

# HomeMade Pickles & Snacks: Taste the Best

Hardware Required:

Processor: Intel i5 or equivalent (minimum). RAM: 4 GB (8 GB recommended for Full Stack MERN). per system). Additional: Audio-visual setup for interactive sessions (microphone, speakers, etc.).

Software Required:

Processor: Intel i5 or equivalent (minimum). RAM: 4 GB (8 GB recommended for Full Stack MERN). Storage: 128 GB SSD or 128 GB HDD. Internet Connectivity: High-speed internet (minimum 10 Mbps per system). Additional: Audio-visual setup for interactive sessions (microphone, speakers, etc.).

## **System Required:**

Projector and Audio System for presentations in all labs/classrooms Classrooms/Labs are equipped with systems or provisions for students to join sessions with their own laptops.

## **Description:**

Home Made Pickles & Snacks — Taste the Best is a cloud-based culinary platform revolutionizing access to authentic, handcrafted pickles and snacks. Addressing the growing demand for preservative-free, traditional recipes, this initiative combines artisanal craftsmanship with cutting-edge technology to deliver farm-fresh flavors directly to consumers. Built on Flask for backend efficiency and hosted on AWS EC2 for scalable performance, the platform offers seamless browsing, ordering, and subscription management. DynamoDB ensures real-time inventory tracking and personalized user experiences, while fostering sustainability through partnerships with local farmers and eco-friendly packaging. From tangy regional pickles to wholesome snacks, every product celebrates heritage recipes, nutritional integrity, and convenience—proving that tradition and innovation can coexist deliciously. "Preserving Traditions, One Jar at a Time."

## **Scenarios:**

### **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**

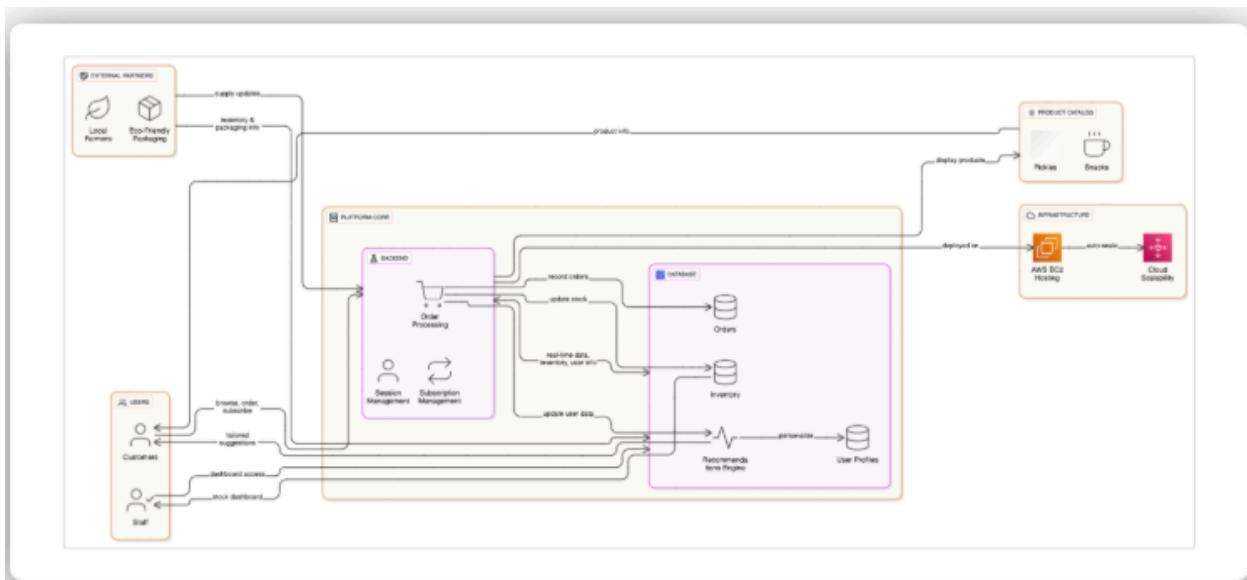
When a customer places an order for a product, the system instantly updates stock levels and records transaction details. For example, a user purchases an item, triggering automatic inventory deduction and order confirmation. Staff members receive updated dashboards to monitor stock availability and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

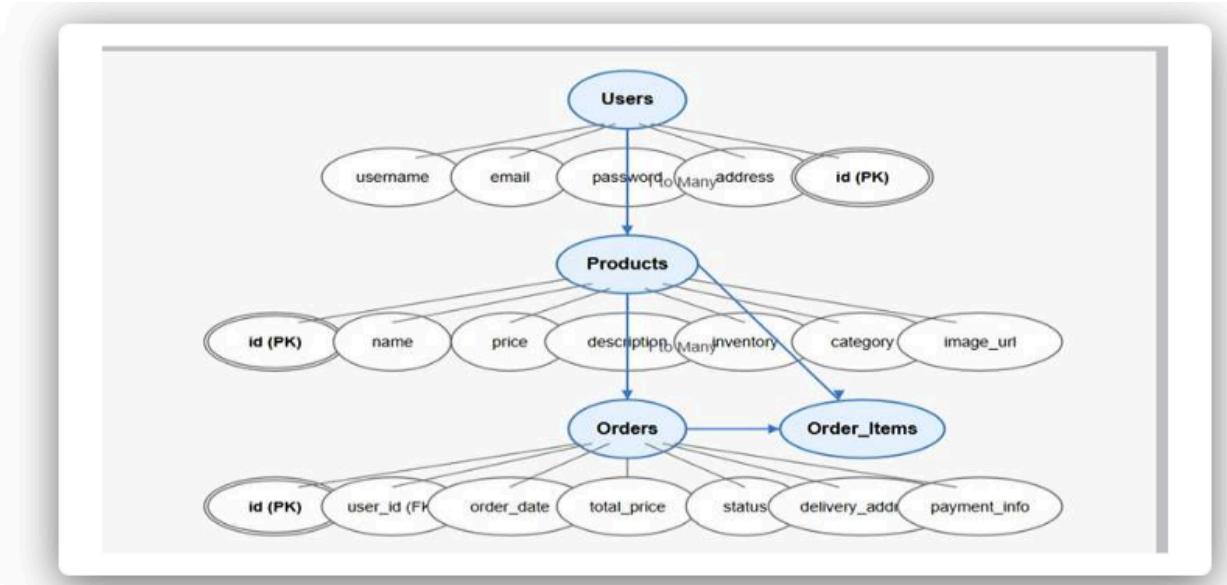
## Architecture

This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for hosting the backend, with a lightweight framework like Flask handling core logic. Application data is stored in Amazon DynamoDB, ensuring fast, reliable access, while user access is managed through AWS IAM for secure authentication and control. Real-time alerts and system notifications are enabled via Amazon SNS, enhancing communication and user engagement.



## Entity Relationship (ER)Diagram

An ER (Entity-Relationship) diagram visually represents the logical structure of a database by defining entities, their attributes, and the relationships between them. It helps organize data efficiently by illustrating how different components of the system interact and relate. This structured approach supports effective database normalization, data integrity, and simplified query design.



