

# PICKLE PARADISE: CLOUD-BASED ONLINE PICKLE & SNACK STORE

## Project Description

Pickle Paradise is an e-commerce platform designed to cater to customers who love traditional Indian pickles and snacks. With increasing demand and the desire to reach a wider audience, the platform was developed using Flask for backend logic, AWS EC2 for deployment, and DynamoDB for scalable database operations. AWS SNS handles real-time customer notifications for orders and inquiries. The system allows users to browse, register, log in, add items to the cart, and place orders—all through a user-friendly web interface.

## Scenarios

### 1. Seamless Online Ordering

Using Flask and EC2, customers can explore a curated selection of pickles (veg, non-veg) and snacks, view prices and images, and place orders with just a few clicks. DynamoDB stores user data and order history, ensuring scalability and speed.

### 2. Instant Notifications via AWS SNS

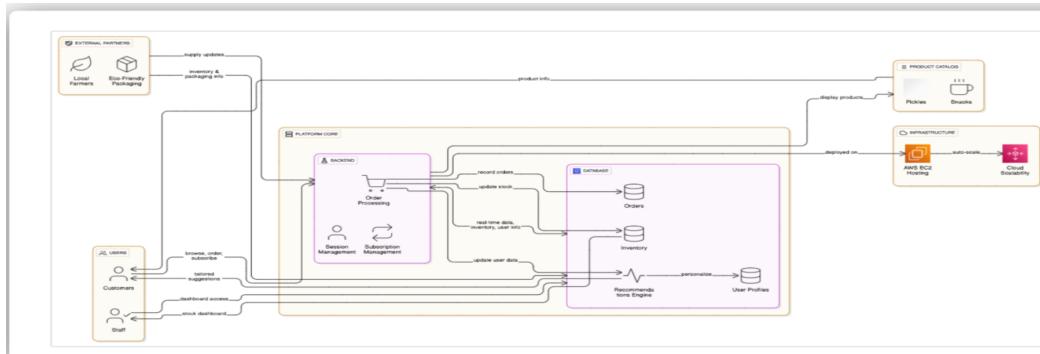
Whenever a user places an order, AWS SNS sends confirmation emails both to the customer and the admin. Flask backend handles the logic while SNS ensures users are updated instantly about their order status.

### 3. Easy Inventory & Category Management

Admins can view and manage inventory (manually or via additional interface), while customers easily switch between pages like Snacks, Veg Pickles, Non-Veg Pickles, Cart, and Checkout. Flask handles page routing and real-time data rendering.

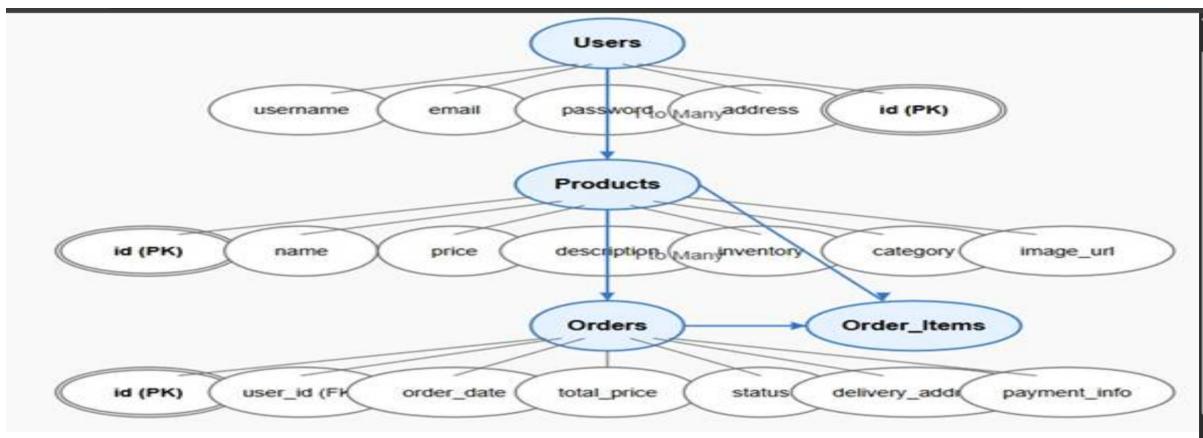
## AWS Architecture

The diagram below represents the complete architecture of the Pickle Paradise system:



## Entity Relationship (ER) Diagram

The ER diagram below outlines the structure of the application's database:



## Pre-requisites

1. AWS Account Setup
2. IAM Configuration
3. Amazon EC2 Basics
4. DynamoDB Basic
5. AWS SNS Setup
6. Git Version Control Basics

## **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

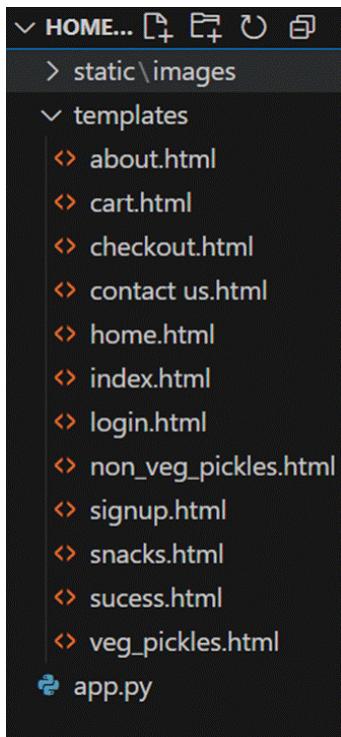
### **Milestone 8. Testing and Deployment**

- 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
from werkzeug.security import generate_password_hash, check_password_hash
import boto3
from datetime import datetime
import json,uuid

app = Flask(__name__)
```

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

```

dynamodb = boto3.resource('dynamodb', region_name='ap-south-1') # e.g., 'us-east-1'
users_table = dynamodb.Table('Users')
orders_table = dynamodb.Table('Orders')

```

```

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

```

- Routes for Web Pages

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

```

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

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

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

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

        user = response['Item']

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

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

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

```

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

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

        # Hash password before storing
        hashed_password = generate_password_hash(password)

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

        return redirect(url_for('login'))

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

```

```

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

    return render_template('signup.html')

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

```

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

```

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

```

- **Home Route:** Home page contains the routing for different categories which are Veg\_pickles,Non\_Veg\_pickles,Snacks.

```

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

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

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

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

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

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

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

```

Restart Visual Studio

- **Check out Route:**

```
@app.route('/checkout', methods=['GET', 'POST'])
def checkout():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    error_message = None # Variable to hold error messages

    if request.method == 'POST':
        try:
            # Extract form data safely
            name = request.form.get('name', '').strip()
            address = request.form.get('address', '').strip()
            phone = request.form.get('phone', '').strip()
            payment_method = request.form.get('payment', '').strip()

            # Validate inputs
            if not all([name, address, phone, payment_method]):
                return render_template('checkout.html', error="All fields are required.")

            if not phone.isdigit() or len(phone) != 10:
                return render_template('checkout.html', error="Phone number must be exactly 10 digits.")

            # Get cart data from hidden inputs
            cart_data = request.form.get('cart_data', '[]')
            total_amount = request.form.get('total_amount', '0')
```

(1) Restart Visual Studio Code to apply the latest changes.

