

# 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. It features rich visuals, engaging descriptions, and an easy-to-navigate layout that highlights the authenticity and flavour of each product. Customers can explore product categories like spicy pickles, sweet preserves, and crunchy munchies, all made with traditional recipes. The site includes a secure shopping cart system and responsive design for mobile accessibility. A blog section shares homemade tips, recipe stories, and cultural food traditions to deepen customer connection. Users can rate products, leave reviews, and sign up for seasonal bundle offers. SEO-optimized content boosts discoverability across search engines. It's a Flavourful blend of heritage, tech, and convenience. If you're planning to build or document this site further, I can help structure sections like product listings, customer experience features, or a compelling homepage narrative.

## 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.

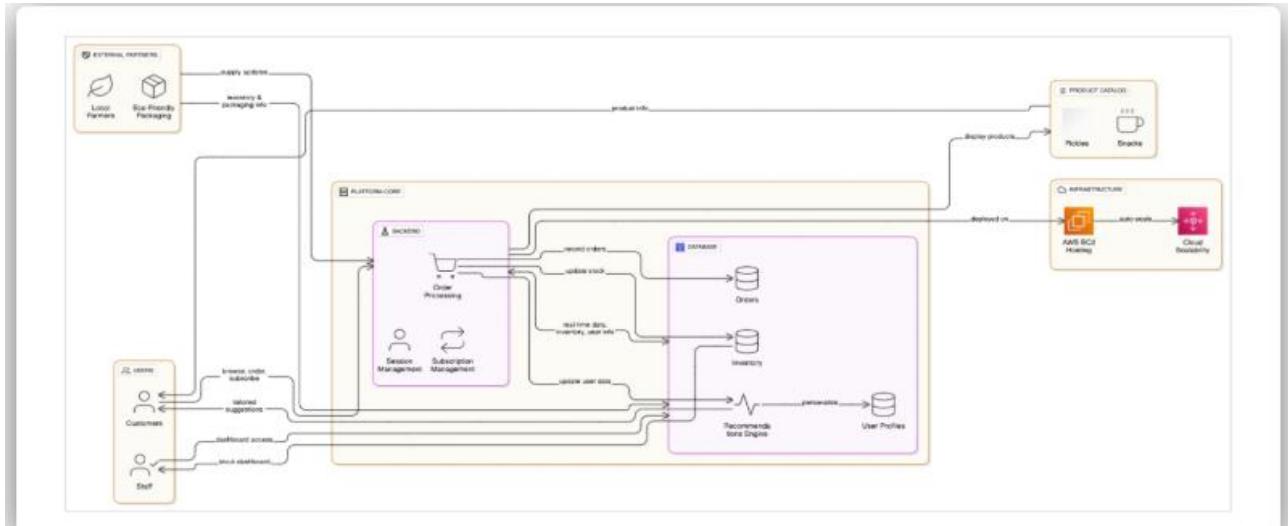
## Scenario 2: Seamless Order Notifications for Kitchen Staff and Customers

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.

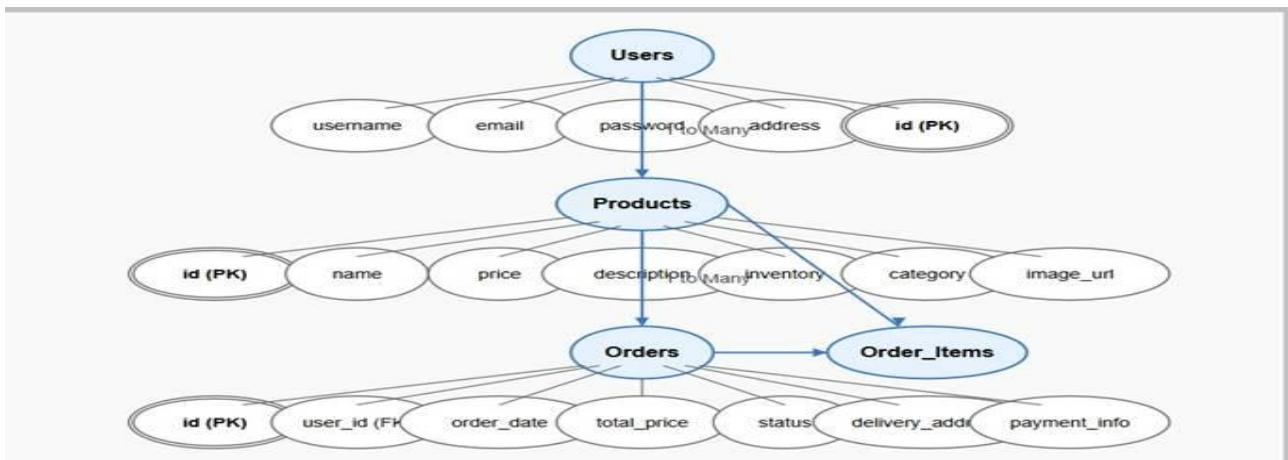
## Scenario 3: Real-Time Inventory and Product Availability

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

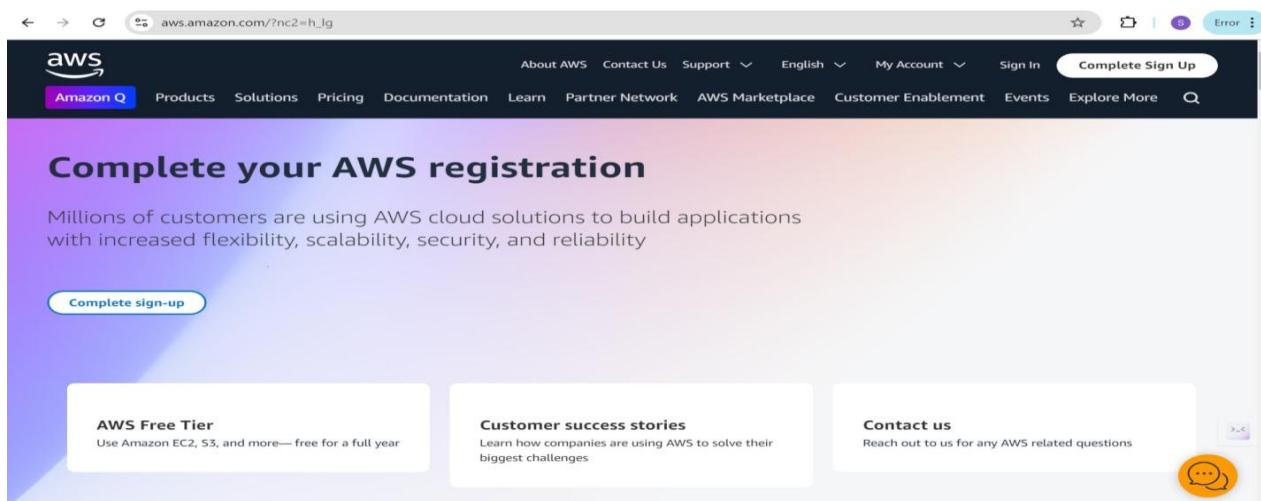
**Activity 7.2:** Run the Flask App

## 8. Testing and Deployment

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

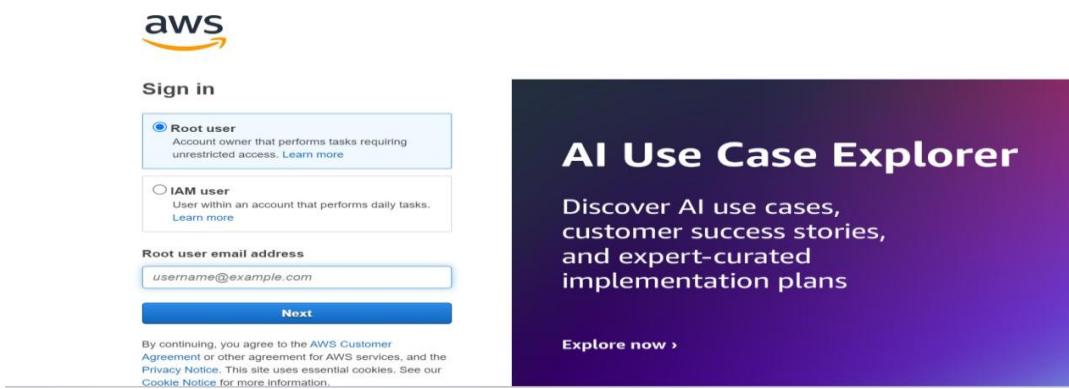
### Milestone 1: AWS Account Setup and Login

- **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](#).

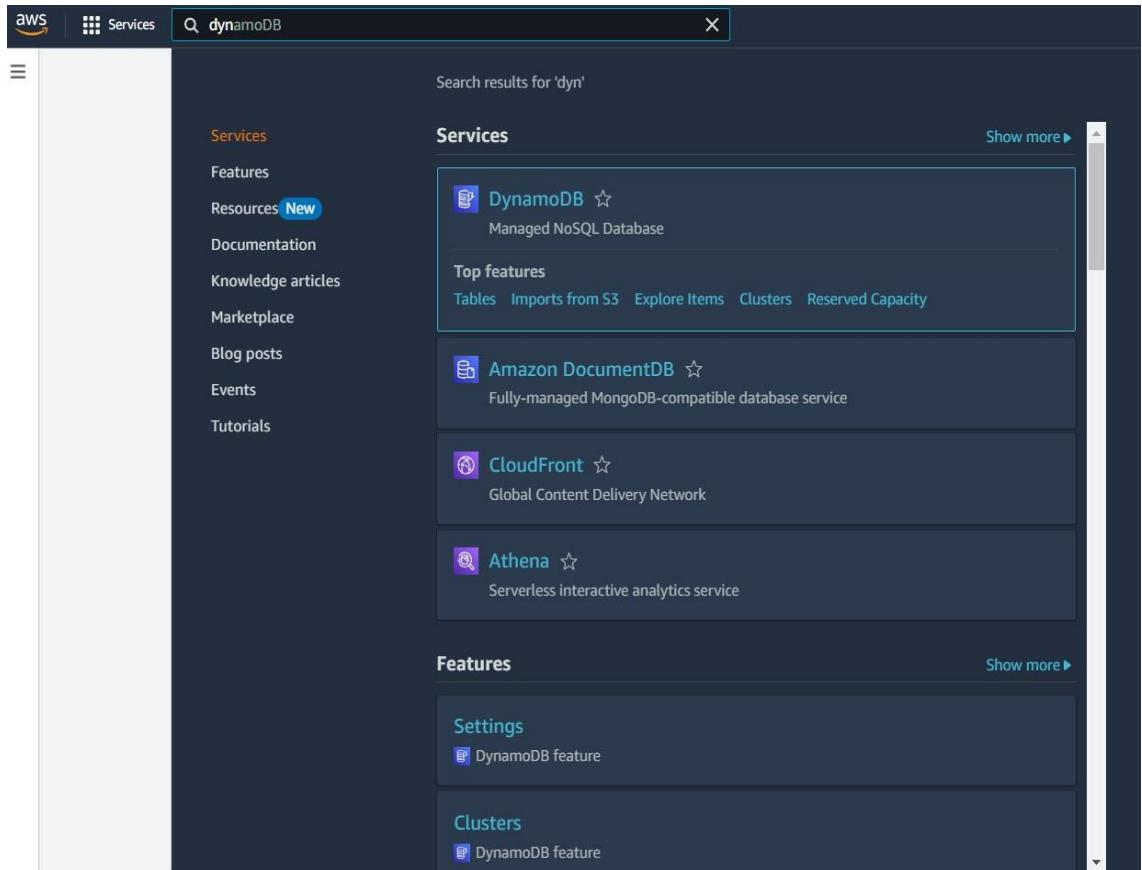


The screenshot shows two side-by-side pages. On the left is the 'Sign in' page for AWS. It has a 'Root user' radio button selected, with a tooltip 'Account owner that performs tasks requiring unrestricted access. Learn more'. There's also an 'IAM user' radio button with a tooltip 'User within an account that performs daily tasks. Learn more'. Below these is a 'Root user email address' input field containing 'username@example.com' and a 'Next' button. At the bottom, there's a small note about cookie consent. On the right is the 'AI Use Case Explorer' page. It features a dark purple gradient background with the title 'AI Use Case Explorer' in white. Below the title is the text 'Discover AI use cases, customer success stories, and expert-curated implementation plans'. At the bottom, there's a 'Explore now >' button.