## Pre-requisites

- AWS Account Setup:  
<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
- AWS IAM (Identity and Access Management):  
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
- AWS EC2 (Elastic Compute Cloud):  
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
- AWS DynamoDB:  
<https://docs.aws.amazon.com/amazondynamodb/Introduction.html>
- Git Documentation:  
<https://git-scm.com/doc>
- VS Code Installation: (download the VS Code using the below link or you can get that in Microsoft store)  
<https://code.visualstudio.com/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 : Web Application Development and Setup**

Backend Development and Application Setup focuses on establishing the core structure of the application. This includes configuring the backend framework, setting up routing, and integrating database connectivity. It lays the groundwork for handling user interactions, data management, and secure access.

#### **Important Instructions:**

- Start by creating the necessary HTML pages and Flask routes (app.py) to build the core functionality of your application.
- During the initial development phase, store and retrieve data using Python dictionaries or lists locally. This will allow you to design, test, and validate your application logic without external database dependencies.
- Ensure your app runs smoothly with local data structures before integrating any cloud services.

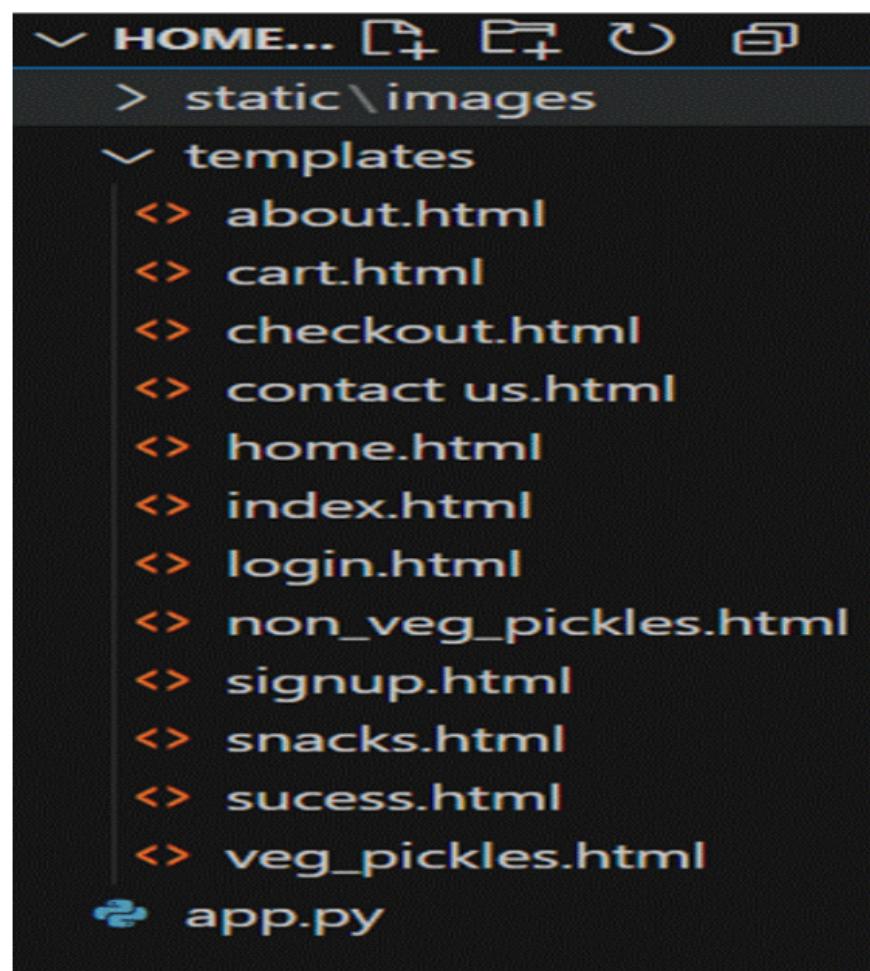
#### **Post Troven Access Activation:**

- Once Troven Labs access is provided (valid for 3 hours), you must immediately proceed with Milestone 1 of your Guided Project instructions.
- At this point, modify your app.py and replace local dictionary/list operations with AWS services (such as DynamoDB, RDS, or others as per project requirements).
- Using the temporary credentials provided by Troven Labs, securely connect your application to AWS resources.

- Since the AWS configuration is lightweight and already instructed in the milestones, you should be able to complete the cloud integration efficiently within the allotted time.

## LOCAL DEPLOYMENT

- File Explorer Structure



```
1 from flask import Flask, render_template, request, redirect, url_for, session, flas
2 import boto3
3 import uuid
4 import smtplib
5 from email.mime.text import MIMEText
6 from email.mime.multipart import MIMEMultipart
7 import os
```

```

7   import os
8
9   app = Flask(__name__)
10  app.secret_key = os.urandom(24)
11

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

# Email Configuration
EMAIL_ADDRESS = 'avanthiakki@gmail.com'
EMAIL_PASSWORD = 'inpq fkgo tmku bwhf' # Replace with your actual Gmail App Passwords

21
22  app = Flask(__name__)
23  app.secret_key = 'your_secret_key_here' # Replace with a strong secret key
24

25  @app.route('/')
26  def index():
27      return render_template('index.html')
28
29  @app.route('/veg_pickles')
30  def veg_pickles():

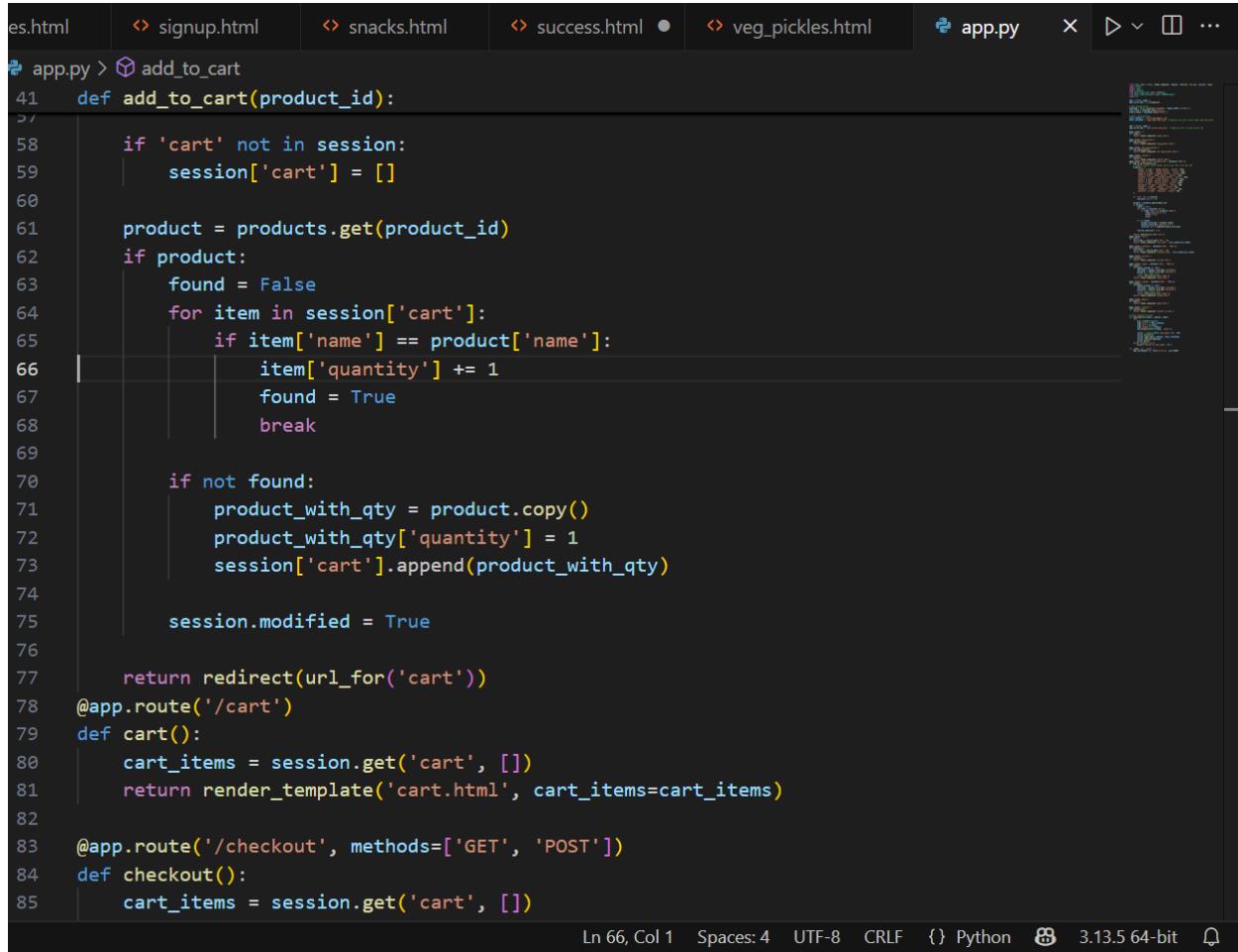

