Home Made Pickles & Snacks: Taste the Best – Cloud-Powered E-Commerce Platform

Project Description

The growing demand for authentic homemade pickles and snacks has highlighted the need for an efficient, scalable e-commerce solution to cater to customers across India. The **Home Made Pickles & Snacks** platform addresses this need by providing a modern web-based application that enables customers to browse, add products to the cart, and place orders seamlessly.

This project uses **Flask** for backend development, **AWS DynamoDB** for storing user and order data, **AWS SNS** for real-time order notifications, and **EC2** for deployment. The platform offers an intuitive UI for customers to explore Veg Pickles, Non-Veg Pickles, and Snacks with pricing and cart functionality.

Scenario 1: Scalable Order Management for High Demand

A cloud-based system ensures seamless order processing during peak user activity. For instance, during a promotional event, hundreds of users simultaneously access the platform to place orders. The backend efficiently processes requests, updates inventory in real-time, and manages user sessions. The cloud infrastructure handles traffic spikes without performance degradation, ensuring smooth transactions and minimizing wait times.

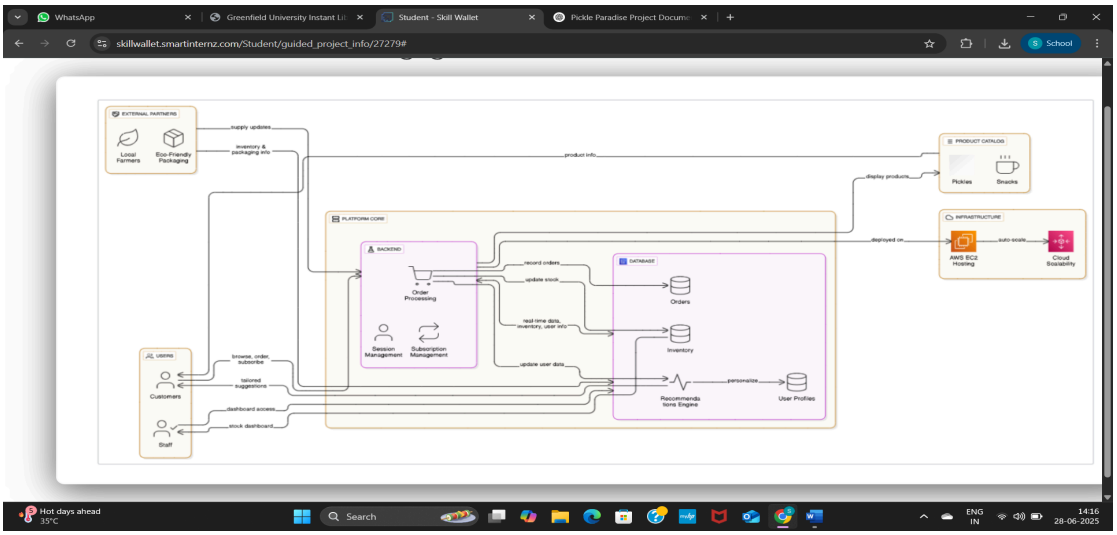
Scenario 2: Real-Time Inventory Tracking and Updates

When a customer places an order for a product, the system instantly updates stock levels and records transaction details. For example, a user purchases an item, triggering automatic inventory deduction and order confirmation. Staff members receive updated dashboards to monitor stock availability and fulfillment progress, ensuring timely restocking and minimizing overselling risks

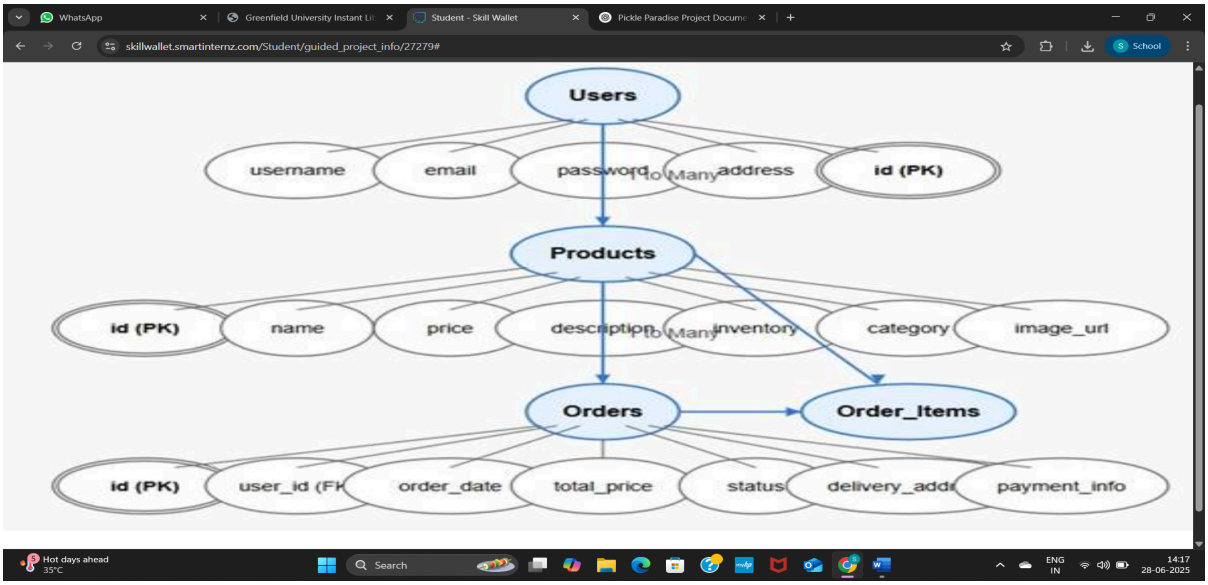
Scenario 3: Personalized User Experience and Recommendations

The platform leverages user behavior data to enhance engagement. A returning customer, for instance, views tailored recommendations based on past purchases and browsing history. The system dynamically adjusts suggestions in real-time, while maintaining fast response rates even during high traffic, creating a frictionless and intuitive shopping experience.

AWS ARCHITECTURE



Entity Relationship (ER)Diagram



Pre-requisites:

- AWSAccountSetup:AWSAccountSetup
- AWSIAM(IdentityandAccessManagement):AWSIAM
- AWSEC2(ElasticComputeCloud):EC2
- AWSDynamoDB:Dynamodb
- GitDocumentation:GitDocumentation
- VS Code Installation: VS Code download

Project WorkFlow:

Milestone 1. Backend Development and Application Setup

- Develop the Backend Using Flask.
- Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login

- Set up an AWS account if not already done.
- Log in to the AWS Management Console

Milestone 3. DynamoDB Database Creation and Setup

- Create a DynamoDB Table.
- Configure Attributes for User Data and Book Requests.

Milestone 4. SNS Notification Setup

- Create SNS topics for book request notifications.
- Subscribe users and library staff to SNS email notifications.

Milestone 5. IAM Role Setup

- Create IAM Role
- Attach Policies

Milestone 6. EC2 Instance Setup

- Launch an EC2 instance to host the Flask application.
- Configure security groups for HTTP, and SSH access.

Milestone 7. Deployment on EC2

- Upload Flask Files
- Run the Flask App

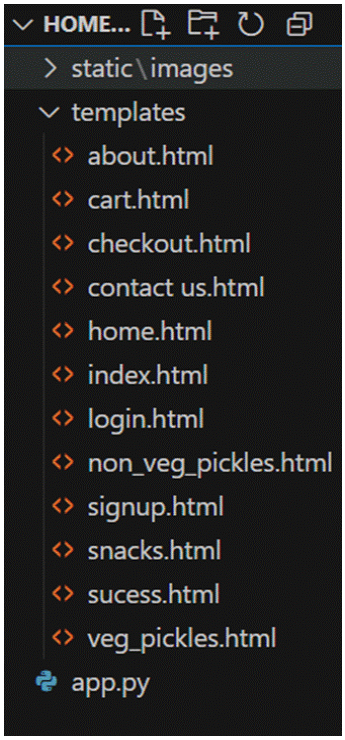
Milestone 8. Testing and Deployment

Conduct functional testing to verify user signup, login, buy/sell stocks and notifications.

Milestone 1. Backend Development and Application Setup

Develop the Backend Using Flask.

File Explorer Structure:



Description of the code :

? Flask App Initialization

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
import boto3
import uuid
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import os

app = Flask(__name__)
```

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

```python
# AWS Configuration
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
user_table = dynamodb.Table('Users')
orders_table = dynamodb.Table('PickleOrders')
```

```python
# Product Data - Unchanged
veg_items = [
    {"name": "Mango Pickle", "price": 150, "img": "mango-pickle.jpg"},
    {"name": "Lemon Pickle", "price": 120, "img": "lemon-pickle.jpg"},
    {"name": "Tomato Pickle", "price": 180, "img": "tomato-pickle.jpg"},
    {"name": "Gongura Pickle", "price": 160, "img": "gongura-pickle.jpg"},
]

non_veg_items = [
    {"name": "Chicken Pickle", "price": 300, "img": "chicken-pickle.jpg"},
    {"name": "Mutton Pickle", "price": 350, "img": "mutton-pickle.jpg"},
    {"name": "Fish Pickle", "price": 320, "img": "fish-pickle.jpg"},
    {"name": "Prawn Pickle", "price": 340, "img": "prawn-pickle.jpg"},
]

snack_items = [
    {"name": "Snack Combo", "price": 200, "img": "snacks-combo.jpg"},
    {"name": "Murukku", "price": 100, "img": "murukku.jpg"},
    {"name": "Mixture", "price": 120, "img": "mixture.jpg"},
    {"name": "Chakli", "price": 130, "img": "chakli.jpg"},
]
```

- Routes for Web Pages

Login Route (GET/POST):Verifies user credentials, increments login count, and redirects to the dashboard on success

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = user_table.get_item(Key={'email': email}).get('Item')
        if user and user['password'] == password:
            session['user'] = email
            flash("Login successful!", "success")
            return redirect(url_for('index'))
        flash("Invalid credentials", "danger")
    return render_template('login.html')
```

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

```
def signup():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user_table.put_item(Item={'email': email, 'password': password})
        send_email(email, "Welcome to Pickle Paradise", "Thank you for signing up!")
        flash("Signup successful! Please login.", "success")
        return redirect(url_for('login'))
    return render_template('signup.html')
```


Logout route: The user can Logout so that the user can get back to the Login Page

```
150
151 @app.route('/logout')
152 def logout():
153     session.pop('user', None)
154     return redirect(url_for('index'))
155
```

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

```
42
43  # Routes
44  @app.route('/')
45  def index():
46      return render_template('index.html')
47
48  @app.route('/veg-pickles')
49  def veg_pickles():
50      return render_template('veg_pickles.html', items=veg_items)
51
52  @app.route('/non-veg-pickles')
53  def non_veg_pickles():
54      return render_template('non_veg_pickles.html', items=non_veg_items)
55
56  @app.route('/snacks')
57  def snacks():
58      return render_template('snacks.html', items=snack_items)
59
```