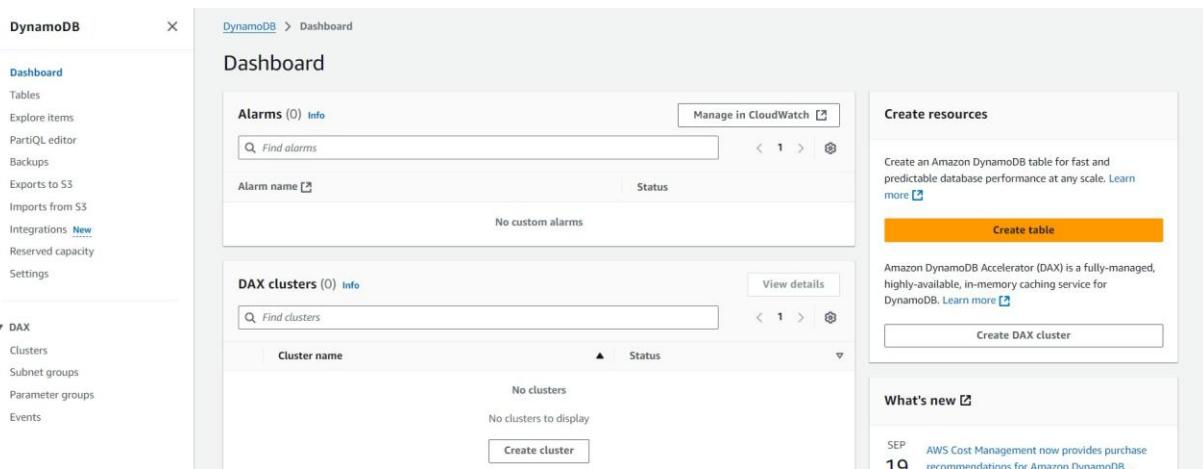
## Milestone 2: DynamoDB Database Creation and Setup

- **Activity 2.1: Navigate to the DynamoDB**

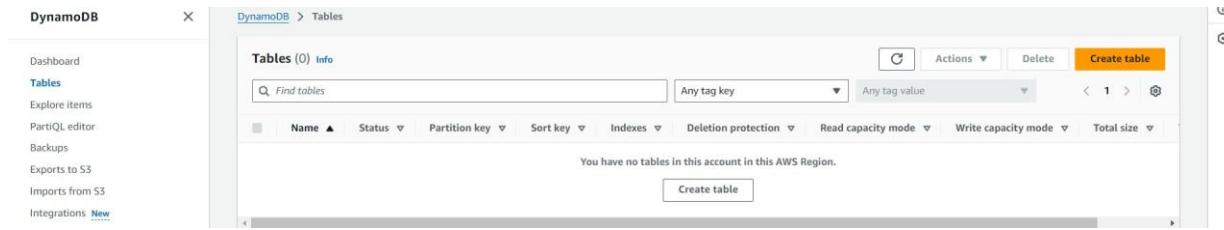
- In the AWS Console, navigate to DynamoDB and click on create tables.



The screenshot shows the AWS Services search results page. The search bar at the top contains the query 'dynamodb'. The results are categorized under 'Services' and 'Features'. Under 'Services', 'DynamoDB' is listed as a Managed NoSQL Database. Other services listed include Amazon DocumentDB, CloudFront, and Athena. Under 'Features', there are sections for 'Settings' (DynamoDB feature) and 'Clusters' (DynamoDB feature). A 'Show more ▶' link is visible at the top right of each category section.

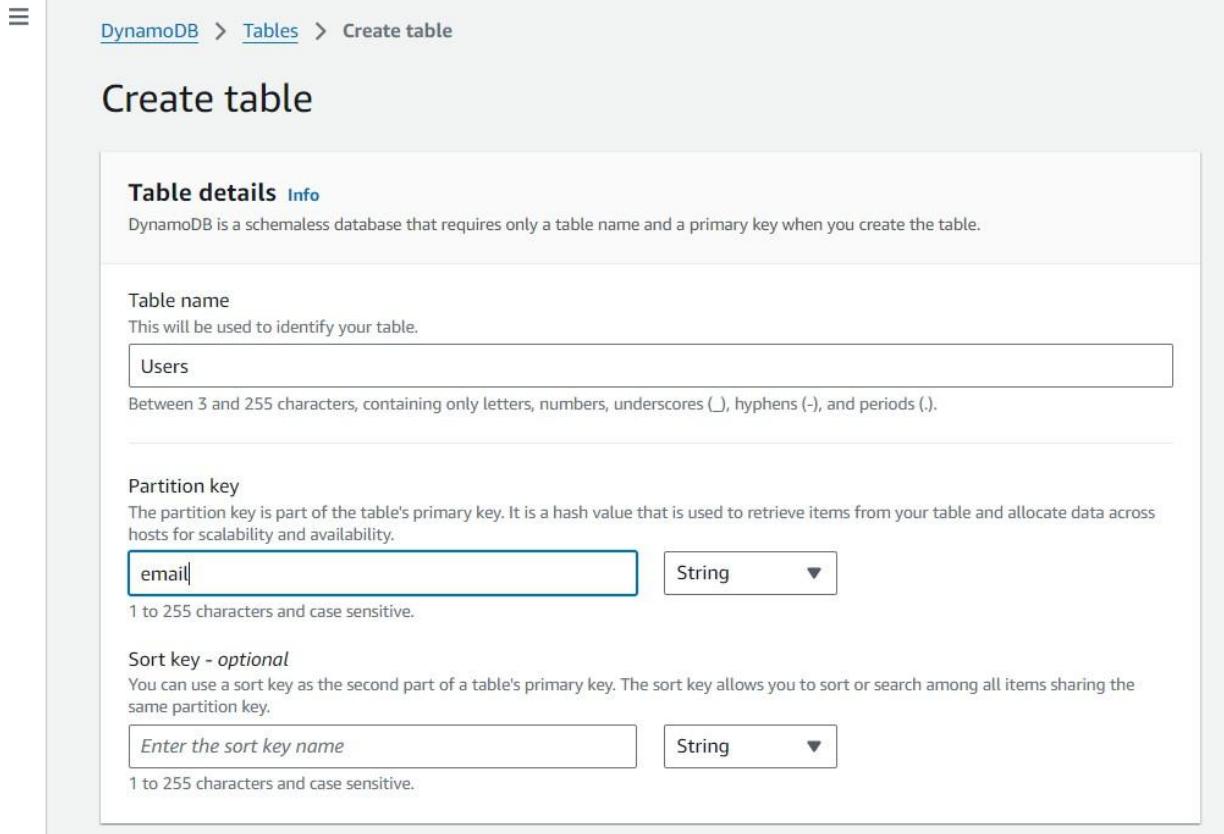


The screenshot shows the AWS DynamoDB Dashboard. The left sidebar includes links for Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, Settings, DAX (Clusters, Subnet groups, Parameter groups, Events), and a 'Create resources' button. The main dashboard area displays sections for Alarms (0), DAX clusters (0), and a 'Create resources' section. The 'Create resources' section includes a 'Create table' button and information about Amazon DynamoDB Accelerator (DAX). A 'What's new' section at the bottom right shows a recent update about AWS Cost Management providing purchase recommendations for Amazon DynamoDB.



The screenshot shows the AWS DynamoDB 'Tables' page. On the left, there is a sidebar with options like 'Dashboard', 'Tables' (which is selected), 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', and 'Integrations'. The main area has a heading 'Tables (0) Info' with a search bar and filters for 'Any tag key' and 'Any tag value'. A message says 'You have no tables in this account in this AWS Region.' At the bottom right of the main area is a large orange 'Create table' button.

- **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', it says '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 shows 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 shows 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 sections have notes at the bottom: 'Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).'  
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 Users table was created successfully.

| Tables (1) <a href="#">Info</a> |       |        |               |          |         |                     |                    |                     |            |  |
|---------------------------------|-------|--------|---------------|----------|---------|---------------------|--------------------|---------------------|------------|--|
|                                 | Name  | Status | Partition key | Sort key | Indexes | Deletion protection | Read capacity mode | Write capacity mode | Total size |  |
| <input type="checkbox"/>        | Users | Active | email (\$)    | -        | 0       | Off                 | Provisioned (5)    | Provisioned (5)     | 0 bytes    |  |

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

DynamoDB > Tables > Create table

## 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.

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.

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 items in the same partition key.

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.

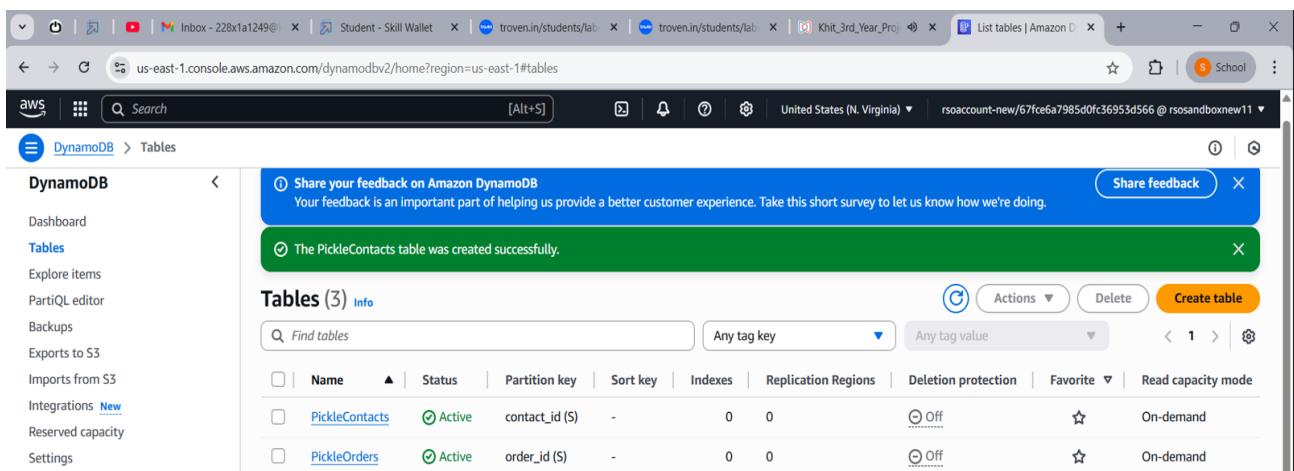
No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

[Cancel](#)

[Create table](#)



