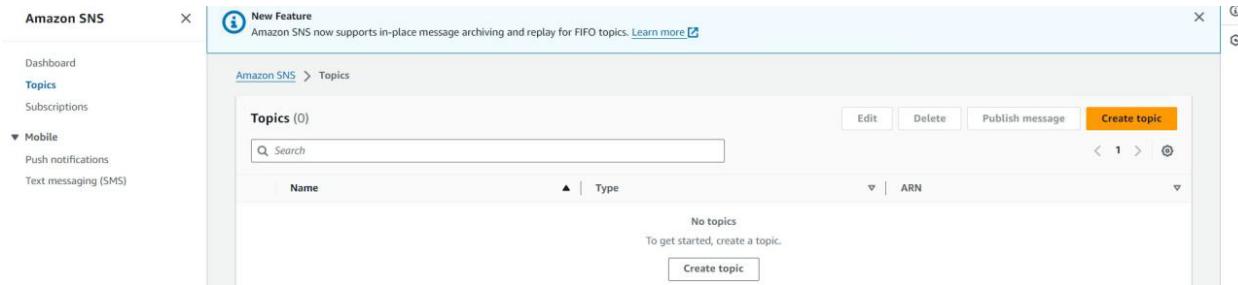
Topic name
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.

Next step

Start with an overview

Pricing

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



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

Milestone 4:Backend Development and Application Setup

- **Activity 4.1:** Develop the backend using Flask

File Explorer Structure

Description: set up the INSTANT LIBRARY project with an app.py file, a static/ folder for images, and a templates/ directory containing all required HTML pages like home, login, signup, pickle-specific pages(e.g.,Veg_Pickles.html,Non_Veg_Pickles.html,Snacks.html), and utility pages (e.g., order.html, contact.html).

Description of the code :

- Flask App Initialization

```
from flask import Flask, request, render_template, redirect, url_for, session, jsonify
import boto3
import uuid
import os
import hashlib
from datetime import datetime
from botocore.exceptions import ClientError
```

Description: import essential libraries including Flask utilities for routing, Boto3 for DynamoDB

operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification

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

Description: initialize the Flask application instance using Flask(name) to start building the web app.

- Dynamodb Setup:

```
# DynamoDB tables - Production ready
order_table = dynamodb.Table('PickleOrders')
user_table = dynamodb.Table('PickleUsers')
contact_table = dynamodb.Table('PickleContacts')
```

Description: initialize the DynamoDB resource for the us-east-1 region and set up access to the Users ,Orders, Contact tables for storing user details and pickles requests.

- SNS Connection

```
@app.route('/notify')
def notify():
    # Send a sample SNS message
    sns.publish(
        TopicArn=SNS_TOPIC_ARN,
        Message='A new order was received on Homemade Pickles & Snacks!',
        Subject='New Pickle Order Alert'
    )
    return "SNS notification sent!"
```

Description: Configure SNS to send notifications when a pickle 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.) Create an 'App password' for the email ID and store it in the SENDER_PASSWORD section.

- **Routes for Web Pages**

- **Home Route:**

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

Description: define the home route / to automatically redirect users to the home page when they access the base URL.

- Signup Route :

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        try:
            name = request.form['name']
            email = request.form['email']
            password = request.form['password']
            hashed_password = hash_password(password)
            timestamp = datetime.utcnow().isoformat()

            # Check if user already exists
            response = user_table.get_item(Key={'email': email})
            if 'Item' in response:
                flash('User already exists', 'error')
                return render_template('signup.html')

            # Save user to DynamoDB
            user_table.put_item(Item={
                'email': email,
                'name': name,
                'password': hashed_password,
                'created_at': timestamp,
                'status': 'active'
            })
```

```
# Send welcome email
welcome_message = f"Dear {name},\n\nWelcome to Homemade Pickles &
Snacks!\n\nYour account has been created successfully. You can now login and
start ordering our delicious homemade pickles and snacks.\n\nThank you for
joining us!\n\nBest regards,\nHomemade Pickles & Snacks Team"
send_email_notification(email, 'Welcome to Homemade Pickles & Snacks!', 
welcome_message)

flash('Account created successfully! Please login.', 'success')
return redirect(url_for('index'))
except Exception as e:
    flash(f'Signup failed: {str(e)}', 'error')
return render_template('signup.html')
```

Description: define signup route to validate registration form fields, hash the user

password using Bcrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

- **Login Route (GET/POST):**

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        try:
            email = request.form['email']
            password = request.form['password']
            hashed_password = hash_password(password)

            # Check user in DynamoDB
            response = user_table.get_item(Key={'email': email})
            if 'Item' in response:
                stored_password = response['Item'].get('password')
                if stored_password == hashed_password:
                    session['user_email'] = email
                    session['user_name'] = response['Item'].get('name')
                    return redirect(url_for('home'))

                flash('Invalid email or password', 'error')
            except Exception as e:
                flash(f'Login failed: {str(e)}', 'error')
        return render_template('login.html')
```

Description: define login route to validate user credentials against DynamoDB, check the password using encryption, update the login count on successful authentication, and redirect users to the home page.

- **Index Route :**

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

Description: The index page serves as the homepage, welcoming users with an overview of your brand and guiding them to key sections like products, contact, or login.

- **Order route:**

```
@app.route('/order', methods=['GET', 'POST'])
def order():
    if request.method == 'POST':
        try:
            name = request.form['name']
            email = request.form['email']
            phone = request.form['phone']
            address = request.form['address']
            city = request.form['city']
            pincode = request.form['pincode']
            item = request.form['item']
            quantity = int(request.form['quantity'])
            notes = request.form.get('notes', '')
            order_id = str(uuid.uuid4())
            timestamp = datetime.utcnow().isoformat()

            # Save to DynamoDB with full order details
            order_table.put_item(Item={
                'order_id': order_id,
                'name': name,
                'email': email,
                'phone': phone,
                'address': address,
                'city': city,
                'pincode': pincode,
                'item': item,
                'quantity': quantity,
            })
```

```

def order():
    pincode = pincode,
    'item': item,
    'quantity': quantity,
    'notes': notes,
    'timestamp': timestamp,
    'status': 'pending',
    'total_amount': quantity * 100 # Sample pricing
})

# Send email notifications
customer_message = f"Dear {name},\n\nYour order has been placed
successfully!\n\nOrder ID: {order_id}\nItem: {item}\nQuantity: {quantity}
\n\nWe'll contact you soon for delivery details.\n\nThank you for choosing
Homemade Pickles & Snacks!"
admin_message = f"New Order Received!\n\nOrder ID: {order_id}\nCustomer:
{name}\nEmail: {email}\nPhone: {phone}\nItem: {item}\nQuantity: {quantity}
\nAddress: {address}, {city} - {pincode}\nNotes: {notes}"

send_email_notification(email, 'Order Confirmation - Homemade Pickles &
Snacks', customer_message)
send_email_notification(ADMIN_EMAIL, f'New Order - {order_id}', admin_message)

session['last_order_id'] = order_id
return redirect(url_for('success'))
except Exception as e:
    flash(f'Order processing failed: {str(e)}', 'error')
    return render_template('order.html')
return render_template('order.html')

```

Description: The order page allows customers to review their selected items, enter shipping details, and confirm payment to complete the purchase. It acts as the final step in the checkout process, ensuring a smooth and secure transaction.

- **Aws-info Route:**

```

@app.route('/aws-info')
def aws_info():
    """Display AWS service information"""
    try:
        # Get EC2 instance info
        instance_id = get_instance_info()

        # Get IAM role info
        try:
            sts = boto3.client('sts', region_name='us-east-1')
            identity = sts.get_caller_identity()
            account_id = identity['Account']
            role_arn = identity.get('Arn', 'No role attached')
        except:
            account_id = 'Unknown'
            role_arn = 'No role attached'

        info = {
            'instance_id': instance_id,
            'account_id': account_id,
            'role_arn': role_arn,
            'region': 'us-east-1'
        }
        return jsonify(info)
    except Exception as e:
        return jsonify({'error': str(e)}), 500

```

Description: The Aws-info route displays details about your website's integration with AWS, such as database status, storage usage, or service configurations. It helps admins monitor backend health and ensure smooth operation of the homemade pickles and snacks platform.