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

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

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

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

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

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

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

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

## Milestone 2. AWS Account Setup and Login

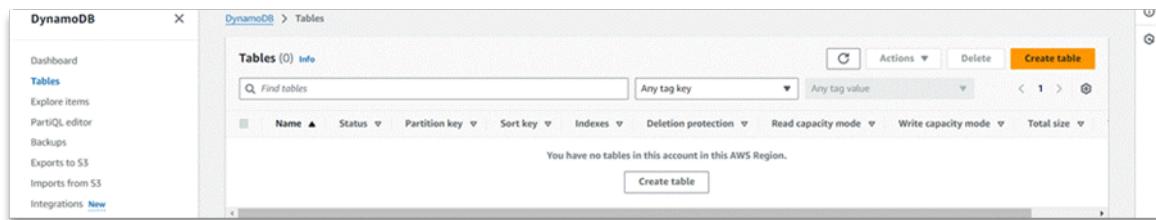
## Milestone 3 : DynamoDB Database Creation and Setup

### Navigate to the DynamoDB

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

The screenshot shows the AWS search interface. The search bar at the top contains the query 'dynamoDB'. Below the search bar, there is a sidebar with links to 'Services', 'Features', 'Resources New', 'Documentation', 'Knowledge articles', 'Marketplace', 'Blog posts', 'Events', and 'Tutorials'. The main content area is titled 'Search results for 'dyn'' and shows a list of services. The first item in the list is 'DynamoDB' with a star icon, described as a 'Managed NoSQL Database'. Below it are other services: 'Amazon DocumentDB' (Fully-managed MongoDB-compatible database service), 'CloudFront' (Global Content Delivery Network), and 'Athena' (Serverless interactive analytics service). There are also sections for 'Features' with 'Settings' and 'Clusters'.

The screenshot shows the DynamoDB Dashboard. The left sidebar has sections for 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations New', 'Reserved capacity', and 'Settings'. A collapsed section for 'DAX' includes 'Clusters', 'Subnet groups', 'Parameter groups', and 'Events'. The main dashboard area has two sections: 'Alarms (0) Info' and 'DAX clusters (0) Info'. Both sections have search bars and status indicators. To the right, there is a 'Create resources' section with a large orange 'Create table' button. Below it is information about the Amazon DynamoDB Accelerator (DAX). At the bottom, there is a 'What's new' section with a note about AWS Cost Management.



## Create a DynamoDB table for storing data

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

**Table details** Info  
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.)

**Partition key**  
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and consistency.

1 to 255 characters and case sensitive.

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

1 to 255 characters and case sensitive.

Table 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.

**DynamoDB > Tables**

**Tables (1) Info**

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size
Users	Active	email (\$)	-	0	Off	Provisioned (\$)	Provisioned (\$)	0 bytes

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

aws | Search [Alt+S]

DynamoDB > Tables > Create table

### Create table

**Table details** Info

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

**Table name**  
This will be used to identify your table.  
**Orders**  
Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.)

**Partition key**  
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and performance.  
**Order\_id** String

1 to 255 characters and case sensitive.

**Sort key - optional**  
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.  
**Enter the sort key name** String  
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

**Tables (2)** Info

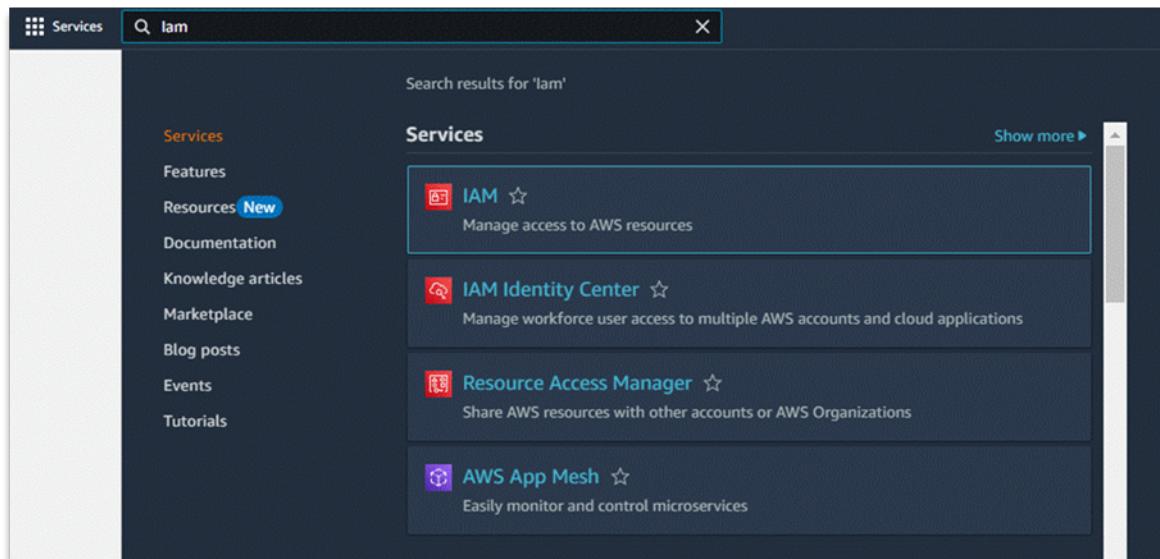
Find tables Any tag key Any tag value

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Action
<input type="checkbox"/>	<a href="#">Orders</a>	<span style="color: green;">Active</span>	order_id (\$)	-	0	0	<span style="border: 1px solid #ccc; padding: 2px;">Off</span>	<span style="border: 1px solid #ccc; padding: 2px;">Edit</span>
<input type="checkbox"/>	<a href="#">Users</a>	<span style="color: green;">Active</span>	username (\$)	-	0	0	<span style="border: 1px solid #ccc; padding: 2px;">Off</span>	<span style="border: 1px solid #ccc; padding: 2px;">Edit</span>

## 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.



The screenshot shows the 'Identity and Access Management (IAM)' service with the 'Roles' tab selected. The page title is 'IAM > Roles'. It includes a search bar, a 'Create role' button, and filters for 'Role name', 'Trusted entities', and 'Last activity'. The main content area displays a table of existing roles.