The screenshot shows the AWS DynamoDB console interface. On the left, there is a navigation sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, and Settings. The 'Tables' option is selected. The main content area shows a message about sharing feedback on DynamoDB. Below that, a green success message states 'The PickleContacts table was created successfully.' The 'Tables (3)' section lists three tables: 'PickleContacts' (Active, contact\_id \$S, - RCU, 0 WCU, Off, On-demand) and 'PickleOrders' (Active, order\_id \$S, - RCU, 0 WCU, Off, On-demand). There are buttons for Actions, Delete, and Create table.

|                            |                          |     |
|----------------------------|--------------------------|-----|
| 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](#)
X
 ⓘ The PickleContacts table was created successfully.
X

### Tables (3) [Info](#)

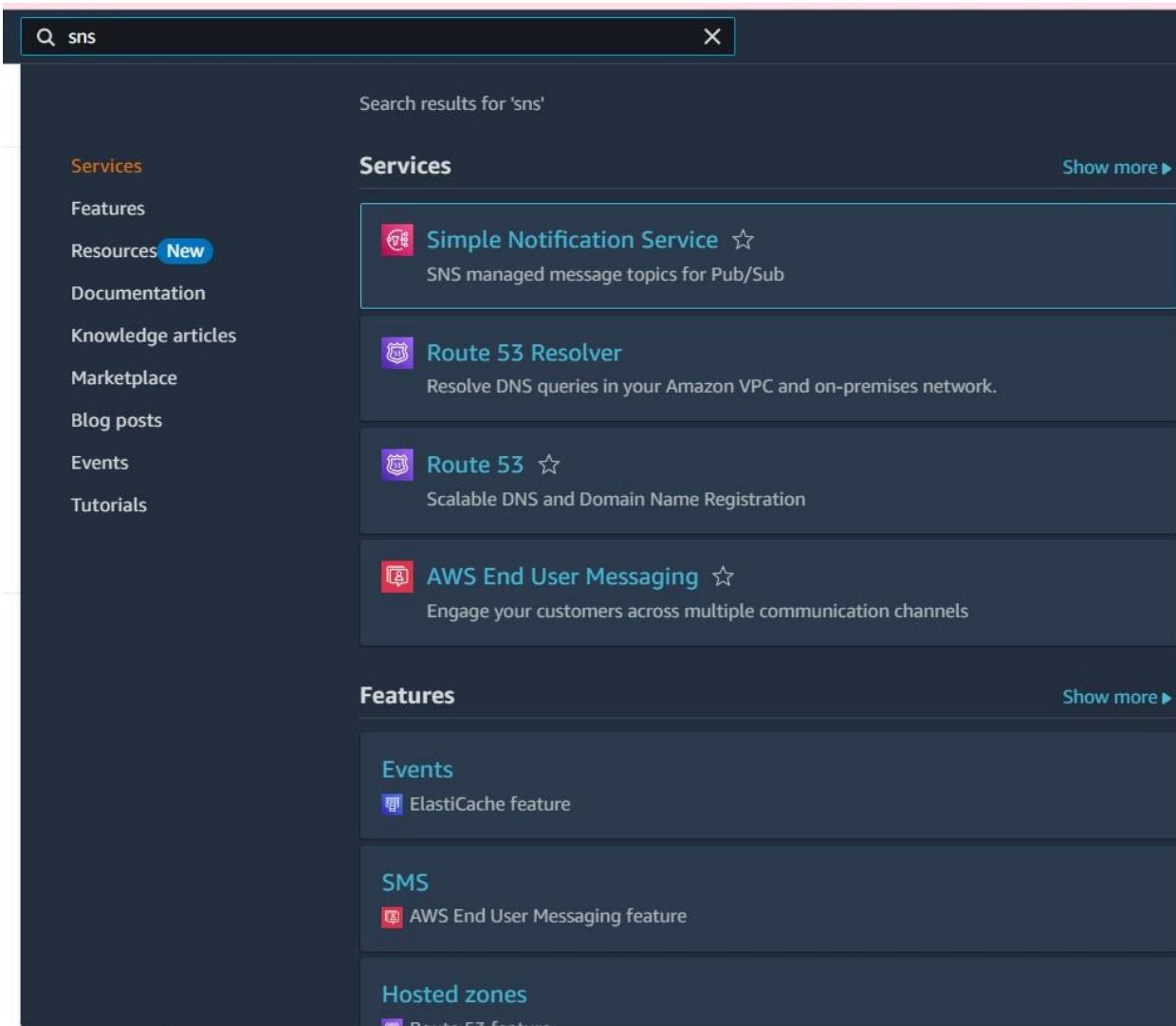

[Actions ▾](#)
[Delete](#)
[Create table](#)

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



The screenshot shows the AWS search interface with the query 'sns' entered in the search bar. The results are displayed under the 'Services' category. The first result is 'Simple Notification Service' (SNS), described as 'SNS managed message topics for Pub/Sub'. Below it are 'Route 53 Resolver' and 'Route 53', both related to DNS services. The third result is 'AWS End User Messaging'. The 'Features' section below includes 'Events' (with an ElastiCache feature listed) and 'SMS' (with an AWS End User Messaging feature listed). The 'Hosted zones' section is partially visible at the bottom.

Q sns

Search results for 'sns'

Services

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

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

Route 53 ☆  
Scalable DNS and Domain Name Registration

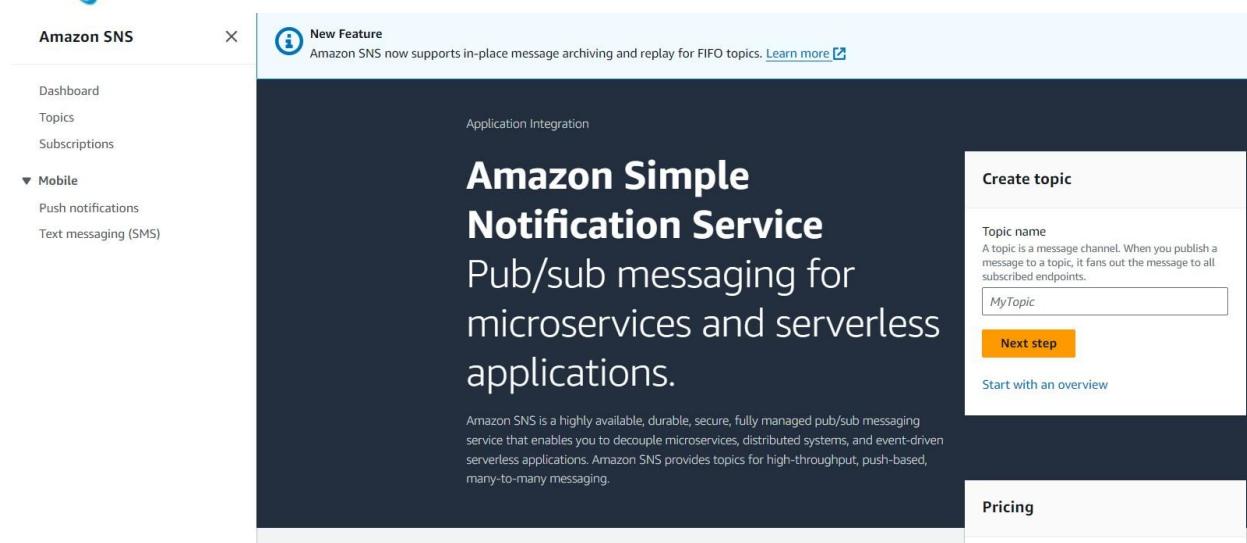
AWS End User Messaging ☆  
Engage your customers across multiple communication channels

Features

Events  
ElastiCache feature

SMS  
AWS End User Messaging feature

Hosted zones  
Route 53 Feature

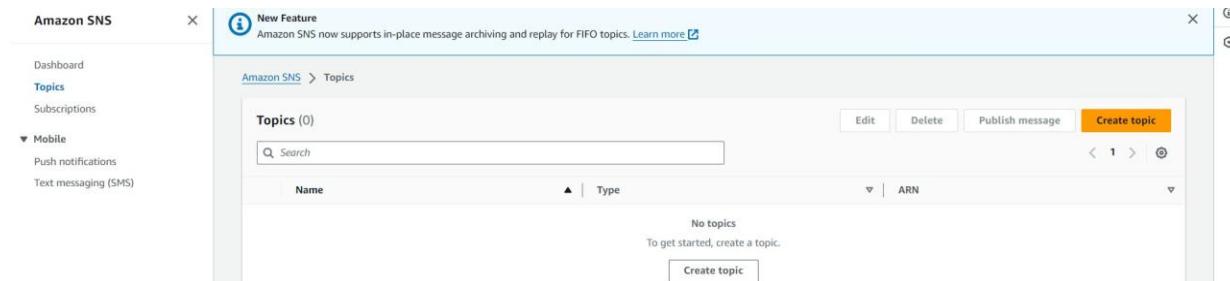


The screenshot shows the Amazon SNS dashboard. On the left, a sidebar menu includes 'Amazon SNS' (selected), 'Dashboard', 'Topics', 'Subscriptions', and 'Mobile' (with 'Push notifications' and 'Text messaging (SMS)'). A 'New Feature' banner at the top right states: 'Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)'.

The main content area features a dark header 'Application Integration' and a large title 'Amazon Simple Notification Service'. Below it, the text reads: 'Pub/sub messaging for microservices and serverless applications.' A descriptive paragraph explains that Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service for decoupling microservices, distributed systems, and event-driven serverless applications.

To the right, a 'Create topic' form has 'Topic name' set to 'MyTopic'. It includes a 'Next step' button and a link to 'Start with an overview'. At the bottom right, a 'Pricing' section is visible.

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



The screenshot shows the 'Topics' list page in the Amazon SNS console. The sidebar menu is identical to the previous dashboard. The main area shows a table with one row: 'Topics (0)'. The table has columns for 'Name', 'Type', and 'ARN'. A 'Create topic' button is located at the bottom of the table. The status bar at the bottom indicates 'No topics' and 'To get started, create a 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, flash, 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:**

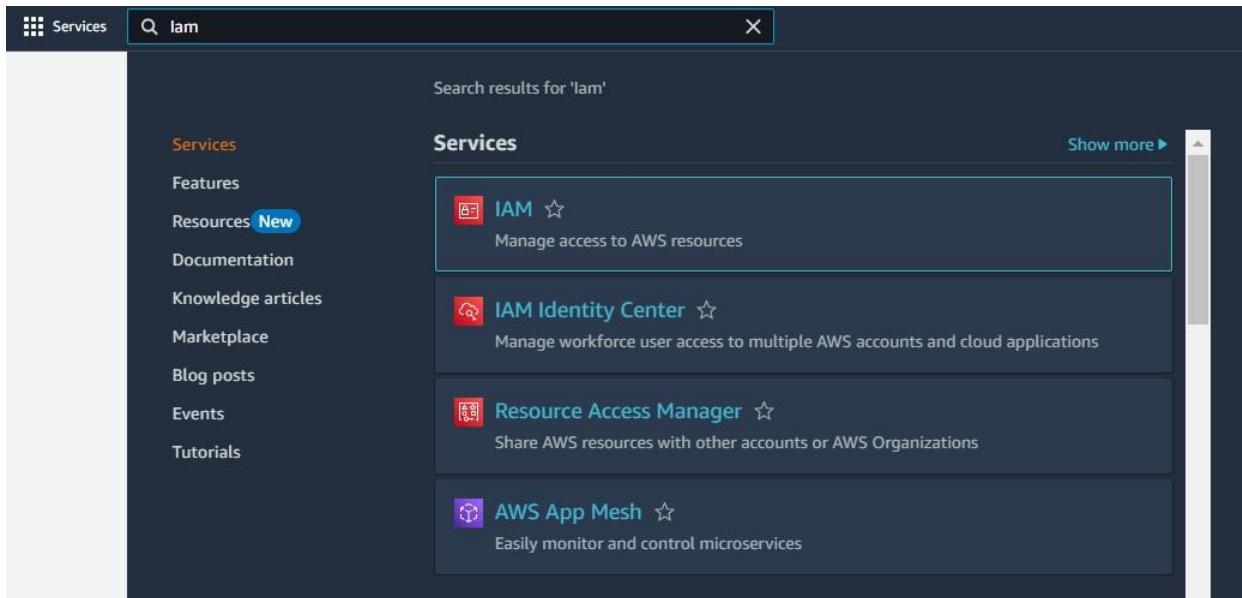
```
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5000))
    debug = os.environ.get('DEBUG', 'False').lower() == 'true'
    app.run(host='0.0.0.0', port=port, debug=debug)
```

