HomeMade Pickles & Snacks: Taste the Best

Hardware 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.).

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,

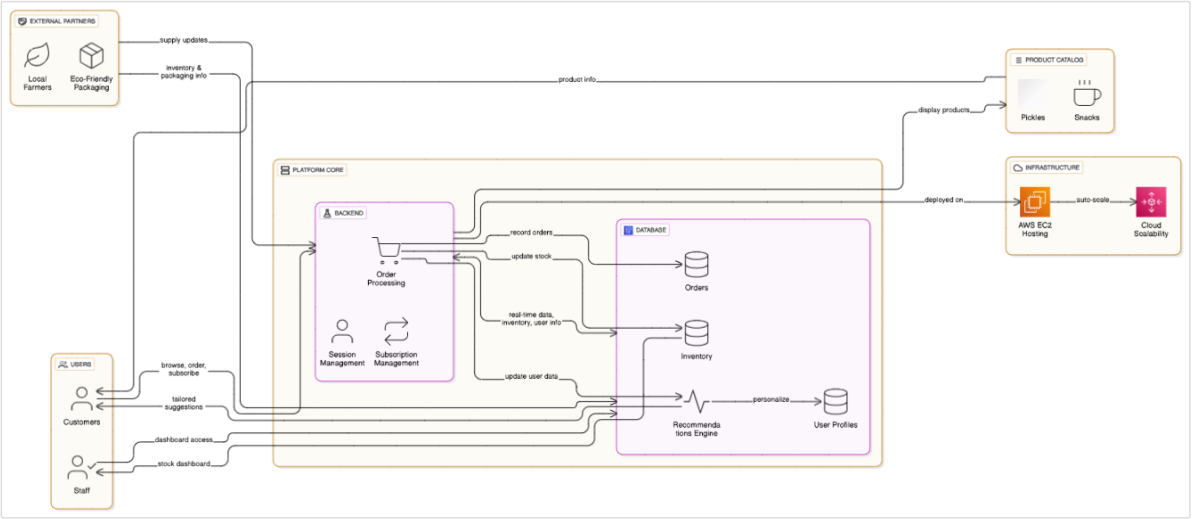
triggering automatic inventory deduction and order confirmation. Staff members receive
updated dashboards to monitor stock availability and fulfillment progress, ensuring timely restocking and minimizing overselling risks.

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.

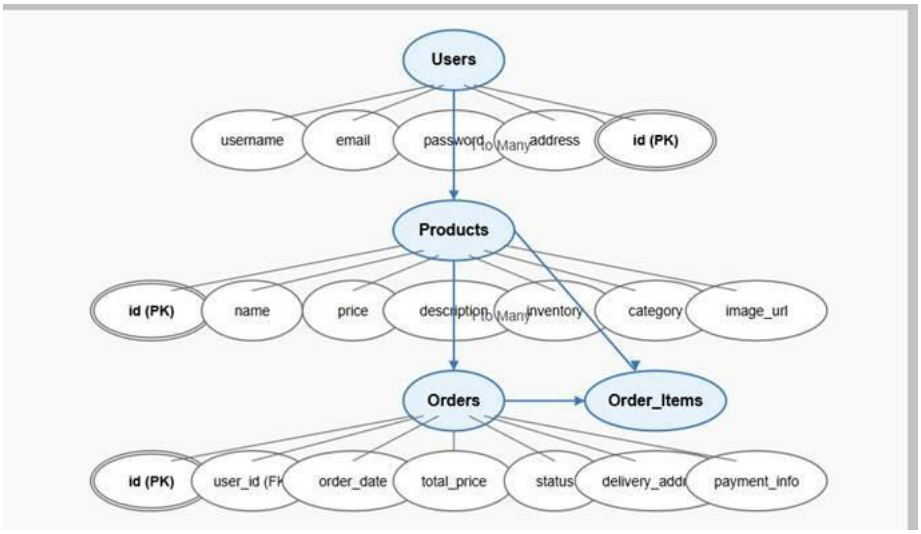
Architecture

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

```
templates
  about.html
  cart.html
  checkout.html
  contact us.html
  home.html
  index.html
  login.html
  non_veg_pickles.html
  signup.html
  snacks.html
  success.html
  veg_pickles.html
app.py
```

Description of the code :

? Flask App Initialization

```
app.py
from flask import Flask, render_template_string, render_template , redirect, url_for, session, request, flash, jsonify
from functools import wraps
import smtplib
import logging
from email.mime.text import MIMEText
from datetime import datetime
from werkzeug.exceptions import BadRequest
from werkzeug.security import generate_password_hash, check_password_hash
import os
import boto3
import uuid
```

```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=int(os.environ.get("PORT", 5000)))
```

- 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 tables
dynamodb = boto3.resource('dynamodb', region_name=region)
orders_table = dynamodb.Table('orders')
users_table = dynamodb.Table('users')
contacts_table = dynamodb.Table('contacts')
reviews_table = dynamodb.Table('reviews')
```

```python
# Product List with Online Image URLs
products = [
    # Non-Veg Pickles
    {
        "id": 1,
        "name": "Spicy Chicken Pickle Boneless",
        "price": 320,
        "image": "https://www.picklesraja.com/wp-content/uploads/2024/11/boon-less-chicken.jpg",
        "description": "Authentic, Spicy, Meaty pickles with high quality and organic ingredients"
    },
    {
        "id": 2,
        "name": "Prawns Pickle",
        "price": 330,
        "image": "https://m.media-amazon.com/images/X/bxt1/M/ibxt1RrE3LsjWSq._SL640_QL75_FMwebp_.jpg",
        "description": "Packed with a generous quantity of prawns in every bottle"
    },
    {
        "id": 3,
        "name": "Korrameru Fish Pickle",
        "price": 400,
        "image": "https://5.imimg.com/data5/ANDROID/Default/2022/1/ZG/CF/RB/145196166/product-jpeg-500x500.jpg",
        "description": "Fresh fish and hygienic preparation"
    },
    {
        "id": 4,
        "name": "Mutton Pickle",
        "price": 330,
        "image": "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSlP2WddI5Nd_xVz2UfUhxe32Xyglpw_cnM9w&s",
        "description": "Juicy mutton pieces"
    },

    # Veg Pickles
    {
        "id": 5,
        "name": "Special Mango Pickle",
```

- Routes for Web Pages

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

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username'].strip()
        password = request.form['password'].strip()
        if users.get(username) == password:
            session['username'] = username
            flash("Logged in successfully!", "success")
            return redirect(url_for('products_page'))
        else:
            flash("Invalid username or password.", "error")
    return render_template_string("""
    <body style="background-image: url('{{ url_for('static', filename='bg.jpg') }}');
            background-size: cover;
            background-position: center;
            font-family: sans-serif;
            padding: 20px;">
    <h2 style="color:#2c3e50;">Login</h2>
    <form method="POST">
        Username: <input type="text" name="username"><br><br>
        Password: <input type="password" name="password"><br><br>
        <button type="submit">Login</button>
    </form>
    <a href="{{ url_for('register') }}">Don't have an account? Register</a>
    """)
```

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

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username'].strip()
        password = request.form['password'].strip()
        if username in users:
            flash("Username already exists.", "error")
        elif not username or not password:
            flash("Please enter both username and password.", "error")
        else:
            users[username] = password
            flash("Registered successfully. Please login.", "success")
            return redirect(url_for('login'))
    return render_template_string("""
    <body style="background-image: url('{{ url_for('static', filename='bg.jpg') }}');
            background-size: cover;
            background-position: center;
            font-family: sans-serif;
            padding: 20px;">
    <h2 style="color:#2c3e50;">Register</h2>
    <form method="POST">
        Username: <input type="text" name="username"><br><br>
        Password: <input type="password" name="password"><br><br>
        <button type="submit">Register</button>
    </form>
    <a href="{{ url_for('login') }}">Already registered? Login</a>
    """)
```

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

```python
@app.route('/logout')
def logout():
    session.clear()
    flash("Logged out successfully.", "success")
    return redirect(url_for('login'))
```

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

```python
@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:

```python
@app.route('/checkout')
@login_required
def checkout():
    cart_items = []
    total = 0

    for product_id, quantity in session.get('cart', {}).items():
        product = next((p for p in products if p['id'] == int(product_id)), None)
        if product:
            cart_items.append({
                'name': product['name'],
                'quantity': quantity,
                'items': cart,
                'price': product['price']
            })
            total += product['price'] * quantity

    return render_template_string("""
    <body style="background-image: url('{{ url_for('static', filename='bg.jpg') }}');
             background-size: cover;
             background-position: center;
             font-family: sans-serif;
             padding: 20px;">
    <h1  style="color:#2c3e50;">Checkout</h1>
    {% if cart_items %}
        <ul style="list-style:none;">
            {% for item in cart_items %}
                <li>{{ item.name }} ✖ {{ item.quantity }} — ₹{{ item.quantity * item.price }}</li>
            {% endfor %}
        </ul>
        <p><strong>Total: ₹{{ total }}</strong></p>
        <form method="POST" action="{{ url_for('place_order') }}">
        Name:<input name="name"><br>
        Address:<input name="address"><br>
        email:<input name="email"><br>
```

```python
p.py > ...
def checkout():
    {% else %}
        <p>Your cart is empty.</p>
    {% endif %}
     save_order_to_dynamodb(order_data)
     send_order_email(email, summary)

        # (Optional) SNS call is defined but not triggered here
        # send_sns_notification("New order received.")

    <a href="{{ url_for('cart') }}">← Back to Cart</a>
    """, cart_items=cart_items, total=total)

@app.route('/place_order', methods=['POST'])
@login_required
def place_order():
    session['cart'] = {}
    flash("🎉 Your order has been placed successfully!", "success")
    session.pop('cart', None)
    logger.info("Order placed and cart cleared.")
    return redirect(url_for('order_success'))

@app.route('/success')
@login_required
def order_success():
    return render_template_string("""
    <body style="background-image: url('{{ url_for('static', filename='bg.jpg') }}');
             background-size: cover;
             background-position: center;
             font-family: sans-serif;
             padding: 20px;">
    <h2>✅ Order Successful!</h2>
    <p>Thank you for your order, {{ session['username'] }}! 😊</p>
    <a href="{{ url_for('products_page') }}">← Back to Products</a><br>
    <a href="{{ url_for('logout') }}">🚪 Logout</a>
    """)
```

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.

Reminder: You must complete the Web Development task before gaining access to Troven. Once accessed, the AWS Console via Troven is available for only 3 hours—please plan your work accordingly.