```

```

app.py > ⌂ add_to_cart
28
29  @app.route('/veg_pickles')
30  def veg_pickles():
31      return render_template('veg_pickles.html')
32
33  @app.route('/non_veg_pickles')
34  def non_veg_pickles():
35      return render_template('non_veg_pickles.html')
36
37  @app.route('/snacks')
38  def snacks():
39      return render_template('snacks.html')
40  @app.route('/add_to_cart/<product_id>', methods=['POST'])
41  def add_to_cart(product_id):
42      # Example product data (you'd usually get this from your DB)
43      products = {
44          'mango': {'name': 'Mango Pickle', 'price': 130},
45          'tomato': {'name': 'Tomato Pickle', 'price': 120},
46          'gongura': {'name': 'Gongura Pickle', 'price': 150},
47          'lemon': {'name': 'Lemon Pickle', 'price': 140},
48          'chicken': {'name': 'Chicken Pickle', 'price': 250},
49          'prawn': {'name': 'Prawn Pickle', 'price': 300},
50          'mutton': {'name': 'Mutton Pickle', 'price': 300},
51          'fish': {'name': 'Fish Pickle', 'price': 260},
52          'murukulu': {'name': 'Murukulu', 'price': 50},
53          'boondi': {'name': 'Boondi', 'price': 80},
54          'chekkalu': {'name': 'Chekkalu', 'price': 90},
55          'mixture': {'name': 'Mixture', 'price': 90}
56      }


```

# Milestone 2 : AWS Account Setup



The screenshot shows a code editor interface with a dark theme. The main area displays Python code for a web application, specifically handling cart operations. The code includes functions for adding items to the cart, viewing the cart, and performing a checkout process. The code uses session management and Jinja templating. The status bar at the bottom provides information about the current file (app.py), line number (Ln 66), column (Col 1), and other settings.

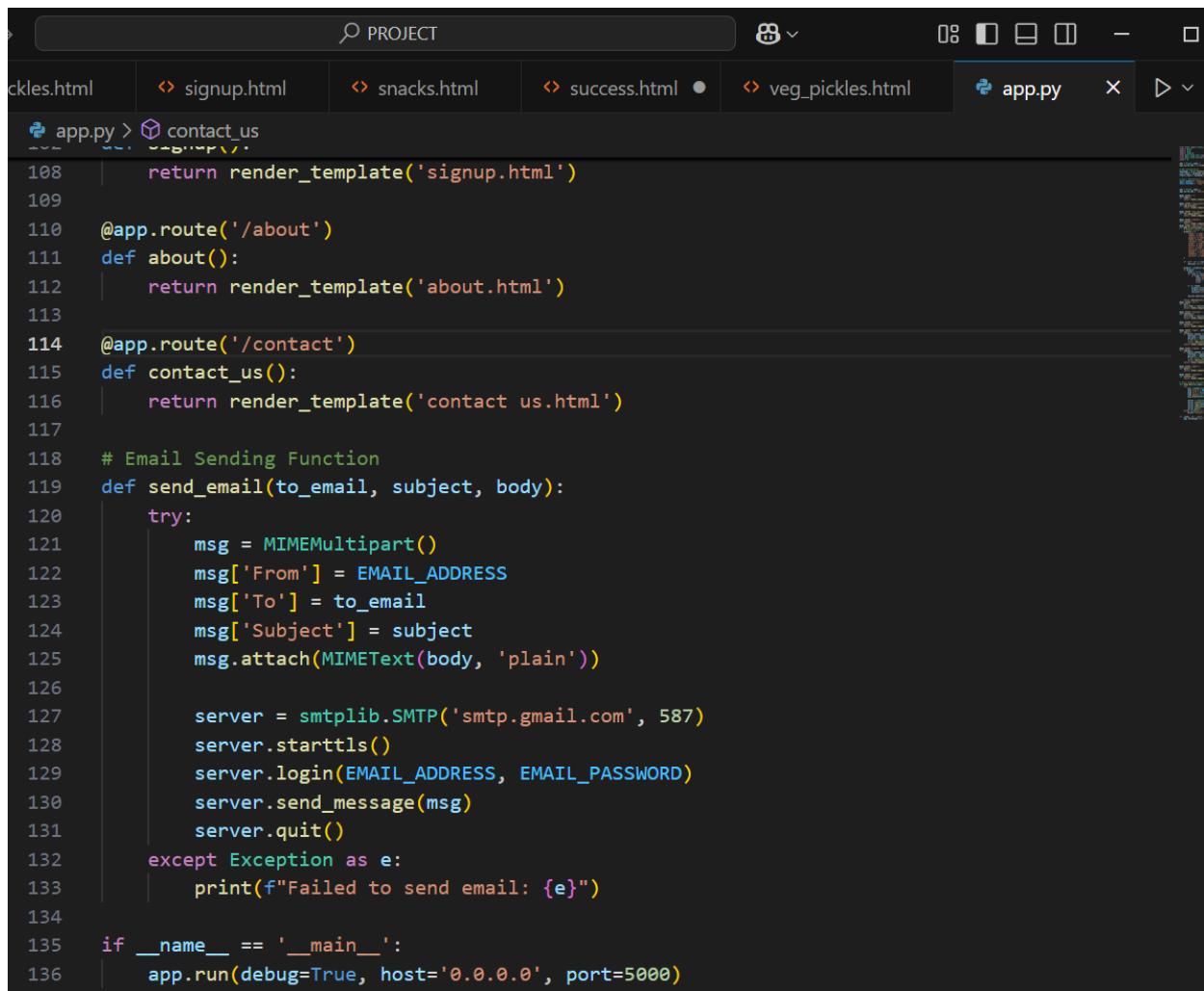
```
es.html    < signup.html    < snacks.html    < success.html    < veg_pickles.html    app.py    X    ⌂    ...  
app.py > add_to_cart  
41     def add_to_cart(product_id):  
58         if 'cart' not in session:  
59             session['cart'] = []  
60  
61         product = products.get(product_id)  
62         if product:  
63             found = False  
64             for item in session['cart']:  
65                 if item['name'] == product['name']:  
66                     item['quantity'] += 1  
67                     found = True  
68                     break  
69  
70             if not found:  
71                 product_with_qty = product.copy()  
72                 product_with_qty['quantity'] = 1  
73                 session['cart'].append(product_with_qty)  
74  
75         session.modified = True  
76  
77     return redirect(url_for('cart'))  
78 @app.route('/cart')  
79 def cart():  
80     cart_items = session.get('cart', [])  
81     return render_template('cart.html', cart_items=cart_items)  
82  
83 @app.route('/checkout', methods=['GET', 'POST'])  
84 def checkout():  
85     cart_items = session.get('cart', [])  
Ln 66, Col 1  Spaces: 4  UTF-8  CRLF  {}  Python  3.13.5 64-bit  ⌂
```