Check out Route:

```python
    return render_template('cart.html', cart=cart_items, total=total)

@app.route('/checkout', methods=['GET', 'POST'])
def checkout():
    if request.method == 'POST':
        name = request.form['fullname']
        email = request.form['email']
        address = request.form['address']
        phone = request.form['phone']
        payment = request.form['payment']
        cart_items = session.get('cart', [])
        total = sum(int(i['price']) for i in cart_items)
        order_id = str(uuid.uuid4())

        orders_table.put_item(Item={
            'order_id': order_id,
            'name': name,
            'email': email,
            'address': address,
            'phone': phone,
            'payment': payment,
            'total': total,
            'items': cart_items
        })

        send_email(email, "Order Confirmation", f"Thank you {name} for your order! Total

        session.pop('cart', None)
        return render_template('success.html', name=name, order_id=order_id)

    return render_template('checkout.html')
```



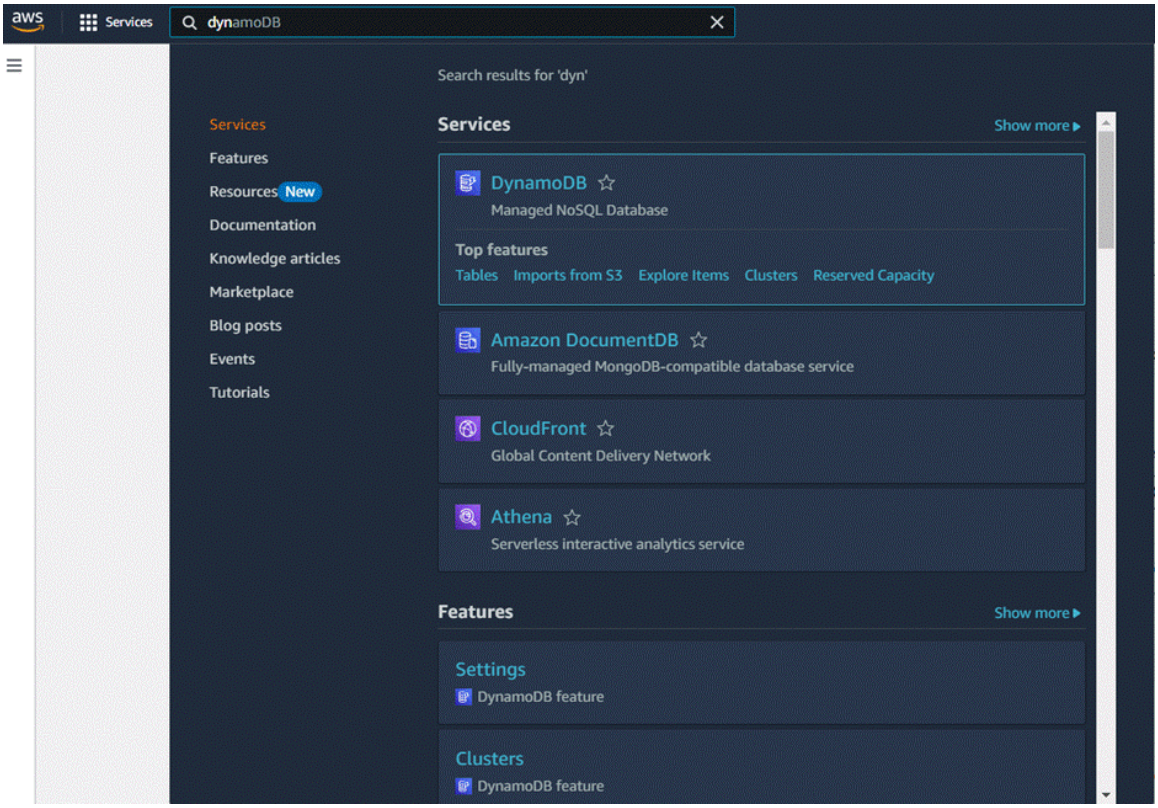
```python
@app.route('/add_to_cart_route', methods=['POST'])
def add_to_cart_route():
    item = request.form.to_dict()
    cart = session.get('cart', [])
    cart.append(item)
    session['cart'] = cart
    return redirect(request.referrer)