- **About route:**

```

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

```

Description: The about route displays a heartfelt introduction to your homemade pickles and snacks brand, sharing its origin, mission, and values. It builds trust and connection by telling your story and celebrating what makes your products unique.

- **Success route:**

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

Description: The success route displays a confirmation message after a successful order or form submission, reassuring users that their action was completed. It adds a friendly closing touch to the user journey with gratitude or next-step instructions.

- **Cart route:**

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

Description: The cart route displays all items a user has added for purchase, allowing them to review quantities and total cost before checkout. It acts as a shopping basket, giving users the option to update or remove products before finalizing their order.

- **Logout route:**

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

Description: The logout route securely ends the user's session, logging them out of their account. It helps protect personal information and ensures a fresh login is required for future activity.

- **Deployment Code:**

```

✓ if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5000))
    debug = os.environ.get('DEBUG', 'False').lower() == 'true'
    app.run(host='0.0.0.0', port=port, debug=debug)

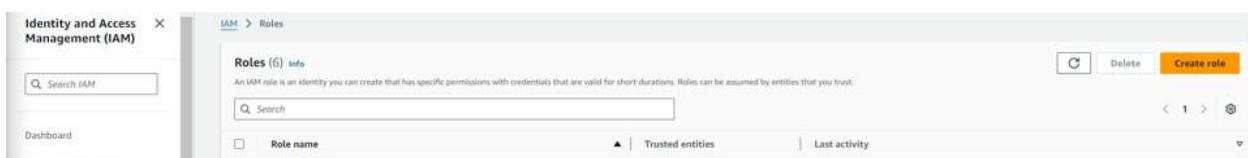
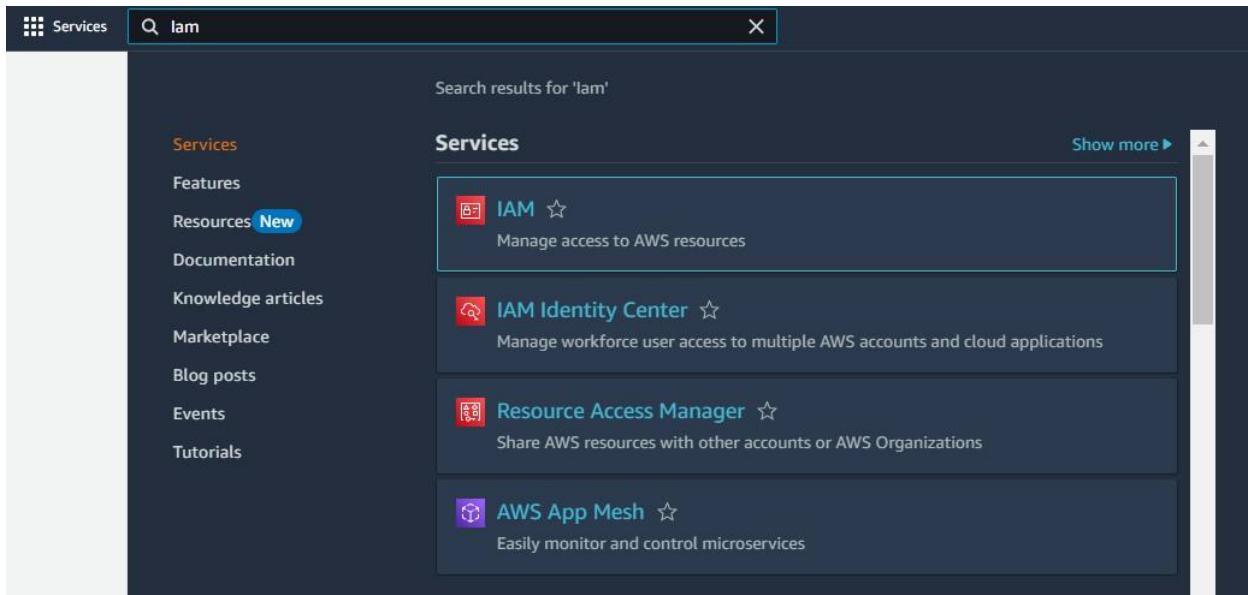
```

Description: The deployment code sets up and launches the homemade pickles and snacks website on a cloud platform like AWS or Heroku, ensuring public access. It typically includes configurations for web hosting, environment variables, and service integration to keep the app running smoothly.

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.



Select trusted entity

Trusted entity type

- AWS service: Allows AWS services (like S3, Lambda, or others) to perform actions in this account.
- AWS account: Allows entities in other AWS accounts (including you or a 3rd party) to perform actions on this account.
- SAML 2.0 federation: Allows users federated with SAML 2.0 from a corporate directory to perform actions on this account.
- Custom trust policy: Creates a custom JSON policy to enable others to perform actions on this account.

Use case

Allows an AWS service (like EC2 Lambda), or others, to perform actions on this account.

Service or use case: EC2

Choose a use case for the specified service:

Use case:

- EC2: Create or modify reserved instances, import and terminate instances on your behalf.
- EC2: Modify for AWS Systems Manager: Allows EC2 to manage AWS Lambda functions and Systems Manager on your behalf.
- EC2 Spot Fleet Role: Allows EC2 Spot Fleet to request and terminate spot instances on your behalf.
- EC2: Auto Scaling: Allows EC2 Auto Scaling to create and manage EC2 user roles on your behalf.
- EC2 - Spot Fleet Tagging: Allows EC2 Spot Fleet to tag instances and change tags to its intended instance on your behalf.
- EC2 - Spot Instances: Allows EC2 Spot Instances to launch and manage spot fleet instances on your behalf.
- EC2 - Spot Fleet: Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.
- EC2 Scheduled Instances: Allows EC2 Scheduled Instances to manage resources on your behalf.

Next Step

• 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.

Add permissions

Permissions policies (1/955) [Info](#)

Choose one or more policies to attach to your new role.

Filter by Type: All types | 2 matches

Policy name	Type
<input checked="" type="checkbox"/> AmazonDynamoDBFullAccess	AWS managed
<input type="checkbox"/> AmazonDynamoDBReadOnlyAccess	AWS managed

Set permissions boundary - optional

Next Step

Add permissions

Permissions policies (2/955) [Info](#)

Choose one or more policies to attach to your new role.

Filter by Type: All types | 5 matches

Policy name	Type
<input checked="" type="checkbox"/> AmazonSNSFullAccess	AWS managed
<input type="checkbox"/> AmazonSQSReadonlyAccess	AWS managed
<input type="checkbox"/> AmazonMSKRole	AWS managed
<input type="checkbox"/> AWSLambdaBeanstalkRoleSNS	AWS managed
<input type="checkbox"/> AWSLambdaDeviceDefenderPublishFindingsToSNSMitigationActions	AWS managed

Set permissions boundary - optional

Next Step

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
`sns_Dynamodb_role`

Description
Add a short description for this role.
Allows EC2 instances to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy

```
[{"Version": "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": "*"}, {"Effect": "Allow", "Principal": "AWSLambda", "Action": "sts:AssumeRole"}]}
```

Step 2: Add permissions

Permissions policy summary

Policy name	Type	Attached as
AmazonSNSFullAccess	AWS managed	Permissions policy
AmazonSQSFullAccess	AWS managed	Permissions policy

Step 3: Add tags

Add tags - optional

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with this resource.

[Add new tag](#)

You can add up to 50 more tags.

[Cancel](#) [Previous](#) [Create role](#)

[IAM](#) > [Roles](#) > [sns_Dynamodb_role](#) [Info](#) [Delete](#)

Allows EC2 instances to call AWS services on your behalf.