A screenshot of a code editor interface, likely PyCharm, displaying a Python Flask application. The code is organized into several functions:

- `cart()`: Returns a rendered template for the cart.
- `checkout()`: Returns a rendered template for the checkout page.
- `success()`: Returns a rendered template for the success page.
- `login()`: Handles both GET and POST requests. If POST, it retrieves username and password, adds login logic, and redirects to the index. Otherwise, it renders the login template.
- `signup()`: Handles both GET and POST requests. If POST, it retrieves username and password, adds signup logic, and redirects to the login page. Otherwise, it renders the signup template.

The code uses session management and render\_template for generating HTML pages. The editor interface includes a top bar with project navigation, a sidebar with file lists, and a right-hand panel for toolbars and settings.

```
app.py > Signup
8 @app.route('/cart')
9 def cart():
0     cart_items = session.get('cart', [])
1     return render_template('cart.html', cart_items=cart_items)
2
3 @app.route('/checkout', methods=['GET', 'POST'])
4 def checkout():
5     cart_items = session.get('cart', [])
6     return render_template('checkout.html', cart_items=cart_items)
7
8 @app.route('/success')
9 def success():
0     return render_template('success.html')
1
2 @app.route('/login', methods=['GET', 'POST'])
3 def login():
4     if request.method == 'POST':
5         username = request.form.get('username')
6         password = request.form.get('password')
7         # Add login logic here
8         return redirect(url_for('index'))
9     return render_template('login.html')
0
1 @app.route('/signup', methods=['GET', 'POST'])
2 def signup():
3     if request.method == 'POST':
4         username = request.form.get('username')
5         password = request.form.get('password')
6         # Add signup logic here
7         return redirect(url_for('login'))
```



The screenshot shows a code editor interface with a dark theme. The top bar has tabs for 'PROJECT' and several files: 'ckles.html', 'signup.html', 'snacks.html', 'success.html', 'veg\_pickles.html', and 'app.py'. The 'app.py' tab is currently selected. The code in 'app.py' is a Python script for a Flask application. It includes routes for '/about', '/contact', and '/'. It also contains a function 'send\_email' that uses the smtplib library to send an email via SMTP. Finally, it has a conditional block to run the app if the script is executed directly.

```
ckles.html    < signup.html    < snacks.html    < success.html    < veg_pickles.html    app.py    X    ▶
```

```
108     return render_template('signup.html')
109
110 @app.route('/about')
111 def about():
112     return render_template('about.html')
113
114 @app.route('/contact')
115 def contact_us():
116     return render_template('contact_us.html')
117
118 # Email Sending Function
119 def send_email(to_email, subject, body):
120     try:
121         msg = MIMEText(body)
122         msg['From'] = EMAIL_ADDRESS
123         msg['To'] = to_email
124         msg['Subject'] = subject
125         msg.attach(MIMEText(body, 'plain'))
126
127         server = smtplib.SMTP('smtp.gmail.com', 587)
128         server.starttls()
129         server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
130         server.send_message(msg)
131         server.quit()
132     except Exception as e:
133         print(f"Failed to send email: {e}")
134
135 if __name__ == '__main__':
136     app.run(debug=True, host='0.0.0.0', port=5000)
```

## Milestone 2 : AWS Account Setup

### Important Notice: Use Troven Labs for AWS Access

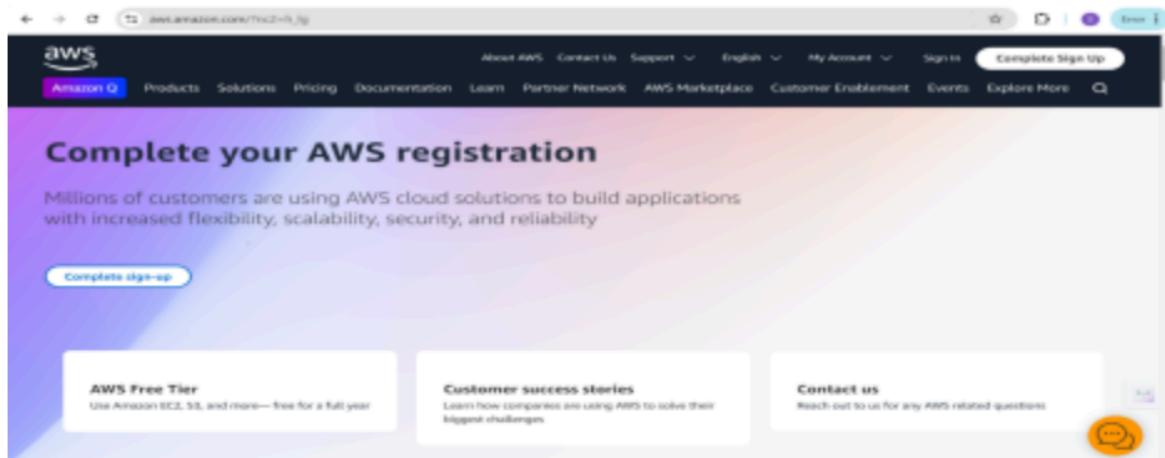
Students are strictly advised not to create their own AWS accounts, as doing so may incur charges. Instead, we have set up a dedicated section called “Labs” on the Troven platform, which provides temporary and cost-free access to AWS services.