@app.route('/remove_from_cart', methods=['POST'])
def remove_from_cart():
    index = int(request.form['index'])
    cart = session.get('cart', [])
    if 0 <= index < len(cart):
        cart.pop(index)
    session['cart'] = cart
    return redirect(url_for('cart'))

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

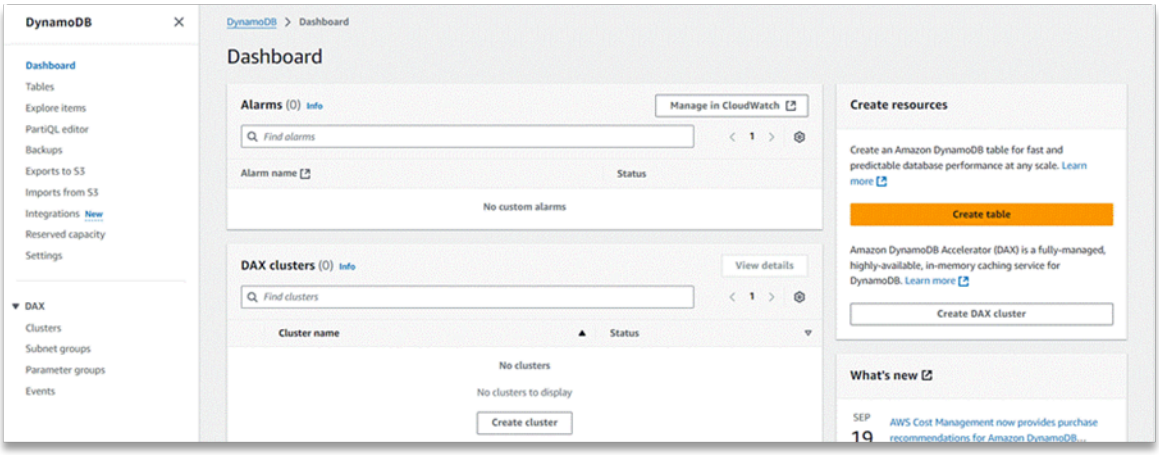
```python
@app.route('/sucess')
def success():
    return render_template('sucess.html')
```

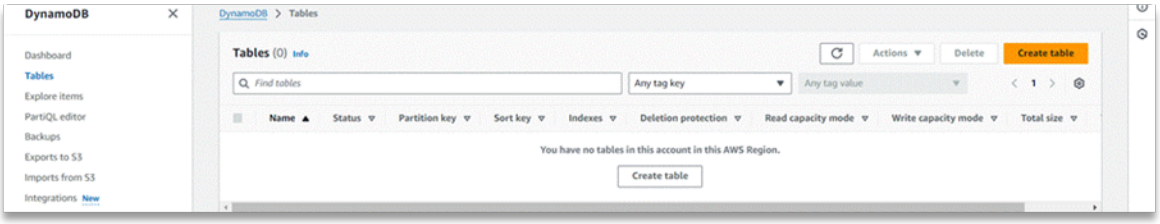
Milestone 2. AWS Account Setup and Login

Milestone 3 : DynamoDB Database Creation and Setup
Navigate to the DynamoDB

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

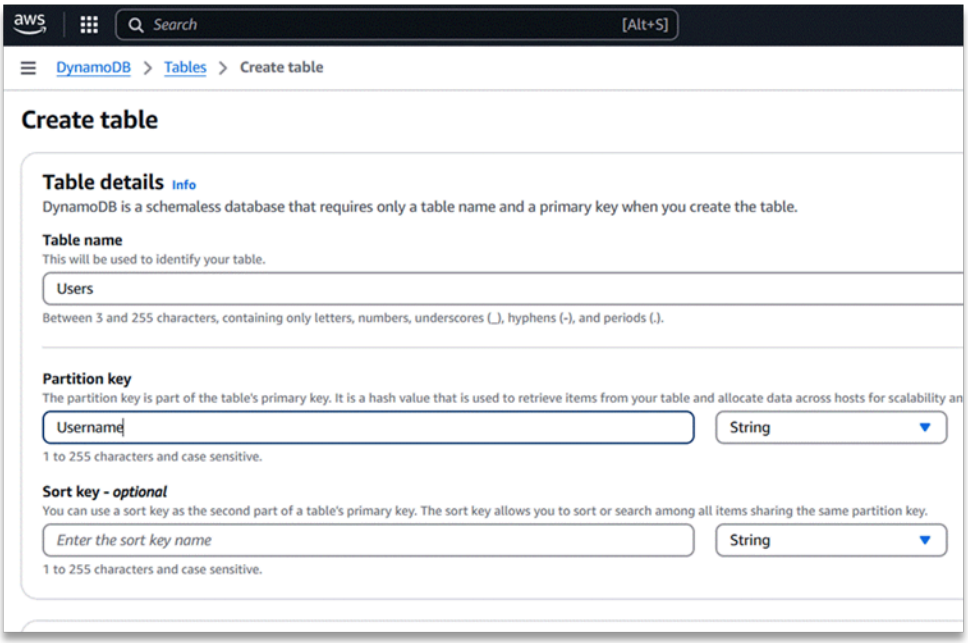


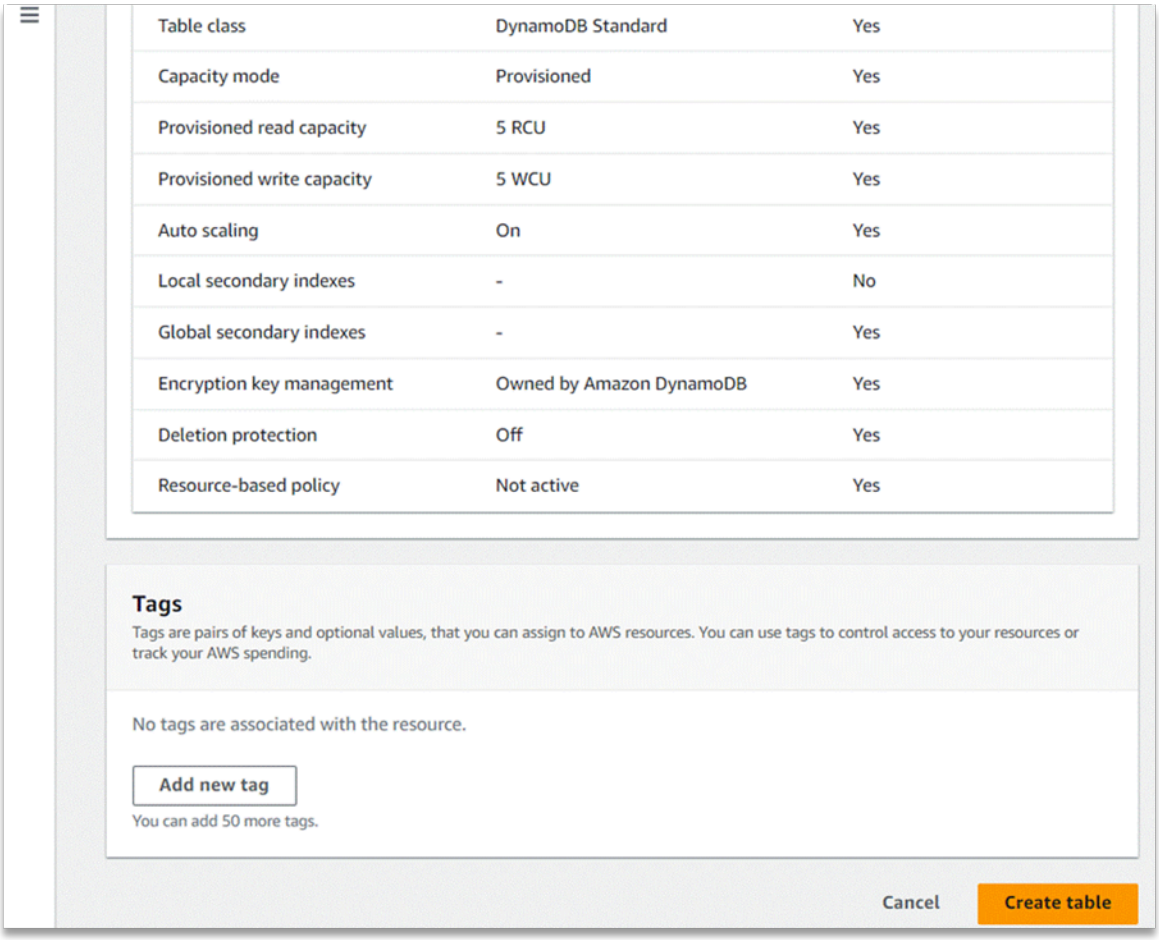


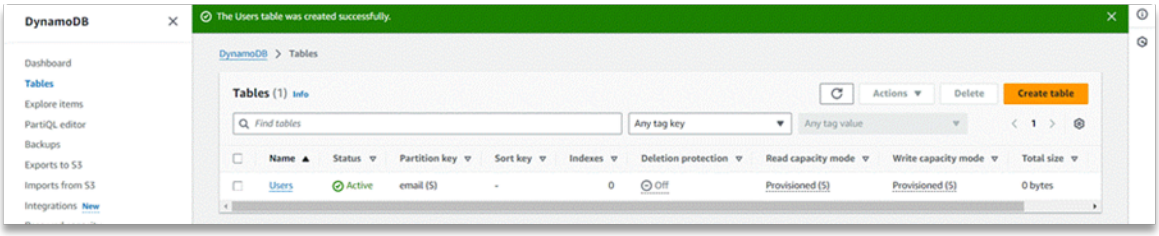


Create a DynamoDB table for storing data

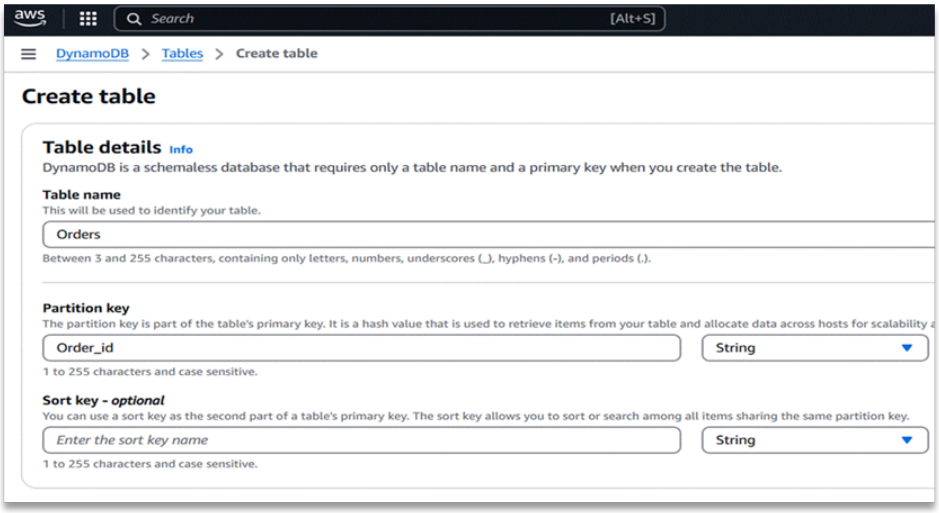
- Create Users table with partition key “Username” with type String and click on create table

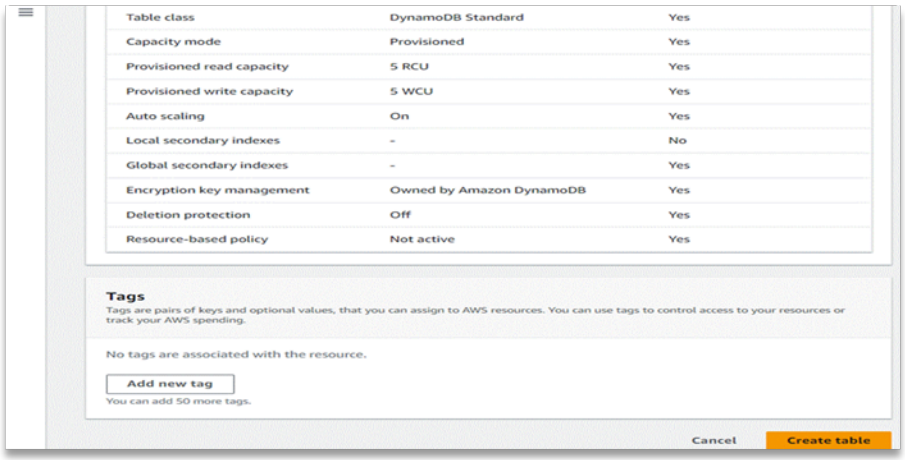


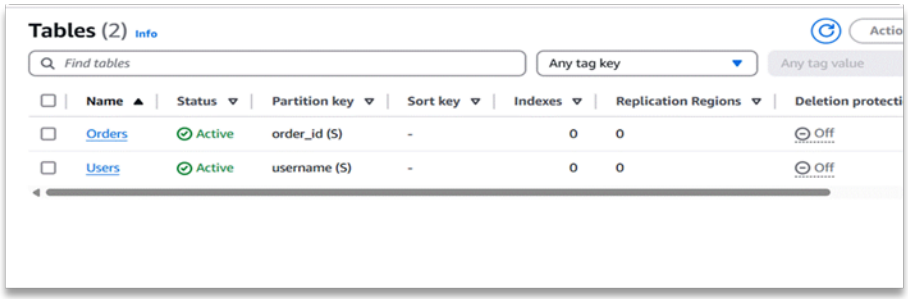




Follow the same steps to create an Orders table with Order_id as the primary key to store Order details.



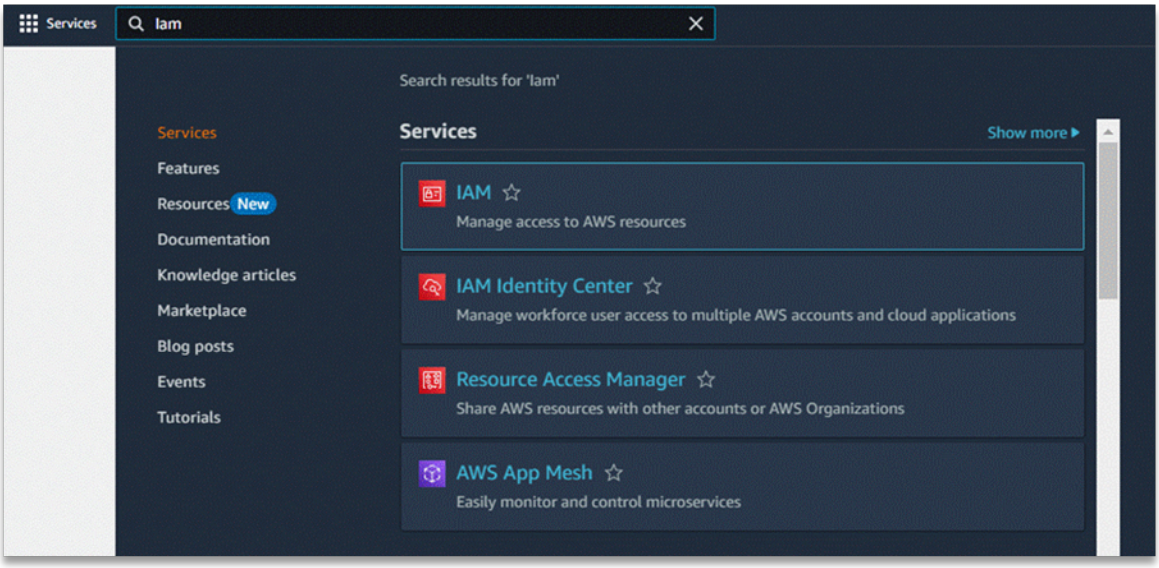


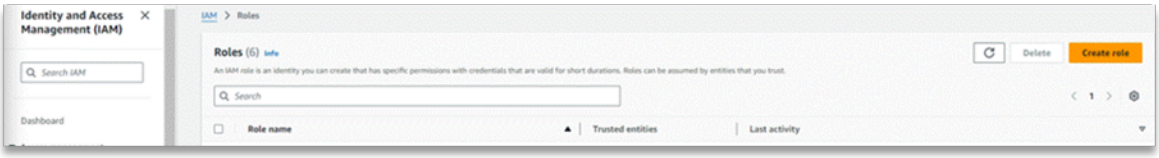


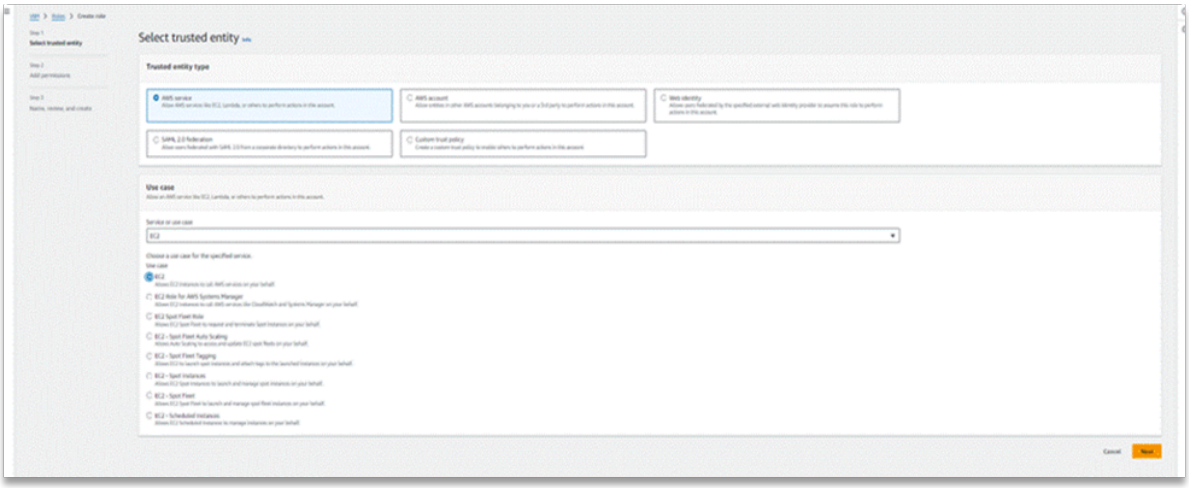
Milestone 4 : IAM Role Setup

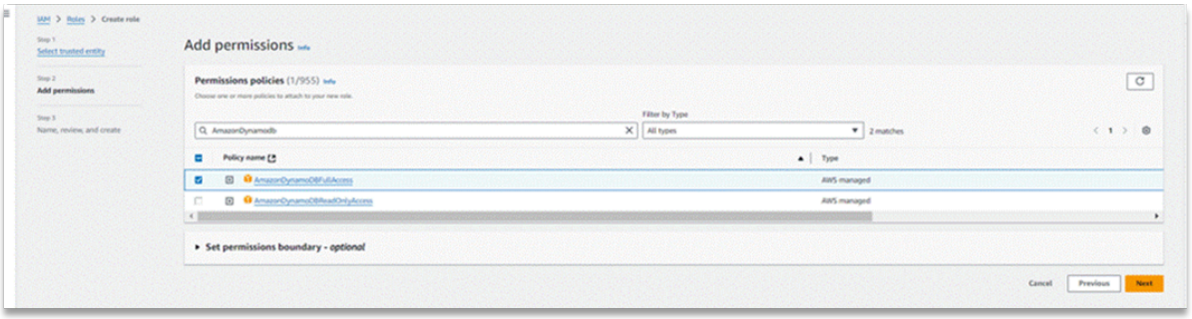