Summary

Creation date October 13, 2024, 23:06 (UTC+05:30)	ARN arn:aws:iam::557690616836:role/sns_Dynamodb_role	Instance profile ARN arn:aws:iam::557690616836:instance-profile/sns_Dynamodb_role
Last activity 6 days ago	Maximum session duration 1 hour	

[Edit](#)

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

Permissions policies (2) [Info](#)
You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	4
AmazonSNSFullAccess	AWS managed	2

[Search](#) [Filter by Type](#) [All types](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#) [17](#) [18](#) [19](#) [20](#) [21](#) [22](#) [23](#) [24](#) [25](#) [26](#) [27](#) [28](#) [29](#) [30](#) [31](#) [32](#) [33](#) [34](#) [35](#) [36](#) [37](#) [38](#) [39](#) [40](#) [41](#) [42](#) [43](#) [44](#) [45](#) [46](#) [47](#) [48](#) [49](#) [50](#) [51](#) [52](#) [53](#) [54](#) [55](#) [56](#) [57](#) [58](#) [59](#) [60](#) [61](#) [62](#) [63](#) [64](#) [65](#) [66](#) [67](#) [68](#) [69](#) [70](#) [71](#) [72](#) [73](#) [74](#) [75](#) [76](#) [77](#) [78](#) [79](#) [80](#) [81](#) [82](#) [83](#) [84](#) [85](#) [86](#) [87](#) [88](#) [89](#) [90](#) [91](#) [92](#) [93](#) [94](#) [95](#) [96](#) [97](#) [98](#) [99](#) [100](#) [101](#) [102](#) [103](#) [104](#) [105](#) [106](#) [107](#) [108](#) [109](#) [110](#) [111](#) [112](#) [113](#) [114](#) [115](#) [116](#) [117](#) [118](#) [119](#) [120](#) [121](#) [122](#) [123](#) [124](#) [125](#) [126](#) [127](#) [128](#) [129](#) [130](#) [131](#) [132](#) [133](#) [134](#) [135](#) [136](#) [137](#) [138](#) [139](#) [140](#) [141](#) [142](#) [143](#) [144](#) [145](#) [146](#) [147](#) [148](#) [149](#) [150](#) [151](#) [152](#) [153](#) [154](#) [155](#) [156](#) [157](#) [158](#) [159](#) [160](#) [161](#) [162](#) [163](#) [164](#) [165](#) [166](#) [167](#) [168](#) [169](#) [170](#) [171](#) [172](#) [173](#) [174](#) [175](#) [176](#) [177](#) [178](#) [179](#) [180](#) [181](#) [182](#) [183](#) [184](#) [185](#) [186](#) [187](#) [188](#) [189](#) [190](#) [191](#) [192](#) [193](#) [194](#) [195](#) [196](#) [197](#) [198](#) [199](#) [200](#) [201](#) [202](#) [203](#) [204](#) [205](#) [206](#) [207](#) [208](#) [209](#) [210](#) [211](#) [212](#) [213](#) [214](#) [215](#) [216](#) [217](#) [218](#) [219](#) [220](#) [221](#) [222](#) [223](#) [224](#) [225](#) [226](#) [227](#) [228](#) [229](#) [230](#) [231](#) [232](#) [233](#) [234](#) [235](#) [236](#) [237](#) [238](#) [239](#) [240](#) [241](#) [242](#) [243](#) [244](#) [245](#) [246](#) [247](#) [248](#) [249](#) [250](#) [251](#) [252](#) [253](#) [254](#) [255](#) [256](#) [257](#) [258](#) [259](#) [260](#) [261](#) [262](#) [263](#) [264](#) [265](#) [266](#) [267](#) [268](#) [269](#) [270](#) [271](#) [272](#) [273](#) [274](#) [275](#) [276](#) [277](#) [278](#) [279](#) [280](#) [281](#) [282](#) [283](#) [284](#) [285](#) [286](#) [287](#) [288](#) [289](#) [290](#) [291](#) [292](#) [293](#) [294](#) [295](#) [296](#) [297](#) [298](#) [299](#) [300](#) [301](#) [302](#) [303](#) [304](#) [305](#) [306](#) [307](#) [308](#) [309](#) [310](#) [311](#) [312](#) [313](#) [314](#) [315](#) [316](#) [317](#) [318](#) [319](#) [320](#) [321](#) [322](#) [323](#) [324](#) [325](#) [326](#) [327](#) [328](#) [329](#) [330](#) [331](#) [332](#) [333](#) [334](#) [335](#) [336](#) [337](#) [338](#) [339](#) [340](#) [341](#) [342](#) [343](#) [344](#) [345](#) [346](#) [347](#) [348](#) [349](#) [350](#) [351](#) [352](#) [353](#) [354](#) [355](#) [356](#) [357](#) [358](#) [359](#) [360](#) [361](#) [362](#) [363](#) [364](#) [365](#) [366](#) [367](#) [368](#) [369](#) [370](#) [371](#) [372](#) [373](#) [374](#) [375](#) [376](#) [377](#) [378](#) [379](#) [380](#) [381](#) [382](#) [383](#) [384](#) [385](#) [386](#) [387](#) [388](#) [389](#) [390](#) [391](#) [392](#) [393](#) [394](#) [395](#) [396](#) [397](#) [398](#) [399](#) [400](#) [401](#) [402](#) [403](#) [404](#) [405](#) [406](#) [407](#) [408](#) [409](#) [410](#) [411](#) [412](#) [413](#) [414](#) [415](#) [416](#) [417](#) [418](#) [419](#) [420](#) [421](#) [422](#) [423](#) [424](#) [425](#) [426](#) [427](#) [428](#) [429](#) [430](#) [431](#) [432](#) [433](#) [434](#) [435](#) [436](#) [437](#) [438](#) [439](#) [440](#) [441](#) [442](#) [443](#) [444](#) [445](#) [446](#) [447](#) [448](#) [449](#) [450](#) [451](#) [452](#) [453](#) [454](#) [455](#) [456](#) [457](#) [458](#) [459](#) [460](#) [461](#) [462](#) [463](#) [464](#) [465](#) [466](#) [467](#) [468](#) [469](#) [470](#) [471](#) [472](#) [473](#) [474](#) [475](#) [476](#) [477](#) [478](#) [479](#) [480](#) [481](#) [482](#) [483](#) [484](#) [485](#) [486](#) [487](#) [488](#) [489](#) [490](#) [491](#) [492](#) [493](#) [494](#) [495](#) [496](#) [497](#) [498](#) [499](#) [500](#) [501](#) [502](#) [503](#) [504](#) [505](#) [506](#) [507](#) [508](#) [509](#) [510](#) [511](#) [512](#) [513](#) [514](#) [515](#) [516](#) [517](#) [518](#) [519](#) [520](#) [521](#) [522](#) [523](#) [524](#) [525](#) [526](#) [527](#) [528](#) [529](#) [530](#) [531](#) [532](#) [533](#) [534](#) [535](#) [536](#) [537](#) [538](#) [539](#) [540](#) [541](#) [542](#) [543](#) [544](#) [545](#) [546](#) [547](#) [548](#) [549](#) [550](#) [551](#) [552](#) [553](#) [554](#) [555](#) [556](#) [557](#) [558](#) [559](#) [560](#) [561](#) [562](#) [563](#) [564](#) [565](#) [566](#) [567](#) [568](#) [569](#) [570](#) [571](#) [572](#) [573](#) [574](#) [575](#) [576](#) [577](#) [578](#) [579](#) [580](#) [581](#) [582](#) [583](#) [584](#) [585](#) [586](#) [587](#) [588](#) [589](#) [590](#) [591](#) [592](#) [593](#) [594](#) [595](#) [596](#) [597](#) [598](#) [599](#) [600](#) [601](#) [602](#) [603](#) [604](#) [605](#) [606](#) [607](#) [608](#) [609](#) [610](#) [611](#) [612](#) [613](#) [614](#) [615](#) [616](#) [617](#) [618](#) [619](#) [620](#) [621](#) [622](#) [623](#) [624](#) [625](#) [626](#) [627](#) [628](#) [629](#) [630](#) [631](#) [632](#) [633](#) [634](#) [635](#) [636](#) [637](#) [638](#) [639](#) [640](#) [641](#) [642](#) [643](#) [644](#) [645](#) [646](#) [647](#) [648](#) [649](#) [650](#) [651](#) [652](#) [653](#) [654](#) [655](#) [656](#) [657](#) [658](#) [659](#) [660](#) [661](#) [662](#) [663](#) [664](#) [665](#) [666](#) [667](#) [668](#) [669](#) [670](#) [671](#) [672](#) [673](#) [674](#) [675](#) [676](#) [677](#) [678](#) [679](#) [680](#) [681](#) [682](#) [683](#) [684](#) [685](#) [686](#) [687](#) [688](#) [689](#) [690](#) [691](#) [692](#) [693](#) [694](#) [695](#) [696](#) [697](#) [698](#) [699](#) [700](#) [701](#) [702](#) [703](#) [704](#) [705](#) [706](#) [707](#) [708](#) [709](#) [710](#) [711](#) [712](#) [713](#) [714](#) [715](#) [716](#) [717](#) [718](#) [719](#) [720](#) [721](#) [722](#) [723](#) [724](#) [725](#) [726](#) [727](#) [728](#) [729](#) [730](#) [731](#) [732](#) [733](#) [734](#) [735](#) [736](#) [737](#) [738](#) [739](#) [740](#) [741](#) [742](#) [743](#) [744](#) [745](#) [746](#) [747](#) [748](#) [749](#) [750](#) [751](#) [752](#) [753](#) [754](#) [755](#) [756](#) [757](#) [758](#) [759](#) [760](#) [761](#) [762](#) [763](#) [764](#) [765](#) [766](#) [767](#) [768](#) [769](#) [770](#) [771](#) [772](#) [773](#) [774](#) [775](#) [776](#) [777](#) [778](#) [779](#) [780](#) [781](#) [782](#) [783](#) [784](#) [785](#) [786](#) [787](#) [788](#) [789](#) [790](#) [791](#) [792](#) [793](#) [794](#) [795](#) [796](#) [797](#) [798](#) [799](#) [800](#) [801](#) [802](#) [803](#) [804](#) [805](#) [806](#) [807](#) [808](#) [809](#) [810](#) [811](#) [812](#) [813](#) [814](#) [815](#) [816](#) [817](#) [818](#) [819](#) [820](#) [821](#) [822](#) [823](#) [824](#) [825](#) [826](#) [827](#) [828](#) [829](#) [830](#) [831](#) [832](#) [833](#) [834](#) [835](#) [836](#) [837](#) [838](#) [839](#) [840](#) [841](#) [842](#) [843](#) [844](#) [845](#) [846](#) [847](#) [848](#) [849](#) [850](#) [851](#) [852](#) [853](#) [854](#) [855](#) [856](#) [857](#) [858](#) [859](#) [860](#) [861](#) [862](#) [863](#) [864](#) [865](#) [866](#) [867](#) [868](#) [869](#) [870](#) [871](#) [872](#) [873](#) [874](#) [875](#) [876](#) [877](#) [878](#) [879](#) [880](#) [881](#) [882](#) [883](#) [884](#) [885](#) [886](#) [887](#) [888](#) [889](#) [890](#) [891](#) [892](#) [893](#) [894](#) [895](#) [896](#) [897](#) [898](#) [899](#) [900](#) [901](#) [902](#) [903](#) [904](#) [905](#) [906](#) [907](#) [908](#) [909](#) [910](#) [911](#) [912](#) [913](#) [914](#) [915](#) [916](#) [917](#) [918](#) [919](#) [920](#) [921](#) [922](#) [923](#) [924](#) [925](#) [926](#) [927](#) [928](#) [929](#) [930](#) [931](#) [932](#) [933](#) [934](#) [935](#) [936](#) [937](#) [938](#) [939](#) [940](#) [941](#) [942](#) [943](#) [944](#) [945](#) [946](#) [947](#) [948](#) [949](#) [950](#) [951](#) [952](#) [953](#) [954](#) [955](#) [956](#) [957](#) [958](#) [959](#) [960](#) [961](#) [962](#) [963](#) [964](#) [965](#) [966](#) [967](#) [968](#) [969](#) [970](#) [971](#) [972](#) [973](#) [974](#) [975](#) [976](#) [977](#) [978](#) [979](#) [980](#) [981](#) [982](#) [983](#) [984](#) [985](#) [986](#) [987](#) [988](#) [989](#) [990](#) [991](#) [992](#) [993](#) [994](#) [995](#) [996](#) [997](#) [998](#) [999](#) [1000](#) [1001](#) [1002](#) [1003](#) [1004](#) [1005](#) [1006](#) [1007](#) [1008](#) [1009](#) [1010](#) [1011](#) [1012](#) [1013](#) [1014](#) [1015](#) [1016](#) [1017](#) [1018](#) [1019](#) [1020](#) [1021](#) [1022](#) [1023](#) [1024](#) [1025](#) [1026](#) [1027](#) [1028](#) [1029](#) [1030](#) [1031](#) [1032](#) [1033](#) [1034](#) [1035](#) [1036](#) [1037](#) [1038](#) [1039](#) [1040](#) [1041](#) [1042](#) [1043](#) [1044](#) [1045](#) [1046](#) [1047](#) [1048](#) [1049](#) <