Once your website is locally deployed and fully functional, you must proceed with integrating AWS services only through the Troven Labs environment. This ensures secure, controlled access to AWS resources without any risk of personal billing.

All steps involving AWS (such as deploying to EC2, connecting to DynamoDB, or using SNS) must be carried out within the Troven Labs platform, as we've configured temporary credentials for each student.

**This is for your understanding only, please refrain from creating an AWS account. A temporary account will be provided via Troven.**

- Go to the AWS website (<https://aws.amazon.com/>).
- Click on the "Create an AWS Account" button.
- Follow the prompts to enter your email address and choose a password.
- Provide the required account information, including your name, address, and phone number.
- Enter your payment information. (Note: While AWS offers a free tier, a credit card or debit card is required for verification.)
- Complete the identity verification process.
- Choose a support plan (the basic plan is free and sufficient for starting).
- Once verified, you can sign in to your new AWS accounts.

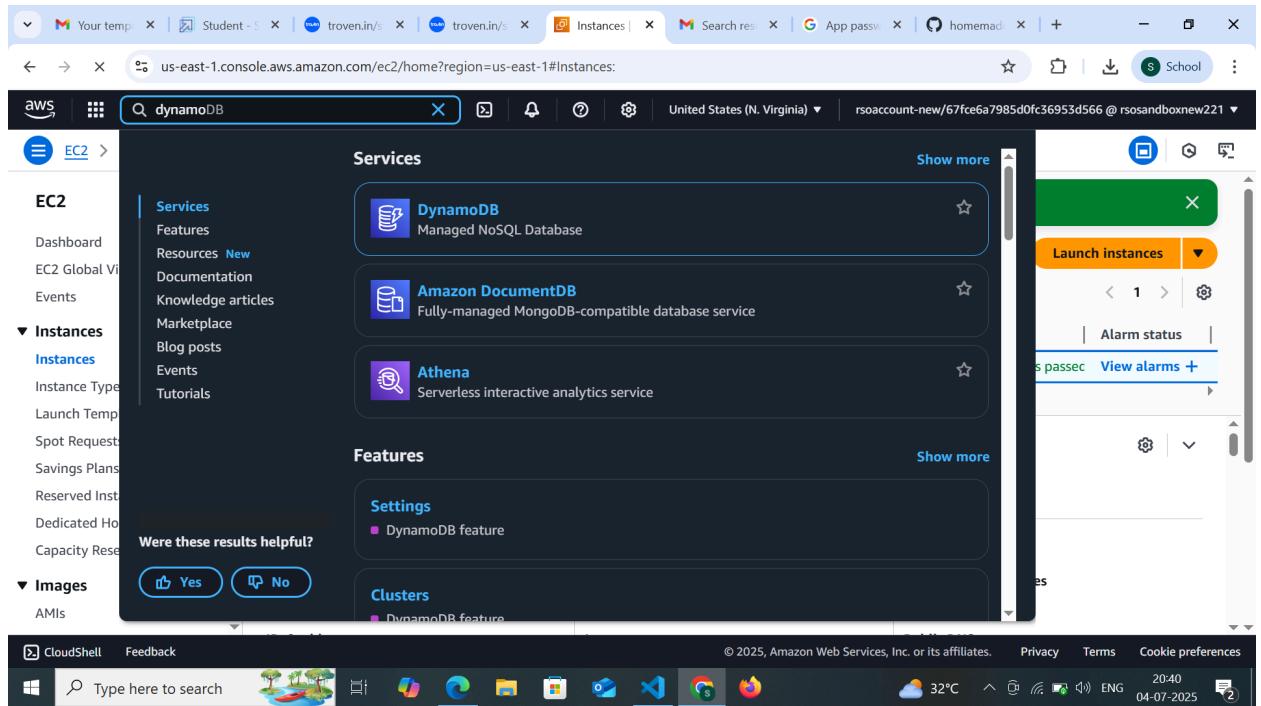


## Milestone 3 : DynamoDB Database Creation and Setup

Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high-performance data storage for seamless backend operations.

### Navigate to the DynamoDB

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



## Create a DynamoDB table for storing data

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The top navigation bar includes the AWS logo, a search bar, and a keyboard shortcut [Alt+S]. The breadcrumb trail indicates the user is at 'DynamoDB > Tables > Create table'. The main section is titled 'Create table' and contains a 'Table details' step. The 'Table name' field is set to 'Users', which is described as a schemaless database requiring only a table name and a primary key. The 'Partition key' is defined as 'Username' of type 'String', and the 'Sort key - optional' field is left empty. The interface uses a clean, modern design with light blue highlights for active fields.

**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.

**Users**

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.

**Username** **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.

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

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](#)

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.

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.

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.

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](#)

Your temp Student - S troven.in/ troven.in/ List tables Search res App passw homemad rsosaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew221

aws Search [Alt+S] United States (N. Virginia) rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew221

DynamoDB > Tables

The Orders table was created successfully.

**Tables (2)** [Info](#)

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite
<input type="checkbox"/>	<a href="#">Orders</a>	<span>Active</span>	order_id (S)	-	0	0	<input type="radio"/> Off	
<input type="checkbox"/>	<a href="#">Users</a>	<span>Active</span>	username (S)	-	0	0	<input type="radio"/> Off	

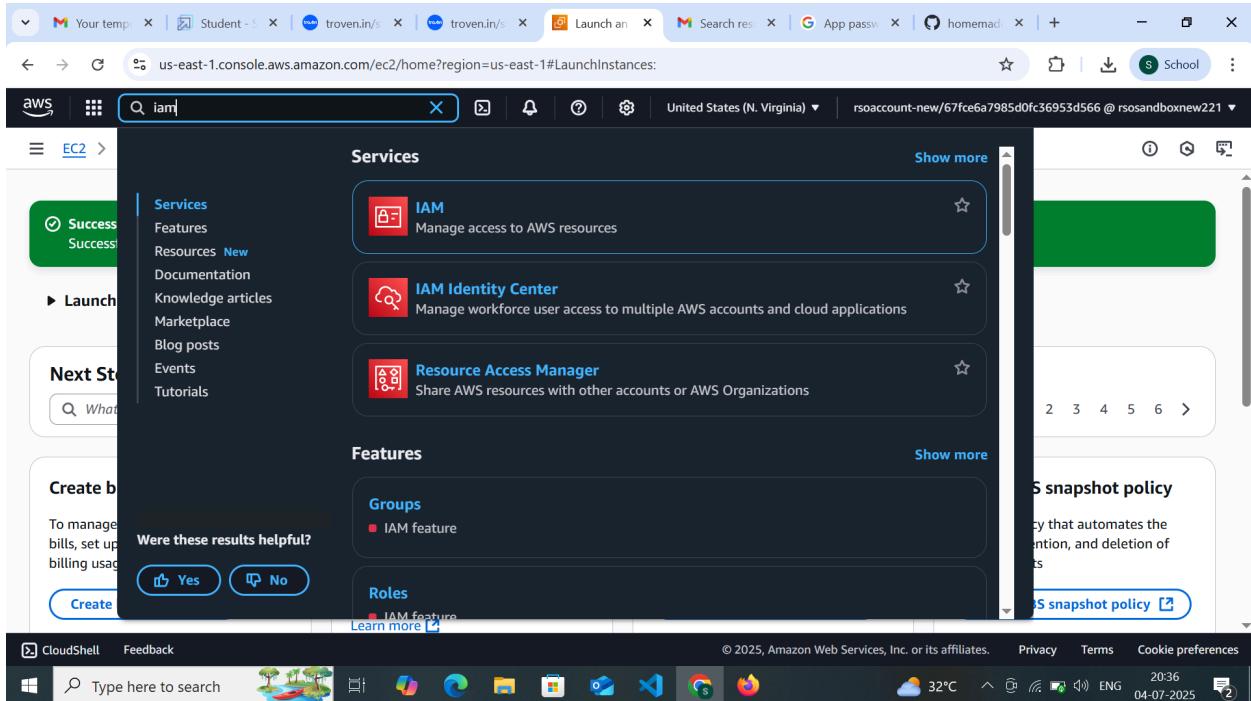
CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 32°C 2043 04-07-2025 ENG