Create IAM Role.

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


Services
Iam
Search results for 'Iam'
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Services
Show more
IAM
Manage access to AWS resources
IAM Identity Center
Manage workforce user access to multiple AWS accounts and cloud applications
Resource Access Manager
Share AWS resources with other accounts or AWS Organizations
AWS App Mesh
Easily monitor and control microservices


Identity and Access Management (IAM)
Search IAM
Dashboard
IAM > Roles
Roles (6) 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.
Search
Delete
Create role
Role name
Trusted entities
Last activity

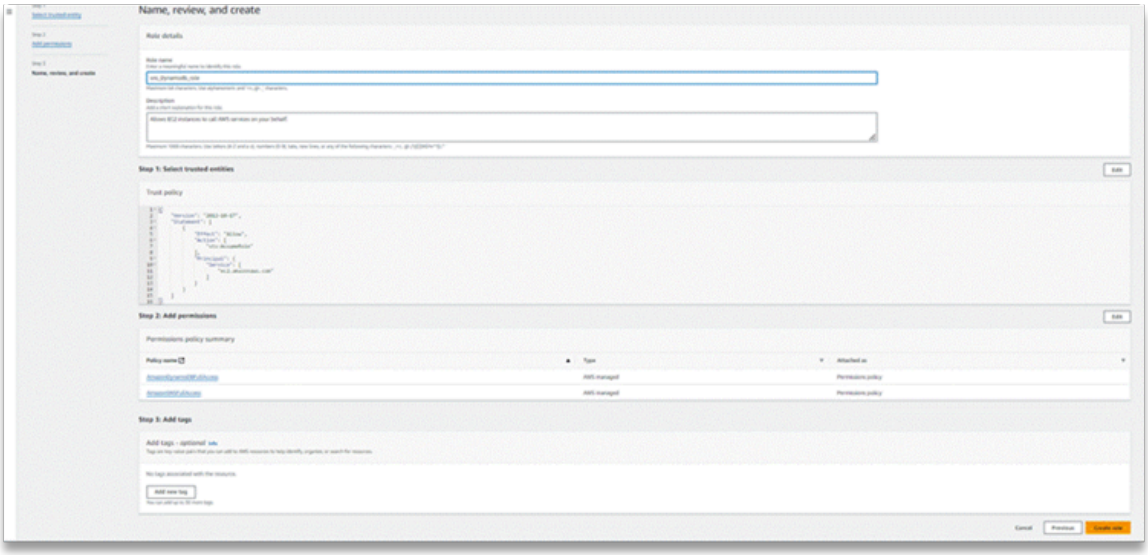

Select trusted entity
Trusted entity type
Use case

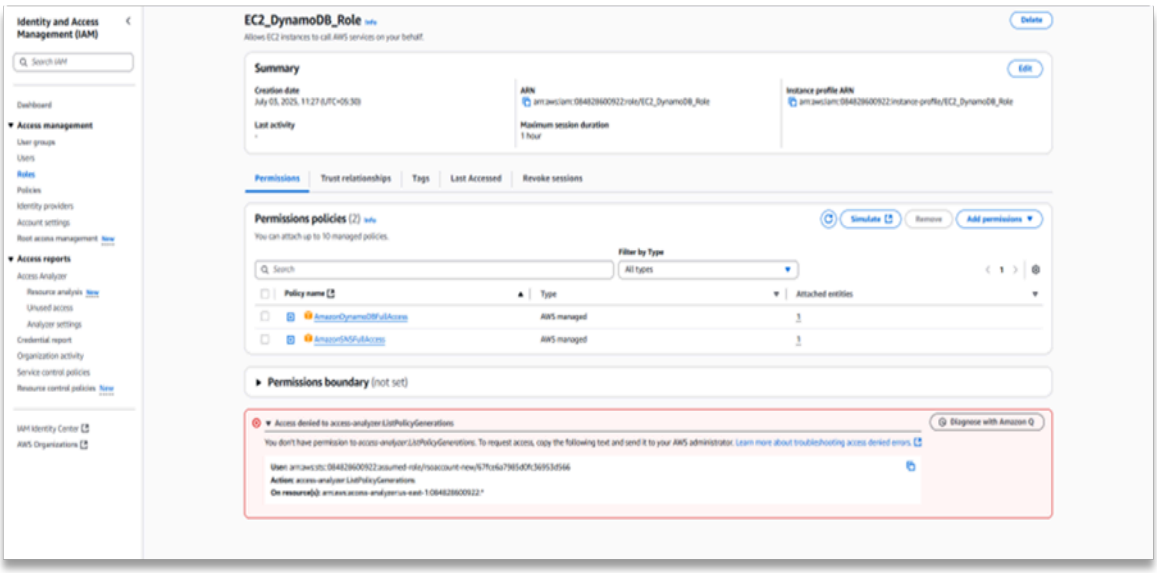


Attach Policies

Attach the following policies to the role:

- AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.



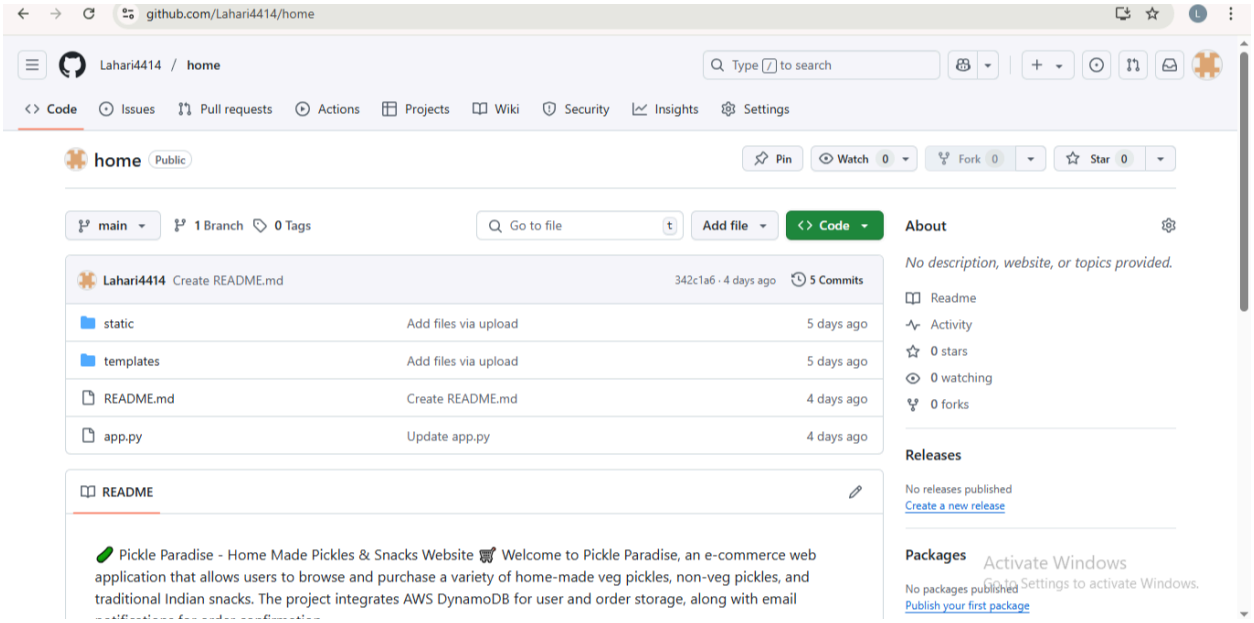


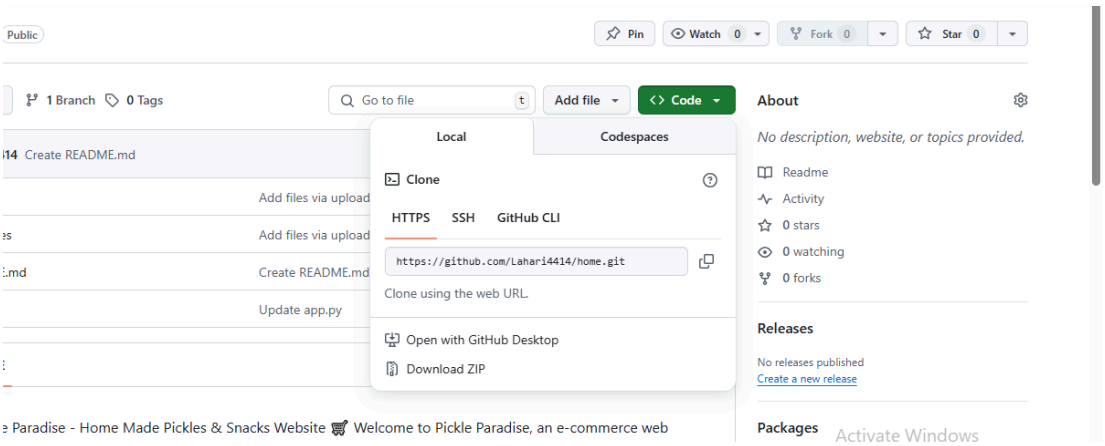
Milestone 5 : EC2 Instance Setup

To set up a public EC2 instance, choose an appropriate Amazon Machine Image (AMI) and instance type. Ensure the security group allows inbound traffic on necessary ports (e.g., HTTP/HTTPS for web applications). After launching the instance, associate it with an Elastic IP for consistent public access, and configure your application or services to be publicly accessible.