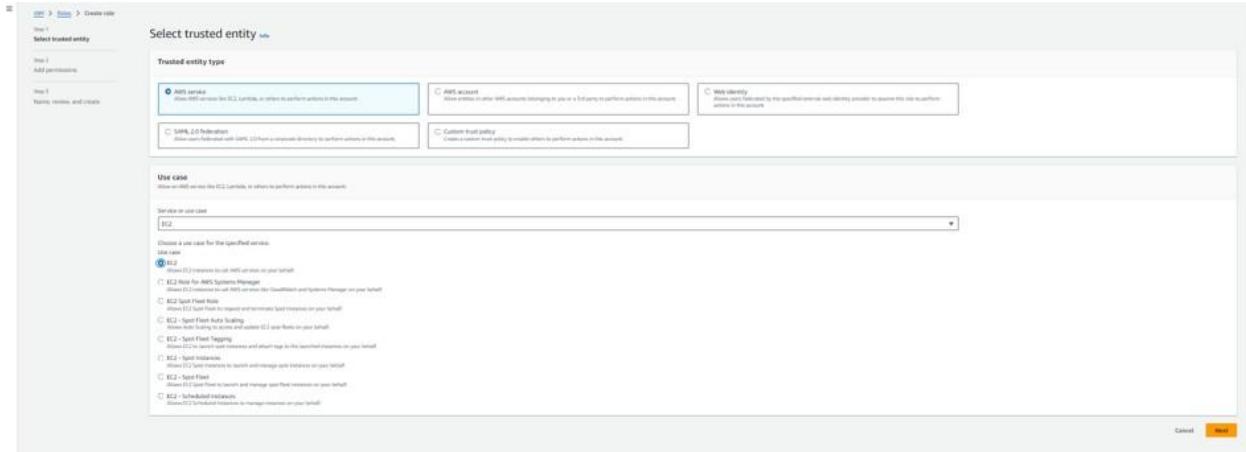
**Description:** The deployment code sets up and launches the homemade pickles and snacks website on a cloud platform like AWS or Heroku, ensuring public access. It typically includes configurations for web hosting, environment variables, and service integration to keep the app running smoothly.

## 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.



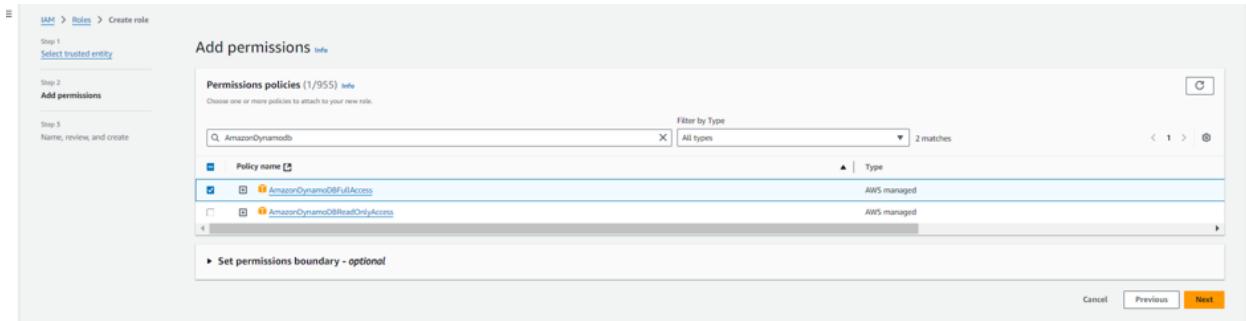


The screenshot shows the 'Select trusted entity' step of the 'Create role' wizard. It includes sections for 'Trusted entity type' (with 'AWS Lambda' selected), 'User case' (with 'EC2 Lambda' selected), and 'Service or use case' (with 'EC2' selected).

## ● 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.



The screenshot shows the 'Add permissions' step of the 'Create role' wizard. It displays a list of 'Permissions policies' (1/955) and allows filtering by type (All types). The 'AmazonDynamoDBFullAccess' policy is selected.

Step 1: Select trusted entity

Step 2: Add permissions

Step 3: Name, review, and create

### Add permissions Info

Permissions policies (2/955) Info

Choose one or more policies to attach to your new role

Filter by Type

Q: vs... X All types ▾ 5 matches ( 1 ) ⌂

| Policy name   | Type        |
|---|-------------|
| <input checked="" type="checkbox"/>  <a href="#">AmazonVSSFullAccess</a>                               | AWS managed |
| <input type="checkbox"/>  <a href="#">AmazonVSSReadOnlyAccess</a>                                      | AWS managed |
| <input type="checkbox"/>  <a href="#">AmazonVSSRole</a>  | AWS managed |
| <input type="checkbox"/>  <a href="#">AWSLambdaBasicExecutionRole</a>                                  | AWS managed |
| <input type="checkbox"/>  <a href="#">AWSLambdaDeviceDefenderPublishFindingsToSNSIntegrationAction</a> | AWS managed |

▶ Set permissions boundary - optional

[Cancel](#) [Previous](#) [Next](#)

Step 1: Select trusted entities

Step 2: Add permissions

Step 3: Name, review, and create

### Name, review, and create

Role details

Role name

Provide a unique name to identify this role.

Description

Provide a description for this role.

Maximum 1000 characters. Use letters A-Z, numbers 0-9, underscores, and hyphens.

Step 1: Select trusted entities Edit

Trust policy

```
1: { "Version": "2012-10-17", 2: "Statement": [ 3: { "Effect": "Allow", 4: "Principal": " *", 5: "Action": "sts:AssumeRole" } ] }
```

Step 2: Add permissions

Permissions policy summary

| Policy name   | Type        | Attached as        |
|---|-------------|--------------------|
| <input checked="" type="checkbox"/>  <a href="#">AmazonVSSFullAccess</a> | AWS managed | Permissions policy |
| <input type="checkbox"/>  <a href="#">AmazonVSSReadOnlyAccess</a>        | AWS managed | Permissions policy |

Step 3: Add tags

Add tags - optional Info

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

No tags associated with this resource.

[Add new tag](#) You can add up to 10 more tags.

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

IAM > Roles > sns\_Dynamodb\_role

### sns\_Dynamodb\_role Info

Allows EC2 Instances to call AWS services on your behalf.

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

**Permissions** | Trust relationships | Tags | Last Accessed | Revoke sessions

**Permissions policies (2) Info**  
 You can attach up to 10 managed policies.

| Policy name              |                          | Type        | Attached entities |
|--------------------------|--------------------------|-------------|-------------------|
| <input type="checkbox"/> | AmazonDynamoDBFullAccess | AWS managed | 4                 |
| <input type="checkbox"/> | AmazonSNSFullAccess      | AWS managed | 2                 |

us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles

**Identity and Access Management (IAM)**

- Dashboard
- Access management**
  - User groups
  - Users
  - Roles**
  - Policies
  - Identity providers
  - Account settings
  - Root access management New
- Access reports**
  - Access Analyzer
  - Resource analysis New
  - Unused access
  - Analyzer settings
  - Credential report
  - Organization activity
- CloudShell Feedback

**Roles (1/8) Info**  
 An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

| 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                             | -             |
| <b>EC2_DynamoDB_Role</b>        | AWS Service: ec2                                  | -             |
| OrganizationAccountAccessRole   | Account: 058264256896                             | 1 hour ago    |
| rsoaccount-new                  | Account: 058264256896                             | 6 minutes ago |

**Roles Anywhere Info**  
 Authenticate your non AWS workloads and securely provide access to AWS services.

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

ENG IN 11:56 03-07-2025

## 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                   |

Local      Codespaces

 Clone      

[HTTPS](https://github.com/Chandra123-123/Homemade.git)    [SSH](#)    [GitHub CLI](#)

<https://github.com/Chandra123-123/Homemade.git> 

Clone using the web URL.