The screenshot shows the AWS IAM Roles page. The left sidebar includes sections for Dashboard, Access management (User groups, Users, Roles, Policies, Identity providers, Account settings), and Access reports (Access Analyzer, Resource analysis, Unused access, Analyzer settings, Credential report, Organization activity). The main content area displays a table of roles:

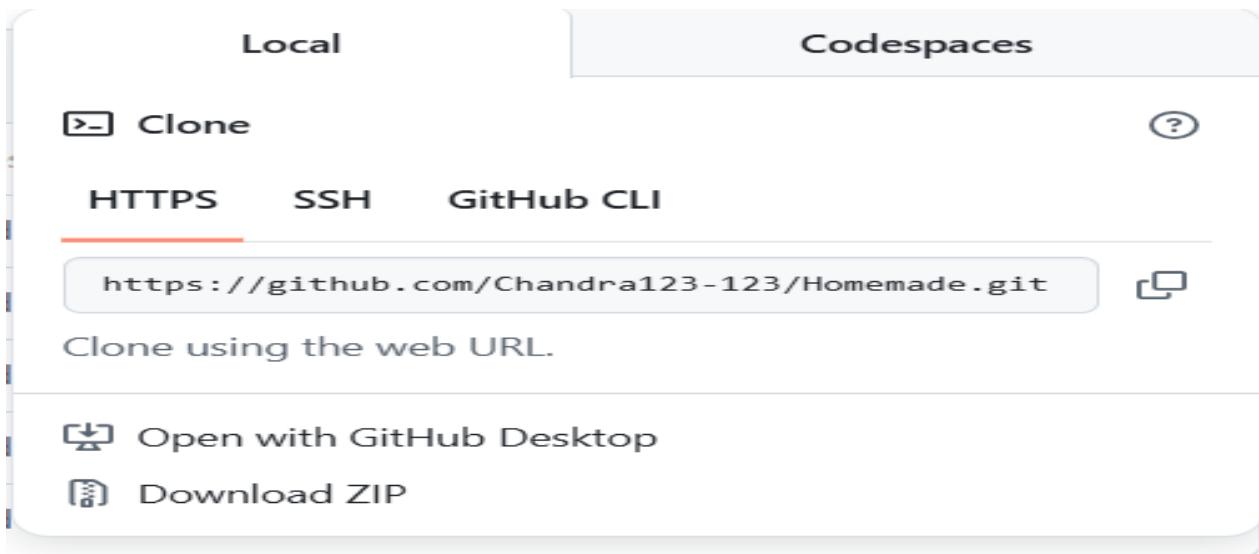
Role name	Trusted entities	Last activity
AWSServiceRoleForOrganizations	AWS Service: organizations (Service-Linked Role)	218 days ago
AWSServiceRoleForSSO	AWS Service: sso (Service-Linked Role)	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
DCEPrincipal-bunnytf	Account: 058264256896	-
<input checked="" type="checkbox"/> EC2_DynamoDB_Role	AWS Service: ec2	-
OrganizationAccountAccessRole	Account: 058264256896	1 hour ago
rsoaccount-new	Account: 058264256896	6 minutes ago

Below the table, there is a section titled "Roles Anywhere" with a "Manage" button.