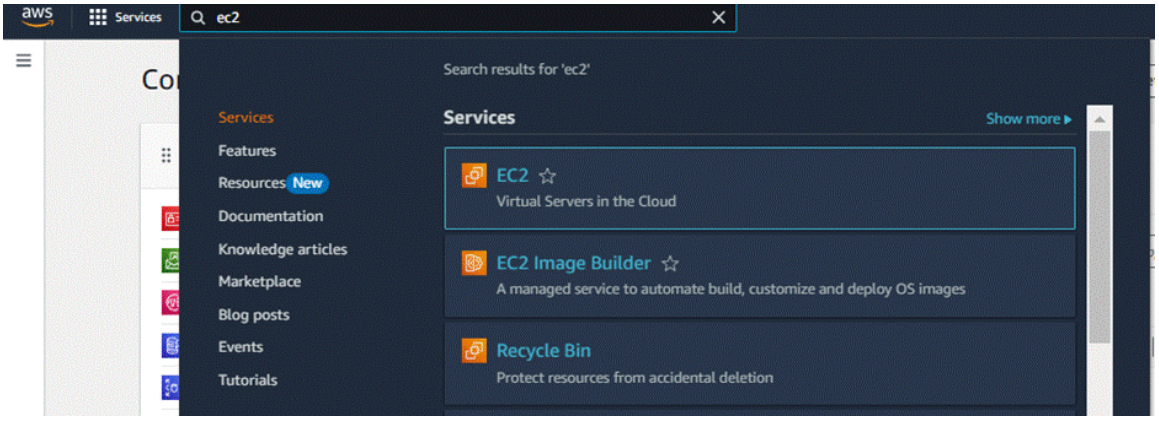
Load your Project Files to GitHub

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

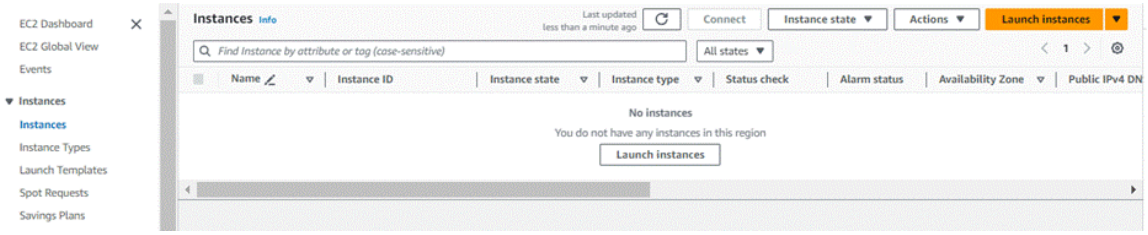


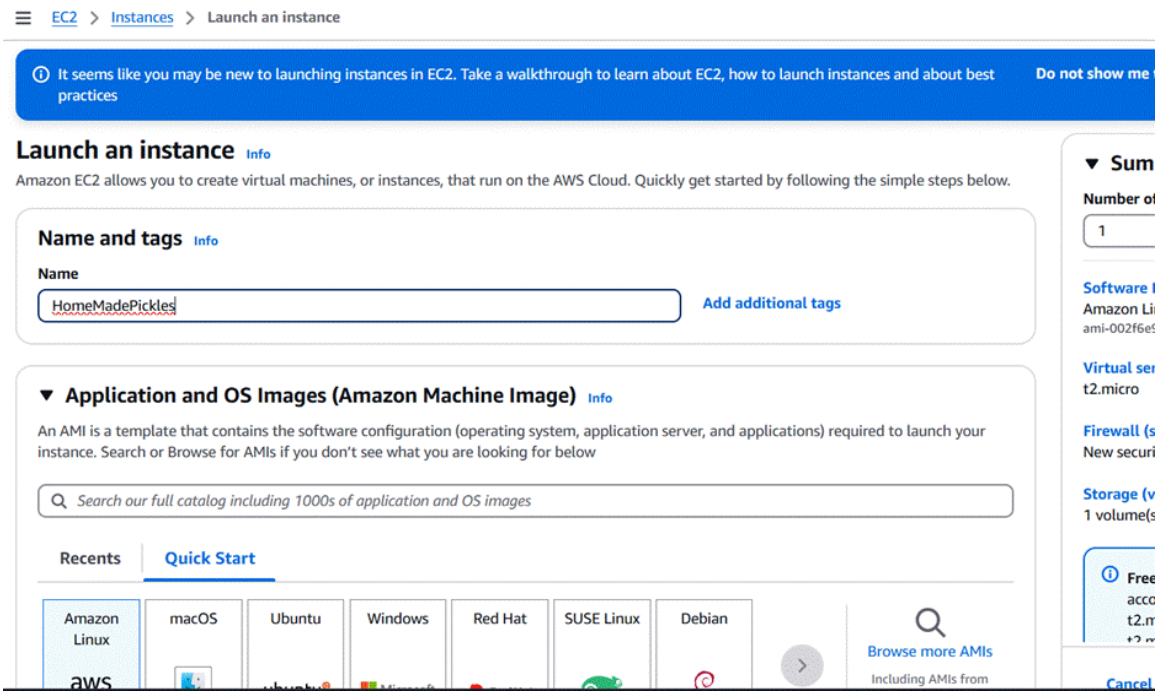


Launch an EC2 instance to host the Flask.

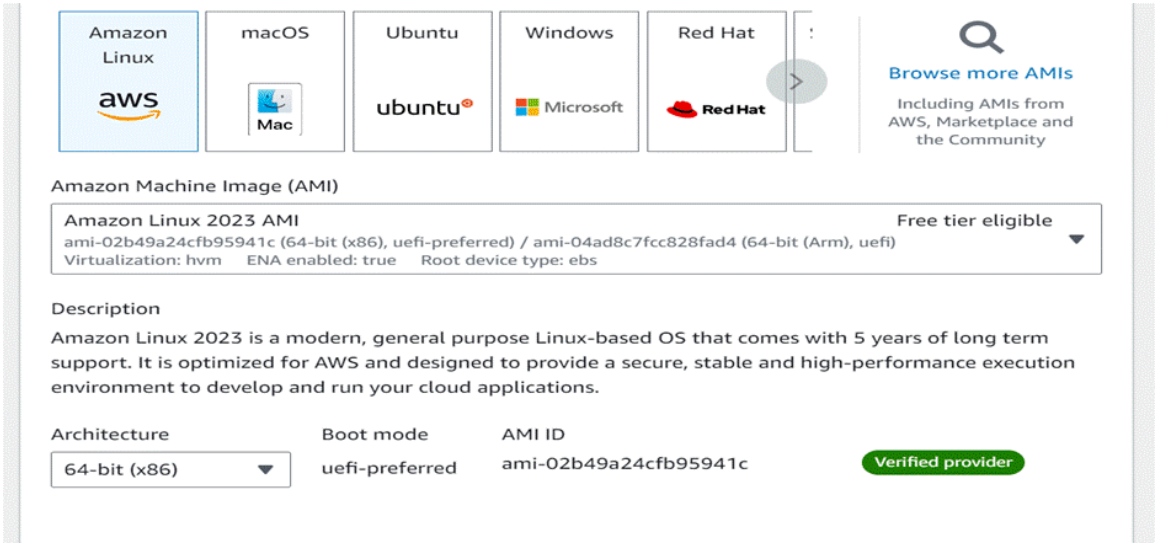


- **Click on Launch instance to launch EC2 instance**

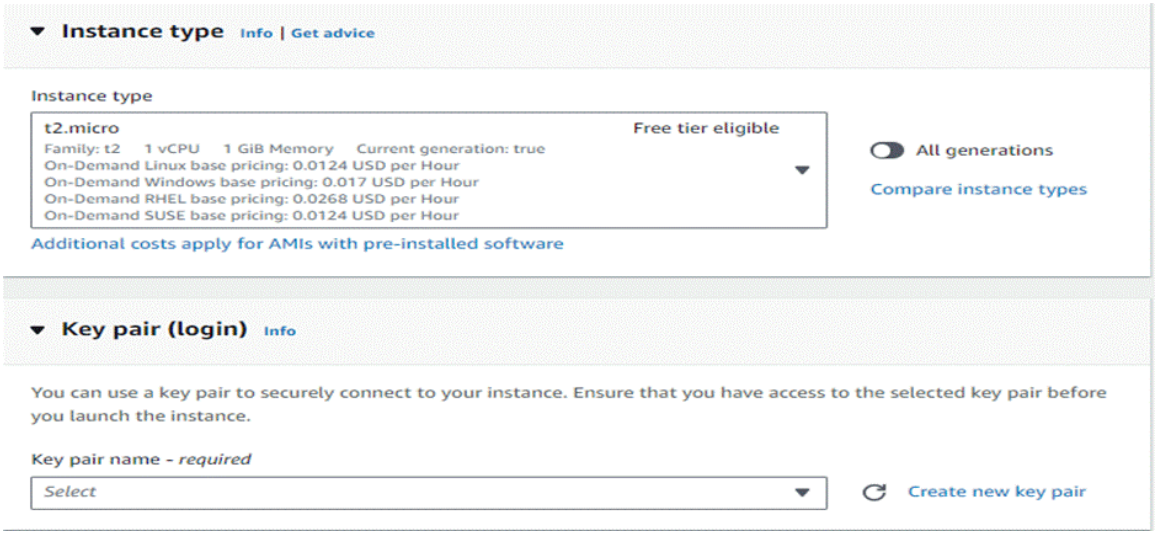


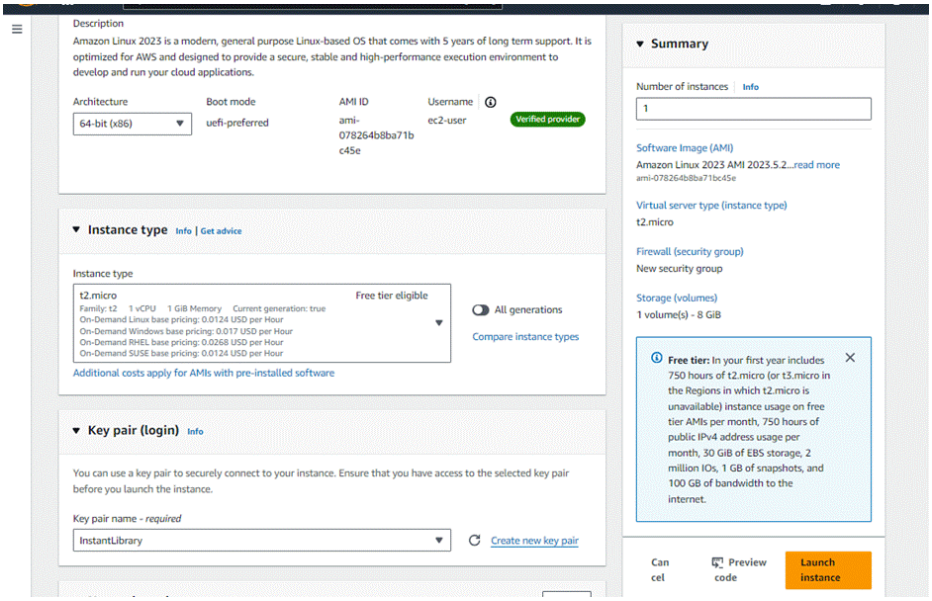


? Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

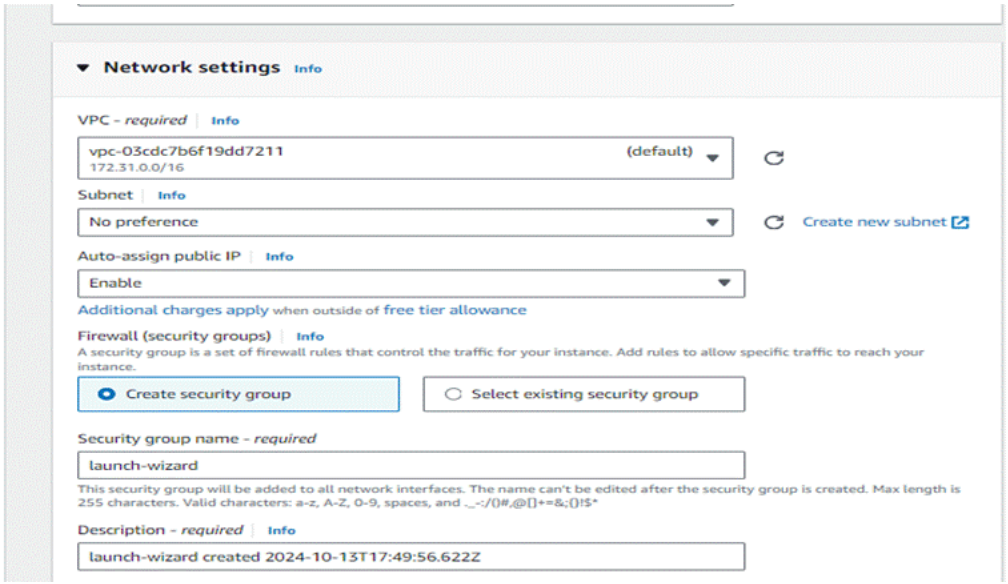


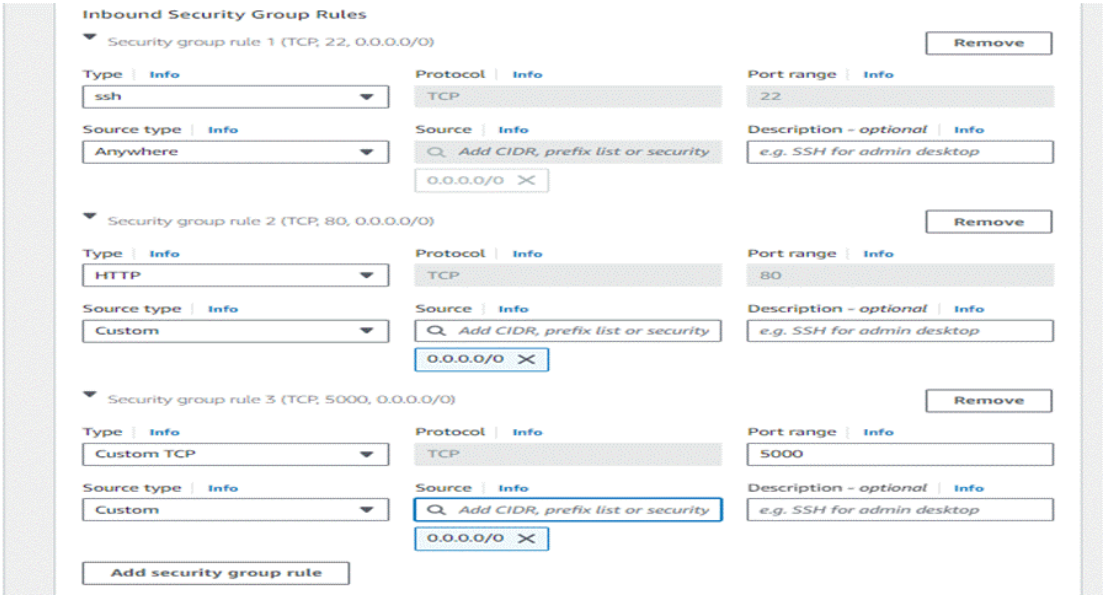
- **Create and download the key pair for Server access.**

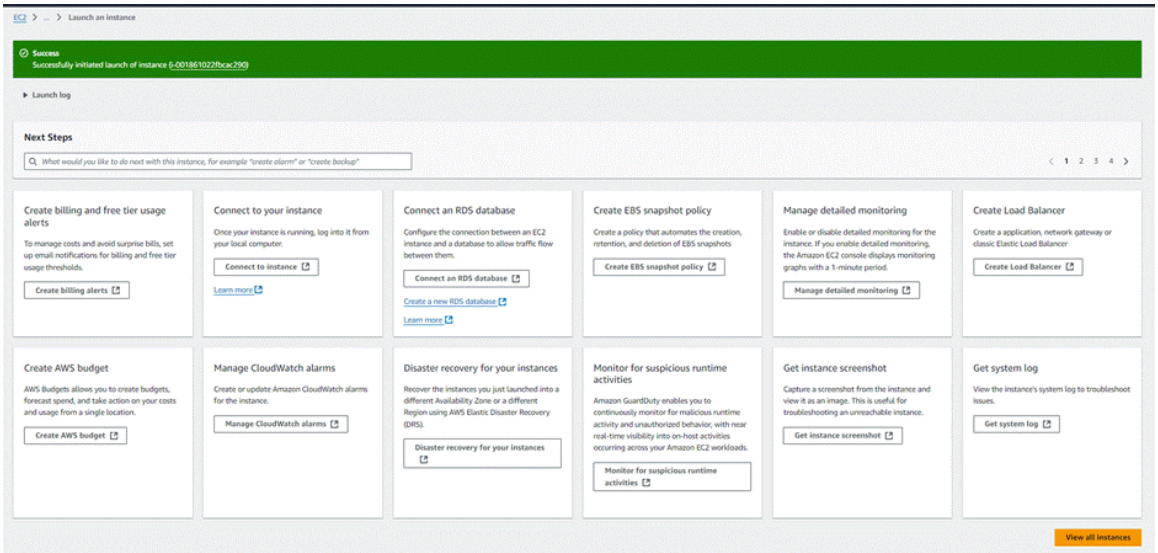




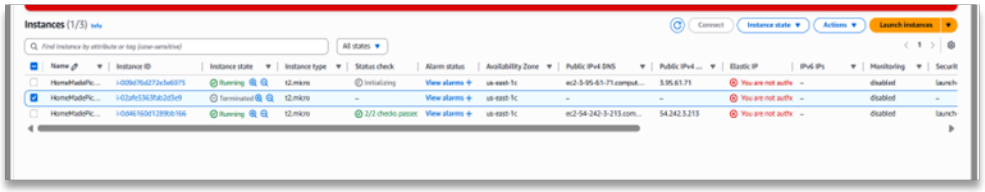
Configure security groups for HTTP, and SSH access.

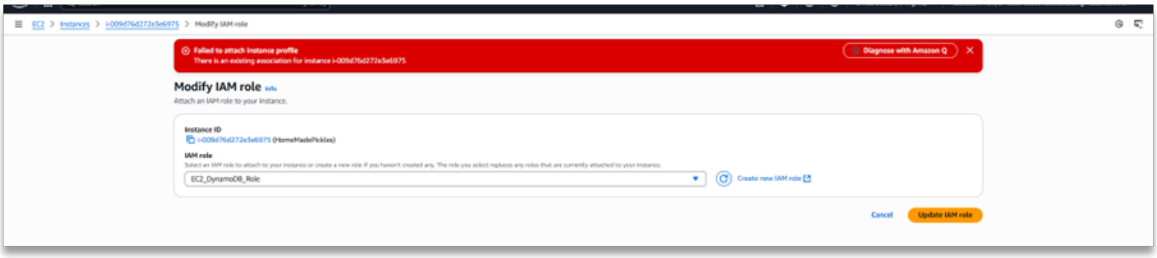

Network settings Info
VPC - required Info
vpc-03cdc7b6f19dd7211 (default)
172.31.0.0/16
Subnet Info
No preference
Create new subnet
Auto-assign public IP Info
Enable
Additional charges apply when outside of free tier allowance
Firewall (security groups) Info
A security group is a set of firewall rules 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

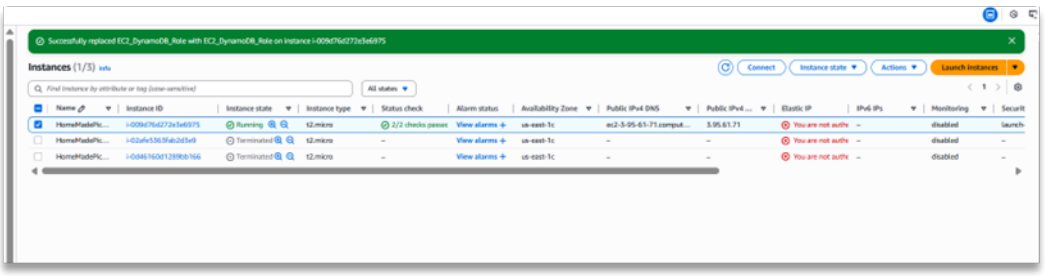

Inbound Security Group Rules
Security group rule 1 (TCP, 22, 0.0.0.0/0)