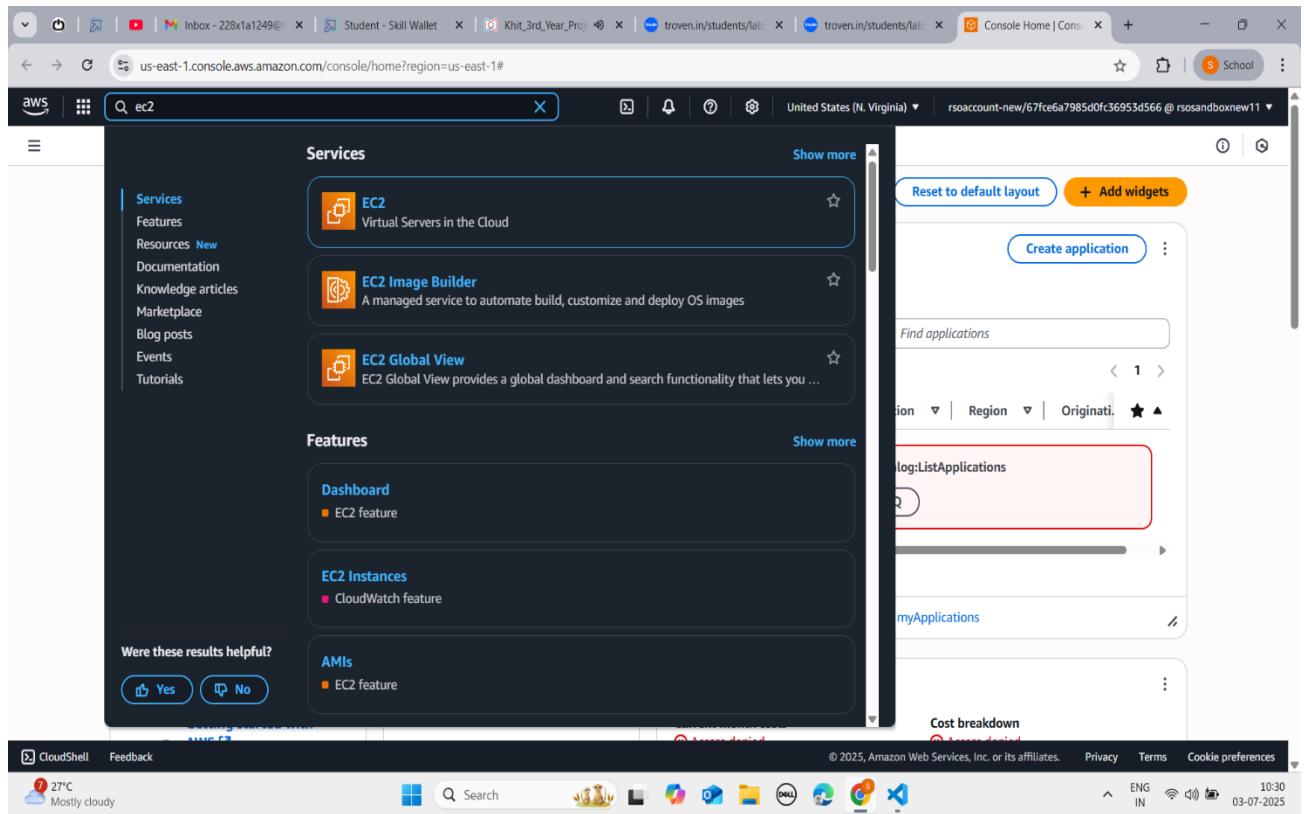
 Open with GitHub Desktop

 Download ZIP

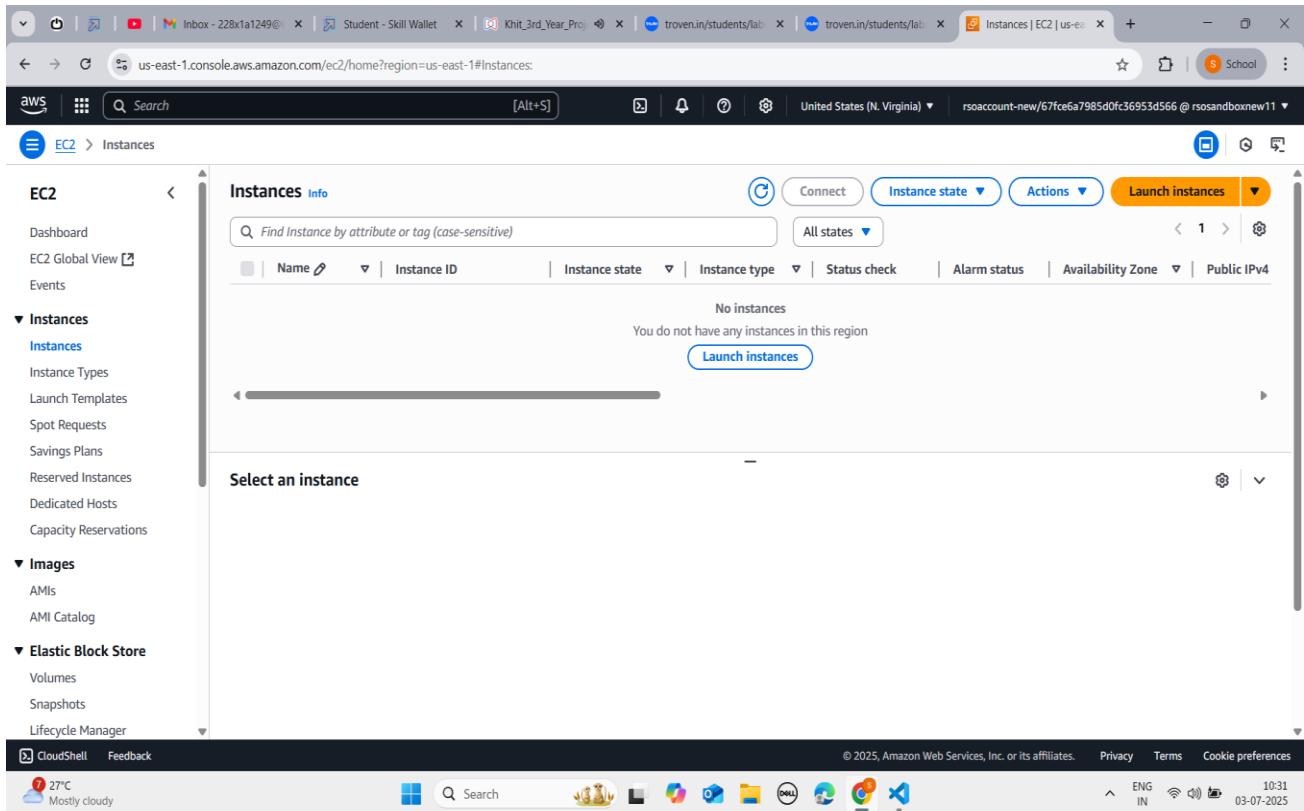
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- **Launch EC2 Instance**

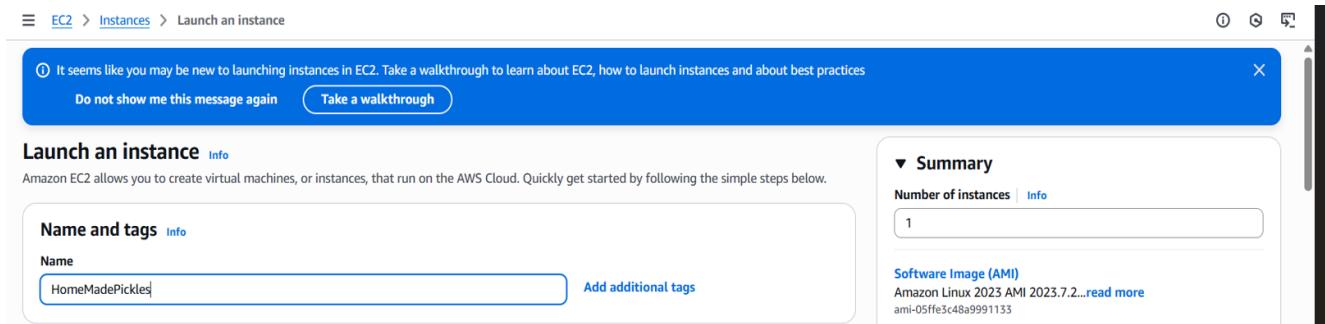
- In the AWS Console, navigate to EC2 and launch a new instance.



- Click on Launch instance to launch EC2 instance

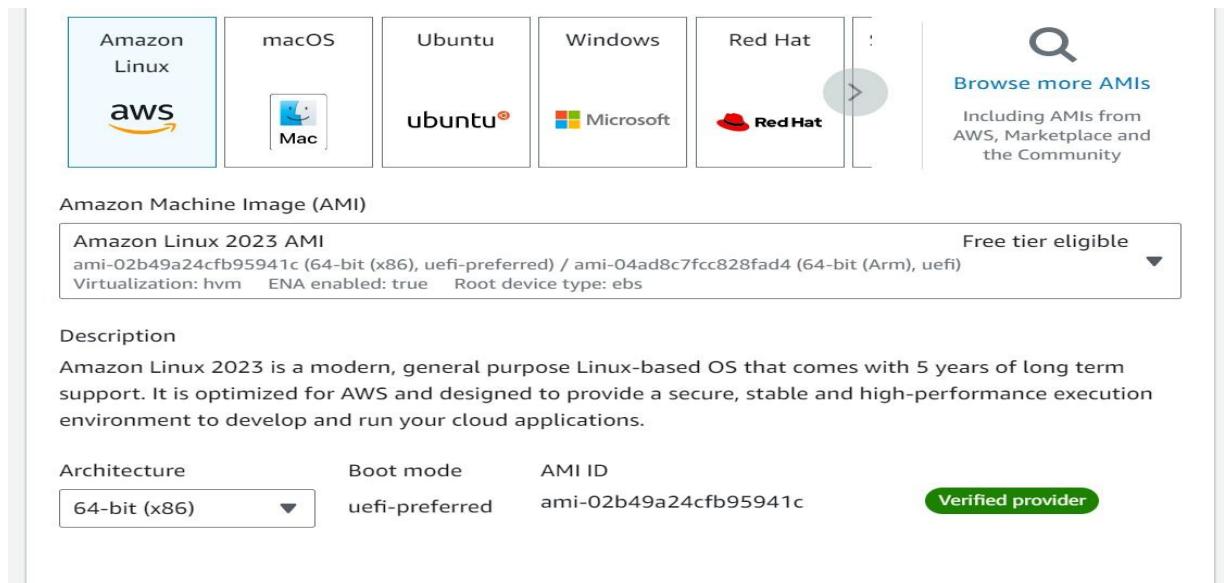


The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed. The main area displays a message: "No instances" and "You do not have any instances in this region". A prominent blue "Launch instances" button is centered below the message. The bottom of the screen shows the AWS navigation bar and system status.



This screenshot shows the first step of the "Launch an instance" wizard, titled "Name and tags". It includes a "Name" input field containing "HomeMadePickles" and a "Add additional tags" link. On the right, the "Summary" section shows "Number of instances: 1" and "Software Image (AMI): Amazon Linux 2023 AMI 2023.7.2...". A blue banner at the top provides a walkthrough for new users.

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



Amazon Machine Image (AMI)

**Amazon Linux 2023 AMI** Free tier eligible ▾

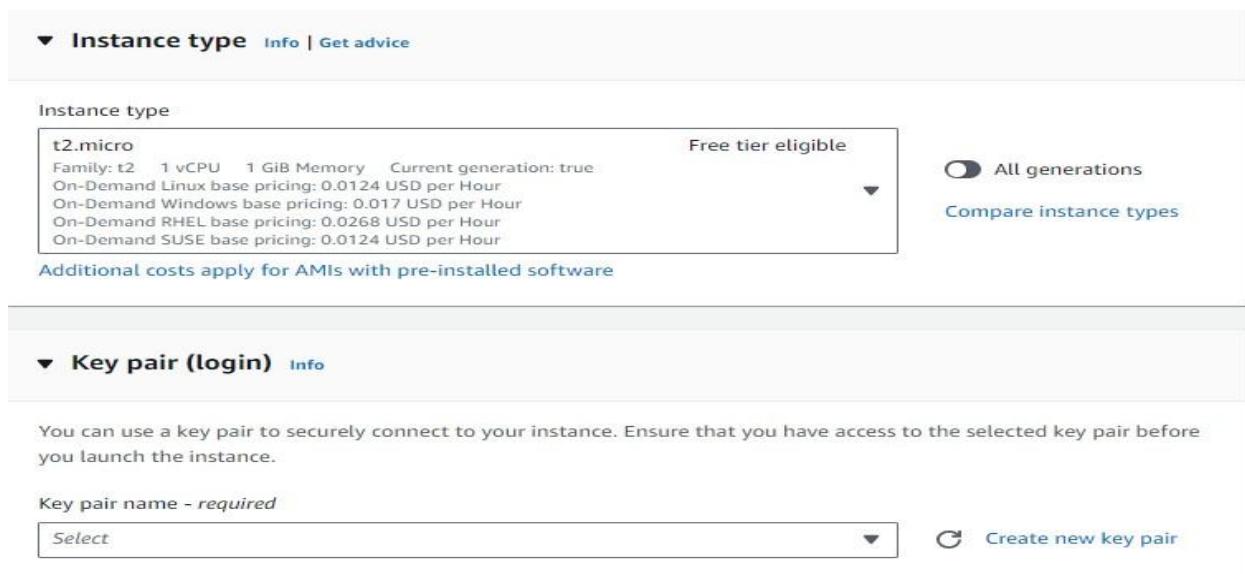
ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)  
 Virtualization: hvm ENA enabled: true Root device type: ebs

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 | Verified provider |

- 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](#)