# Milestone 4 : IAM Role Setup

IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows controlled access and ensures security best practices in managing AWS resources.

## Create IAM Role.

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



The image consists of three vertically stacked screenshots from the AWS IAM 'Create role' wizard:

- Select trusted entity**: Shows the 'Trusted entity type' section with 'AWS Lambda' selected. It also shows the 'Use role' section where 'AmazonDynamoDBFullAccess' is chosen.
- Add permissions**: Shows the 'Permissions policies' section with 'AmazonDynamoDBFullAccess' selected. A search bar at the top shows 'AmazonDynamoDB'. The 'Set permissions boundary - optional' field is present at the bottom.
- Review policy**: Shows the final review screen with the role name 'AmazonDynamoDBRole' and the attached policy 'AmazonDynamoDBFullAccess'.

## Attach Policies

Attach the following policies to the role:

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

The image consists of two vertically stacked screenshots from the AWS IAM service.

**Screenshot 1: Name, review, and create**

This screenshot shows the final step of creating a new IAM role. The role name is "sns\_Dynamodb\_role". The description is "Allows EC2 instances to call AWS services on your behalf". The trust policy is set to "AWS Lambda" with the ARN "arn:aws:lambda:us-east-1:123456789012:role/service-role/AmazonDynamoDBFullAccess". The permissions policy summary shows three managed policies attached:

- AmazonDynamoDBFullAccess (AWS-managed)
- AmazonSNSFullAccess (AWS-managed)
- AmazonSQSFullAccess (AWS-managed)

**Screenshot 2: sns\_Dynamodb\_role**

This screenshot shows the summary page for the "sns\_Dynamodb\_role". It includes the ARN (arn:aws:iam::557690616836:role/sns\_Dynamodb\_role), Instance profile ARN (arn:aws:iam::557690616836:instance-profile/sns\_Dynamodb\_role), Creation date (October 13, 2024, 23:06 (UTC+09:00)), Last activity (5 days ago), and Maximum session duration (1 hour). The Permissions tab is selected, showing the attached policies:

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

## 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.

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

Name	Last commit message	Last commit date
..		
static	Initial project upload	last week
templates	Initial project upload	last week
app.py	Update app.py	5 days ago

The screenshot shows a GitHub repository page for 'homemade-pickles'. The repository is public and has 1 branch and 0 tags. The commit history shows an initial project upload and an update to app.py. The 'Code' tab is selected. A context menu is open over the 'Clone' button, showing options for Local, HTTPS, SSH, and GitHub CLI, along with a URL and download links. The right sidebar includes sections for About (no description), Activity (0 stars, 0 watching, 0 forks), Releases (no releases published), and Packages (no packages published). The bottom of the screen shows a Windows taskbar with various pinned icons.

## Launch an EC2 instance to host the Flask

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

The screenshot shows two consecutive pages from the AWS EC2 service.

**Top Page (Search Results):**

- The URL is [us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances](https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances).
- The search bar at the top contains "ec2".
- The main content area is titled "Services" and lists:

  - EC2**: Virtual Servers in the Cloud
  - EC2 Image Builder**: A managed service to automate build, customize and deploy OS images
  - EC2 Global View**: EC2 Global View provides a global dashboard and search functionality that lets you ...

- The "Features" section includes:

  - Dashboard**: EC2 feature
  - EC2 Instances**: CloudWatch feature

- A message "Were these results helpful?" with "Yes" and "No" buttons is present.

**Bottom Page (Launch Wizard):**

- The URL is [us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances](https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances).
- The search bar at the top contains "Search".
- The main content area is titled "Launch an instance".
- The "Name and tags" section shows "Name: HomeMadePickles" and a "Add additional tags" button.
- The "Application and OS Images (Amazon Machine Image)" section shows "Software Image (AMI): Amazon Linux 2023.7.2...read more" and "ami-05ffe3c48a9991133".
- The "Virtual server type (instance type)" is set to "t2.micro".
- The "Firewall (security group)" is set to "New security group".
- The "Storage (volumes)" section is collapsed.
- Buttons include "Cancel", "Launch instance" (highlighted in orange), and "Preview code".

The screenshot shows the AWS EC2 Instances page with the URL <https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances>. The top navigation bar includes tabs for 'Inbox (316)', 'Student +5', 'troven.in/s', 'troven.in/s', 'Launch an...', 'Khit\_3...', 'App passiv...', 'homemad...', and a '+' sign. The main content area shows the 'Amazon Machine Image (AMI)' section, where 'Amazon Linux 2023 kernel-6.1 AMI' is selected. Other AMI options shown include 'Mac', 'ubuntu®', 'Microsoft', 'Red Hat', and 'SUSE'. A note says 'Including AMIs from AWS, Marketplace and the Community'. Below this, the 'Description' section details 'Amazon Linux 2023 (kernel-6.1) 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.' The 'Virtualization' settings are listed as 'hvm', 'ENA enabled: true', and 'Root device type: ebs'. The 'Architecture' table shows '64-bit (...', 'Boot mode uefi-preferred', 'AMI ID ami-05ffe3c48a9991133', 'Publish Date 2025-06-20', and 'Username ec2-user'. A green button at the bottom left says 'Verified provider'. On the right, the 'Summary' section shows 'Number of instances 1', 'Software Image (AMI) Amazon Linux 2023 7.2...', 'Virtual server type (instance type) t2.micro', 'Firewall (security group) New security group', and 'Storage (volumes)'. Buttons for 'Cancel', 'Launch instance', and 'Preview code' are at the bottom right.

The screenshot shows the AWS CloudShell interface with multiple tabs open. The main focus is the EC2 'Launch an instance' wizard. The first step, 'Instance type', is selected. It shows a dropdown menu for 't2.micro' with details: Family: t2, 1 vCPU, 1 GiB Memory, Current generation: true. There's also a toggle for 'All generations' and a 'Compare instance types' link. The second step, 'Key pair (login)', is shown below, with a note about using a key pair for secure connection. The third step, 'Software Image (AMI)', is partially visible on the right. The bottom right contains 'Cancel', 'Launch instance', and 'Preview code' buttons.