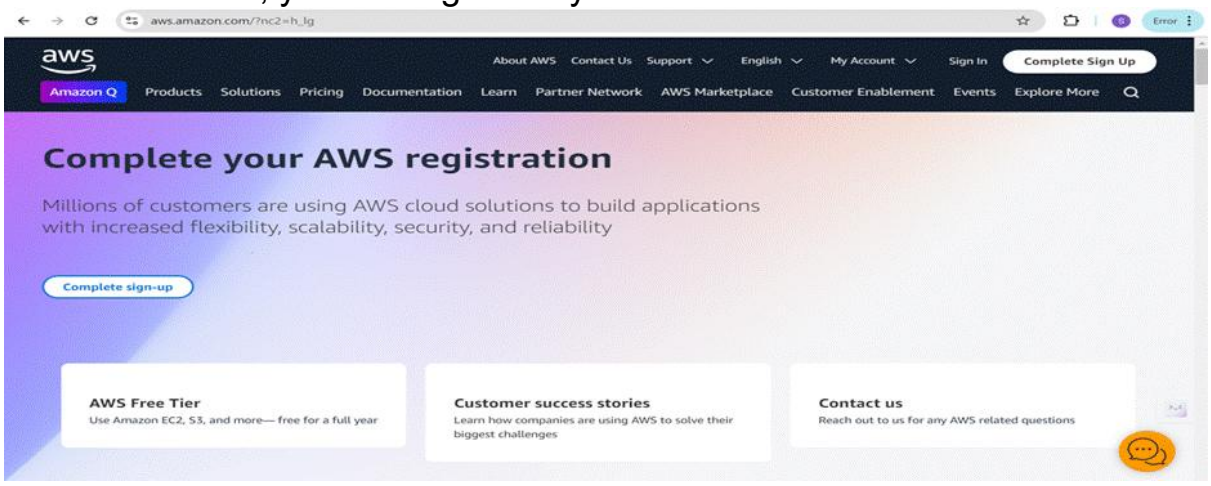
Please follow the provided guidelines and access AWS exclusively through Troven to avoid unnecessary issues.
Please refer the below link –

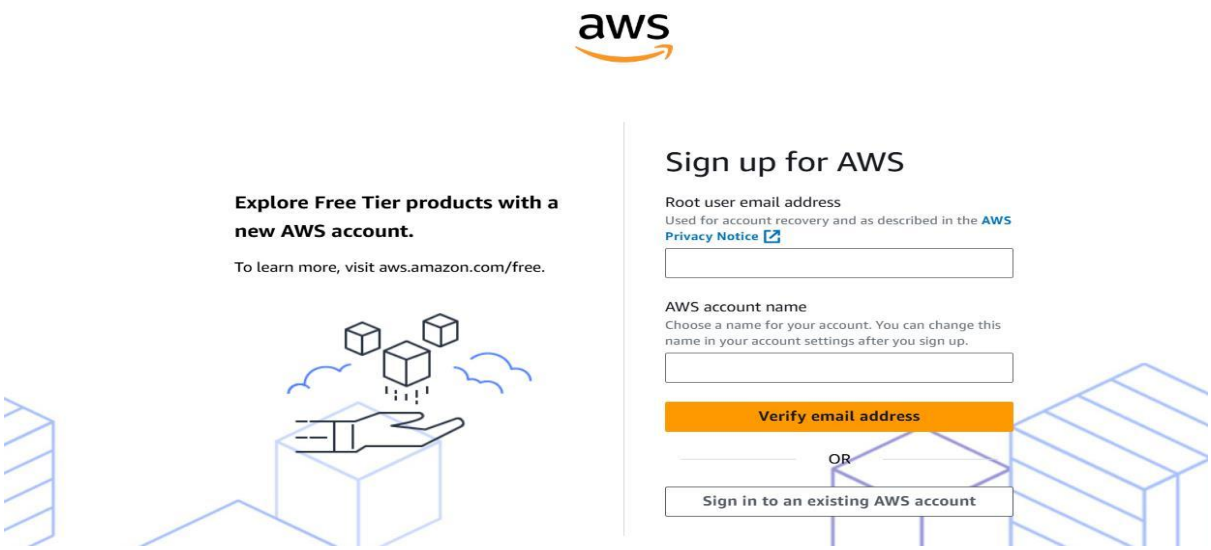
https://drive.google.com/file/d/1HzWc7AMJ2BrxhV-uaw5s0vWtcd-28qgl/view?usp=sharing

AWS Account Setup and Login

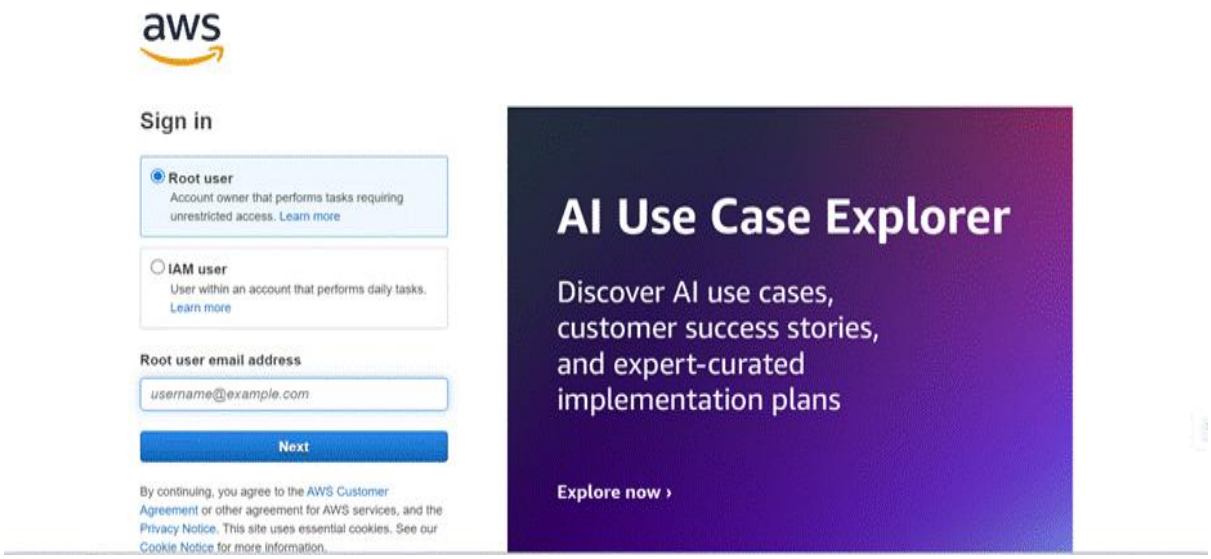
This is for your understanding only, please refrain from creating an AWS account.
A temporary account will be provided via Troven.

- 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 account





- Log in to the AWS Management Console
- After setting up your account, log in to the AWS Management Console.

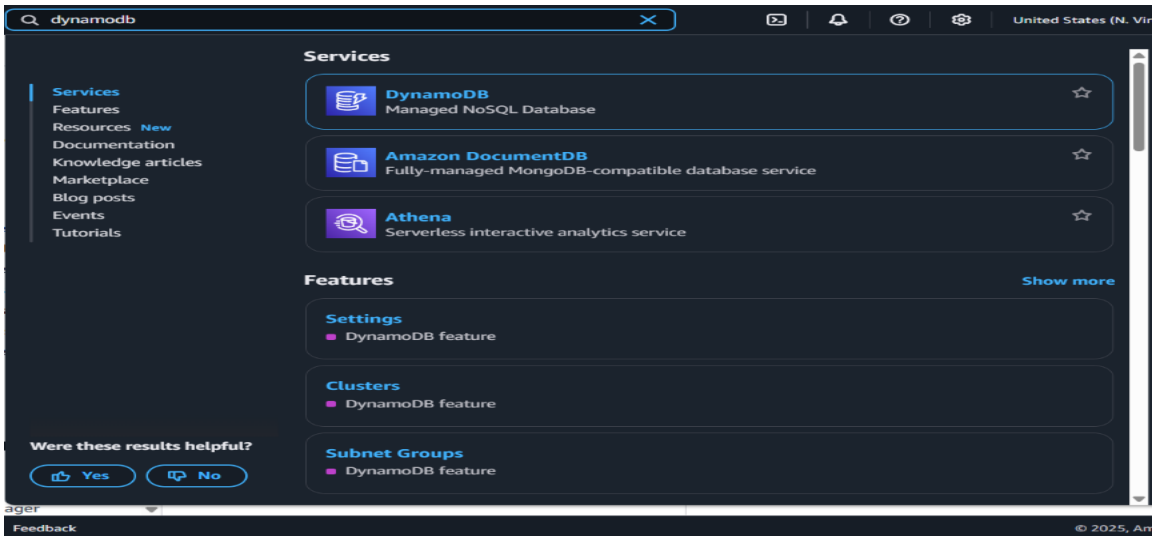


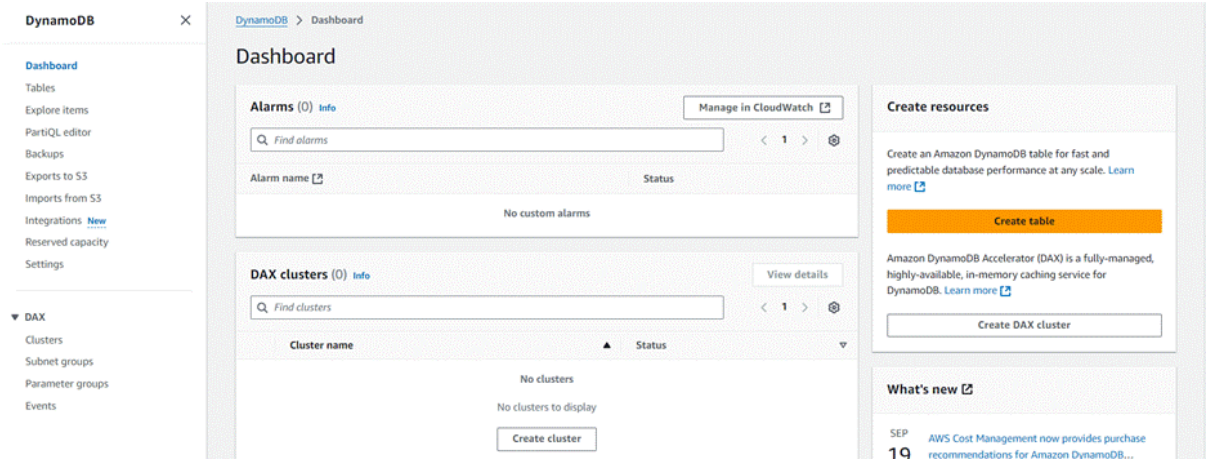
Milestone 3 : DynamoDB Database Creation and Setup

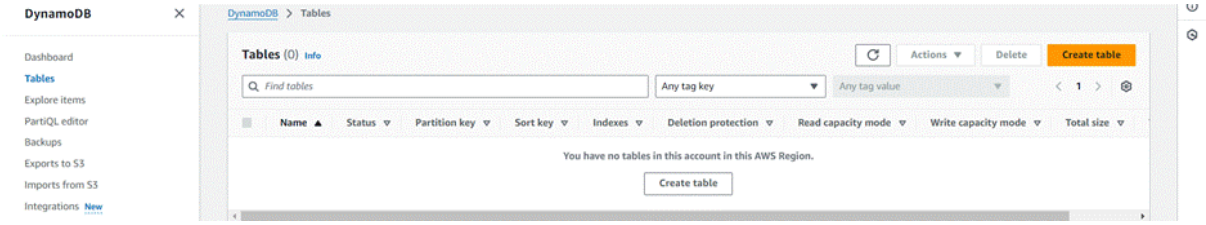
Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high - performance data storage for seamless backend operations.