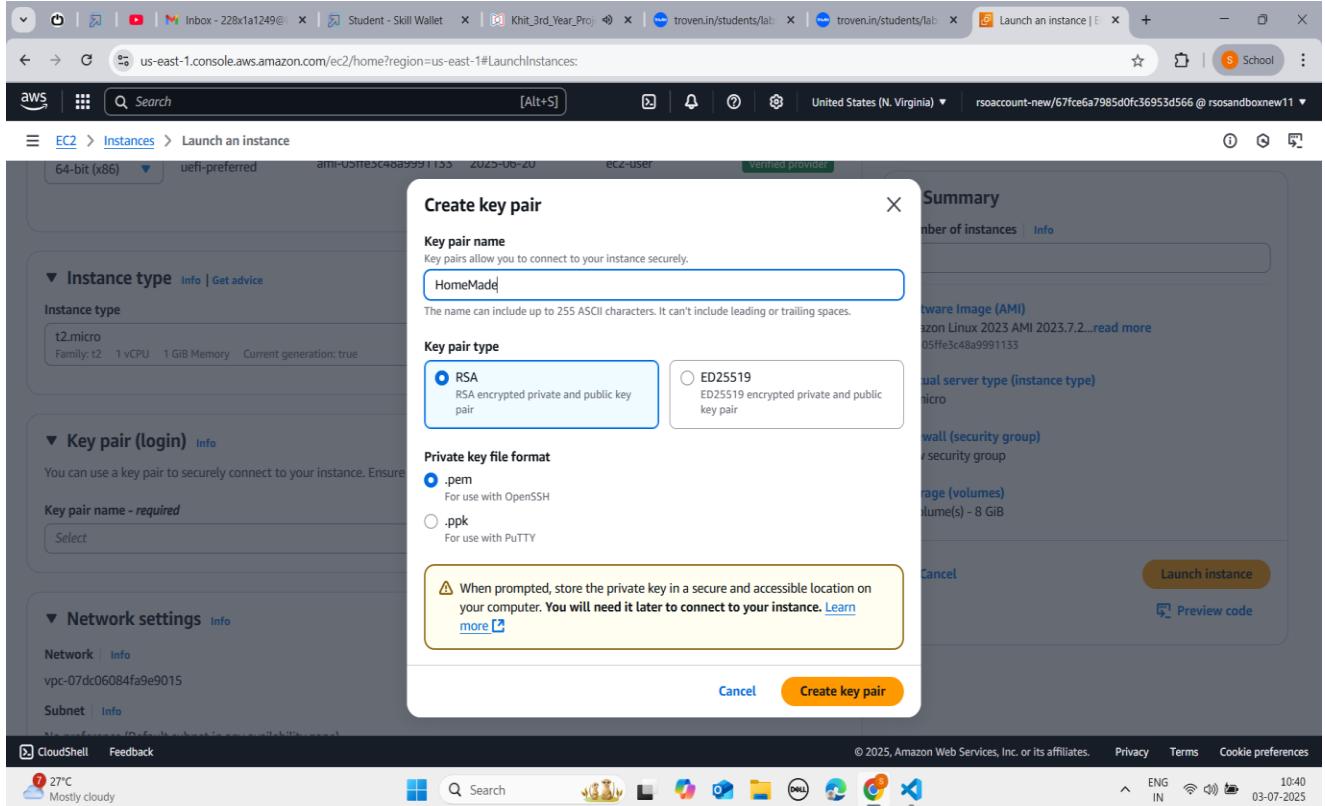
Additional costs apply for AMIs with pre-installed software

**Key pair (login)** [Info](#)

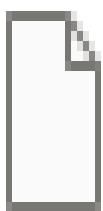
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select Create new key pair



The screenshot shows the AWS EC2 'Launch an instance' wizard. The current step is 'Create key pair'. The 'Key pair name' field is filled with 'HomeMade'. The 'Key pair type' section shows 'RSA' selected (highlighted in blue), with 'ED25519' as an alternative. The 'Private key file format' section shows '.pem' selected (highlighted in blue). A note at the bottom of this section states: '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](#)'.



# HomeMade.pem

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

**▼ Network settings [Info](#)**

VPC - required [Info](#)  
 vpc-03cdc7b6f19dd7211 (default) ▾ 

Subnet [Info](#)  
 No preference ▾ 

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

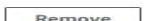
**Inbound Security Group Rules**

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

| Type <a href="#">Info</a>   | Protocol <a href="#">Info</a>     | Port range <a href="#">Info</a>             |
|---|-----------------------------------|---|
| ssh   | TCP                               | 22  |
| Source type <a href="#">Info</a>  | Source <a href="#">Info</a>       | Description - optional <a href="#">Info</a> |
| Anywhere  | Add CIDR, prefix list or security | e.g. SSH for admin desktop                  |
| 0.0.0.0/0  |                                   |   |

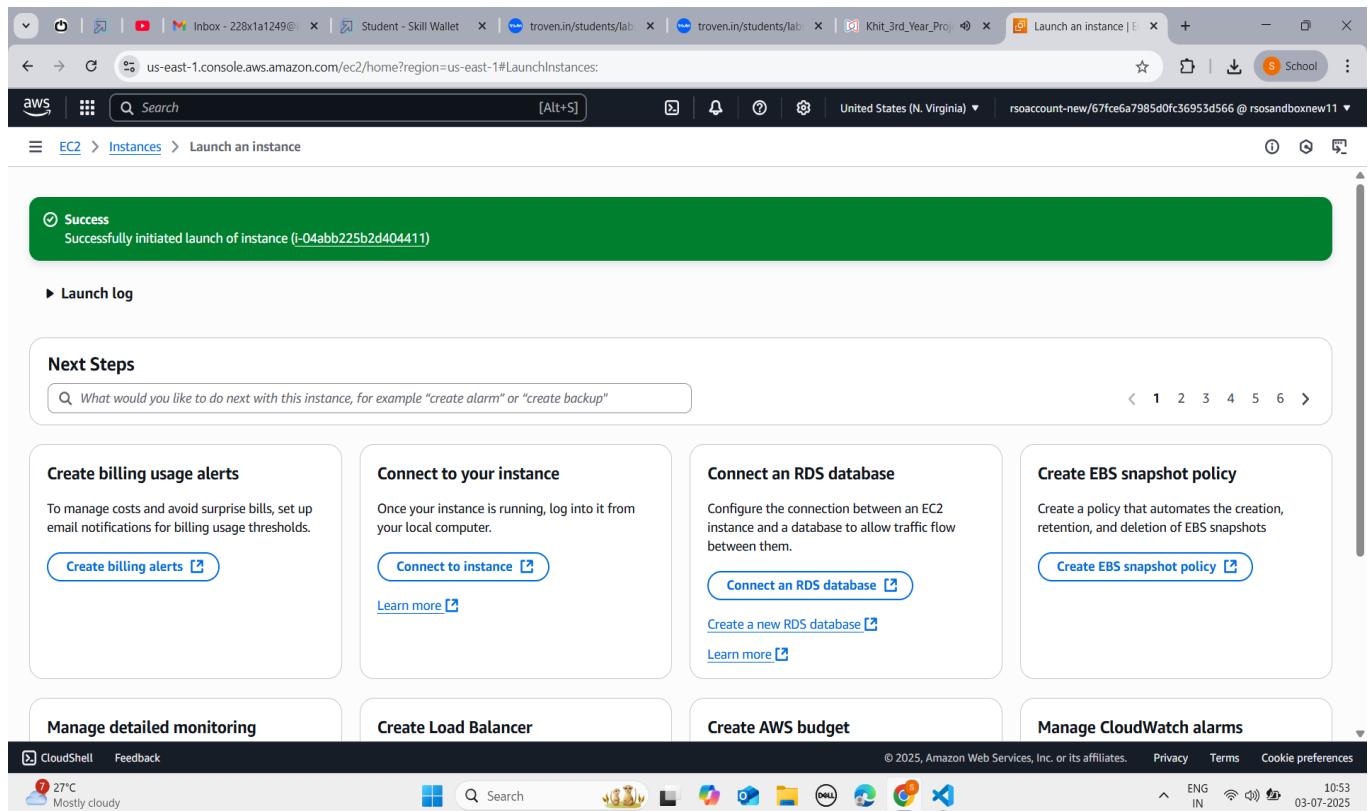
**▼ Security group rule 2 (TCP, 80, 0.0.0.0/0)** 

| Type <a href="#">Info</a>   | Protocol <a href="#">Info</a>     | Port range <a href="#">Info</a>             |
|---|-----------------------------------|---|
| HTTP  | TCP                               | 80  |
| Source type <a href="#">Info</a>  | Source <a href="#">Info</a>       | Description - optional <a href="#">Info</a> |
| Custom  | Add CIDR, prefix list or security | e.g. SSH for admin desktop                  |
| 0.0.0.0/0  |                                   |   |

**▼ Security group rule 3 (TCP, 5000, 0.0.0.0/0)** 

| Type <a href="#">Info</a>   | Protocol <a href="#">Info</a>     | Port range <a href="#">Info</a>             |
|---|-----------------------------------|---|
| Custom TCP  | TCP                               | 5000  |
| Source type <a href="#">Info</a>  | Source <a href="#">Info</a>       | Description - optional <a href="#">Info</a> |
| Custom  | Add CIDR, prefix list or security | e.g. SSH for admin desktop                  |
| 0.0.0.0/0  |                                   |   |

**Add security group rule**



The screenshot shows the AWS EC2 Instances Launch log page. A green success message at the top states: "Successfully initiated launch of instance (i-04abb225b2d404411)". Below this, there is a "Launch log" section. 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". The "Connect to your instance" section includes a "Connect to instance" button and a "Learn more" link. The "Create EBS snapshot policy" section includes a "Create EBS snapshot policy" button and a "Learn more" link. The bottom of the page shows the AWS navigation bar with links for CloudShell, Feedback, Weather (27°C, Mostly cloudy), Search, Notifications, and other services like S3, Lambda, and CloudWatch.

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.

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

aws Search [Alt+S] United States (N. Virginia) rsaccount-new/67fce6a7985d0fc36953d566 @ rsandboxnew11

**EC2 > Instances**

Successfully attached EC2\_DynamoDB\_Role to instance i-04abb225b2d404411

**Instances (1/1) Info**

Find Instance by attribute or tag (case-sensitive) All states ▾

| Name           | Instance ID         | Instance state | Instance type | Status check      | Alarm status  | Availability Zone | Public IP   |
|----------------|---------------------|----------------|---------------|-------------------|---------------|-------------------|-------------|
| HomeMadePic... | i-04abb225b2d404411 | Running        | t2.micro      | 2/2 checks passed | View alarms + | us-east-1b        | ec2-34-2... |

**i-04abb225b2d404411 (HomeMadePickles)**

**Details** Status and alarms Monitoring Security Networking Storage Tags

**Instance summary**

|                     |                               |   |
|---------------------|-------------------------------|---|
| Instance ID         | Public IPv4 address           | Private IPv4 addresses                                    |
| i-04abb225b2d404411 | 34.235.165.139   open address | 172.31.28.8   |
| IPv6 address        | Instance state                | Public DNS  |
| -                   | Running                       | ec2-34-235-165-139.compute-1.amazonaws.com   open address |

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 11:34 03-07-2025

27°C Mostly cloudy

- 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

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.