Milestone 6: EC2 Instance Setup

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

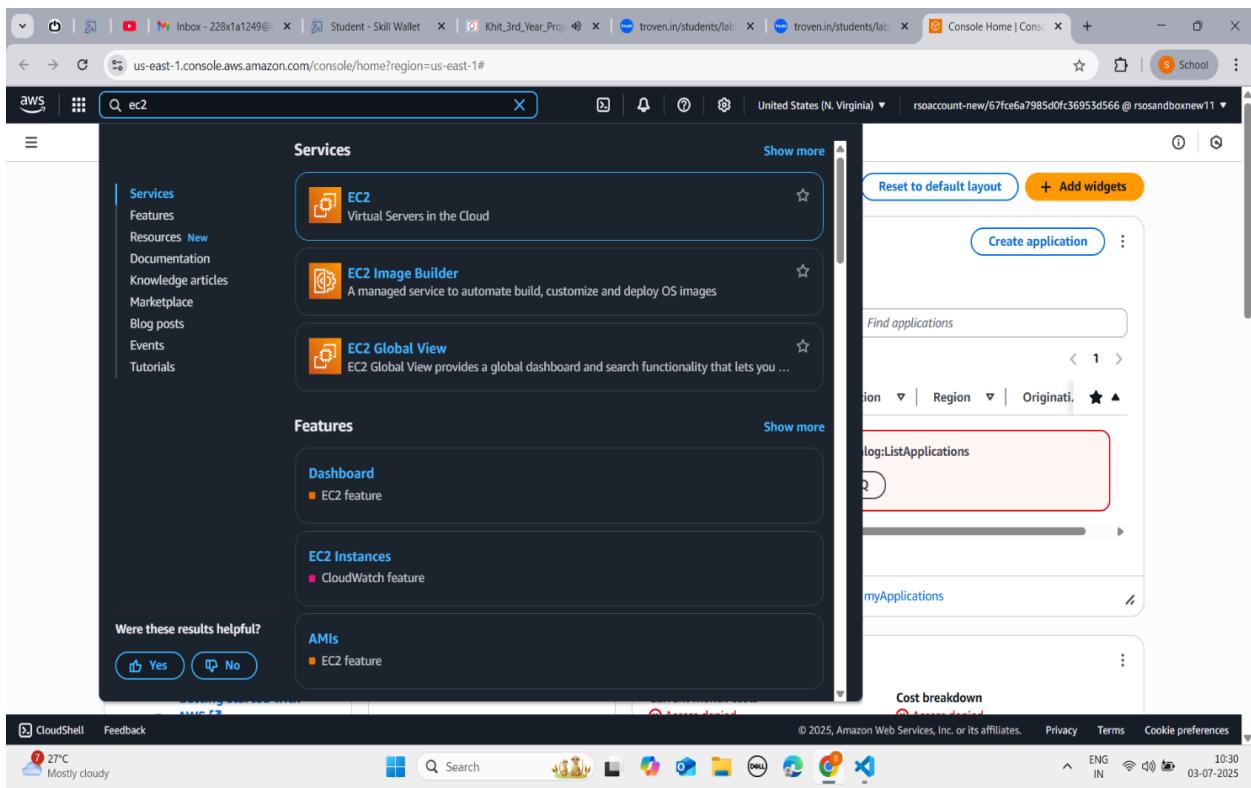
static/images	Delete static/images/images - Copy.jpg
templates	Add files via upload
README-AWS.md	Add files via upload
app.py	Add files via upload
aws-setup.py	Add files via upload
deploy.py	Add files via upload



- **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

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed. The main area has a header "Instances Info" with a search bar and filters for "Name", "Instance ID", "Instance state", "Instance type", "Status check", "Alarm status", "Availability Zone", and "Public IPv4". A message says "No instances" and "You do not have any instances in this region". There is a blue "Launch instances" button. Below it is a section titled "Select an instance". The bottom of the page includes a footer with CloudShell, Feedback, and weather information (27°C, Mostly cloudy). The footer also shows the year 2025, privacy terms, cookie preferences, and system status (ENG IN, 10:31, 03-07-2025).

The screenshot shows the "Launch an instance" wizard. Step 1: Name and tags. It asks, "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". There are "Do not show me this message again" and "Take a walkthrough" buttons. The main form has a "Name and tags" section with a "Name" field containing "HomeMadePickles" and a "Add additional tags" button. To the right is a "Summary" section showing "Number of instances: 1" and "Software Image (AMI): Amazon Linux 2023 AMI 2023.7.2...read more". The ami-05ffe5c48a9991133 link is visible.

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

Amazon Linux macOS Ubuntu Windows Red Hat ...

[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	Verified provider
64-bit (x86) ▾	uefi-preferred	ami-02b49a24cfb95941c	

- 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](#)



- **Activity 6.2:**Configure security groups for HTTP, and SSH access.

▼ Network settings [Info](#)

VPC - required [Info](#)

vpc-03cdc7b6f19dd7211
172.31.0.0/16

(default) ▾



Subnet [Info](#)

No preference



Create new subnet [\[+\]](#)

Auto-assign public IP [Info](#)

Enable



Additional charges apply when outside of free tier allowance

Firewall (security groups) [Info](#)

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

Create security group

Select existing security group

Security group name - required

launch-wizard

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

Description - required [Info](#)

launch-wizard created 2024-10-13T17:49:56.622Z

Inbound Security Group Rules

- ▼ Security group rule 1 (TCP, 22, 0.0.0.0/0)

Type Info	Protocol Info	Port range Info
ssh	TCP	22
Source type Info	Source Info	Description - optional Info
Anywhere	Add CIDR, prefix list or security 0.0.0.0/0 X	e.g. SSH for admin desktop
- ▼ Security group rule 2 (TCP, 80, 0.0.0.0/0)

Type Info	Protocol Info	Port range Info
HTTP	TCP	80
Source type Info	Source Info	Description - optional Info
Custom	Add CIDR, prefix list or security 0.0.0.0/0 X	e.g. SSH for admin desktop
- ▼ Security group rule 3 (TCP, 5000, 0.0.0.0/0)

Type Info	Protocol Info	Port range Info
Custom TCP	TCP	5000
Source type Info	Source Info	Description - optional Info
Custom	Add CIDR, prefix list or security 0.0.0.0/0 X	e.g. SSH for admin desktop

Add security group rule

The screenshot shows the AWS EC2 Instances page with a green success message: "Successfully initiated launch of instance (i-04abb225b2d404411)". Below the message, there's a "Launch log" section with a link to "View log". Under "Next Steps", there are several options: "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", and "Manage CloudWatch alarms". At the bottom, there are links for "CloudShell", "Feedback", and system status indicators.

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. On the left, there's a navigation sidebar with options like Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, and Elastic Block Store. The main content area has a green success message: "Successfully attached EC2_DynamoDB_Role to instance i-04abb225b2d404411". Below this, a table lists one instance: "HomeMadePic... i-04abb225b2d404411" (Running, t2.micro, 2/2 checks passed). The instance details page for "i-04abb225b2d404411 (HomeMadePickles)" is shown, with tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. Under Details, it shows the Instance ID (i-04abb225b2d404411), Public IPv4 address (34.235.165.159), Private IPv4 address (172.31.28.8), and Public DNS (ec2-34-235-165-139.compute-1.amazonaws.com). The status is Running. The bottom of the screen shows the AWS footer with links for CloudShell, Feedback, and various icons, along with system information like weather (27°C, Mostly cloudy), time (11:34), date (03-07-2025), and language (ENG IN).