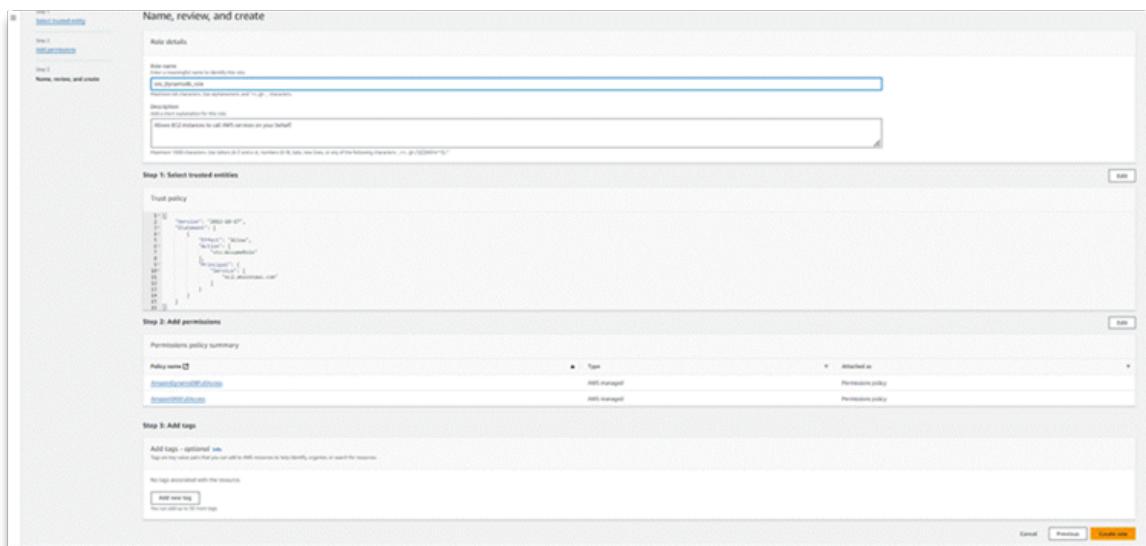
The screenshot shows the 'Select trusted entity' step in the 'Create role' wizard. It includes a 'Trusted entity type' section with options for 'AWS service', 'AWS account', and 'Web identity'. Below this is a 'Use case' section with a dropdown menu set to 'EC2' and a list of EC2-related use cases.

The screenshot shows the 'Add permissions' step in the 'Create role' wizard. It includes a 'Permissions policies' section with a search bar, a 'Filter by Type' dropdown, and a list of AWS managed policies. The policies listed are 'AmazonDynamoDBFullAccess' and 'AmazonDynamoDBReadWriteAccess'. At the bottom, there is a note about setting a 'permissions boundary'.

## Attach Policies

Attach the following policies to the role:

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

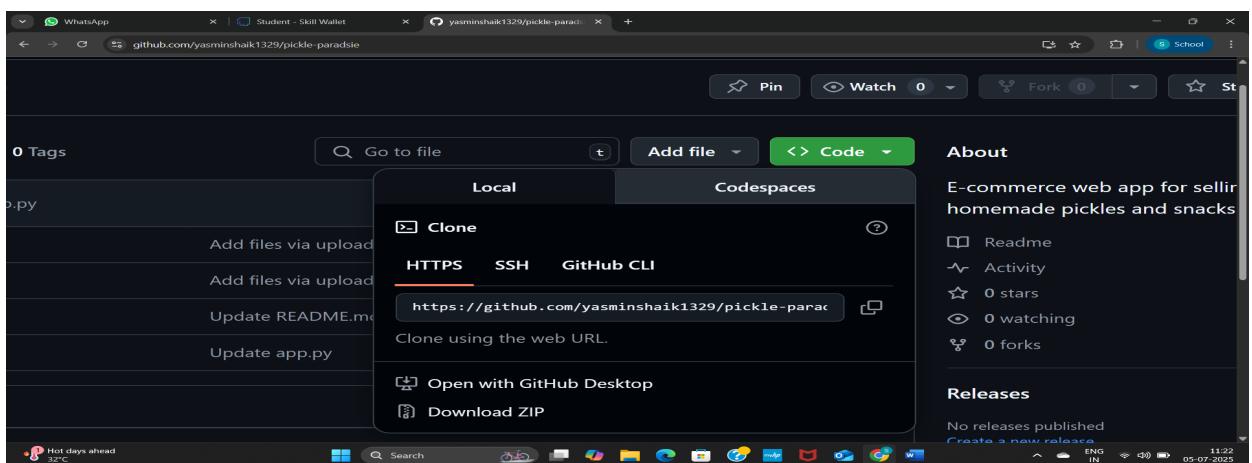
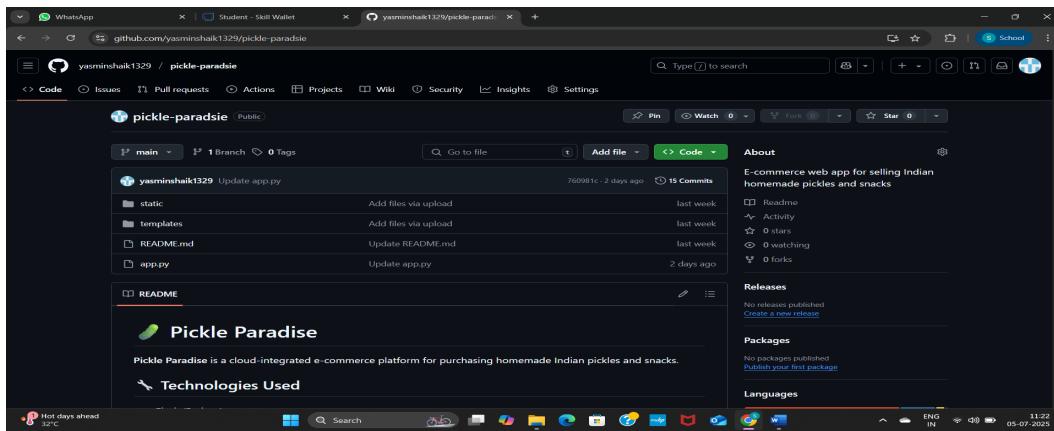