Navigate to the DynamoDB

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

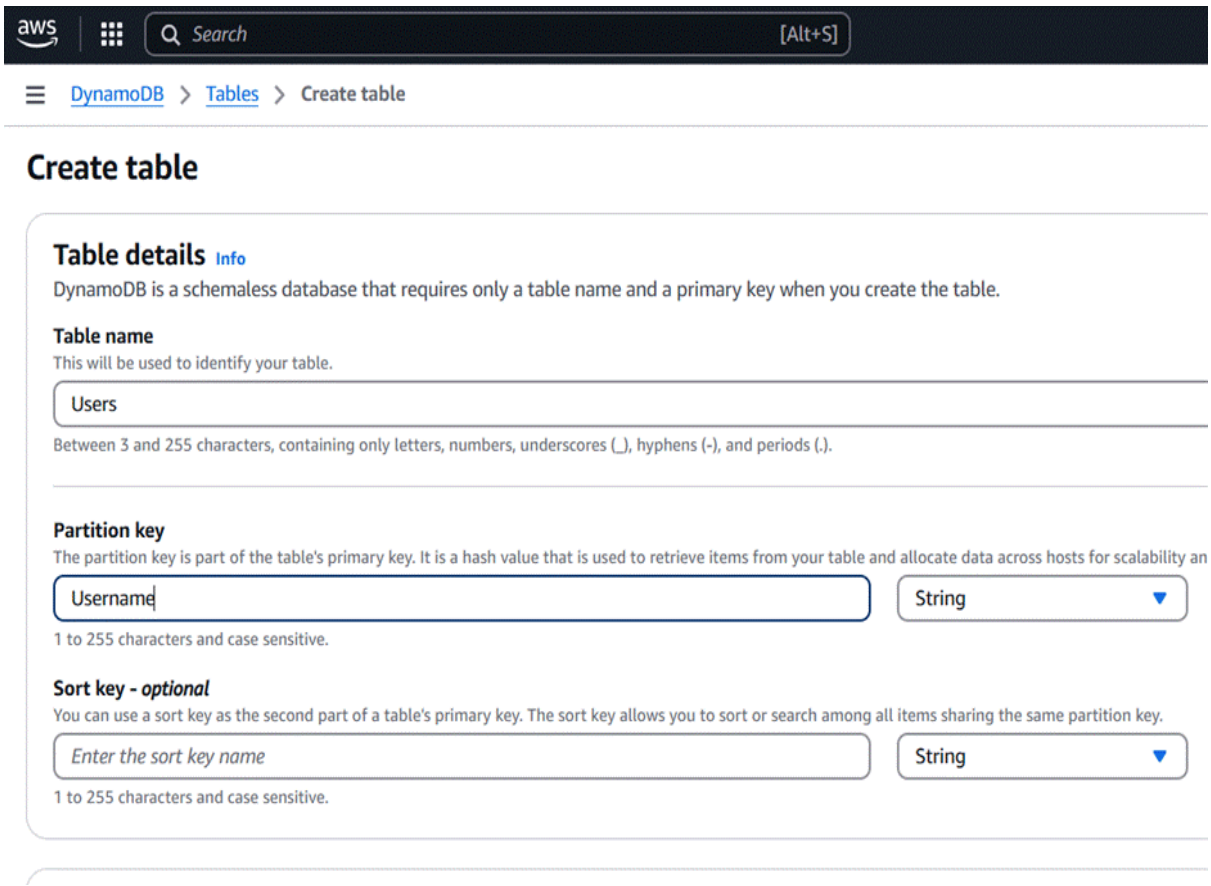






Create a DynamoDB table for storing data

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



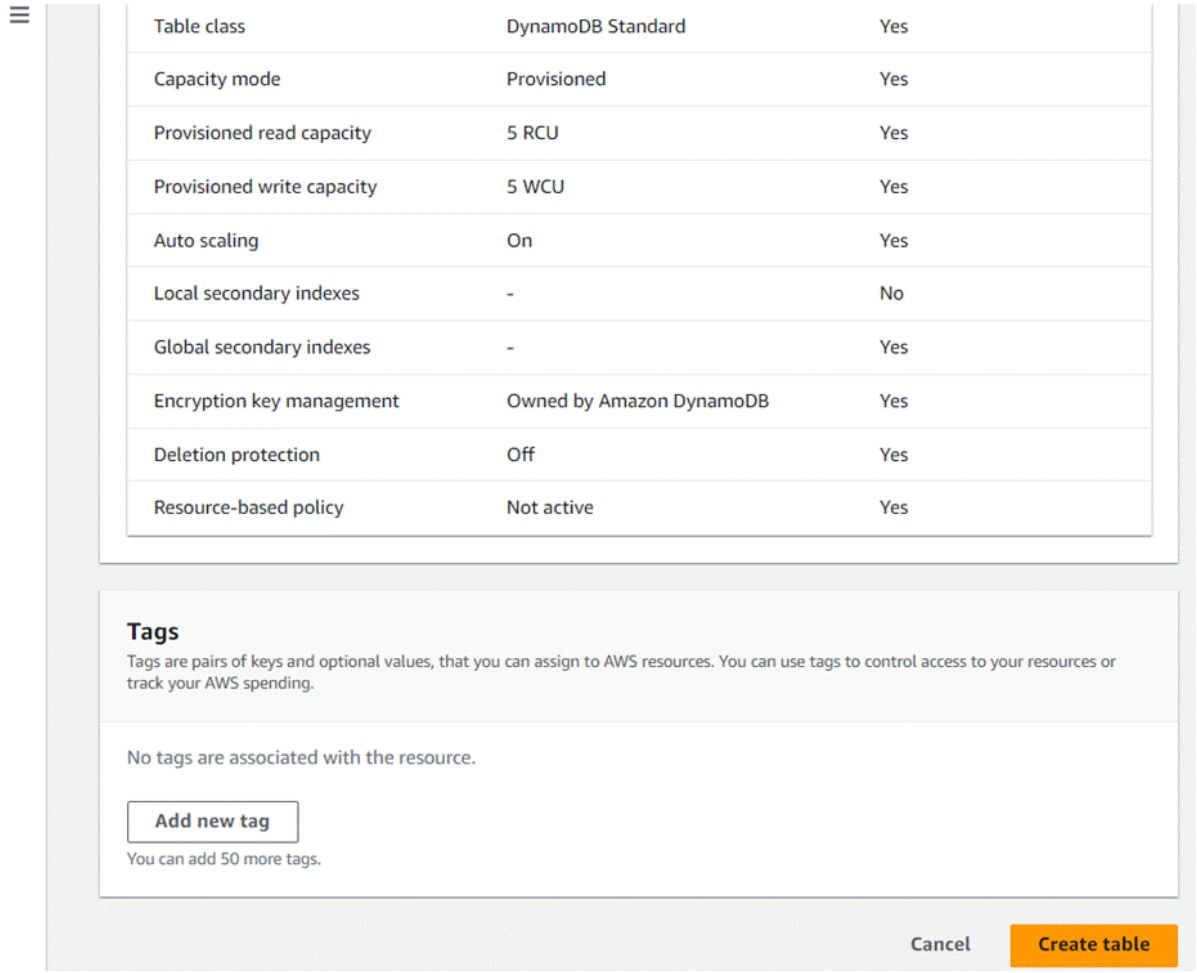


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags

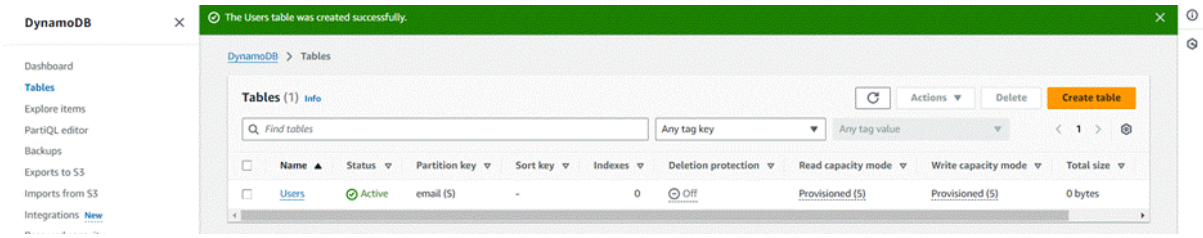
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel Create table



- 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 a

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

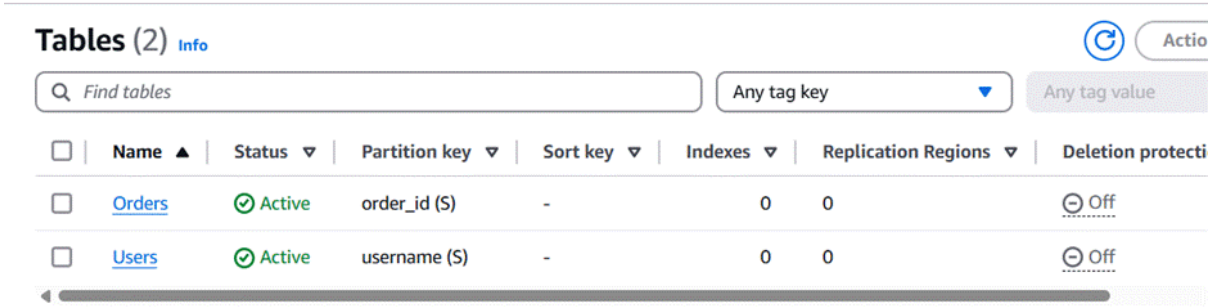
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel | Create table