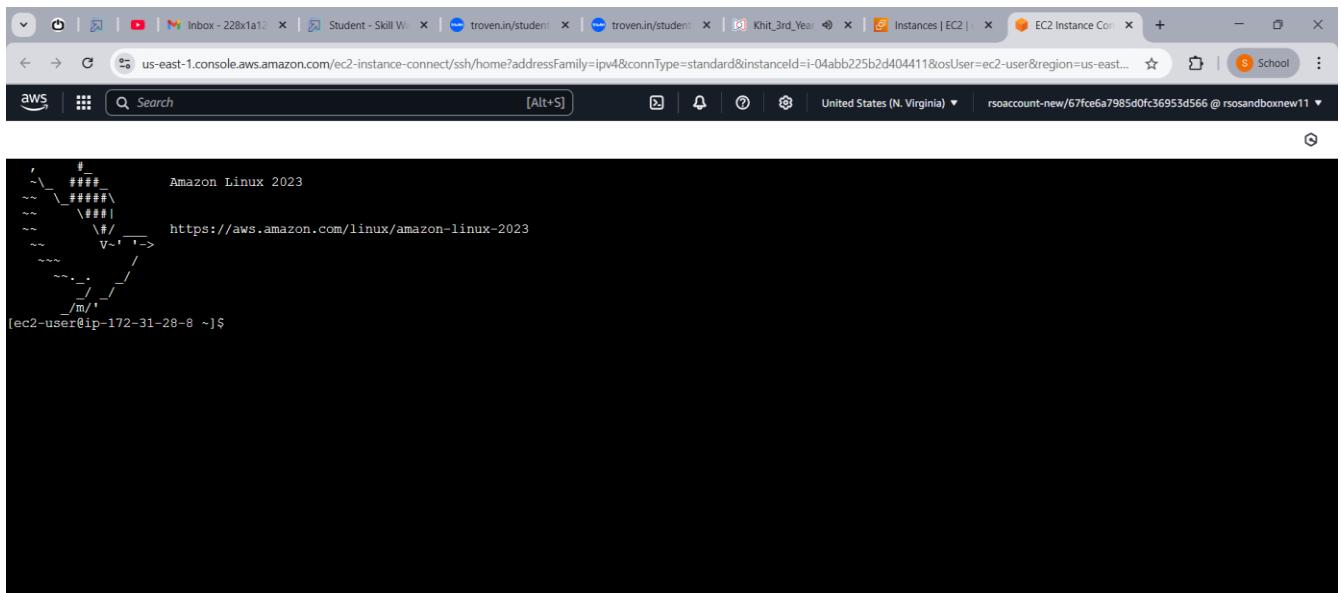
Public IPv4 address

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**



```

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

[ec2-user@ip-172-31-28-8 ~]$

```

i-04abb225b2d404411 (HomeMadePickles)

Public IPs: 34.235.165.139 Private IPs: 172.31.28.8



```
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/Homemade.git'

Note: change your-github-username and your-repository-name with your credentials

here: 'git clone https://github.com/Chandra123-123/Homemade.git'

- This will download your project to the EC2 instance.

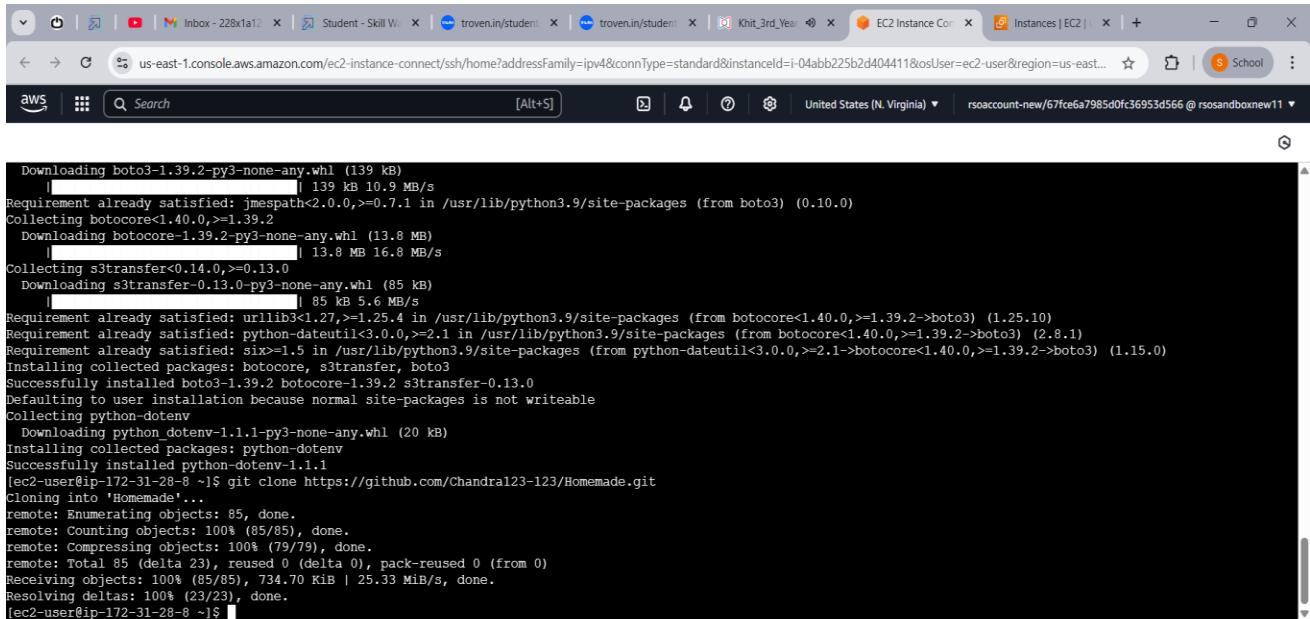
#### To navigate to the project directory, run the following command:

```
cd Homemade
```

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

#### Run the Flask Application

```
sudo flask run --host=0.0.0.0 --port=5000
```



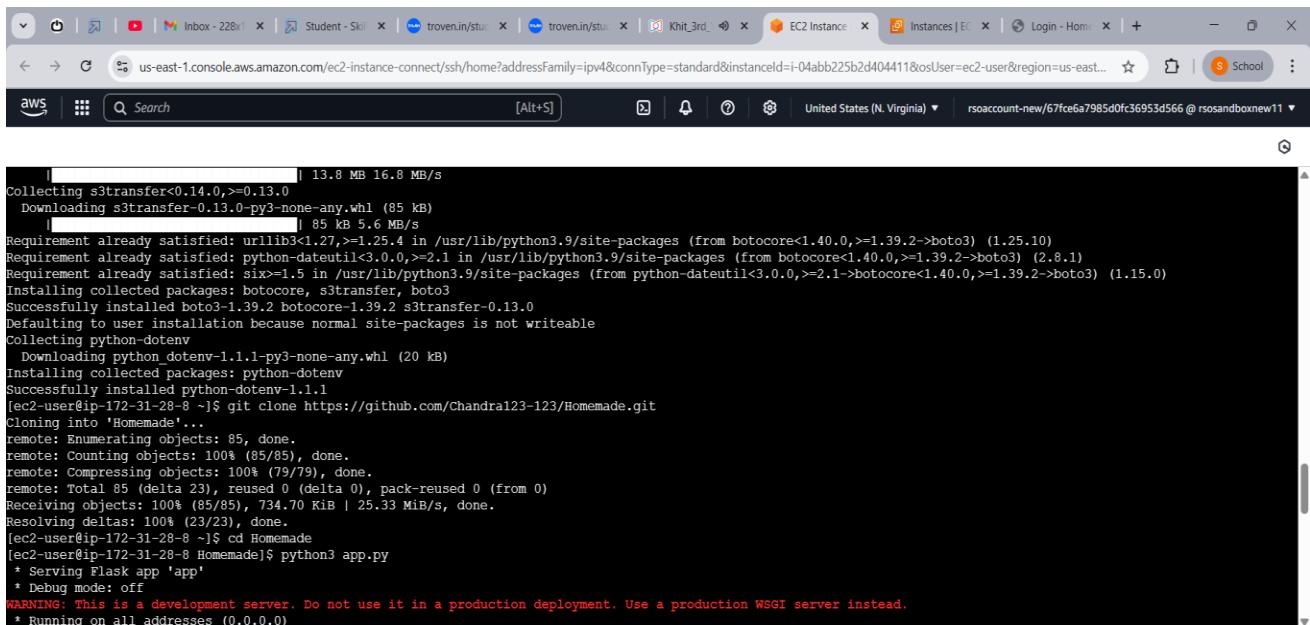
```

        Downloading boto3-1.39.2-py3-none-any.whl (139 kB)
        |██████████| 139 kB 10.9 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Collecting botocore<1.40.0,>=1.39.2
    Downloading botocore-1.39.2-py3-none-any.whl (13.8 MB)
    |██████████| 13.8 MB 16.8 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
    Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
    |██████████| 85 kB 5.6 MB/s
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 botocore-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/Chandra123-123/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 ~]$ 

```

### i-04abb225b2d404411 (HomeMadePickles)

PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8

```

        |██████████| 13.8 MB 16.8 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
    Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
    |██████████| 85 kB 5.6 MB/s
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 botocore-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/Chandra123-123/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)

Public IPs: 34.235.165.139 Private IPs: 172.31.28.8

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  
~~ \### 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)