- Now connect the EC2 with the files

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 IPv4 address and a VPC endpoint.

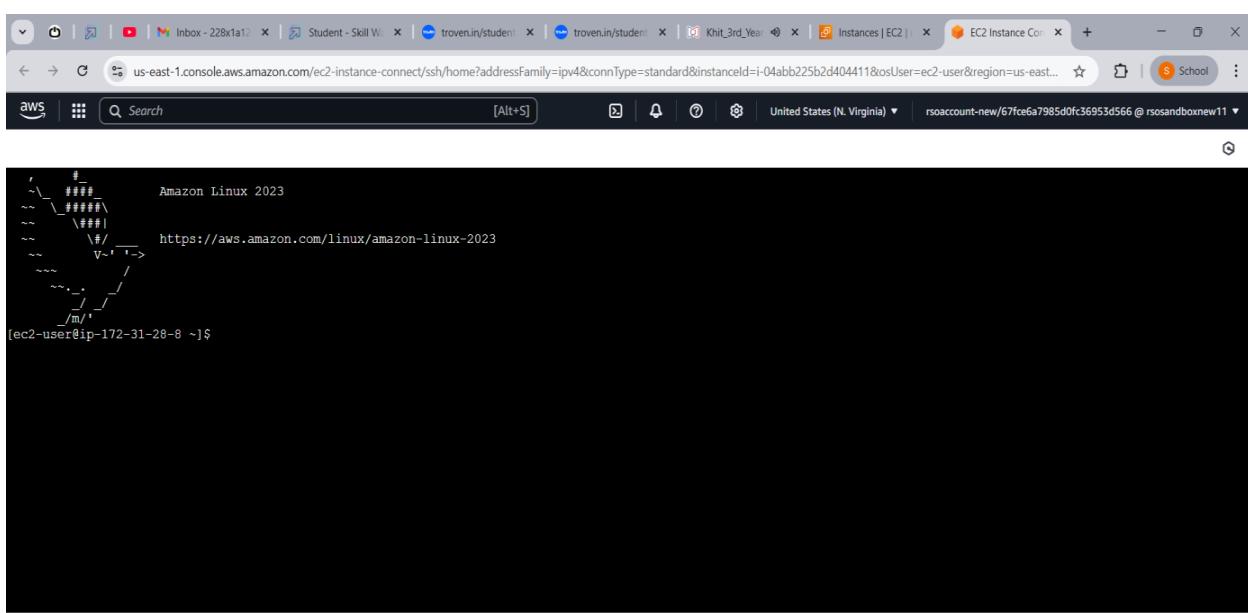
Public IPv4 address
 13.200.229.59

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.
 X

 **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**



i-04abb225b2d404411 (HomeMadePickles)

Public IPs: 34.235.165.139 Private IPs: 172.31.28.8



```

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

[ec2-user@ip-172-31-28-8 ~]$ sudo yum update -y
sudo yum install python3 git
sudo pip3 install flask boto3
sudo yum install python3-pip -y
pip3 install flask
pip3 install boto3
pip3 install python-dotenv
Amazon Linux 2023 Kernel Livepatch repository
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:00:03 ago on Thu Jul 3 07:06:08 2025.
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.

Package                                         Architecture   Version           Repository      Size
Installing:
i-04abb225b2d404411 (HomeMadePickles)          x86_64        3.9.23-1.amzn2023.0.1.x86_64
PublicIPs: 34.235.165.139  PrivateIPs: 172.31.28.8

CloudShell  Feedback  © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
30°C  Mostly cloudy  ENG IN  13:35  03-07-2025

```

```

Install 8 Packages

Total download size: 7.5 M
Installed size: 37 M
Is this ok [y/N]: y
Downloading Packages:
(1/8): git-2.47.1-1.amzn2023.0.3.x86_64.rpm
(2/8): perl-Error-0.17029-5.amzn2023.0.2.noarch.rpm
(3/8): git-core-doc-2.47.1-1.amzn2023.0.3.noarch.rpm
(4/8): perl-File-Find-1.37-477.amzn2023.0.7.noarch.rpm
(5/8): perl-Git-2.47.1-1.amzn2023.0.3.noarch.rpm
(6/8): perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64.rpm
(7/8): git-core-2.47.1-1.amzn2023.0.3.x86_64.rpm
(8/8): perl-lib-0.65-477.amzn2023.0.7.x86_64.rpm

```

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
sudo yum install python3-pip -y

```

pip3 install Flask

pip3 install boto3

pip3 install python-dotenv

Verify Installations:

```
flask --version git --version
```

Activity 7.2:Clone Your Flask Project from GitHub

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

- Run:** 'git clone https://github.com/Chandra123-123/H

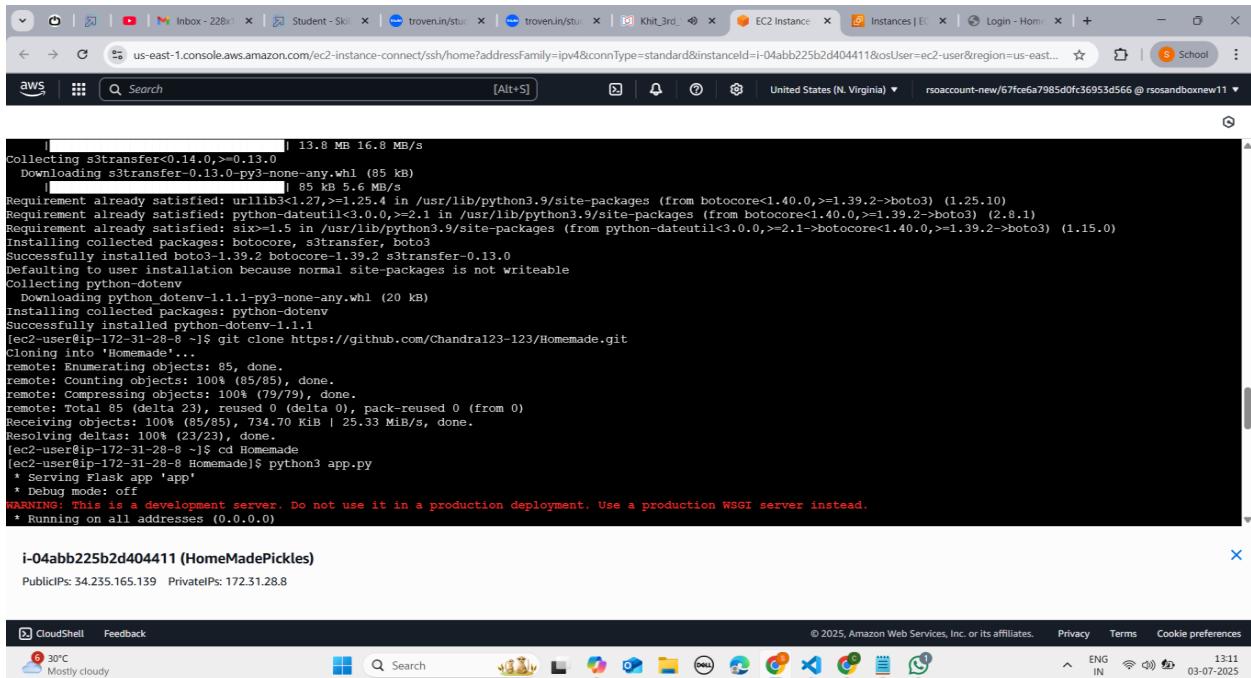
- This will download your project to the EC2 instance.

directory, run the

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

Run the Flask Application

Run the Flask Application