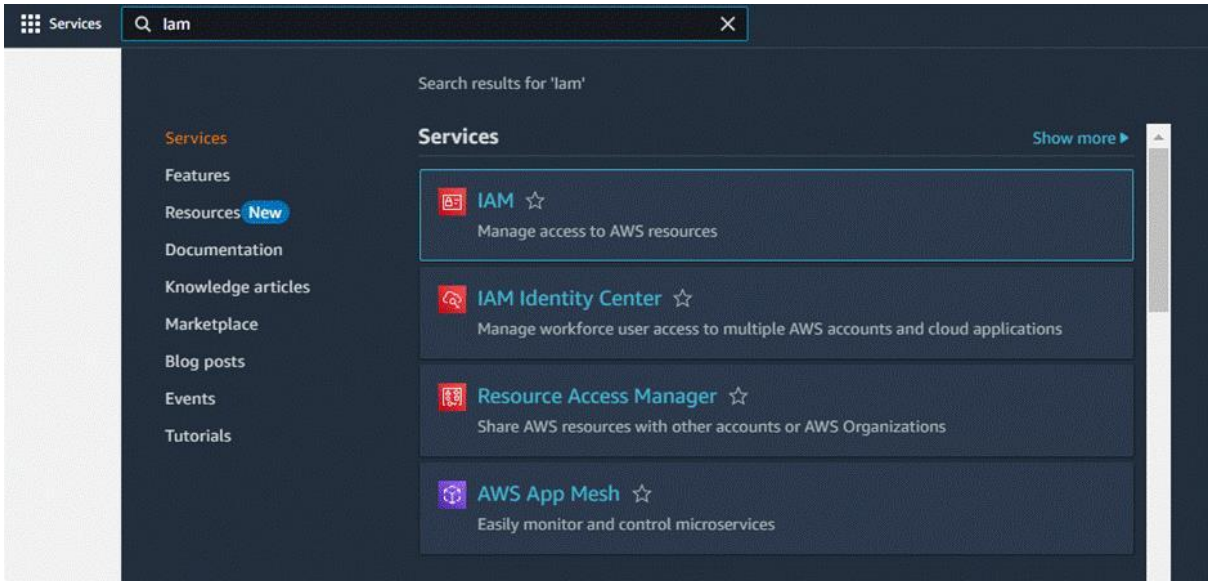


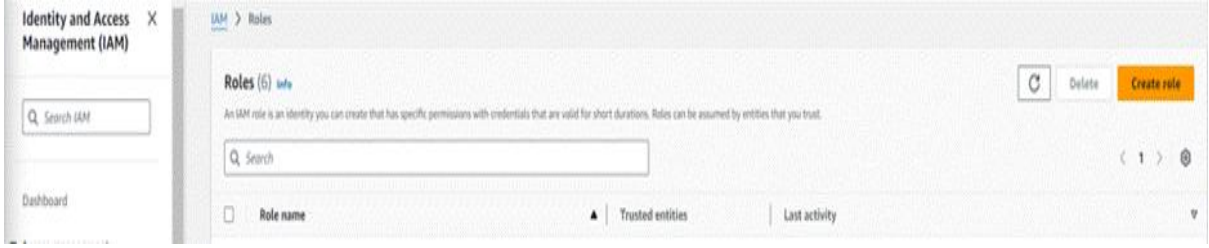
Milestone 4 : IAM Role Setup

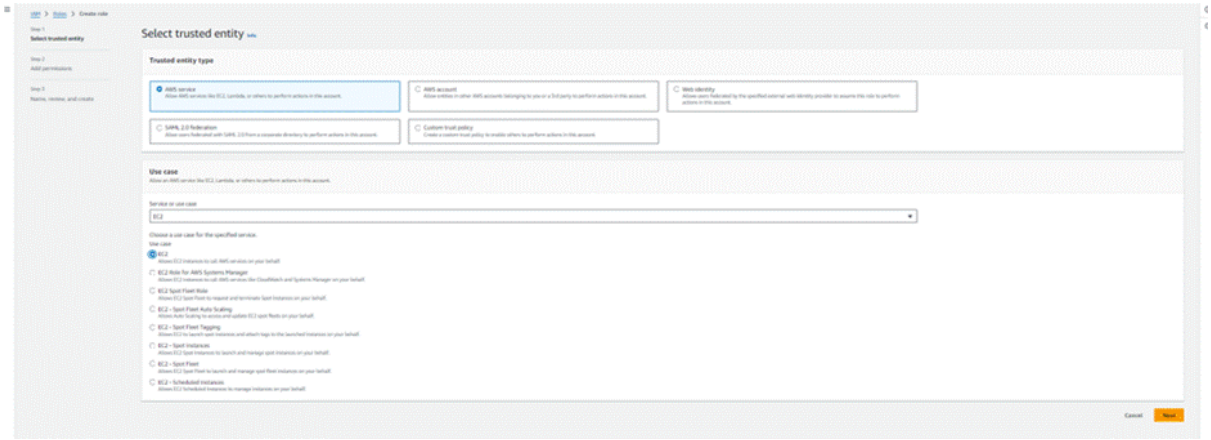
IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows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

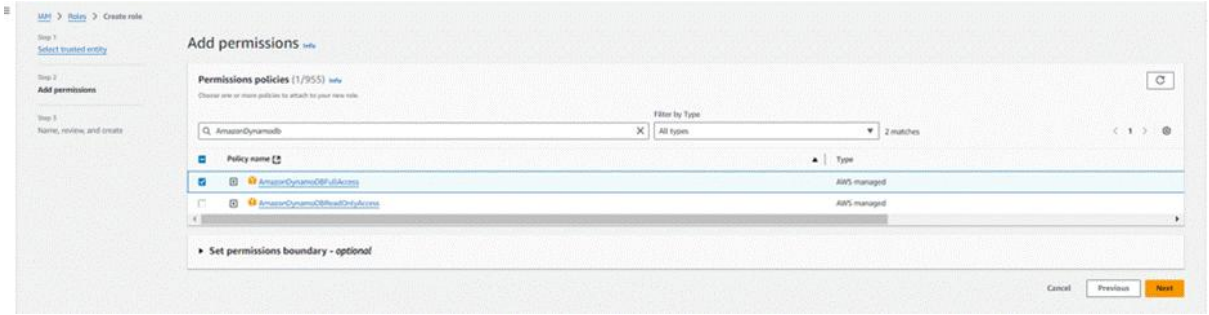
Create IAM Role.

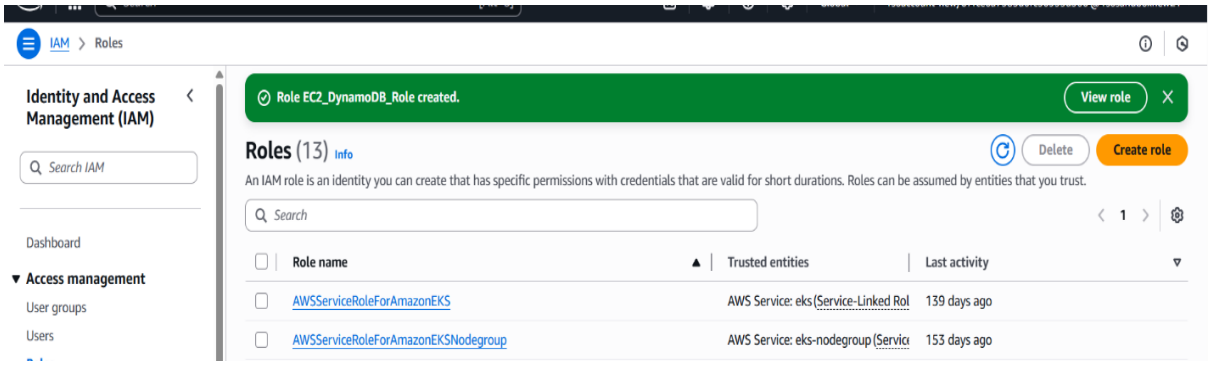
- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB.




Identity and Access Management (IAM)
Search IAM
Dashboard
IAM > Roles
Roles (6) Info
Delete
Create role
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.
Search
Role name
Trusted entities
Last activity


Select trusted entity
Trusted entity type
Use case

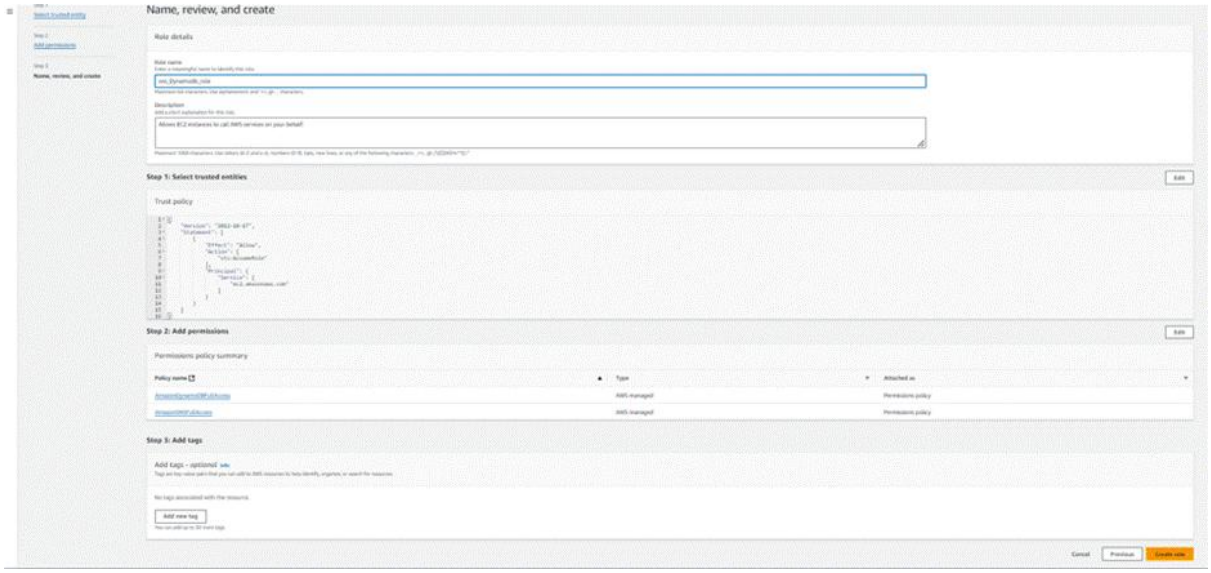

IAM > Roles > Create role
Add permissions
Permissions policies (1/955)
AmazonDynamodb
Policy name
Type
AWS-managed
Set permissions boundary - optional
Cancel
Previous
Next