Public IPs: 34.235.165.139 Private IPs: 172.31.28.8

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 6 30°C Mostly cloudy ENG IN 13:35 03-07-2025

```
[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)

Public IPs: 34.235.165.139 Private IPs: 172.31.28.8

### Verify the Flask app is running:

<http://34.235.165.139>

- Run the Flask app on the EC2 instance



i-04abb225b2d404411 (HomeMadePickles)

Public IPs: 34.235.165.139 Private IPs: 172.31.28.8

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 6 30°C Mostly cloudy  Search        ENG IN 13:11 03-07-2025

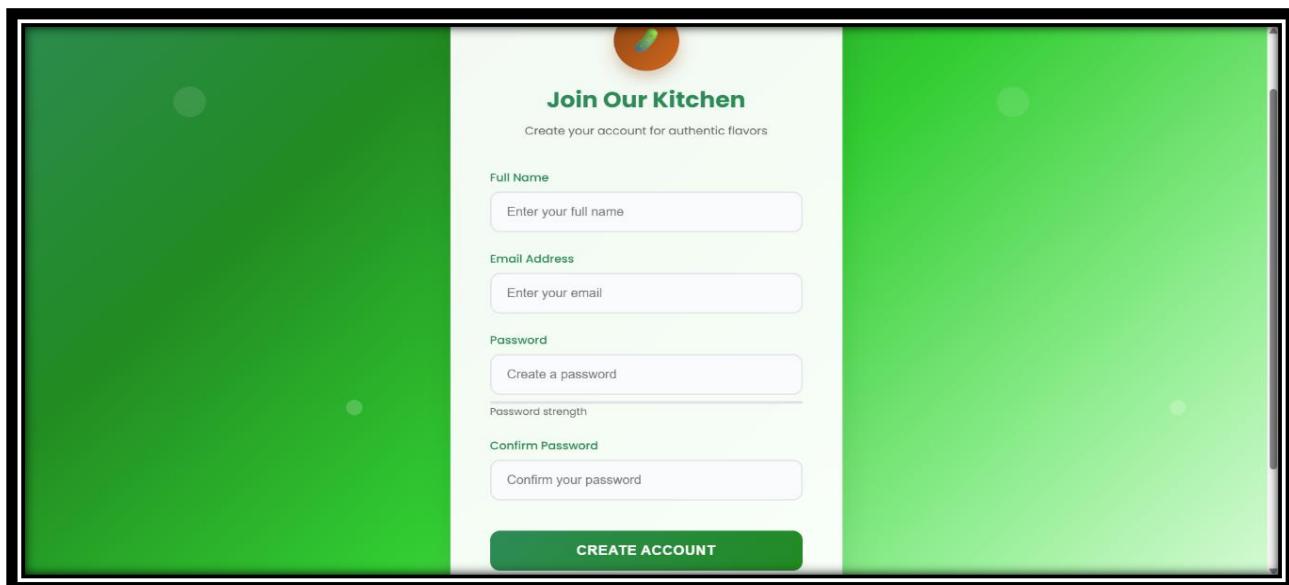
Access the website through:

Public IPs: <http://34.235.165.139>

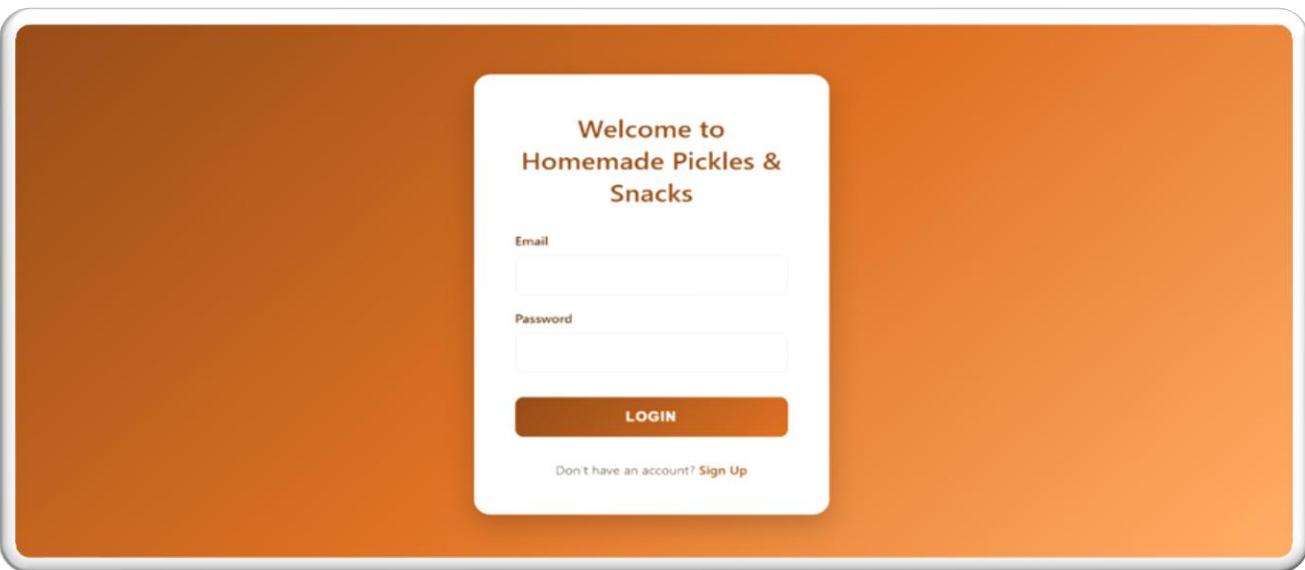
## Milestone 8: Testing and Deployment

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

### Signup Page:



## Login Page:



## Home page:

# Homemade Pickles & Snacks

Authentic tastes from our kitchen to yours!

About      Contact      Cart

### Featured Products

**VEG\_PICKLES**



KERALA FISH PICKLE  
PRAWN PICKLE  
ANCHOVY PICKLE | KOZHIKA  
SARDINE FISH PICKLE ( MATHI )  
TENMINNAM FISH PICKLE  
CHICKEN PICKLE  
GARLIC PICKLE  
TOMATO PICKLE  
ENNA MANGA PICKLE

Spicy and tangy chunks packed with tradition.

**NON\_VEG\_PICKLES**



Not VEG  
PRAWN  
FISH  
KALGOORI

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 a website page with a brown header bar containing the title "Homemade Pickles & Snacks". Below the header is a navigation menu with links to "Home", "Veg Pickles", "Non-Veg Pickles", "Snacks", and "Contact". The main content area has a green background and features a white rounded rectangle containing the heading "Our Story". Inside this box, there is descriptive text about the brand's history and values, followed by two smaller paragraphs detailing their commitment to quality and tradition.

**Homemade Pickles & Snacks**

Home    Veg Pickles    Non-Veg Pickles    Snacks    Contact

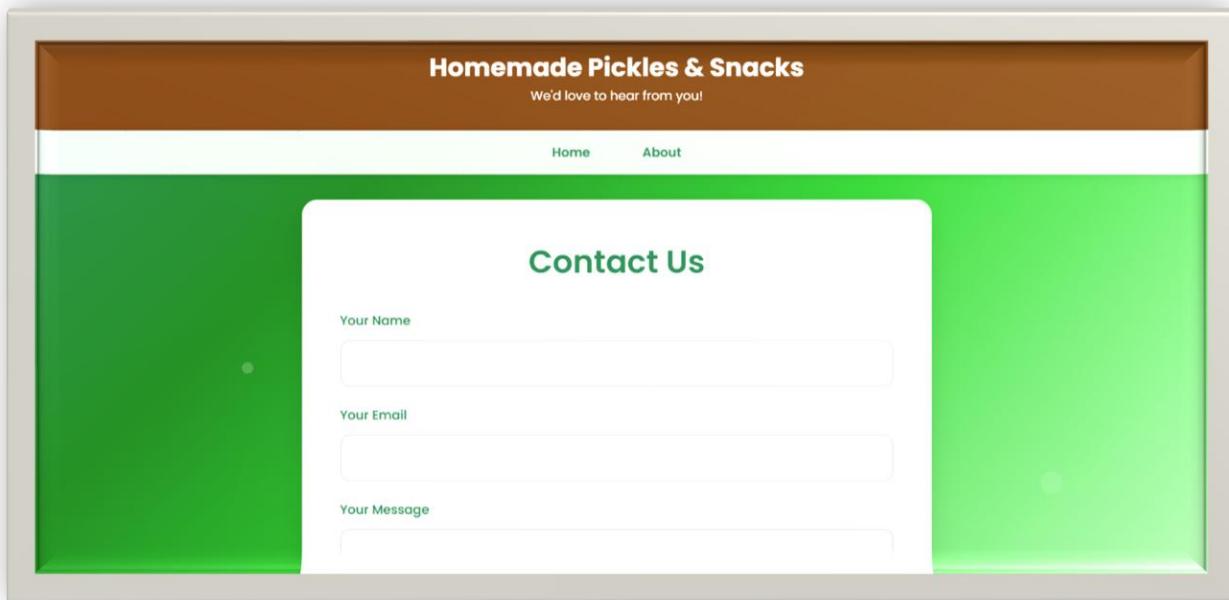
**Our Story**

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:



## Veg-Pickles page :

### Homemade Pickles & Snacks

Purely Plant-Based, Bursting with Flavor

Home    Non-Veg Pickles    Snacks    About    Cart    Contact

#### Our Vegetarian Pickles



**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.

Andhra mango pickle  
Gongura pickle  
Lemon pickle

Andhra mango pickle  
Gongura pickle  
Lemon pickle

## 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 garam masala and traditional spices.



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

Chicken Pickle    Mutton Pickle    Prawn Pickle

## 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:

**Homemade Pickles & Snacks**  
Your Cart

Home      Contact

| PRODUCT        | QUANTITY   | UNIT PRICE | SUBTOTAL |
|----------------|--|------------|----------|
| Amla Pickle    | <input type="button" value="-"/> <input type="button" value="1"/> <input type="button" value="+"/> | ₹60        | ₹60      |
| Gongura Pickle | <input type="button" value="-"/> <input type="button" value="1"/> <input type="button" value="+"/> | ₹50        | ₹50      |
| Mutton Pickle  | <input type="button" value="-"/> <input type="button" value="1"/> <input type="button" value="+"/> | ₹120       | ₹120     |
| Kakinada Kaja  | <input type="button" value="-"/> <input type="button" value="1"/> <input type="button" value="+"/> | ₹80        | ₹80      |
| Karam Boondi   | <input type="button" value="-"/> <input type="button" value="1"/> <input type="button" value="+"/> | ₹60        | ₹60      |

**Total: ₹370**

## Check-out Page:



The screenshot shows a web-based check-out interface for a store named "Homemade Pickles & Snacks". The top navigation bar is brown and contains the store name and a "Checkout" link. Below the navigation is a white header bar with links for "Home", "Cart", and "Contact". The main content area has a green background and features a white rectangular form titled "Billing & Shipping Details". This form includes three input fields: "Full Name", "Email", and "Phone Number", each with a corresponding placeholder text below it. At the bottom of the form, there is a large, faint watermark-like text that reads "Smart Internz".

**Homemade Pickles & Snacks**

Checkout

Home Cart Contact

**Billing & Shipping Details**

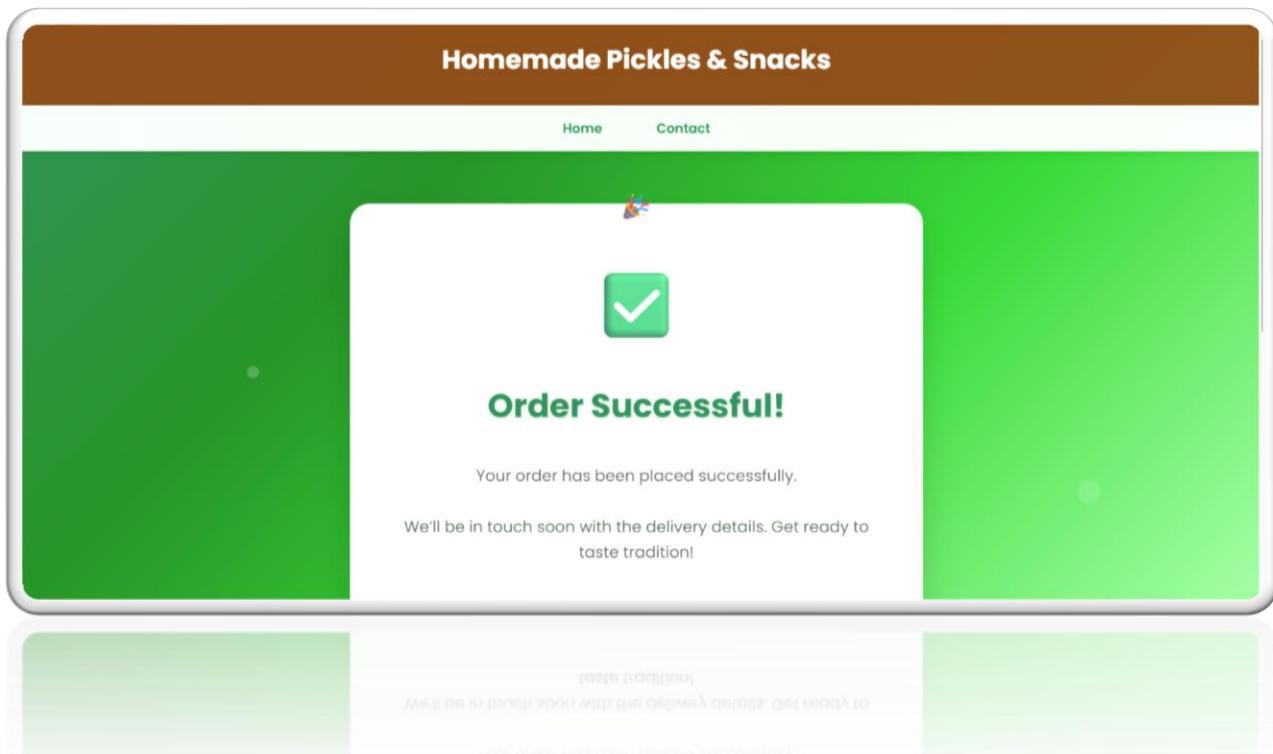
Full Name

Email

Phone Number

Smart Internz

## 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.