## Milestone 5 : EC2 Instance Setup

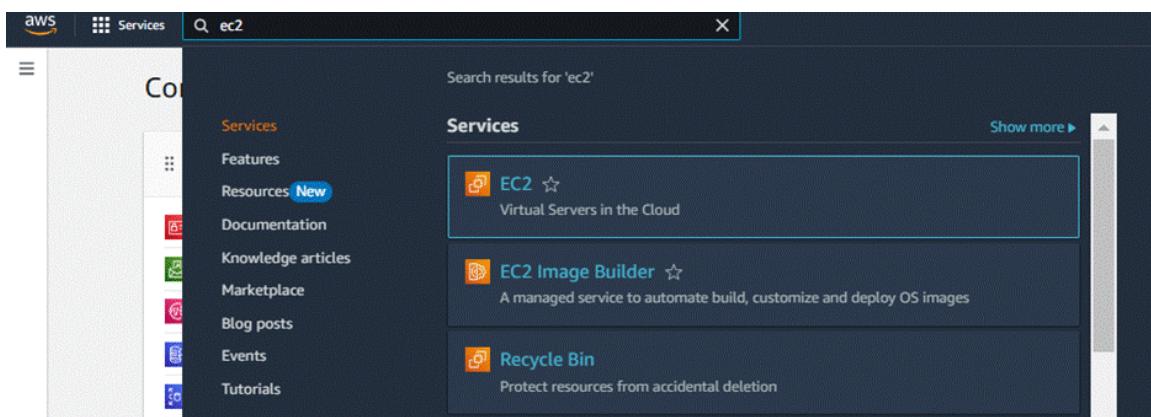
To set up a public EC2 instance, choose an appropriate Amazon Machine Image (AMI) and instance type. Ensure the security group allows inbound traffic on necessary ports (e.g., HTTP/HTTPS for web applications). After launching the instance, associate it with an Elastic IP for consistent public access,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

## Load your Project Files to GitHub

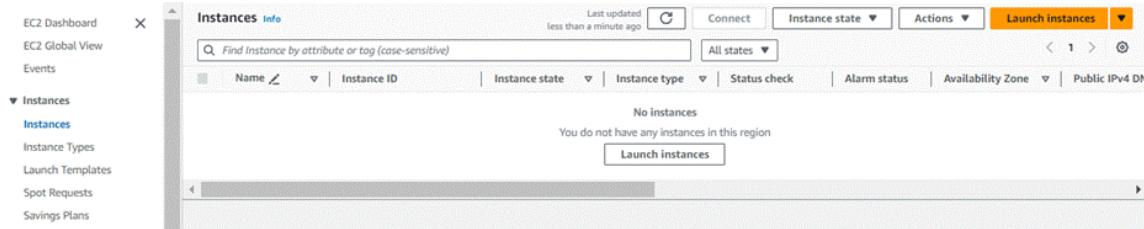
- Note: Load your Flask app and Html files into GitHub repository.



Launch an EC2 instance to host the Flask.



● Click on Launch instance to launch EC2 instance



EC2 > Instances > Launch an instance

ⓘ It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices [Do not show me](#)

### Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

**Name and tags [Info](#)**

Name  Add additional tags

**Application and OS Images (Amazon Machine Image) [Info](#)**

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Recents Quick Start

Amazon Linux	macOS	Ubuntu	Windows	Red Hat	SUSE Linux	Debian
aws						

> Browse more AMIs Including AMIs from

▼ Sum  
Number of 1

Software I  
Amazon Li ami-002f6e5

Virtual ser t2.micro

Firewall (s New secur

Storage (v 1 volume(s)

ⓘ Free accc t2.rr +2 ~

[Cancel](#)

? 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: 64-bit (x86) Boot mode: uefi-preferred AMI ID: 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](#)

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	Username	<a href="#">Verified provider</a>
64-bit (x86)	uefi-preferred	ami-078264b8ba71bc45e	ec2-user	

**Instance type** [Info](#) | [Get advice](#)

Instance type  
t2.micro  
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

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  
 [Create new key pair](#)

**Summary**

Number of instances [Info](#)

Software Image (AMI)  
Amazon Linux 2023 AMI 2023.5.2... [read more](#)  
ami-078264b8ba71bc45e

Virtual server type (instance type)  
t2.micro

Firewall (security group)  
New security group

Storage (volumes)  
1 volume(s) - 8 GiB

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

Cancel [Preview code](#) [Launch instance](#)

## Configure security groups for HTTP, and SSH access.

**Network settings** [Info](#)

VPC - required [Info](#)  
 [Edit](#)

Subnet [Info](#)  
 [Create new subnet](#)

Auto-assign public IP [Info](#)

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

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](#)

**Inbound Security Group Rules**

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

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

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

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

**Add security group rule**

**EC2 > ... > Launch an instance**

**Success**  
Successfully initiated launch of instance i-00186102fbac290

Launch log

**Next Steps**

Q. What would you like to do next with this instance, for example "create alarm" or "create backup?"

Create billing and free tier usage alerts	Connect to your instance	Connect an RDS database	Create EBS snapshot policy	Manage detailed monitoring	Create Load Balancer
To manage costs and avoid surprise bills, set up email notifications for billing and free tier usage thresholds.	Once your instance is running, log into it from your local computer.	Configure the connection between an EC2 instance and a database to allow traffic flow between them.	Create a policy that automates the creation, retention, and deletion of EBS snapshots.	Enable or disable detailed monitoring for the instance. If you enable detailed monitoring, the Amazon EC2 console displays monitoring graphs with a 1-minute period.	Create a application, network gateway or classic Elastic Load Balancer
<a href="#">Create billing alerts</a>	<a href="#">Connect to instance</a> <a href="#">Learn more</a>	<a href="#">Connect an RDS database</a> <a href="#">Create a new RDS database</a> <a href="#">Learn more</a>	<a href="#">Create EBS snapshot policy</a>	<a href="#">Manage detailed monitoring</a>	<a href="#">Create Load Balancer</a>
Create AWS budget	Manage CloudWatch alarms	Disaster recovery for your instances	Monitor for suspicious runtime activities	Get instance screenshot	Get system log
AWS Budgets allows you to create budgets, forecast spend, and take action on your costs and usage from a single location.	Create or update Amazon CloudWatch alarms for the instance.	Recover the instances you just launched into a different Availability Zone or a different Region using AWS Elastic Disaster Recovery (EDRS).	Amazon GuardDuty enables you to continuously monitor for malicious runtime activity and unauthorized behavior, with near real-time visibility into on-host activities occurring across your Amazon EC2 workloads.	<a href="#">Get instance screenshot</a>	<a href="#">Get system log</a>
<a href="#">Create AWS budget</a>	<a href="#">Manage CloudWatch alarms</a>	<a href="#">Disaster recovery for your instances</a>	<a href="#">Monitor for suspicious runtime activities</a>		

**View all instances**

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.

Instances (1/3) <a href="#">Info</a>												
Find instance by attribute or tag (case-sensitive)				All states		Actions						
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 IP	Elastic IP	IPv6 IPs	Monitoring	Secure
HomeMadePic... <a href="#">View details</a>	i-009476d72a5e6975	<span>Running</span> <a href="#">View</a> <a href="#">Logs</a>	12.micro	<span>Initializing</span>	<a href="#">View alarms</a> <a href="#">+</a> us-east-1c	us-east-1c	ec2-5-95-61-71.compute...	3.95.61.71	<span>You are not auth.</span>	-	disabled	<a href="#">Launch</a>
HomeMadePic... <a href="#">View details</a>	i-02d2f5363fbab2d49	<span>Terminated</span> <a href="#">View</a> <a href="#">Logs</a>	12.micro	-	<a href="#">View alarms</a> <a href="#">+</a> us-east-1c	-	-	-	<span>You are not auth.</span>	-	disabled	-
HomeMadePic... <a href="#">View details</a>	i-0846160d1299b8166	<span>Running</span> <a href="#">View</a> <a href="#">Logs</a>	12.micro	<span>2/2 checks passed</span>	<a href="#">View alarms</a> <a href="#">+</a> us-east-1c	us-east-1c	ec2-54-242-3-213.com..	54.242.3.213	<span>You are not auth.</span>	-	disabled	<a href="#">Launch</a>

Failed to attach Instance profile  
There is an existing association for instance i-009476d72a5e6975

Modify IAM role [Info](#)  
Attach an IAM role to your INSTANCE.