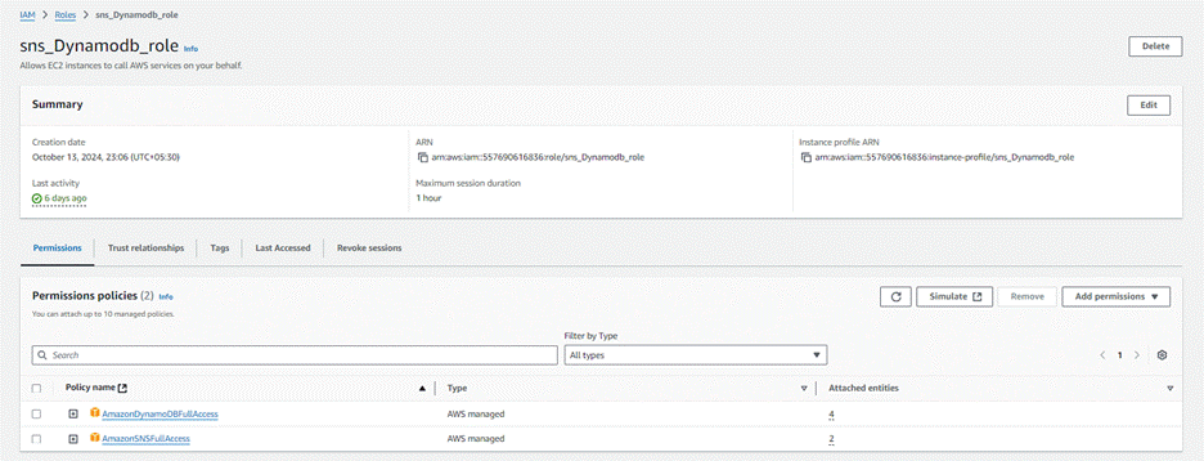

IAM > Roles
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Role EC2_DynamoDB_Role created.
View role
Roles (13) Info
Delete
Create role
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.
Search
Role name
Trusted entities
Last activity
AWSServiceRoleForAmazonEKS
AWS Service: eks (Service-Linked Rol
139 days ago
AWSServiceRoleForAmazonEKSNodegroup
AWS Service: eks-nodegroup (Servic
153 days ago

Attach Policies

Attach the following policies to the role:

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



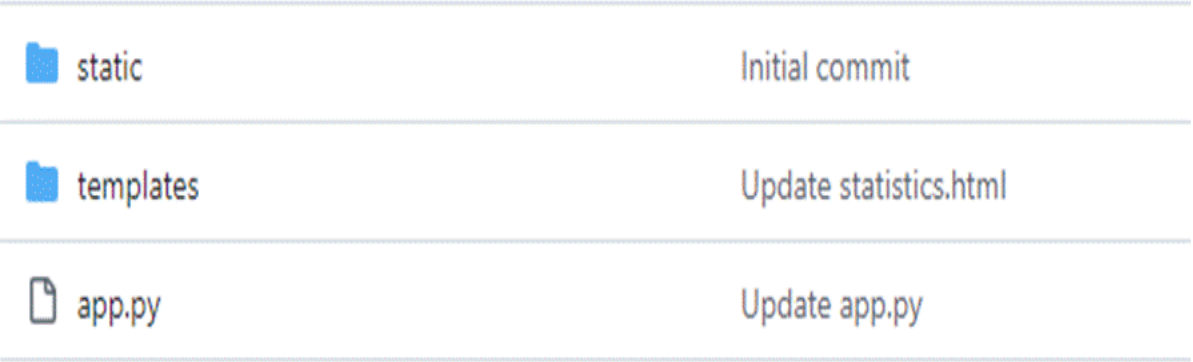


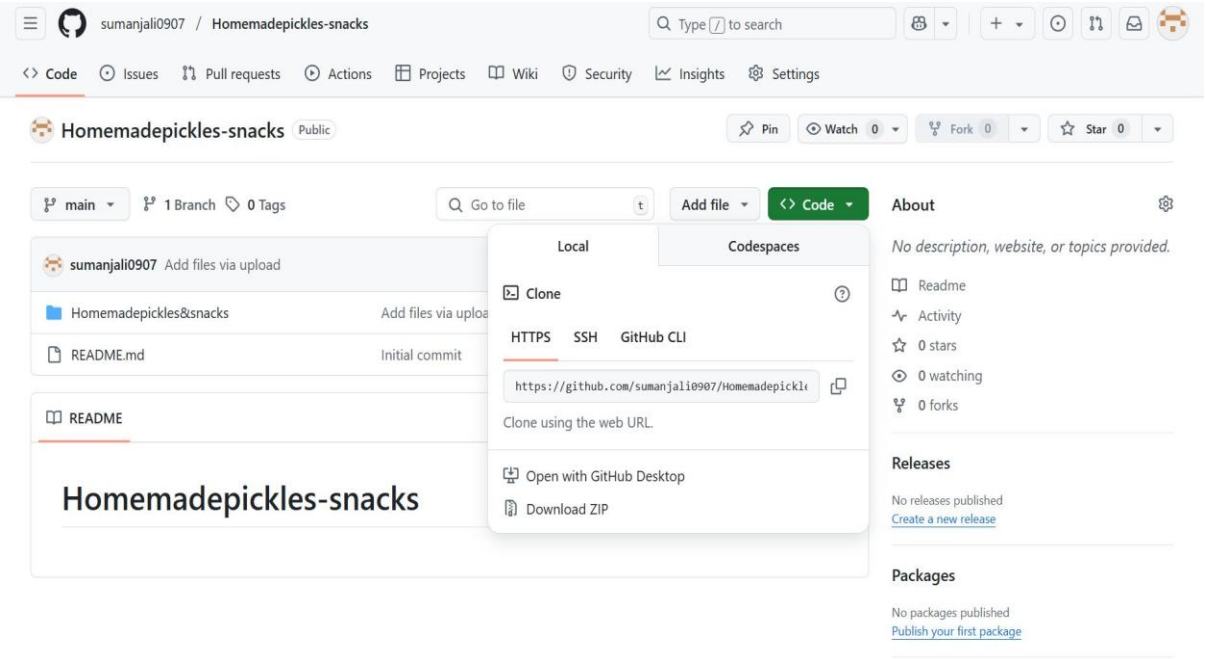
Milestone 5 : EC2 Instance Setup

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.

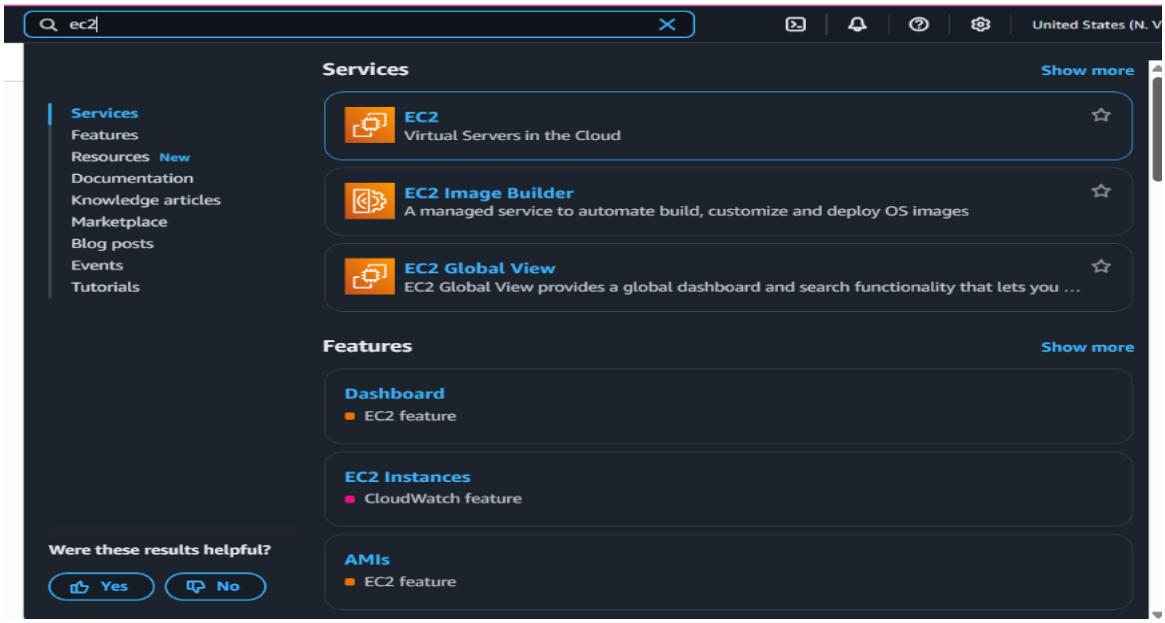
- Note: Load your Flask app and Html files into GitHub repository



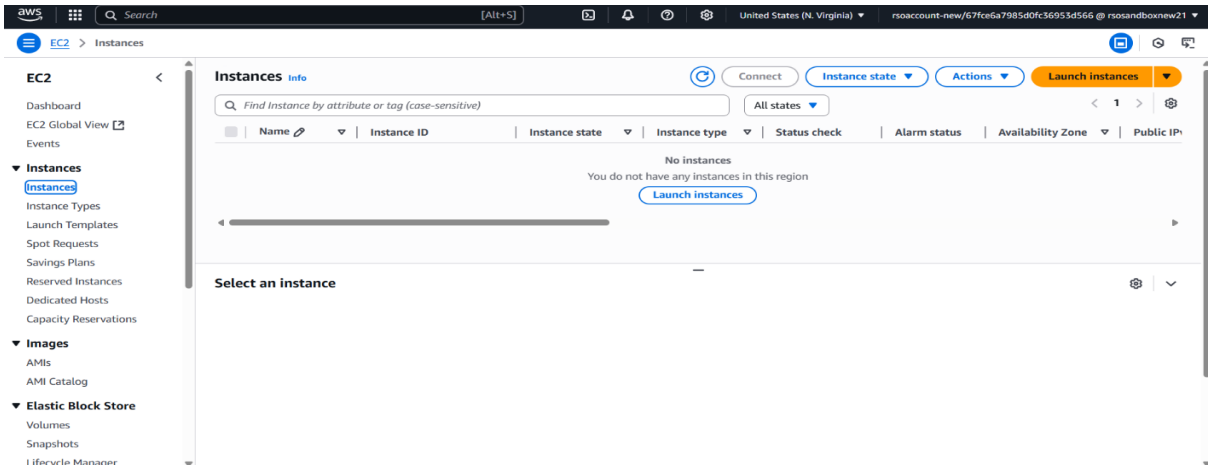


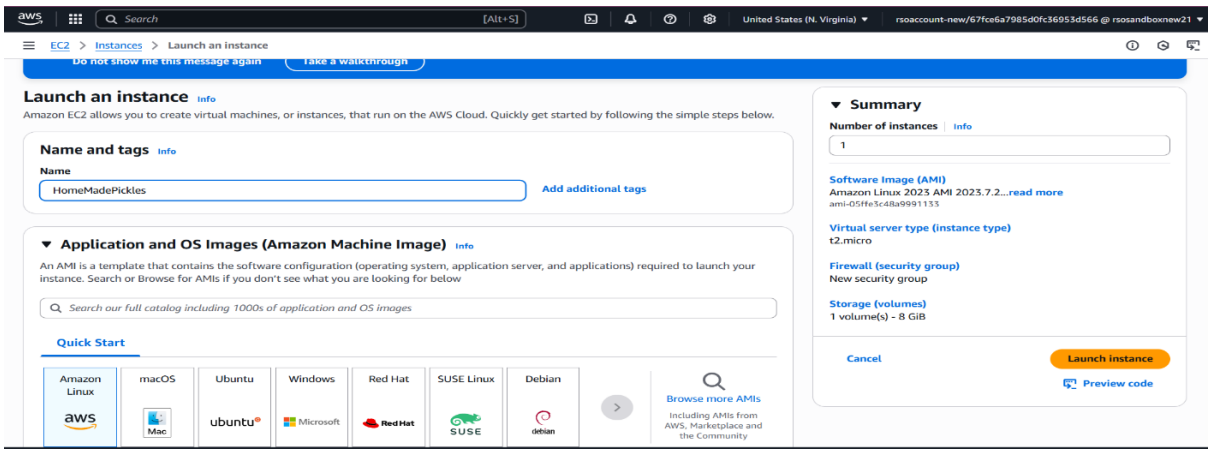
Launch an EC2 instance to host the Flask