CloudShell Feedback

Type here to search

Inbox (316) Student - troven.in/s troven.in/s Launch an Khit\_3i App passw homema School

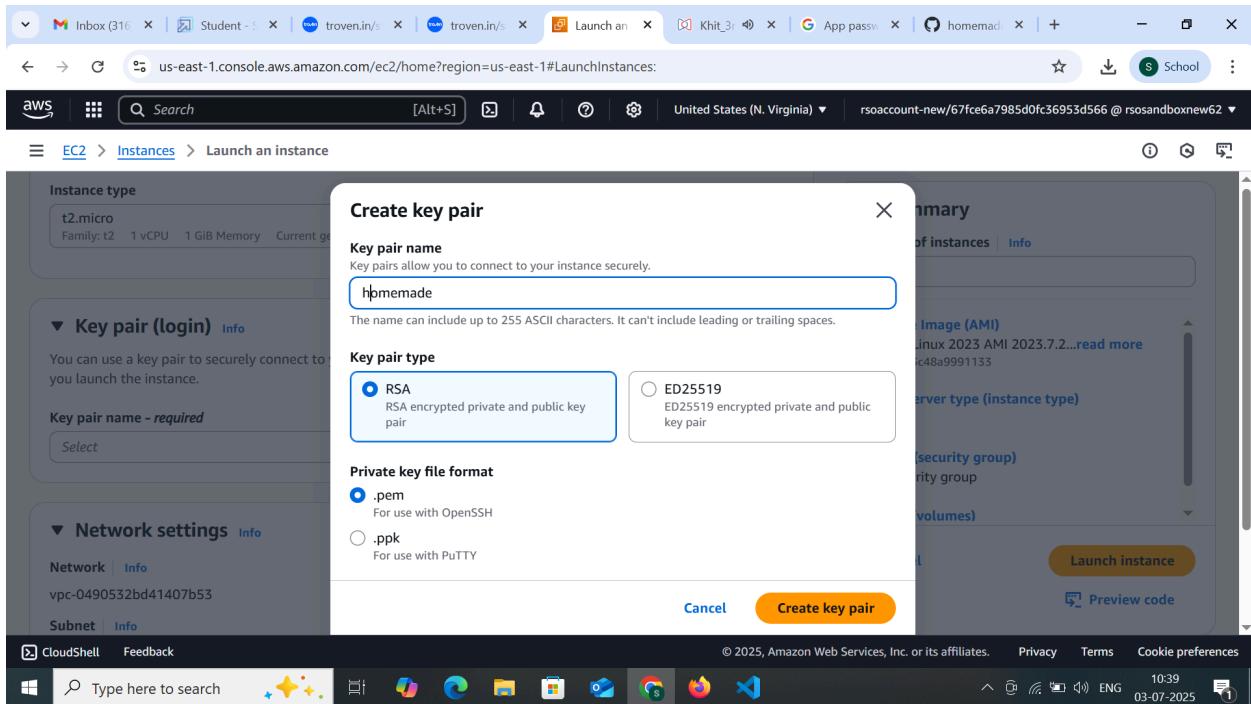
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

CloudShell Feedback

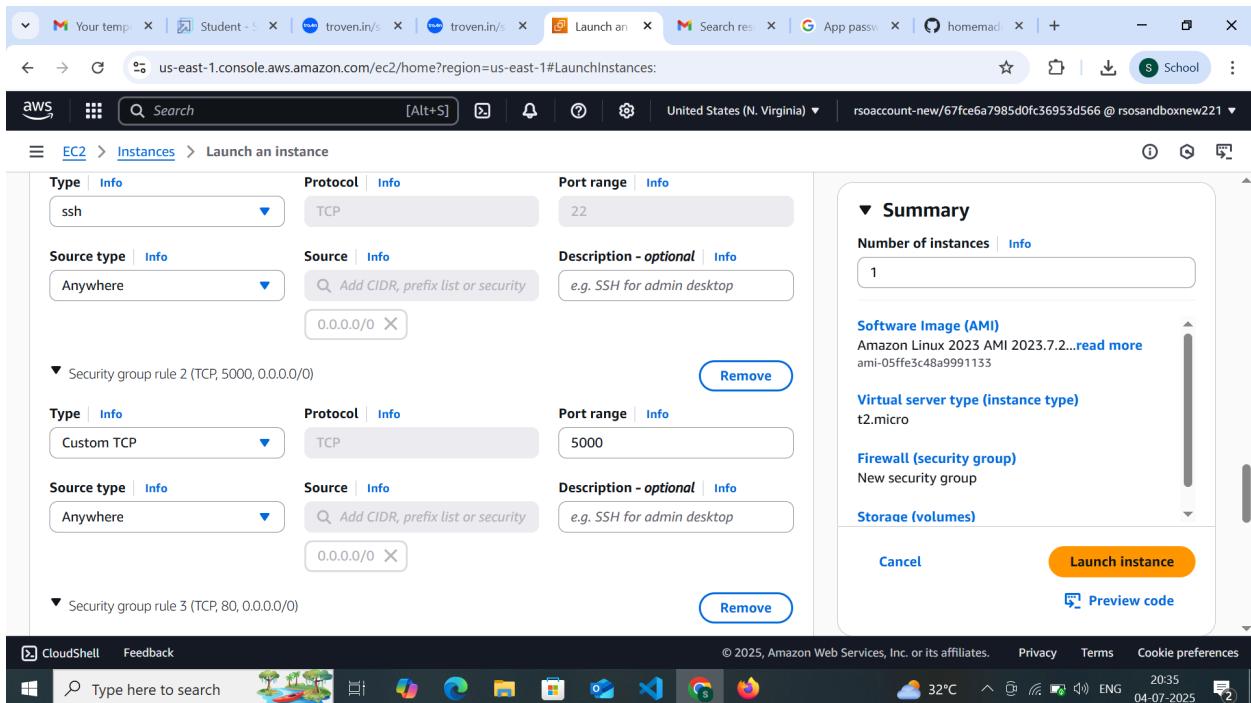
Type here to search

CloudShell Feedback

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



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



To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected, navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the EC2 Dashboard, click the Connect button. From

the connection methods presented, choose EC2 Instance Connect. Finally, click Connect again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

The image consists of three vertically stacked screenshots of the AWS CloudWatch Metrics Insights interface. Each screenshot shows the results of a query for the AWS Lambda function 'GetAWSLogsMetrics'. The results are displayed in a table with columns for 'Time' (UTC), 'Function Name', 'Metric Name', and 'Value'.

Time (UTC)	Function Name	Metric Name	Value
2023-09-18T10:00:00Z	GetAWSLogsMetrics	ApproximateInvokeCount	1
2023-09-18T10:00:00Z	GetAWSLogsMetrics	ApproximateSuccessCount	1
2023-09-18T10:00:00Z	GetAWSLogsMetrics	ApproximateFailureCount	0
2023-09-18T10:00:00Z	GetAWSLogsMetrics	Approximate吞吐量 (ApproximateThroughput)	1.0

The first screenshot shows the overall metrics table. The second and third screenshots show the detailed results for the 'Value' column, which includes the raw log data for each metric. The raw log data for 'ApproximateInvokeCount' shows the JSON output of the Lambda function, which includes the count of successful invocations and the approximate throughput.