Instance ID: i-009476d72a5e6975 (HomeMadePicles)  
IAM role: Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

[EC2\\_DynamoDB\\_Role](#) [Create new IAM role](#)

[Cancel](#) [Update IAM role](#)

Successfully replaced EC2\_DynamoDB\_Role with EC2\_DynamoDB\_Role on instance i-009476d72a5e6975

Instances (1/3) <a href="#">Info</a>												
Find instance by attribute or tag (case-sensitive)				All states		Actions						
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 IP	Elastic IP	IPv6 IPs	Monitoring	Secure
HomeMadePic... <a href="#">View details</a>	i-009476d72a5e6975	<span>Running</span> <a href="#">View</a> <a href="#">Logs</a>	12.micro	<span>2/2 checks passed</span>	<a href="#">View alarms</a> <a href="#">+</a> us-east-1c	us-east-1c	ec2-5-95-61-71.compute...	3.95.61.71	<span>You are not auth.</span>	-	disabled	<a href="#">Launch</a>
HomeMadePic... <a href="#">View details</a>	i-02d2f5363fbab2d49	<span>Terminated</span> <a href="#">View</a> <a href="#">Logs</a>	12.micro	-	<a href="#">View alarms</a> <a href="#">+</a> us-east-1c	-	-	-	<span>You are not auth.</span>	-	disabled	-
HomeMadePic... <a href="#">View details</a>	i-0846160d1299b8166	<span>Running</span> <a href="#">View</a> <a href="#">Logs</a>	12.micro	<span>2/2 checks passed</span>	<a href="#">View alarms</a> <a href="#">+</a> us-east-1c	us-east-1c	ec2-54-242-3-213.com..	54.242.3.213	<span>You are not auth.</span>	-	disabled	-

## 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

```

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

[ec2-user@ip-172-31-17-101 ~]$ sudo yum update -y
sudo yum install python3 git
sudo yum install python3
sudo yum install python3-glib
git3 install Flask
git3 install boto3
Amazon Linux 2023 Kernel Livepatch repository
Dependencies resolved.
Nothing to do.
Complete!
Last download expiration check: 0@0000 days ago on Thu Jul 3 07:06:08 2025.
Package python3-3.9.23-1.amazon2023.0.1.x86_64 is Already installed.
Dependencies received.

Transaction Summary
Install 8 Packages
Total download size: 7.5 M
Downloaded size: 37 M
Is this ok? [y/n]: y
Downloading Packages:
(1/8) : git-2.47.1-1.amzn2023.0.3.x86_64.rpm 1.1 MB/s | 52 MB 00:00
(2/8) : git3-0.12.0-1.amzn2023.0.2.search.rpm 1.1 MB/s | 41 MB 00:00
(3/8) : git3-0.12.0-1.amzn2023.0.3.search.rpm 1.1 MB/s | 2.6 MB 00:00
i-009d76d272e3e6975 (HomeMadePickles)
Public IP: 3.95.61.71 Private IP: 172.31.17.101

168 kB/s | 17 kB 00:00

```

## Clone Your Flask Project from GitHub

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

- Run: '<https://github.com/yasminshaik1329/pickle-paradsie.git>'
- Note: change your-github-username and your-repository-name with your credentials here:<https://github.com/yasminshaik1329/pickle-paradsie.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

```

aws Search [Alt+S] United States (N. Virginia) rsoaccount-new@67fc6a7985d0fc36953d566 @ nosandboxnew14

Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
Collecting markupsafe>=2.1.1
  Downloading MarkupSafe-3.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting zipp>=3.20
  Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: zipp, markupsafe, werkzeug, jinja2, itsdangerous, importlib-metadata, click, blinker, Flask
  Using cached zipp-3.23.0-py3-none-any.whl (10 kB)
  Using cached MarkupSafe-3.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
  Using cached werkzeug-3.1.3-py3-none-any.whl (224 kB)
  Using cached jinja2-3.1.1-py3-none-any.whl (139 kB)
  Using cached click-8.1.0-py3-none-any.whl (139 kB)
  Using cached importlib-metadata-4.7.0-py3-none-any.whl (139 kB)
  Using cached flask-2.2.0-py3-none-any.whl (139 kB)
  Using cached blinker-1.4.0-py3-none-any.whl (139 kB)
  Using cached itsdangerous-2.0.0-py3-none-any.whl (139 kB)
  Using cached werkzeug-3.1.3-py3-none-any.whl (224 kB)
  Using cached jinja2-3.1.1-py3-none-any.whl (139 kB)
  Using cached importlib-metadata-4.7.0-py3-none-any.whl (139 kB)
  Using cached flask-2.2.0-py3-none-any.whl (139 kB)
  Using cached blinker-1.4.0-py3-none-any.whl (139 kB)
  Using cached itsdangerous-2.0.0-py3-none-any.whl (139 kB)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Requirement already satisfied: urllib3<1.27.0,>=1.25.4 in /usr/lib/python3.9/site-packages (from boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=>2.1 in /usr/lib/python3.9/site-packages (from 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->boto3) (1.15.0)
Installing collected packages: python-dateutil, boto3, s3transfer
Successfully installed boto3-1.19.2 python-dateutil-2.2.0 s3transfer-1.39.2
[ec2-user@ip-172-31-17-101 ~]$ git clone https://github.com/ymminshaike1229/pickle-paradise.git
Cloning into 'pickle-paradise'...
remote: Enumerating objects: 73, done.
remote: Counting objects: 100%, done.
remote: Compressing objects: 100%, done.
remote: Total 73 (delta 13), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (73/73), 9.75 MiB / 21.24 MiB/s, done.
Resolving deltas: 100% (13/13), done.
[ec2-user@ip-172-31-17-101 ~]$
```

i-009d76d272e3e6975 (HomeMadePickles)  
PublicIP: 3.95.61.71 PrivateIP: 172.31.17.101

Verify the Flask app is running:

<http://your-ec2-public-ip>

- Run the Flask app on the EC2 instance

```

aws Search [Alt+S] United States (N. Virginia) rsoaccount-new@67fc6a7985d0fc36953d566 @ nosandboxnew14

-bash: kill: (1234) - No such process
[ec2-user@ip-172-31-17-101 pickle-paradise]$ kill -9 12345
-bash: kill: (12345) - No such process
[ec2-user@ip-172-31-17-101 pickle-paradise]$ sudo lsof -i :5000
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
python3 32830 ec2-user 7u IPv4 58689 0t0 TCP *:complex-main (LISTEN)
python3 32831 ec2-user 7u IPv4 58689 0t0 TCP *:complex-main (LISTEN)
python3 32831 ec2-user 8u IPv4 58689 0t0 TCP *:complex-main (LISTEN)
python3 32831 ec2-user 9u IPv4 64403 0t0 TCP ip-172-31-17-101.ec2.internal:complex-main->broadband.actcorp.in:16437 (ESTABLISHED)
python3 32831 ec2-user 11u IPv4 64404 0t0 TCP ip-172-31-17-101.ec2.internal:complex-main->broadband.actcorp.in:16440 (ESTABLISHED)
[ec2-user@ip-172-31-17-101 pickle-paradise]$ kill -9 32831
[ec2-user@ip-172-31-17-101 pickle-paradise]$ kill -9 32831
-bash: kill: (32831) - No such process
[ec2-user@ip-172-31-17-101 pickle-paradise]$ sudo yum install lsof -y
Last metadata expiration check: 1:37:48 ago on Thu Jul 3 07:06:08 2025.
Package lsof-4.94.0-1.amzn2023.0.3.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-17-101 pickle-paradise]$ sudo lsof -i :5000
[ec2-user@ip-172-31-17-101 pickle-paradise]$ python3 app.py
  * Serving Flask app 'app'
  * Debug mode: on
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://172.31.17.101:5000
Press CTRL+C to quit
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: 139-491-322
```

i-009d76d272e3e6975 (HomeMadePickles)  
PublicIP: 3.95.61.71 PrivateIP: 172.31.17.101

## Milestone 7 : Testing and Deployment

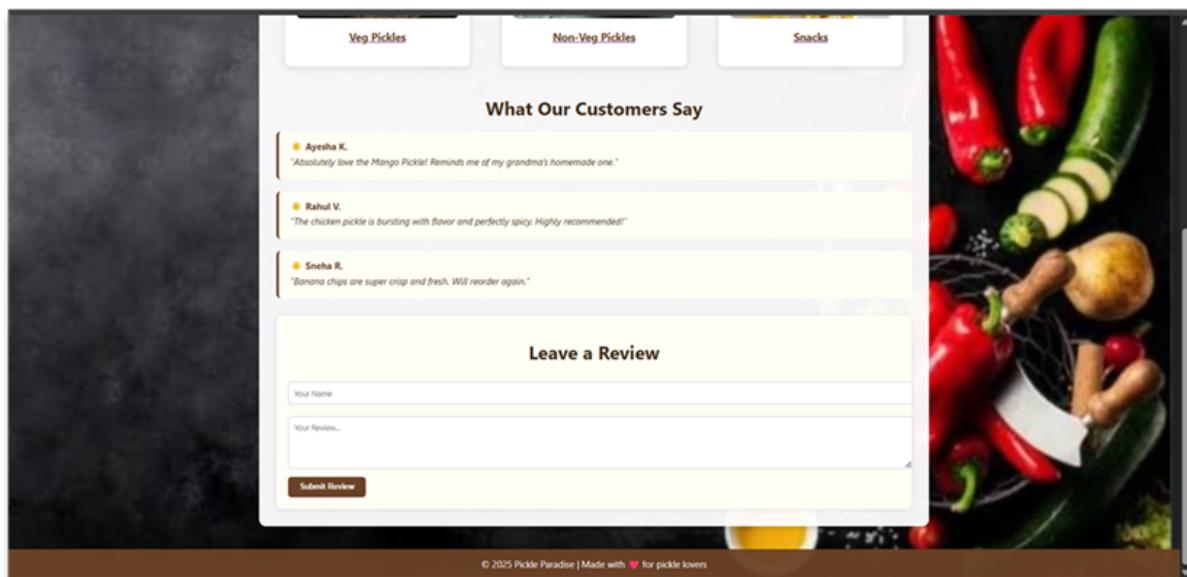
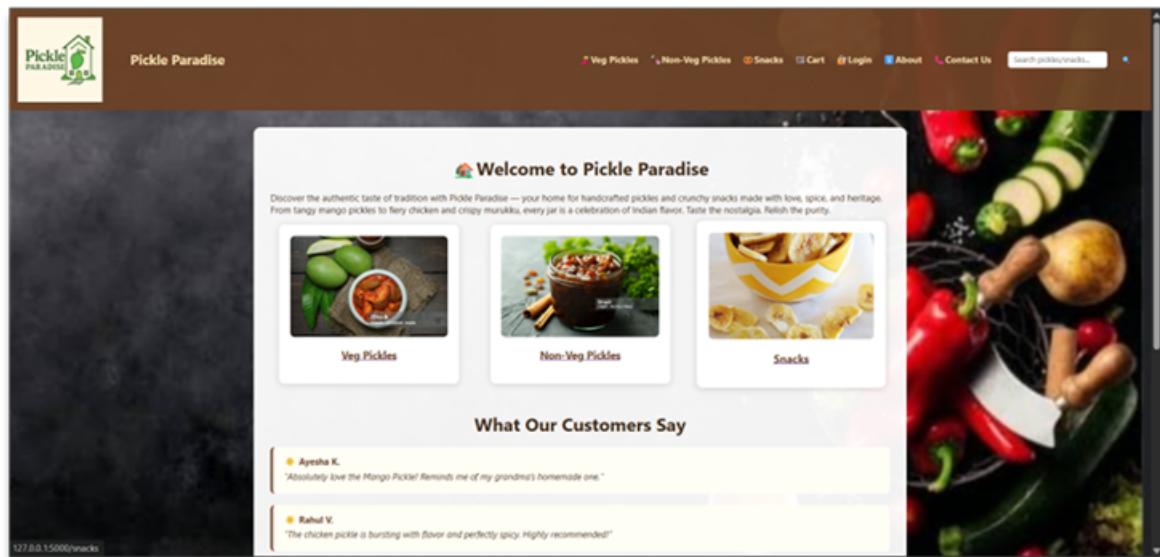
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and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

smoothly in the production environment.