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

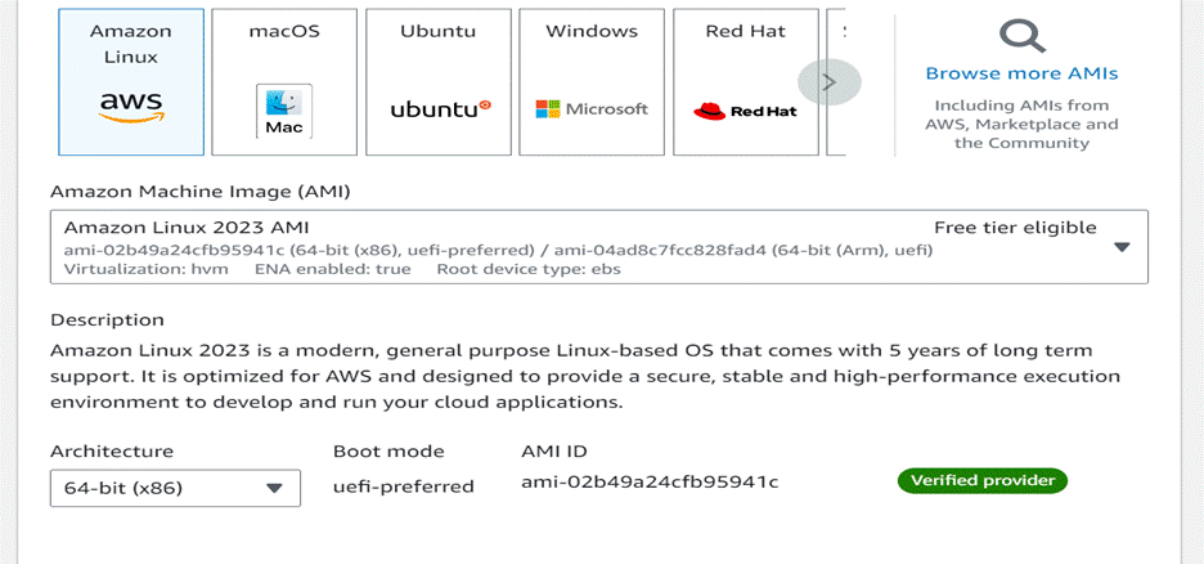


- Click on Launch instance to launch EC2 instance

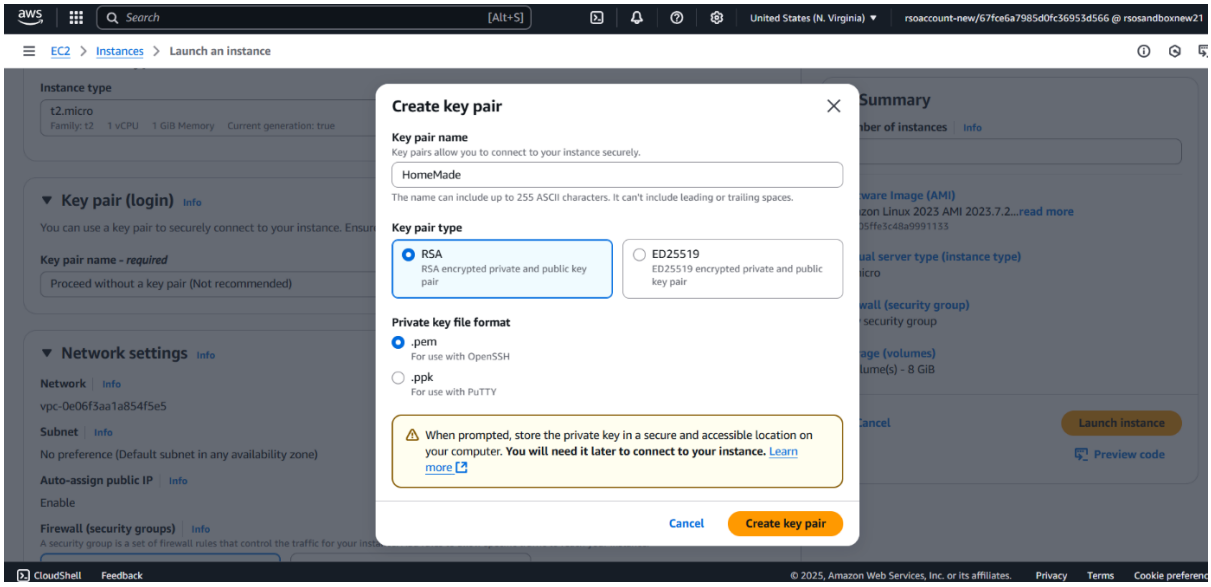


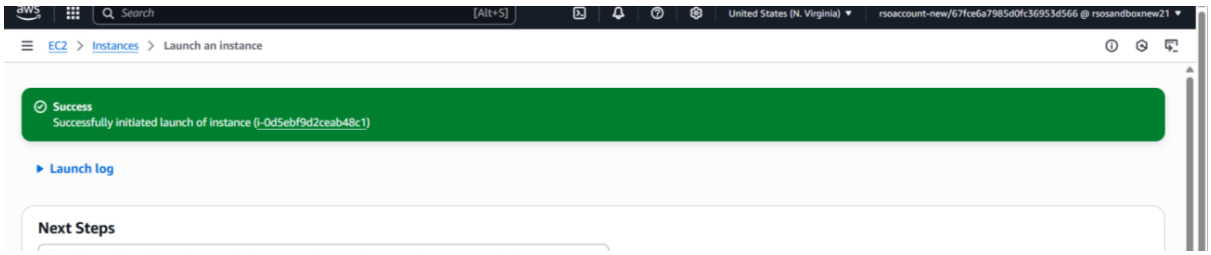


? Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



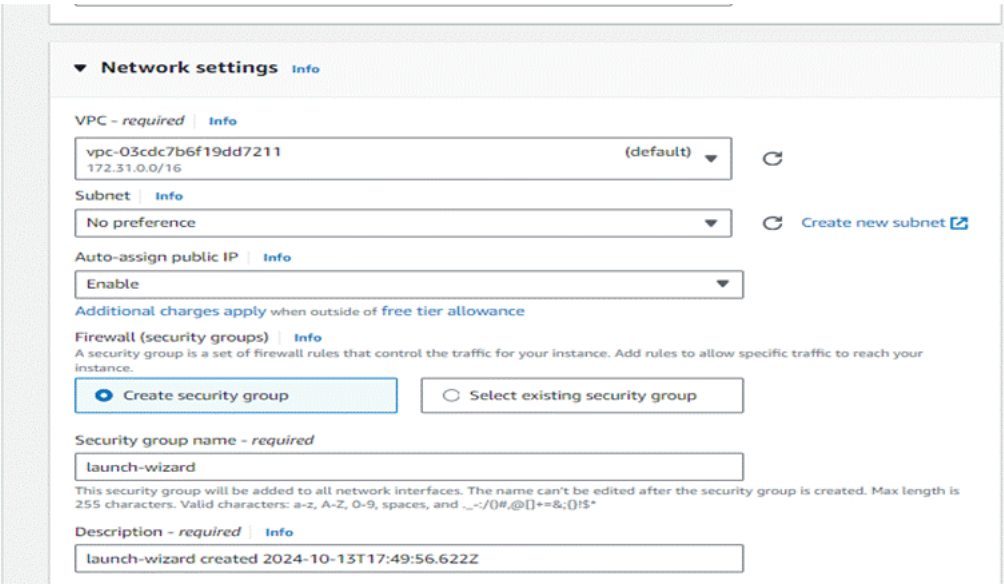
- Create and download the key pair for Server access.