Remove
Type Info
ssh
Protocol Info
TCP
Port range Info
22
Source type Info
Anywhere
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Security group rule 2 (TCP, 80, 0.0.0.0/0)
Remove
Type Info
HTTP
Protocol Info
TCP
Port range Info
80
Source type Info
Custom
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Security group rule 3 (TCP, 5000, 0.0.0.0/0)
Remove
Type Info
Custom TCP
Protocol Info
TCP
Port range Info
5000
Source type Info
Custom
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Add security group rule



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







Now connect the EC2 with the files

Install Software on the EC2 Instance
Install Python3, Flask, and Git:
On Amazon Linux 2:
sudo yum update -y
sudo yum install python3 git
sudo pip3 install flask boto3

Verify Installations:

flask --version
git --version

Clone Your Flask Project from GitHub

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

- Run: https://github.com/Lahari4414/home.git

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

  here: https://github.com/Lahari4414/home.git

- This will download your project to the EC2 instance.

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

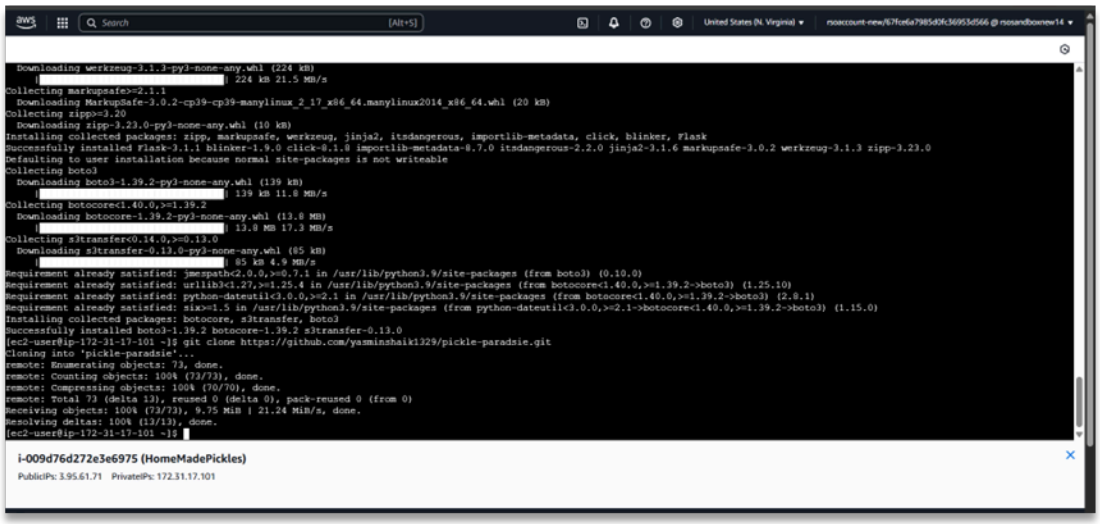
- cd Homemadepicklesandsnacks
- cd “Home Made Pickles1”

Create a Virtual Environment:

- python3 -m venv venv