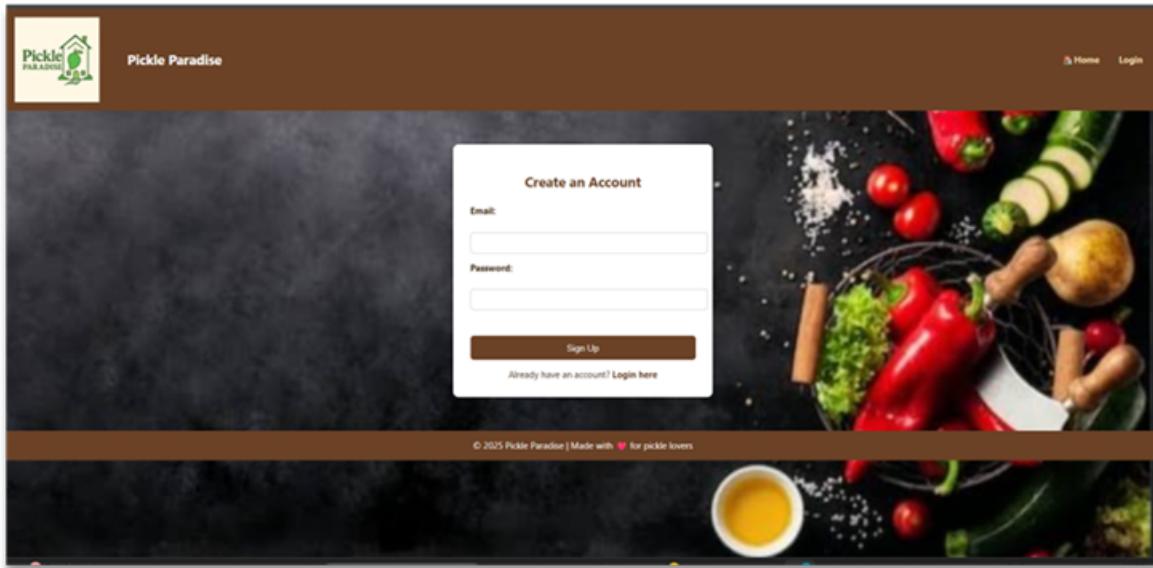
## Functional testing to verify the Project

Welcome page:

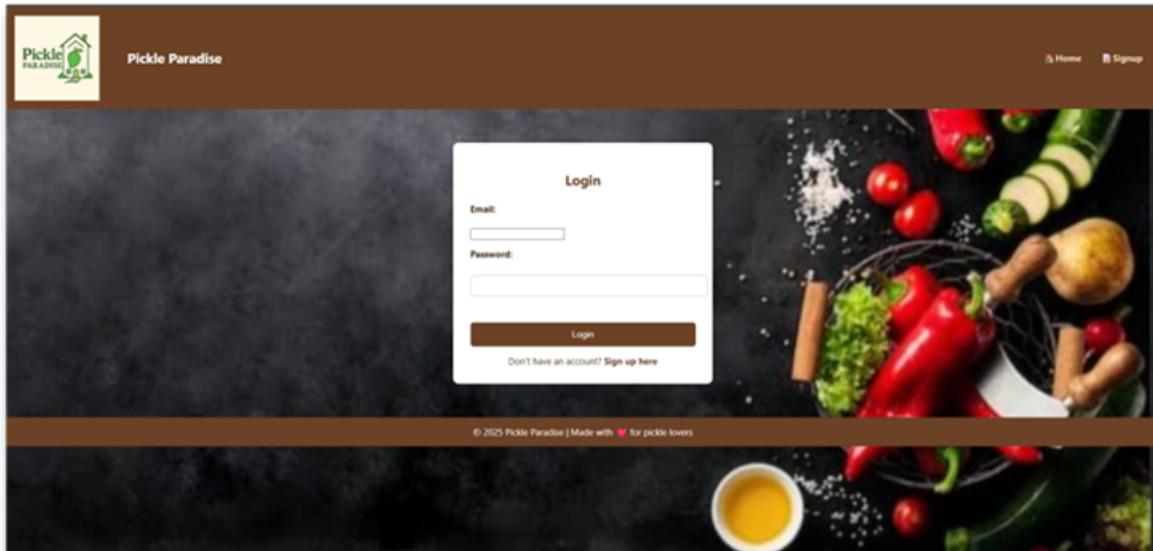
**Home page:**



## Signup page:



## Login page:



## Veg pickles:

The screenshot shows the 'Veg Pickles' section of the Pickle Paradise website. At the top, there's a navigation bar with links for Home, Non Veg, Snacks, Cart, Login, About, and Contact. A search bar is also present. The main content area features a large image of various vegetables like red and green bell peppers, tomatoes, and a bottle of pickle. Below this, a title 'Veg Pickles' is followed by a descriptive paragraph: 'Taste the essence of nature with our Veg Pickles — from spicy mango and zesty lemon to fiery red chili and tangy gongura. Each jar is hand-packed with love, tradition, and a burst of South Indian flavor!'. Three product cards are displayed: 'Mango Pickle' (spicy mango chunks), 'Lemon Pickle' (zesty lemon pieces), and 'Tomato Pickle' (ripe tomatoes blended with fiery masala). Each card includes a small image of the pickle, a star rating, a brief description, price (₹120, ₹150, ₹195 respectively), weight (500g), and an 'Add to Cart' button.

This screenshot shows the 'Veg Pickles' section with five products listed. The layout is identical to the first screenshot, featuring a large vegetable background image on the right. The products listed are: 'Amla Pickle' (vitamin-rich gooseberries), 'Mixed Veg Pickle' (a medley of crunchy vegetables), 'Red Chilli Pickle' (whole red chilies stuffed and soaked in aromatic oil), 'Garlic Pickle' (whole garlic pods steeped in tangy masala), and 'Gongura Pickle' (tangy gongura leaves blended with spice-rich oil). Each product card includes an image, a star rating, a brief description, price (₹185, ₹110, ₹95, ₹95, ₹125 respectively), weight (500g), and an 'Add to Cart' button.

## Non-Veg pickles:

The screenshot shows the 'Non-Veg Pickles' section of the Pickle Paradise website. At the top, there's a navigation bar with the logo 'Pickle Paradise', a search bar, and links for Home, Veg, Snacks, Cart, Login, About, and Contact. The main content area has a dark background featuring various vegetables like red bell peppers, zucchini, and tomatoes. A white card displays three products:

- Prawn Pickle**: Spicy, tangy prawns soaked in aromatic coastal masala. Price: ₹450, Weight: 500g. Add to Cart button.
- Chicken Pickle**: Juicy chicken with bold Andhra spices. Price: ₹400, Weight: 500g. Add to Cart button.
- Fish Pickle**: Boneless fish in a sharp, spicy, and tangy masala mix. Price: ₹550, Weight: 500g. Add to Cart button.

This screenshot shows the same 'Non-Veg Pickles' section from a different perspective or a later page. The layout is identical, with the dark vegetable-themed background and the white card displaying the same three products:

- Fish Pickle**: Boneless fish in a sharp, spicy, and tangy masala mix. Price: ₹550, Weight: 500g. Add to Cart button.
- Mutton Pickle**: Juicy mutton pieces blended with bold, fiery spices. Price: ₹800, Weight: 500g. Add to Cart button.
- Crab Pickle**: Delicate crab meat pickled in rich, zesty Andhra flavors. Price: ₹500, Weight: 500g. Add to Cart button.
- Egg Pickle**: Boiled eggs infused with flavorful, spiced oil and chilies. Price: ₹250, Weight: 500g. Add to Cart button.