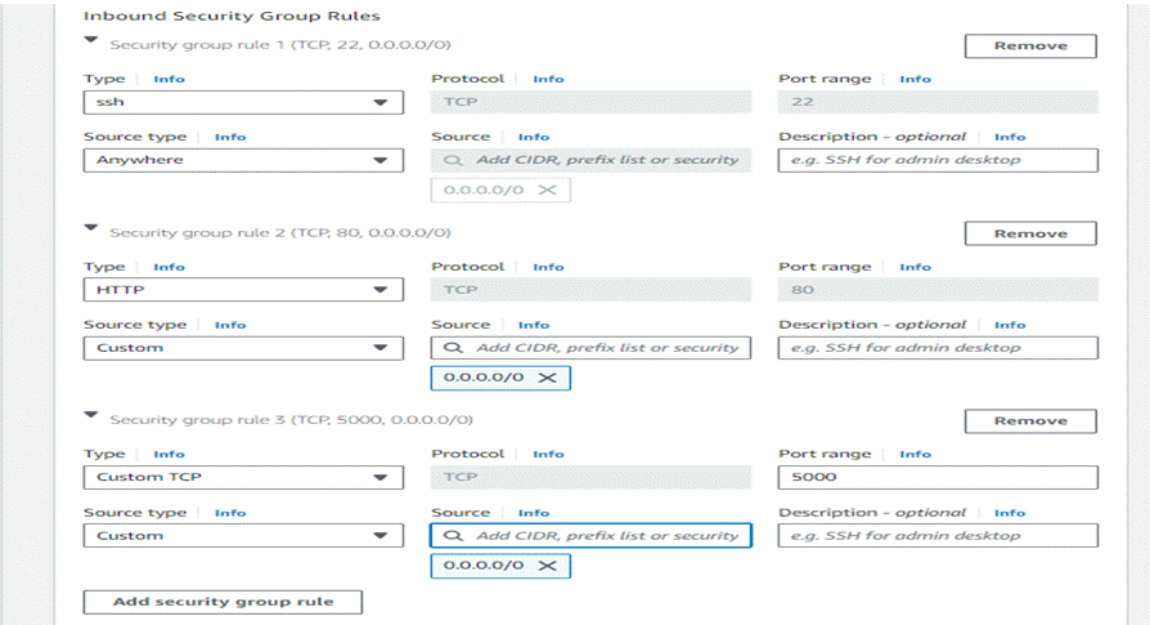


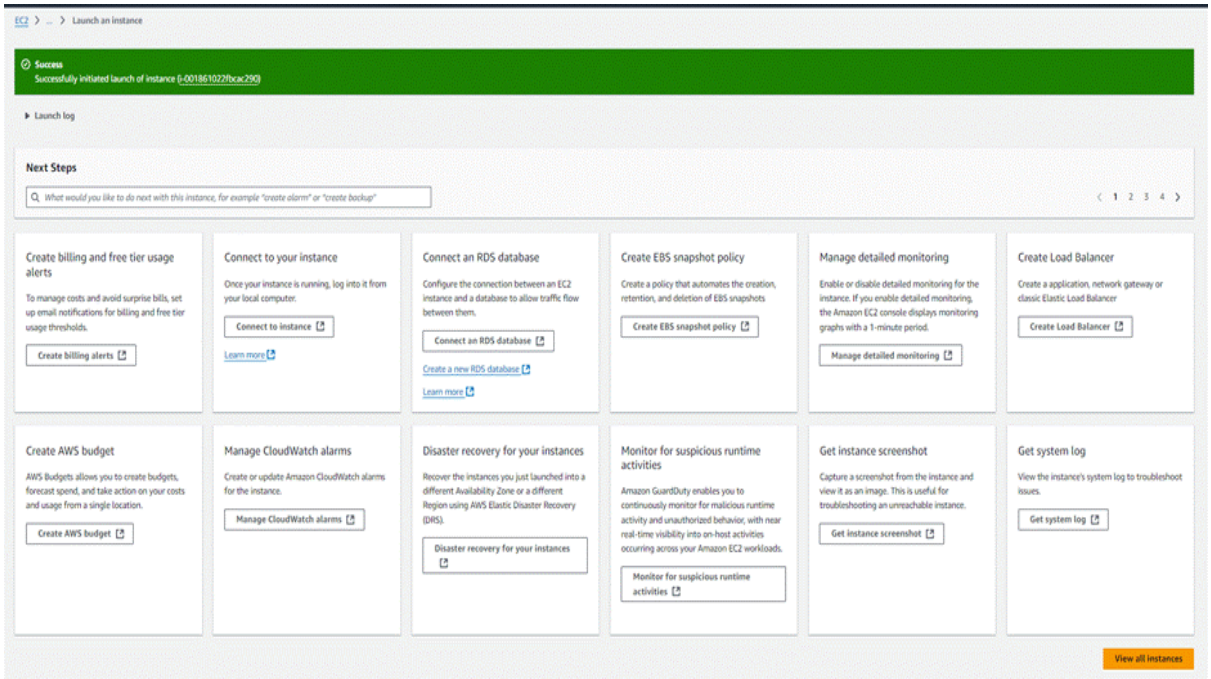


HomeMade.pem

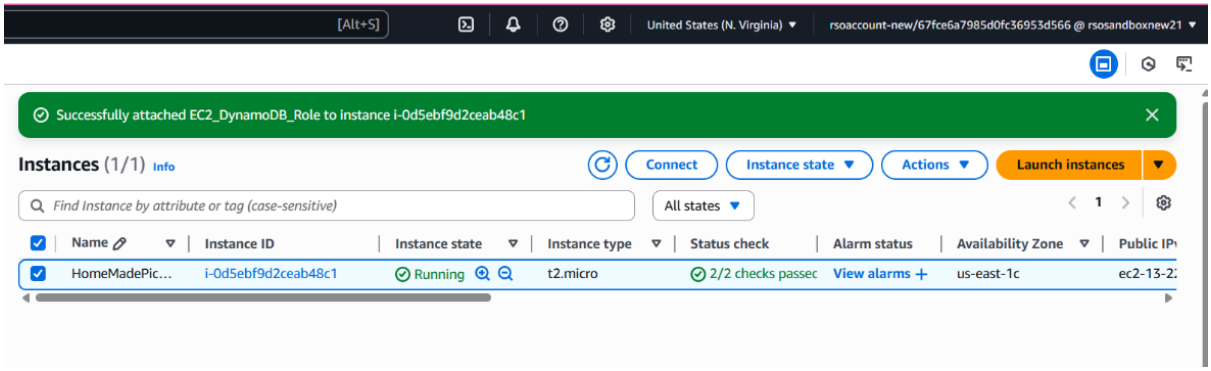
Configure security groups for HTTP, and SSH access.

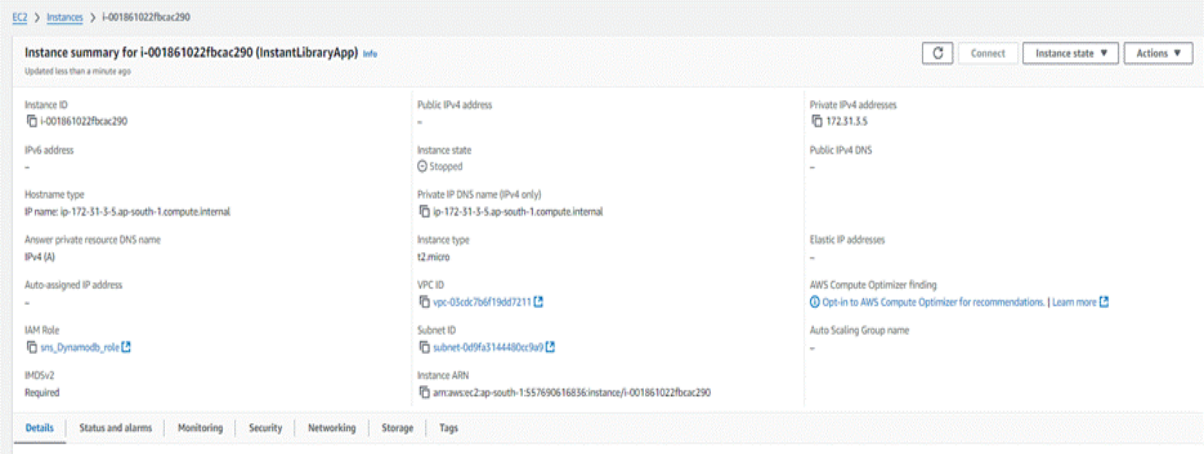




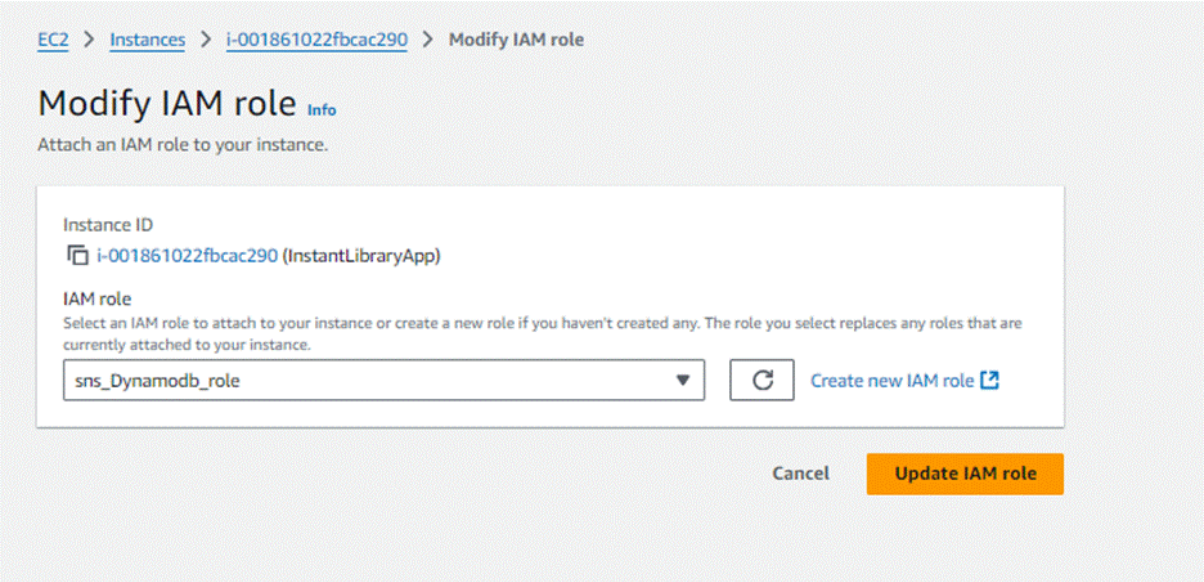


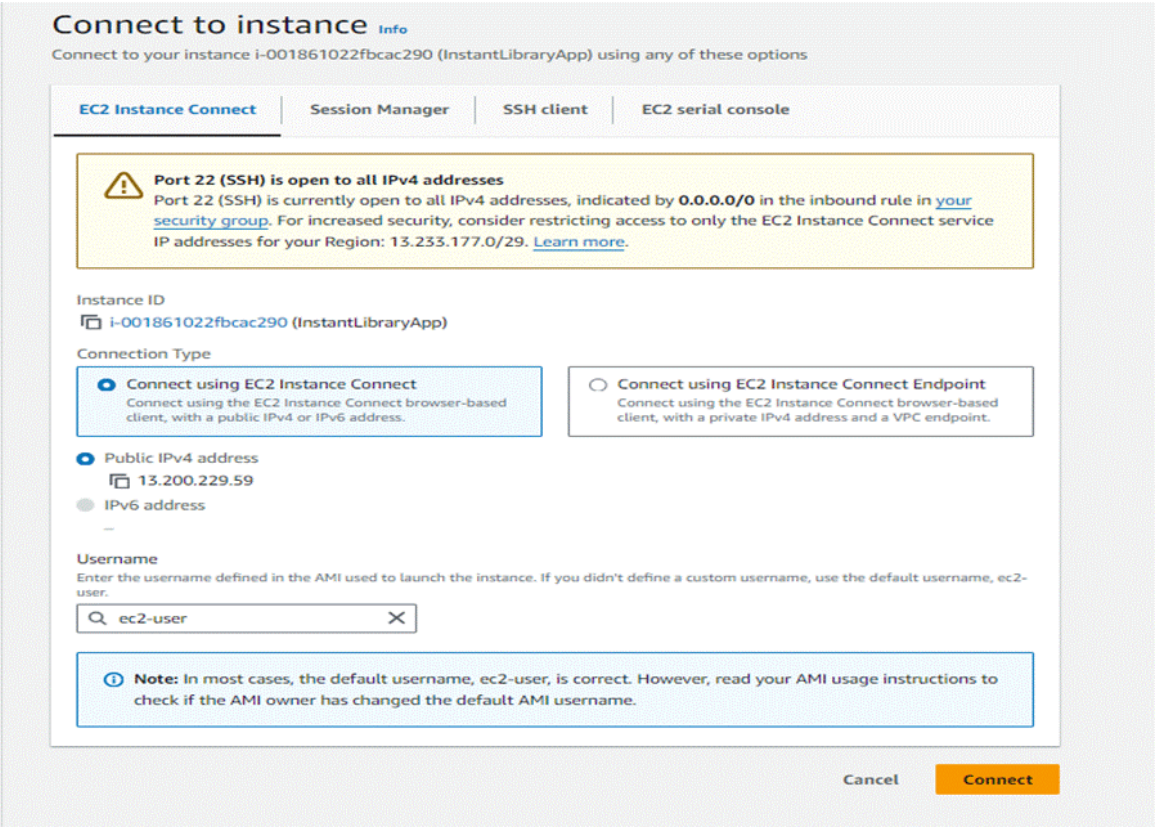
- To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected, 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.