- source venv/bin/activate
- sudo yum install python3 git
- sudo pip3 install flask boto3

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

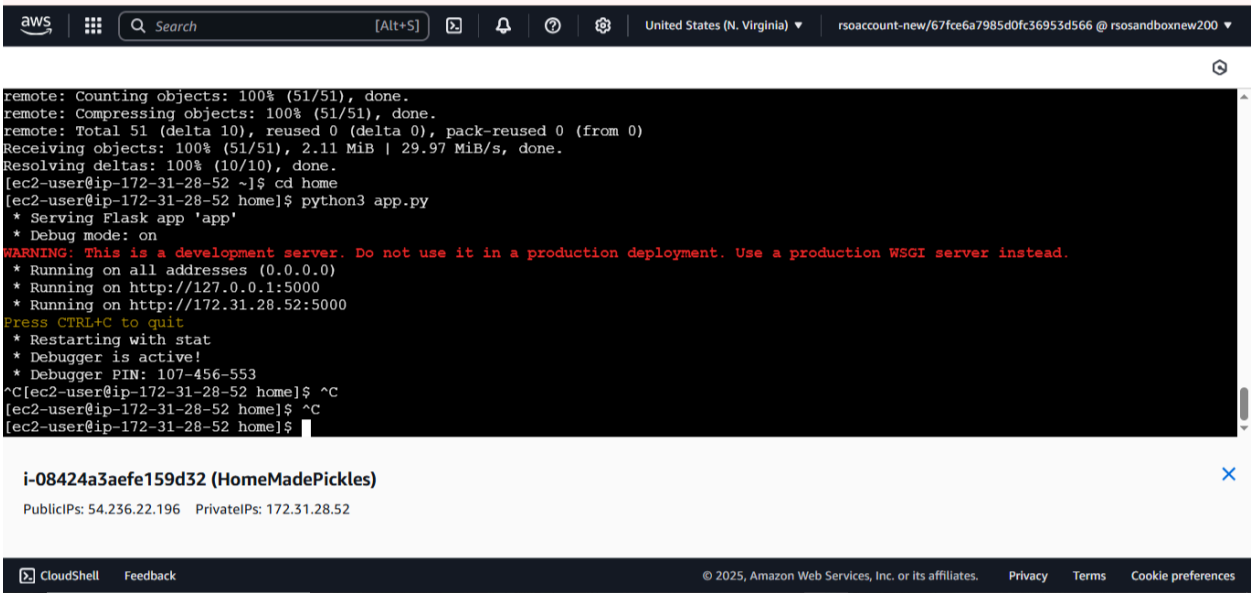
- Run the Flask Application
- sudo flask run --host=0.0.0.0 --port=5000



Verify the Flask app is running:

http://your-ec2-public-ip

- Run the Flask app on the EC2 instance



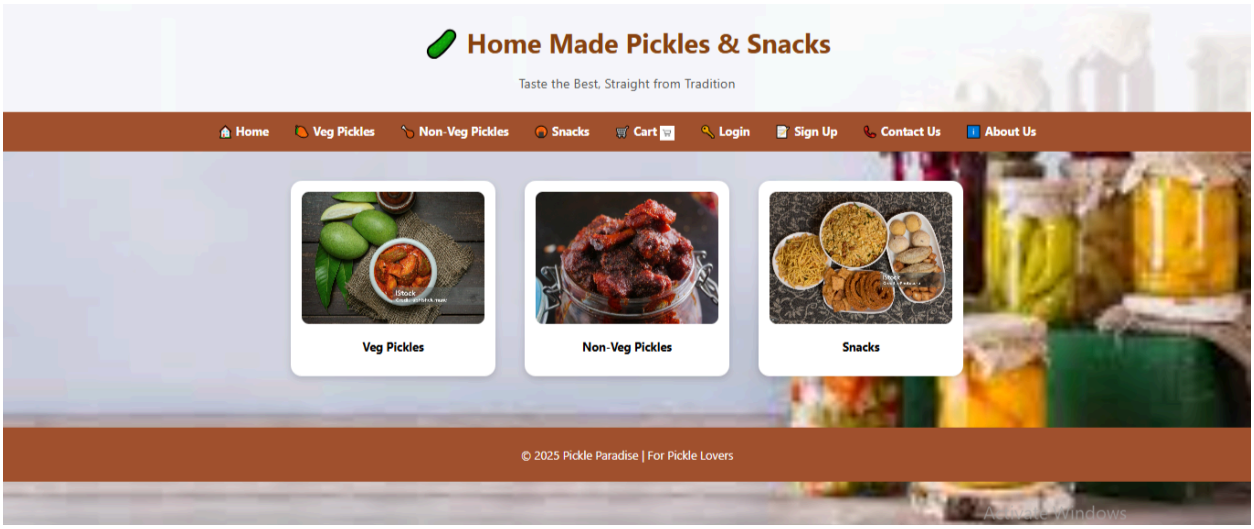
Milestone 7 : Testing and Deployment

Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment.

Functional testing to verify the Project

Welcome page:

Home page:

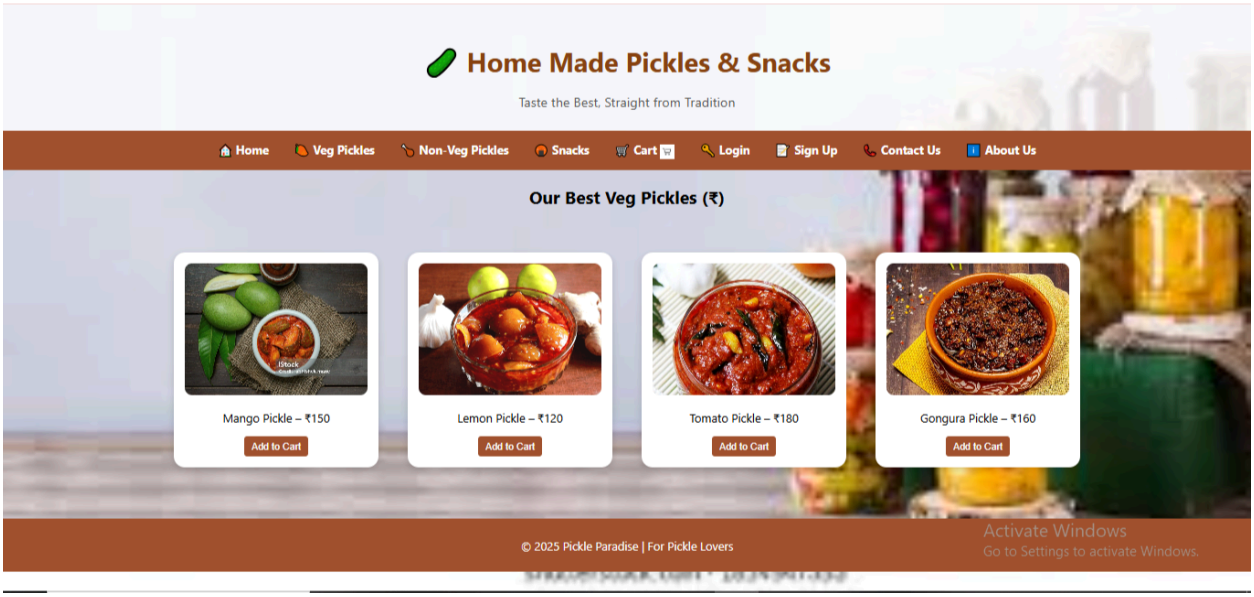


Signup page:

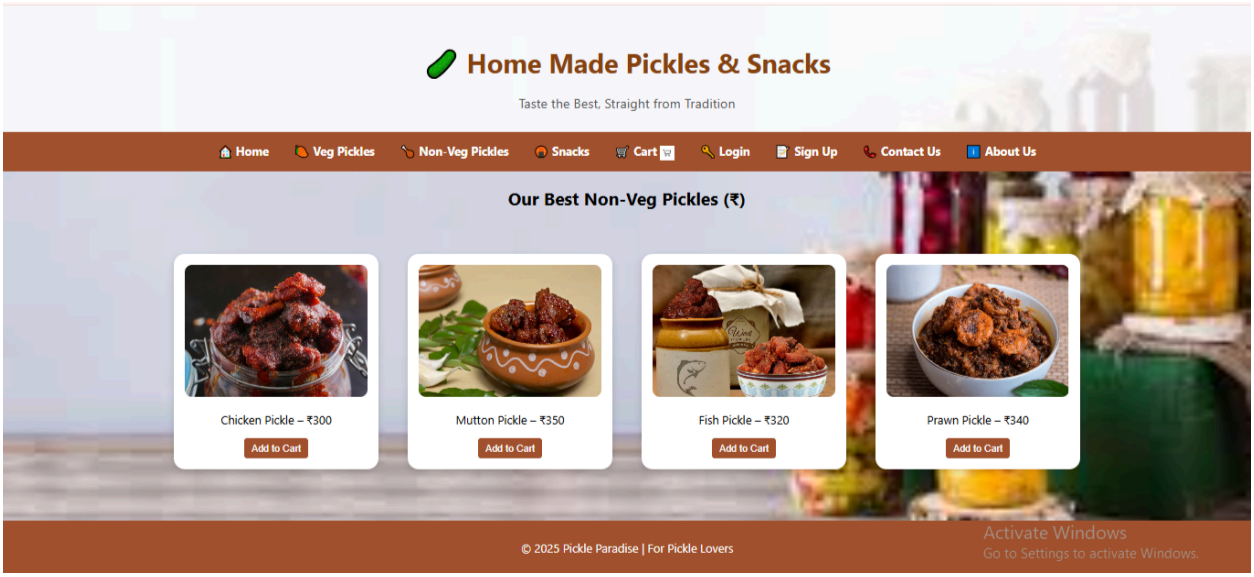
Login page:


Home Made Pickles & Snacks
Taste the Best, Straight from Tradition
Login
Username
Password
Login
shutterstock.com · 1814547353

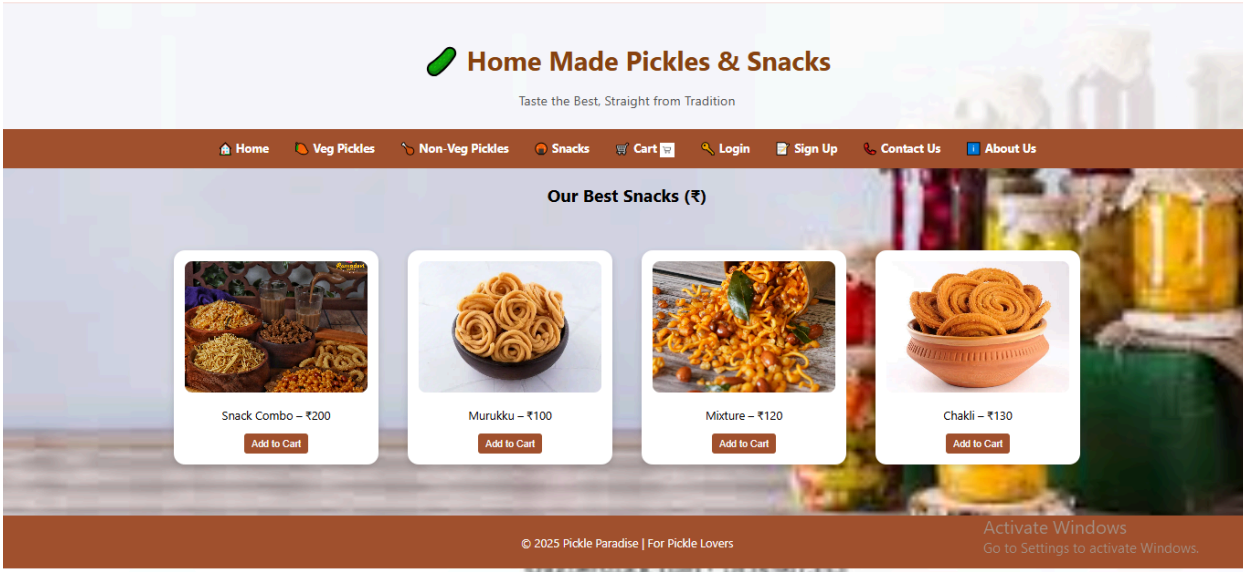
Veg pickles:


Home Made Pickles & Snacks
Taste the Best, Straight from Tradition
Home
Veg Pickles
Non-Veg Pickles
Snacks
Cart
Login
Sign Up
Contact Us
About Us
Our Best Veg Pickles (₹)
Mango Pickle – ₹150
Add to Cart
Lemon Pickle – ₹120
Add to Cart
Tomato Pickle – ₹180
Add to Cart
Gongura Pickle – ₹160
Add to Cart
© 2025 Pickle Paradise | For Pickle Lovers

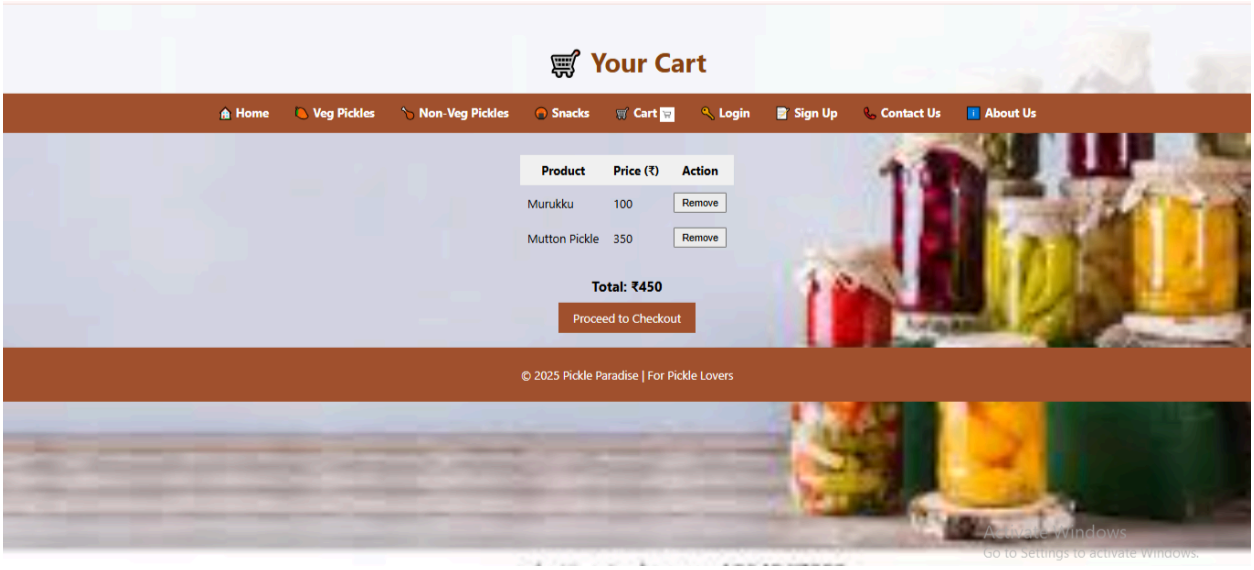
Non-Veg pickles:



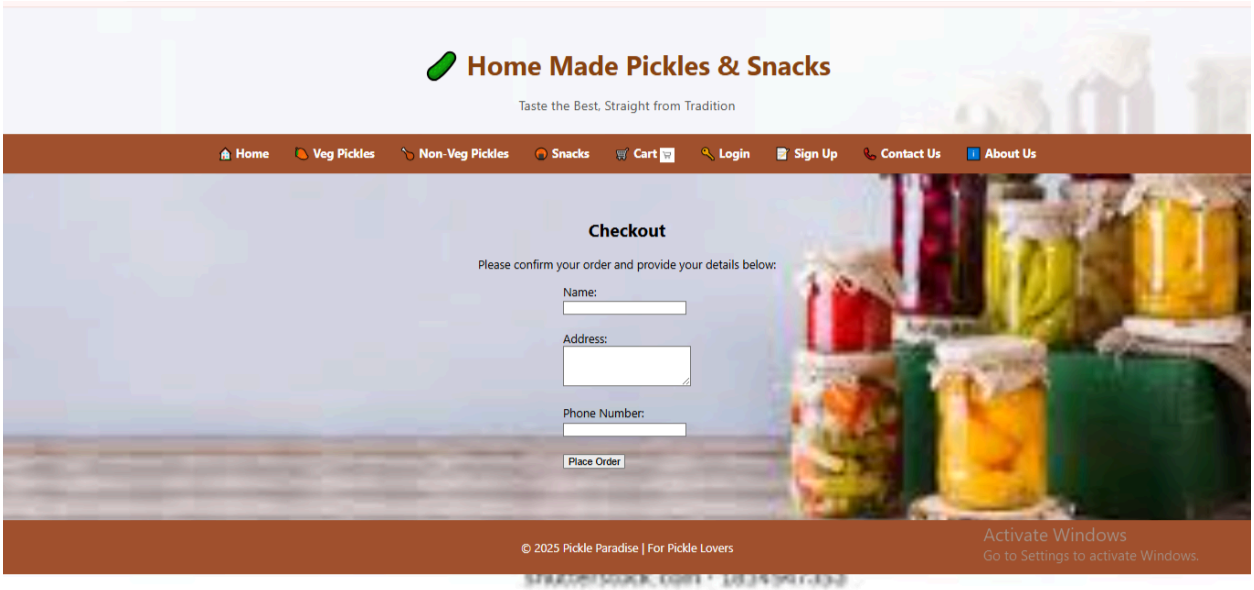
Snacks:



Cart Page:



Checkout Page:



Conclusion

The **Homemade Pickles: Taste the Best** project demonstrates the successful development and deployment of a scalable, cloud-based e-commerce platform using modern web technologies and AWS services. By leveraging **Flask** for backend development, **AWS EC2** for hosting, **DynamoDB** for secure data management, and **SNS** for real-time notifications, the system provides a seamless and responsive experience for both customers and administrators.

This project not only solves the limitations of traditional small-scale selling but also opens up a professional and automated channel for reaching wider audiences. Customers can easily browse, select, and order high-quality homemade products with real-time updates and secure transactions, while sellers can manage orders more efficiently.

The architecture ensures reliability, scalability, and performance — making it future-ready as customer demand and product variety increase. Through this project, the effective integration of cloud services and Python web development is clearly showcased, reflecting practical implementation of theoretical knowledge and real-world problem-solving skills.

In conclusion, this solution provides a **modern, reliable, and efficient e-commerce experience**, supporting both digital transformation for small businesses and convenience for customers.