EC2 > Instances > i-001861022fbcac290 > Modify IAM role

## Modify IAM role Info

Attach an IAM role to your instance.

Instance ID  
 i-001861022fbcac290 (InstantLibraryApp)

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.

## Connect to instance Info

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

[EC2 Instance Connect](#) [Session Manager](#) [SSH client](#) [EC2 serial console](#)

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

Instance ID  
 i-001861022fbcac290 (InstantLibraryApp)

Connection Type

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

Connect using EC2 Instance Connect Endpoint  
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IPv4 address  
 13.200.229.59

IPv6 address

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

**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.

```

Requirement already satisfied: jinja2>=3.1.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (3.0.2)
Requirement already satisfied: importlib-metadata>=3.6.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (8.7.0)
Requirement already satisfied: werkzeug>=3.1.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (3.1.3)
Requirement already satisfied: blinker>=1.9.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (1.9.0)
Requirement already satisfied: zipp>=3.20 in /home/ec2-user/.local/lib/python3.9/site-packages (from importlib-metadata>=3.6.0->Flask) (3.23.0)
[ec2-user@ip-172-31-17-125 PROJECT]$ 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://172.31.17.125:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-181-203
^C[ec2-user@ip-172-31-17-125 PROJECT]$ ^C
[ec2-user@ip-172-31-17-125 PROJECT]$ 

```

i-0ee97f45ced0e202d (HomeMadePickles)

PublicIPs: 54.236.94.61 PrivateIPs: 172.31.17.125

## Milestone 6 : Deployment on EC2

Deployment on an EC2 instance involves launching a server, configuring security groups for public access, and uploading your application files. After setting up necessary dependencies and environment variables, start your application and ensure it's running on the correct port. Finally, bind your domain or use the public IP to make the application accessible online.

## 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: git clone https://github.com/BabyAvanthiGayathriAkki/homemade-pickles.git
- Note: change your-github-username and your-repository-name with your credentials here: "https://github.com/BabyAvanthiGayathriAkki/homemade-pickles.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

```

Requirement already satisfied: jinja2>=3.1.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (3.0.2)
Requirement already satisfied: importlib-metadata>=3.6.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (8.7.0)
Requirement already satisfied: werkzeug>=3.1.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (3.1.3)
Requirement already satisfied: blinker>=1.9.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from Flask) (1.9.0)
Requirement already satisfied: zipp>=3.20 in /home/ec2-user/.local/lib/python3.9/site-packages (from importlib-metadata>=3.6.0->Flask) (3.23.0)
[ec2-user@ip-172-31-17-125 PROJECT]$ 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://172.31.17.125:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-181-203
^C[ec2-user@ip-172-31-17-125 PROJECT]$ ^C
[ec2-user@ip-172-31-17-125 PROJECT]$ ^C
[ec2-user@ip-172-31-17-125 PROJECT]$ 
```

i-0ee97f45ced0e202d (HomeMadePickles)

PublicIPs: 54.236.94.61 PrivateIPs: 172.31.17.125

Verify the Flask app is running:

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

- Run the Flask app on the EC2 instance

Access the website through:

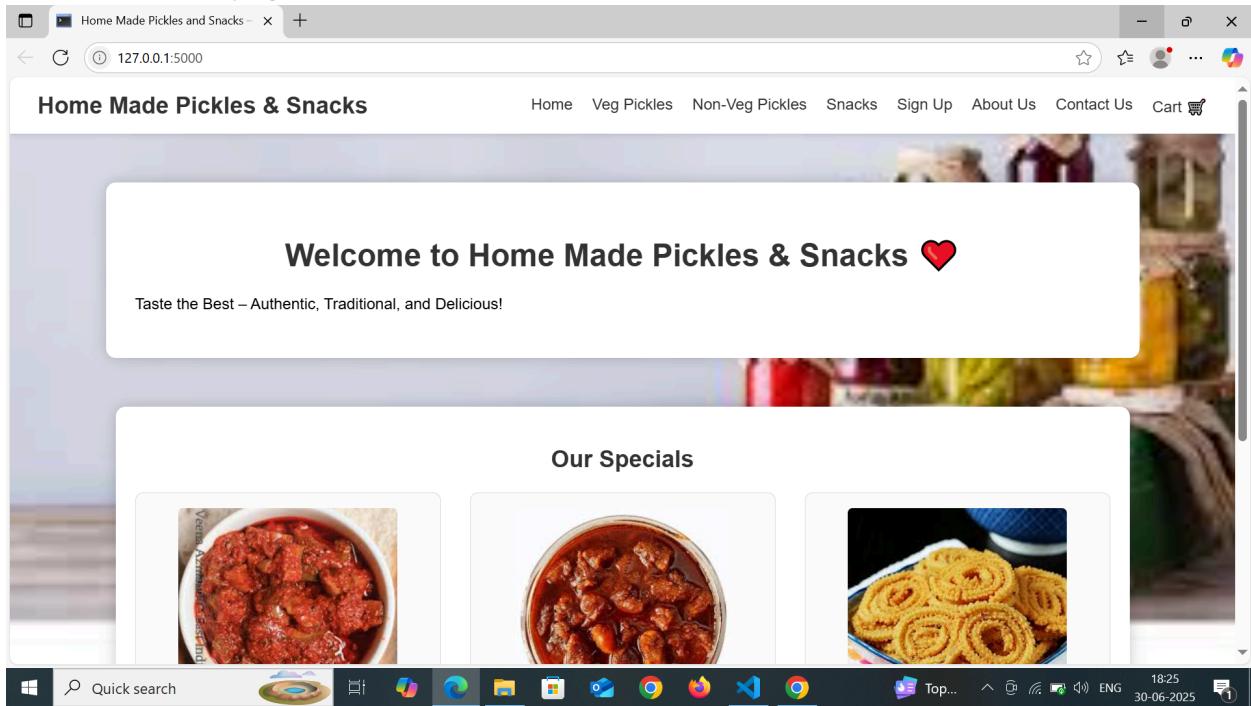
PublicIPs: <http://54.236.94.61:5000>

## 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:



## Conclusion

The Homemade Pickles and Snacks platform has been meticulously crafted to deliver a seamless and delightful experience for food enthusiasts seeking authentic, handcrafted flavors. By leveraging modern web technologies such as Flask for backend logic, secure user authentication, and dynamic cart management, the platform ensures a user-friendly interface for browsing, customizing, and ordering artisanal pickles and snacks.

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.

By combining time-honored recipes with modern e-commerce capabilities, this website not only preserves culinary heritage but also adapts to the digital age, ensuring that every jar of pickle or snack reaches customers with the same care and quality as a homemade meal. As the platform evolves, it stands ready to scale, introduce new product lines, and foster a community of food lovers united by a passion for authentic flavors.

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