```
[ec2-user@ip-172-31-28-8 ~]$ pip install boto3 s3transfer
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.2->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.2->boto3) (2.8.1)
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,>=1.39.2->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.2 botocore-1.39.2 s3transfer-0.13.0
Defaulting to user installation because normal site-packages is not writeable
Collecting python-dotenv
  Downloading python_dotenv-1.1.1-py3-none-any.whl (20 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.1.1
[ec2-user@ip-172-31-28-8 ~]$ git clone https://github.com/Chandral23-123/Homemade.git
Cloning into 'Homemade'...
remote: Enumerating objects: 85, done.
remote: Counting objects: 100% (85/85), done.
remote: Compressing objects: 100% (79/79), done.
remote: Total 85 (delta 23), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (85/85), 734.70 KiB | 25.33 MiB/s, done.
Resolving deltas: 100% (23/23), done.
[ec2-user@ip-172-31-28-8 ~]$ cd Homemade
[ec2-user@ip-172-31-28-8 Homemade]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
```

i-04abb225b2d404411 (HomeMadePickles)

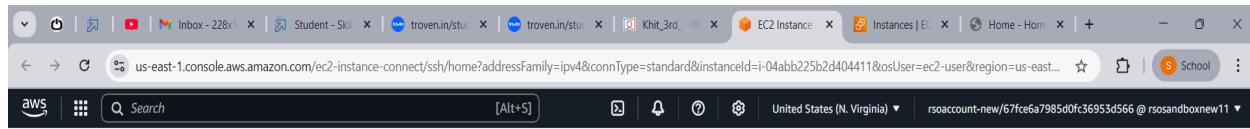
PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8



```
[ec2-user@ip-172-31-28-8 ~]$ cd Homemade
[ec2-user@ip-172-31-28-8 Homemade]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.28.8:5000
Press CTRL+C to quit
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET / HTTP/1.1" 200 -
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET /favicon.ico HTTP/1.1" 404 -
[]
```

i-04abb225b2d404411 (HomeMadePickles)

PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8



```
'~\_\#\#\#_          Amazon Linux 2023
~~ \#\#\#\`\
~~ \#\#\#
~~ \|#_
~~ v-,_->
~~ /_
~~ ._/
~~ /_/
~~ /m/
[ec2-user@ip-172-31-28-8 ~]$ sudo yum update -y
sudo yum install python3 git
sudo pip3 install flask boto3
sudo yum install python3-pip -y
pip3 install Flask
pip3 install boto3
pip3 install python-dotenv
Amazon Linux 2023 Kernel Livepatch repository
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:00:03 ago on Thu Jul 3 07:06:08 2025.
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
=====
 Package           Architecture      Version       Repository      Size
=====
Installing:
i-04abb225b2d404411 (HomeMadePickles)
```

PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8