## Snacks:

The screenshot shows the 'Snacks' page of the Pickle Paradise website. At the top, there's a navigation bar with the logo 'Pickle PARADISE', a search bar, and links for Home, Veg, Snacks, Non Veg, Login, About, and Contact. Below the navigation is a large, vibrant image of various vegetables like red bell peppers, zucchini, and onions. The main content area is titled 'Snacks' and features three product cards:

- Mixture**: A crunchy blend of sev, nuts, and spices — the ultimate anytime snack.  
Price: ₹10  
Weight: 250g  
[Add to Cart](#)
- Murukku**: Crispy spiral sticks made from rice flour and sesame — delightfully addictive.  
Price: ₹10  
Weight: 250g  
[Add to Cart](#)
- Banana Chips**: Thinly sliced raw bananas fried to golden crispness with a hint of salt.  
Price: ₹15  
[Add to Cart](#)

This screenshot shows another view of the 'Snacks' page on the Pickle Paradise website. The layout is similar to the first one, with the navigation bar at the top and a background image of vegetables on the right. The main content area is titled 'Snacks' and features three product cards:

- Ribbon Pakoda**: Flat, ribbon-shaped snack with a spicy, crunchy bite in every strip.  
Price: ₹15  
Weight: 250g  
[Add to Cart](#)
- Kara Sev**: Spiced chickpea flour noodles deep-fried for a fiery, crunchy treat.  
Price: ₹15  
Weight: 250g  
[Add to Cart](#)
- Thattai**: Flat, crunchy discs made with rice flour, spices, and lentils — perfectly savory.  
Price: ₹10  
Weight: 200g  
[Add to Cart](#)

## Cart Page:

The screenshot shows the shopping cart page for 'Pickle Paradise'. At the top, there's a navigation bar with the logo 'Pickle Paradise' and links for Home, Veg, Non-Veg, and Snacks. The main content area features a dark background with a vibrant image of various vegetables like red bell peppers, green beans, and onions. A white modal box titled 'Your Shopping Cart' displays a table of items:

Item	Price (₹)	Weight	Quantity	Subtotal (₹)	Action
Ribbon Pakoda	75	250g	<input type="button" value="−"/> 1 <input type="button" value="+"/>	₹75	<input type="button" value="Remove"/>
Brinjal Pickle	450	300g	<input type="button" value="−"/> 1 <input type="button" value="+"/>	₹450	<input type="button" value="Remove"/>
Mango Pickle	120	300g	<input type="button" value="−"/> 2 <input type="button" value="+"/>	₹1200	<input type="button" value="Remove"/>

Total: ₹1765

[Proceed to Checkout](#)

At the bottom, a footer note reads: © 2025 Pickle Paradise | Made with ❤️ for pickle lovers.

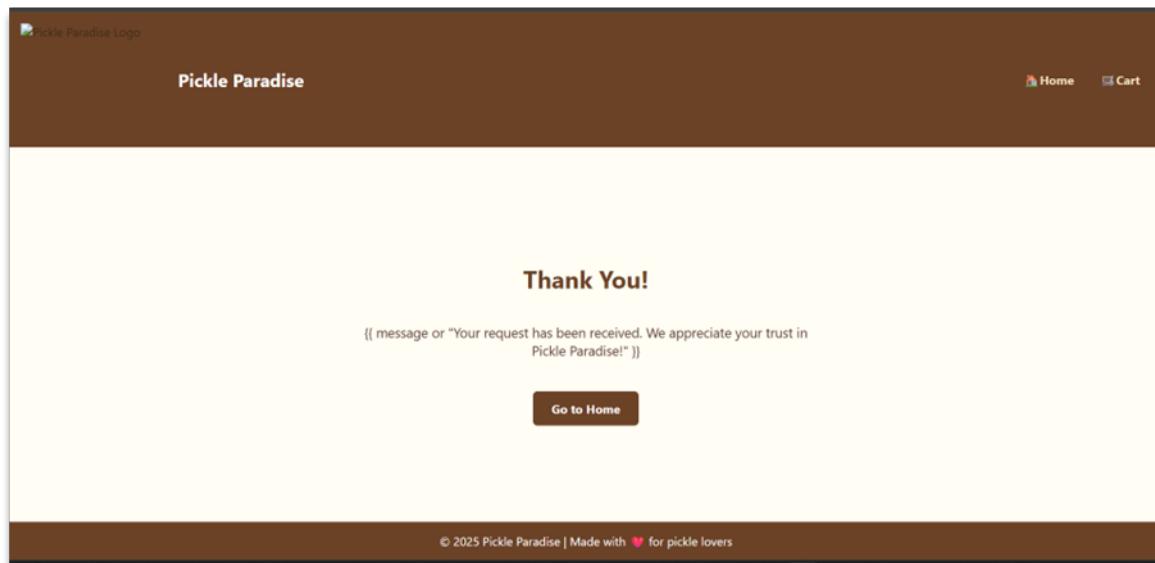
## Checkout Page:

The screenshot shows the checkout page for 'Pickle Paradise'. The layout is similar to the cart page, with the 'Pickle Paradise' logo at the top and a navigation bar with Home, Veg Pickles, Non-Veg Pickles, Snacks, and a 'Cart' link. The background image of vegetables is present. A white form box titled 'Checkout' contains fields for:

- Full Name
- Email
- Address
- City
- Pincode
- Phone

Below these fields is a 'Payment Method' section with three options: UPI, Credit Card, and Cash on Delivery. The 'Cash on Delivery' option is selected, indicated by a checked radio button. At the bottom right of the form is a large orange button labeled 'Place Order'.

## Success Page:



## Conclusion

Pickle Paradise demonstrates how cloud-native architecture can empower small businesses with reliable, scalable, and fast digital platforms. Leveraging AWS and Python's Flask framework, the platform ensures excellent user experience, automated notifications, and simple backend management.

The integration of cloud-ready architecture (e.g., AWS for future scalability) and robust session management allows the platform to handle high traffic efficiently while maintaining real-time updates for orders and inventory. Features like weight-based pricing, category-specific searches, and instant checkout streamline the shopping process, empowering customers to explore a diverse range of traditional and innovative recipes with ease.

This project addresses the growing demand for homemade, preservative-free food products by bridging the gap between small-scale producers and discerning customers. The platform's intuitive design and secure payment workflows enhance trust and convenience, while backend tools enable effortless inventory tracking and order fulfillment for administrators.

In essence, this project redefines the way homemade delicacies are shared and enjoyed, offering a flavorful bridge between tradition and technology.