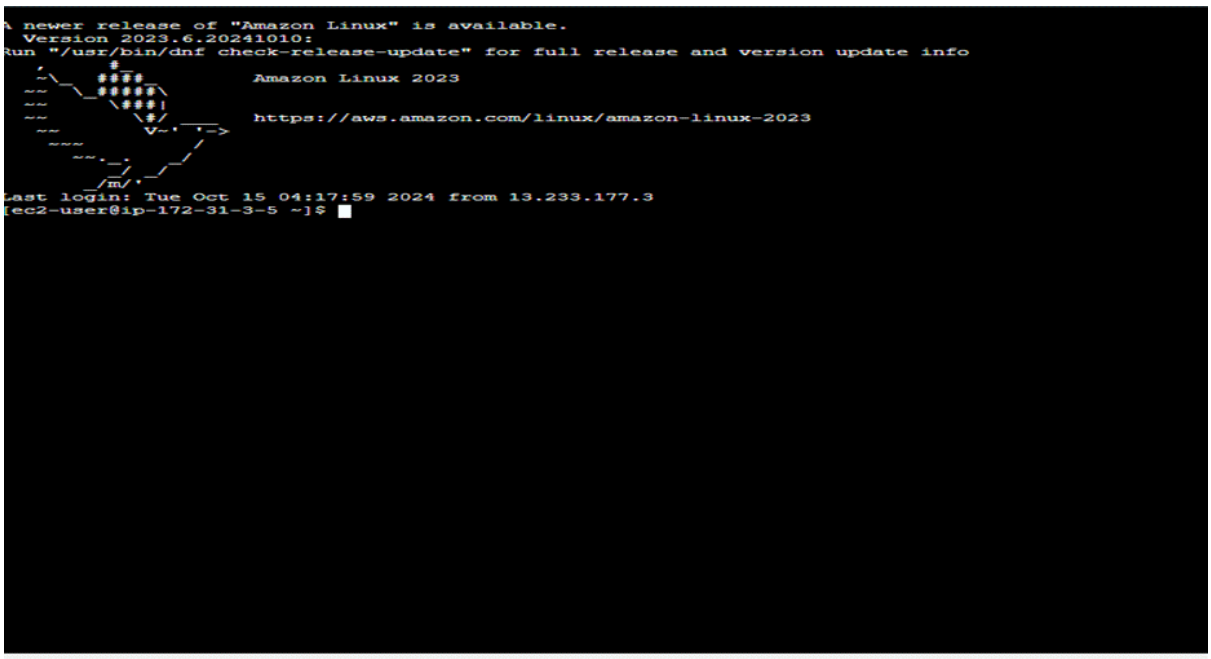



EC2 > Instances > i-001861022fbcac290
Instance summary for i-001861022fbcac290 (InstantLibraryApp) Info
Updated less than a minute ago
Connect
Instance state
Actions
Instance ID
i-001861022fbcac290
IPv6 address
–
Hostname type
IP name: ip-172-31-3-5.ap-south-1.compute.internal
Answer private resource DNS name
IPv4 (A)
Auto-assigned IP address
–
IAM Role
sns_Dynamodb_role
IMDSv2
Required
Public IPv4 address
–
Instance state
Stopped
Private IP DNS name (IPv4 only)
ip-172-31-3-5.ap-south-1.compute.internal
Instance type
t2.micro
VPC ID
vpc-03cdc7b6f19dd7211
Subnet ID
subnet-0d9fa3144480cc9a9
Instance ARN
arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290
Private IPv4 addresses
172.31.3.5
Public IPv4 DNS
–
Elastic IP addresses
–
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
Auto Scaling Group name
–
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags


EC2 > Instances > i-001861022fbcac290
Instance summary for i-001861022fbcac290 (InstantLibraryApp) Info
Updated less than a minute ago
Connect
Instance state
Actions
Connect
Manage instance state
Instance settings
Networking
Security
Image and templates
Monitor and troubleshoot
Change security groups
Get Windows password
Modify IAM role
Instance ID
i-001861022fbcac290
IPv6 address
–
Hostname type
IP name: ip-172-31-3-5.ap-south-1.compute.internal
Answer private resource DNS name
IPv4 (A)
Auto-assigned IP address
–
IAM Role
sns_Dynamodb_role
IMDSv2
Required
Public IPv4 address
–
Instance state
Stopped
Private IP DNS name (IPv4 only)
ip-172-31-3-5.ap-south-1.compute.internal
Instance type
t2.micro
VPC ID
vpc-03cdc7b6f19dd7211
Subnet ID
subnet-0d9fa3144480cc9a9
Instance ARN
arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290
Private IPv4 addresses
172.31.3.5
Public IPv4 DNS
–
Elastic IP addresses
–
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
Auto Scaling Group name
–


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.
sns_Dynamodb_role
Create new IAM role
Cancel
Update IAM role


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.
ec2-user
Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.
Cancel
Connect


A newer release of "Amazon Linux" is available.
Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$

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

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/sumanjali0907/Homemadepickles-snacks.git'

  Here: __’https://github.com/sumanjali0907/Homemadepickles-snacks.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

```
A newer release of "Amazon Linux" is available.
  Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$ git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git
fatal: destination path 'InstantLibrary' already exists and is not an empty directory.
[ec2-user@ip-172-31-3-5 ~]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ flask run --host=0.0.0.0 --port=80
 * Debug mode: off
Permission denied
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
^C[ec2-user@ip-172-31-3-5 InstantLibrary]$
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```

Verify the Flask app is running:

http://your-ec2-public-ip

- Run the Flask app on the EC2 instance

```
ModuleNotFoundError: No module named 'boto3'

[ec2-user@ip-172-31-15-149 Home made pickles1]$ sudo pip3 install boto3
Collecting boto3
  Downloading boto3-1.37.23-py3-none-any.whl (139 kB)
     |████████████████████████████████| 139 kB 8.1 MB/s
Collecting s3transfer<0.12.0,>=0.11.0
  Downloading s3transfer-0.11.4-py3-none-any.whl (84 kB)
     |████████████████████████████████| 84 kB 6.7 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Collecting botocore<1.38.0,>=1.37.23
  Downloading botocore-1.37.23-py3-none-any.whl (13.4 MB)
     |████████████████████████████████| 13.4 MB 38.3 MB/s
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.38.0,>=1.37.23->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.38.0,>=1.37.23->boto3) (2.8.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.38.0,>=1.37.23->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.37.23 botocore-1.37.23 s3transfer-0.11.4
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual
environment instead: https://pip.pypa.io/warnings/venv
[ec2-user@ip-172-31-15-149 Home made pickles1]$ sudo flask run --host=0.0.0.0 --port=5000
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.15.149:5000
Press CTRL+C to quit
```

Access the website through:

PublicIPs: http://127.0.0.1:5000

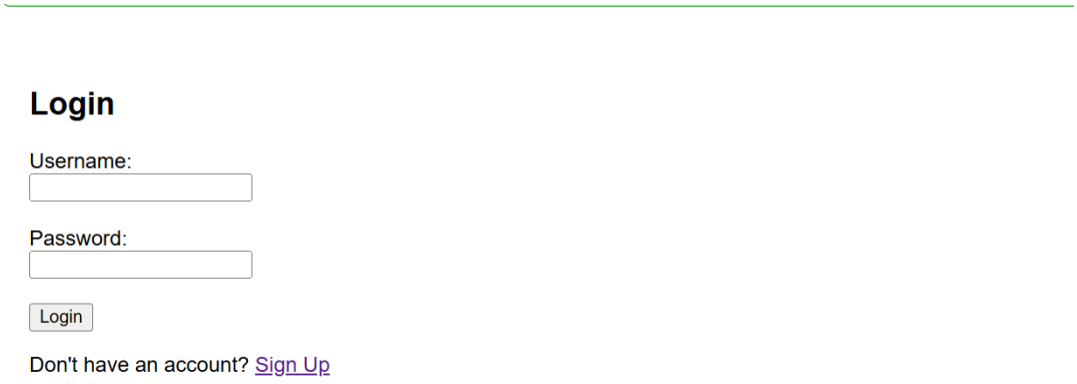
Milestone 7 : Testing and Deployment

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 :-

Signup :-

Login page:



Veg-pickles , non-veg pickles , snacks:

Our Pickles

Spicy Mango Pickle

Price: ₹150

Add to Cart

Lemon Pickle

Price: ₹120

Add to Cart

Tomato Pickle

Price: ₹130

Add to Cart

Chicken Pickle

Price: ₹250

Add to Cart

Our Snacks

Chakkaralu

Price: ₹80

Add to Cart

Chekkalu

Price: ₹90

Add to Cart

Murukulu

Price: ₹100

Add to Cart

Sunnunda

Price: ₹110

Add to Cart

Cart page:

Your Cart

- Chicken Pickle — ₹250

Total: ₹250

Checkout

Checkout page:

All fields are required.

Checkout

Full Name:

Shipping Address:

Email:

Order Summary:

- Chicken Pickle — ₹250

Total: ₹250

← Back to Products Place Order

Order success page:

Order Successful!

Thank you for your order,suma.

Order ID: ORDER1

Back to Products

Contact page:

Contact Us

Email: contact@homemadepickles.com

Phone: +91-9876543210

Address: Tenali, Andhra Pradesh, India

Review page:

Contact Us

Email: contact@homemadepickles.com

Phone: +91-9876543210

Address: Tenali, Andhra Pradesh, India

About page: “We chose to give the About page a different background to make it stand out, as it tells the unique story of our homemade pickles. This helps users clearly understand what makes our products special."

About Homemade Pickles and Snacks

Homemade Pickles and Snacks is a homegrown brand that brings you authentic Indian pickles made with traditional recipes and the finest ingredients. Our pickles are made in small batches to ensure quality and freshness.

Every jar is prepared with care and love using recipes handed down over generations in our family.

We use no artificial preservatives — just natural ingredients and traditional methods to create flavors that remind you of home and childhood.

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