```
[ec2-user@ip-172-31-28-8 ~]$ cd Homemade
[ec2-user@ip-172-31-28-8 Homemade]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.28.8:5000
Press CTRL+C to quit
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET / HTTP/1.1" 200 -
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET /favicon.ico HTTP/1.1" 404 -
```

i-04abb225b2d404411 (HomeMadePickles)

PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8

Verify the Flask app is running: <http://34.235.165.139>

- Run the Flask app on the EC2 instance

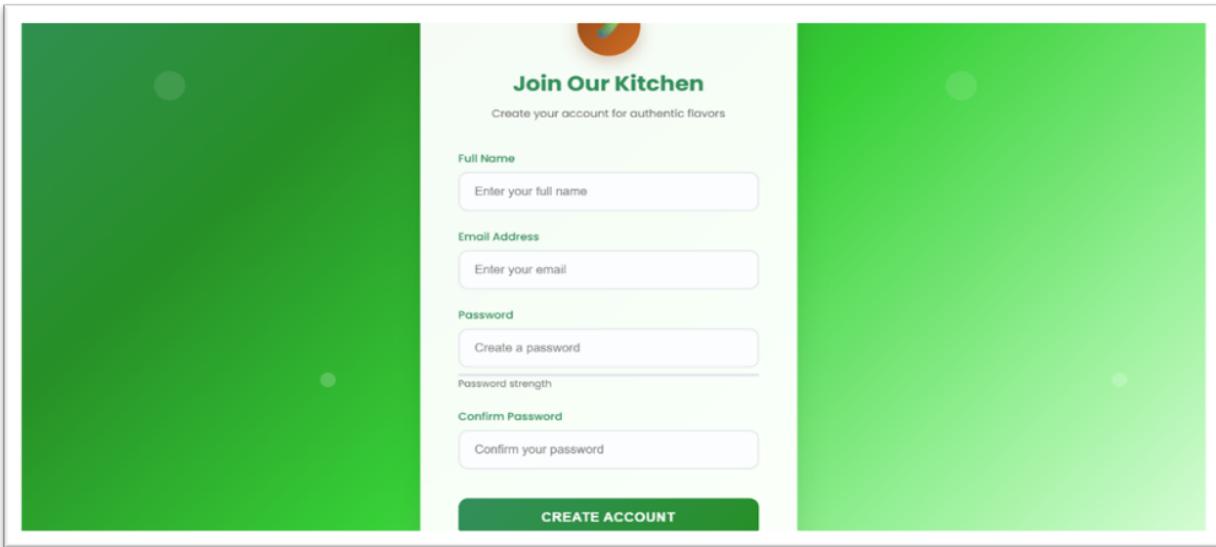
Access the website through:

Public IPs: <http://34.235.165.139>

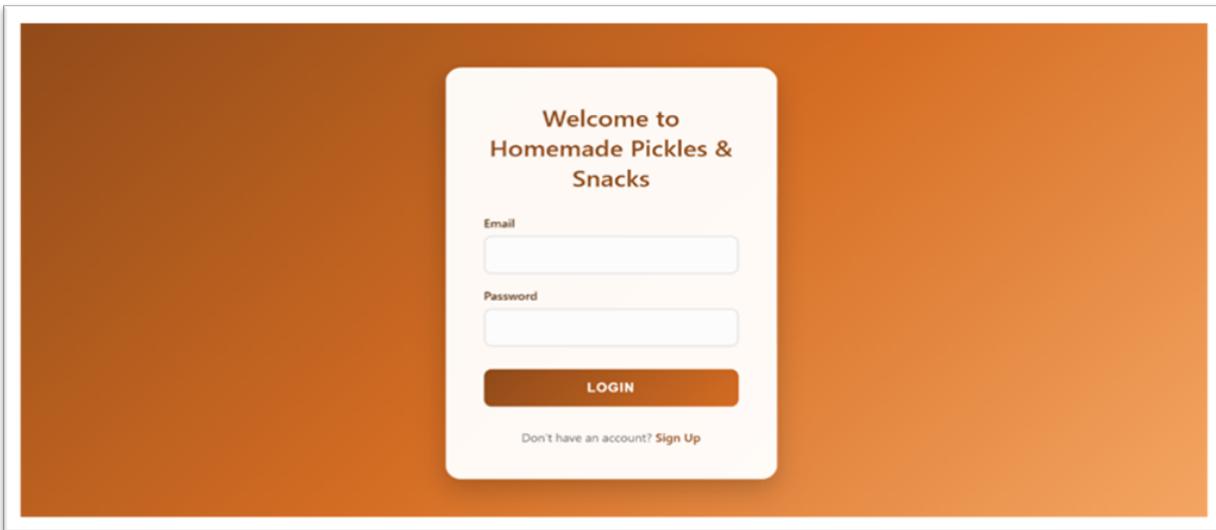
Milestone 8: Testing and Deployment

- **Activity 8.1:** Conduct functional testing to verify user signup, login, pickles requests, and notifications.

Signup Page :



Login Page :



Home page :

The screenshot shows the homepage of a website for "Homemade Pickles & Snacks". The header features a brown banner with the title "Homemade Pickles & Snacks" and a subtitle "Authentic tastes from our kitchen to yours!". Below the banner is a navigation bar with links for "About", "Contact", and "Cart". The main content area has a green background and is titled "Featured Products". It contains three sections: "VEG_PICKLES" (with an image of various jars), "NON_VEG_PICKLES" (with an image of four different non-vegan pickle varieties labeled "PRAWN", "KALGUDI", and "FISH"), and "SNACKS" (with an image of various South Indian snacks). Each section includes a brief description.

Featured Products

VEG_PICKLES

KERALA FISH PICKLE
TOMATO PICKLE
ANCHOVY PICKLE / KOZHIKA
MANGO MURUKU / PACHAKAM
MENSAH / FISH PICKLE (MATHRI)
CHICKEN PICKLE
DAIRY FREE PICKLE
CUT MANGO PICKLE
DRAMA MANGA PICKLE

Spicy and tangy chunks packed with tradition.

NON_VEG_PICKLES

Non-Veg
PRAWN
KALGUDI
FISH

Rich flavorful, fully loaded with masalas spices and fiery--perfect

SNACKS

Crunchy South Indian snack & juicy and delicious Sweets & seasoned to

About Us page:

The screenshot shows the "About Us" page of the website. The header is identical to the homepage, featuring the title "Homemade Pickles & Snacks" and the subtitle "Authentic tastes from our kitchen to yours!". The navigation bar includes links for "Home", "Veg Pickles", "Non-Veg Pickles", "Snacks", and "Contact". The main content area is titled "Our Story" and contains a paragraph about the company's history and the care taken in their products. Below this, there are two smaller paragraphs: one about the ingredients and techniques used, and another about the company's mission to satisfy taste buds and warm hearts.

Our Story

Born in the heart of a small kitchen, Homemade Pickles & Snacks was crafted out of love for authentic flavors passed down through generations. What began as weekend experiments with age-old recipes soon blossomed into a passion for preserving homemade taste in every jar and packet.

Every pickle we ferment and every snack we fry is made using carefully sourced ingredients and traditional techniques, ensuring freshness, hygiene, and a bold burst of nostalgia in every bite. From spicy mango pickles to crunchy murukulu, we bring you the essence of grandma's kitchen—packaged with care.

Whether you're craving a spicy sidekick to your meal or a savory companion for your chai break, we've got something to tingle your taste buds and warm your heart.

Contact Page:

The screenshot shows a contact form titled "Contact Us" on a website. The header reads "Homemade Pickles & Snacks" and "We'd love to hear from you!". The navigation bar includes links for "Home" and "About". The contact form fields are labeled "Your Name", "Your Email", and "Your Message", each with an associated input field.

Veg-Pickles page:

The screenshot displays a section titled "Our Vegetarian Pickles" on the website. It features three pickle varieties: "Andhra Mango Pickle", "Gongura Pickle", and "Lemon Pickle", each with a small image and a brief description. The header of this section also includes the tagline "Purely Plant-Based, Bursting with Flavor".

- Andhra Mango Pickle**
Raw mango chunks in fiery red chili-garlic masala.
- Gongura Pickle**
Tangy gongura leaves pickled with spices & tradition.
- Lemon Pickle**
Zesty lemons steeped in mustard, fenugreek & oil.

Non-Veg-Pickles page:

Homemade Pickles & Snacks
Bold, Spicy, and Packed with Flavor!

Home Veg Pickles Snacks About Cart Contact

Our Non-Veg Pickles



Chicken Pickle
Juicy chicken chunks marinated in a fiery spice blend, slow-cooked to perfection.



Mutton Pickle
Tender mutton pieces infused with aromatic masala and traditional spices.



Prawn Pickle
Succulent prawns pickled with mustard, red chili, and a dash of vinegar.

Snacks Page :

Homemade Pickles & Snacks
South Indian Crunch & Munch Specials

Home Veg Pickles Non-Veg Pickles About Cart Contact

Our Handmade Snacks



Masala Murukulu
Spicy and crunchy spirals made from rice flour and urad dal.



Karam Boondi
Golden crispy gram flour pearls seasoned with bold flavors.



Mixture
A signature blend of sev, peanuts, curry leaves & spice.

Cart Page:

The screenshot shows a web page titled "Homemade Pickles & Snacks" with a brown header bar. Below the header, there are navigation links for "Home" and "Contact". The main content area displays a table of items in the cart:

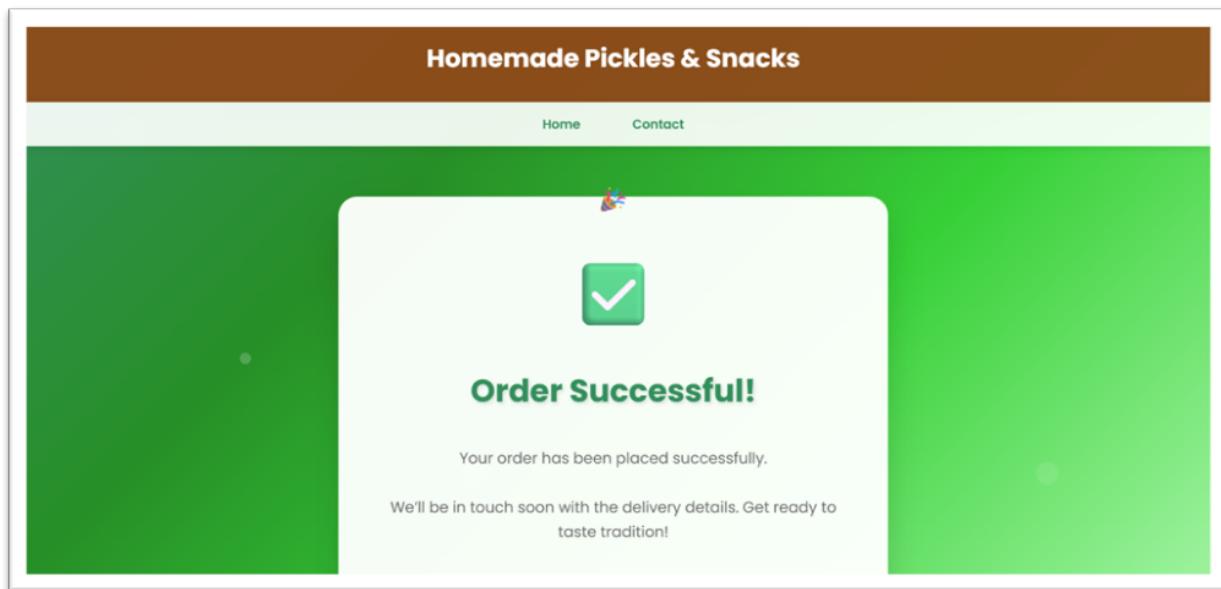
PRODUCT	QUANTITY	UNIT PRICE	SUBTOTAL
Amla Pickle	- 1 +	₹60	₹60
Gongura Pickle	- 1 +	₹50	₹50
Mutton Pickle	- 1 +	₹120	₹120
Kakinada Kaja	- 1 +	₹80	₹80
Karam Boondi	- 1 +	₹60	₹60

A green button at the bottom right of the table area says "Total: ₹370".

Check-out Page:

The screenshot shows a web page titled "Homemade Pickles & Snacks" with a brown header bar. Below the header, there are navigation links for "Home", "Cart", and "Contact". The main content area has a title "Billing & Shipping Details" and three input fields for "Full Name", "Email", and "Phone Number".

Success Page:



Your order has been placed successfully! We Will get backto you soon

Exit:

Session Ended

Please close this tab.

Conclusion:

The Homemade Pickles and Snacks Website has been successfully developed and deployed using a robust cloud-native architecture to ensure high performance, scalability, and customer satisfaction. By leveraging **AWS EC2** for reliable hosting, **DynamoDB** for secure product and order data management, and **SNS** for real-time customer and staff notifications, the platform delivers a seamless end-to-end experience for home made food lovers. This system addresses the need for efficient online access to traditional food offerings, allowing customers to conveniently browse, order, and track their favourite pickles and snacks. The cloud infrastructure enables the site to scale effortlessly during high-demand periods like festive seasons or promotions, without compromising responsiveness or user experience.

The integration of **Flask with AWS services** ensures that backend operations—such as order processing, inventory updates, and customer messaging—function smoothly in real time. Thorough testing has validated the stability of all core features, from product browsing and secure checkout to order confirmation notifications. In conclusion, this website serves as a modern platform that blends tradition with technology, providing a delightful shopping experience while streamlining business operations. It stands as a strong example of how cloud-based solutions can elevate local, handmade products to a broader digital audience.

