ACHAAR HOUSE

Traditional Flavours , Delivered Fresh

Project Description:

ACHAAR HOUSE — 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 flavours 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."

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 time.

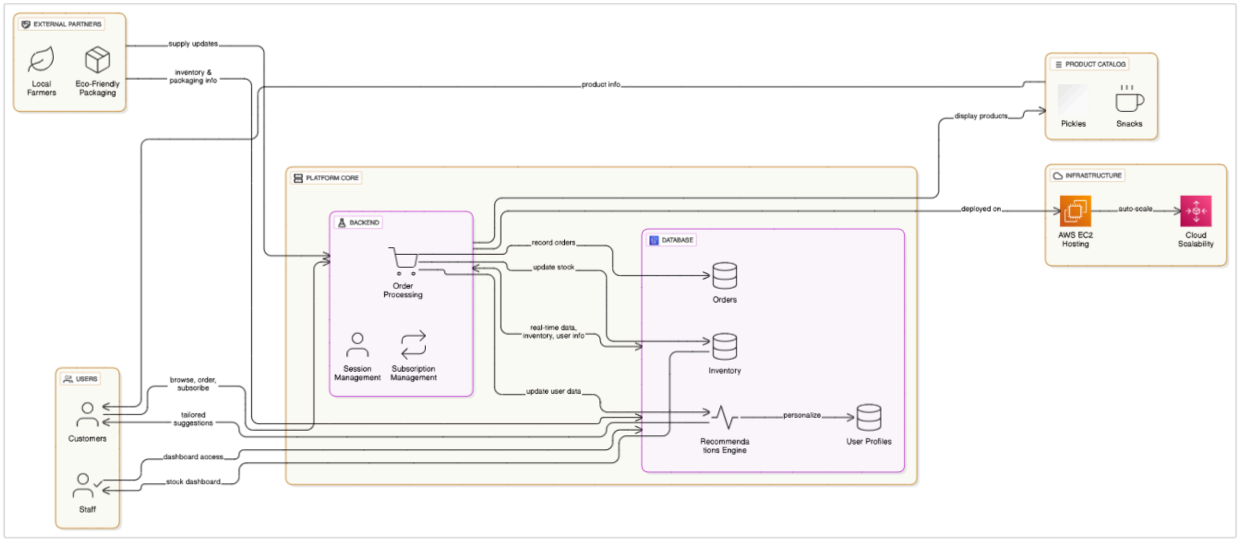
scenario 2: Real-Time Inventory Tracking and Updates

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 risk

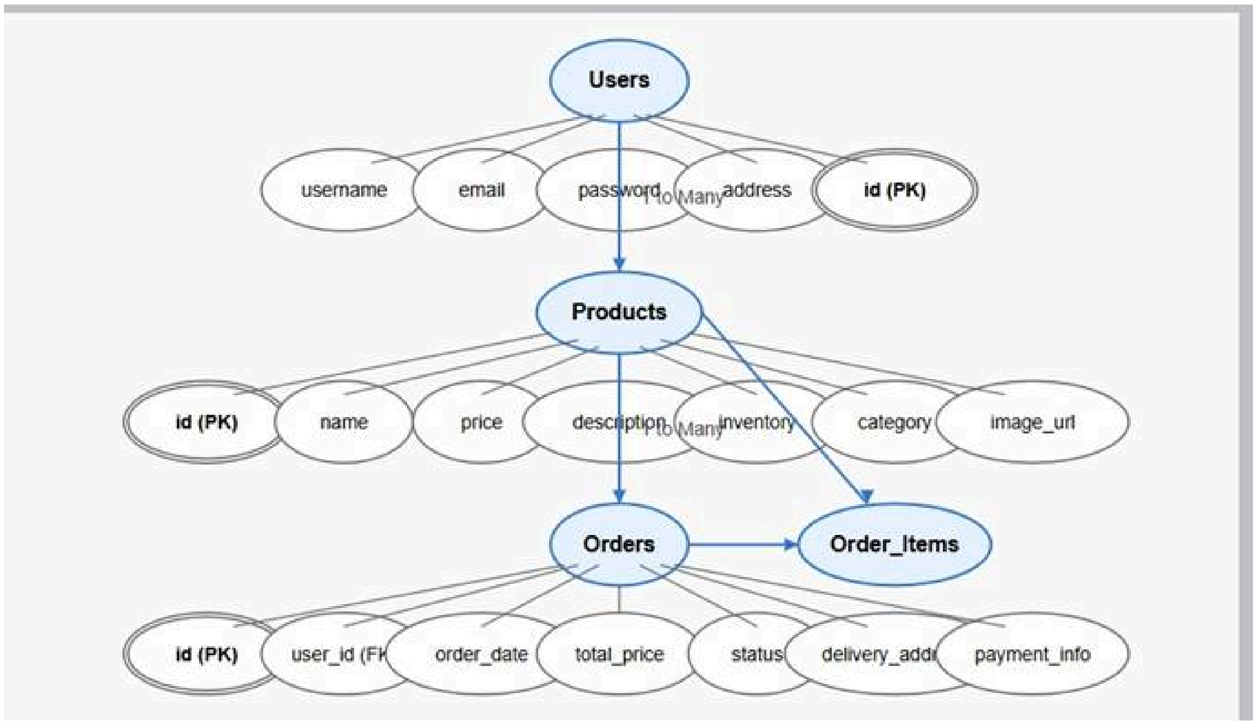
Scenario 3: Personalized User Experience and Recommendations

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

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Pre-requisites:

- **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.

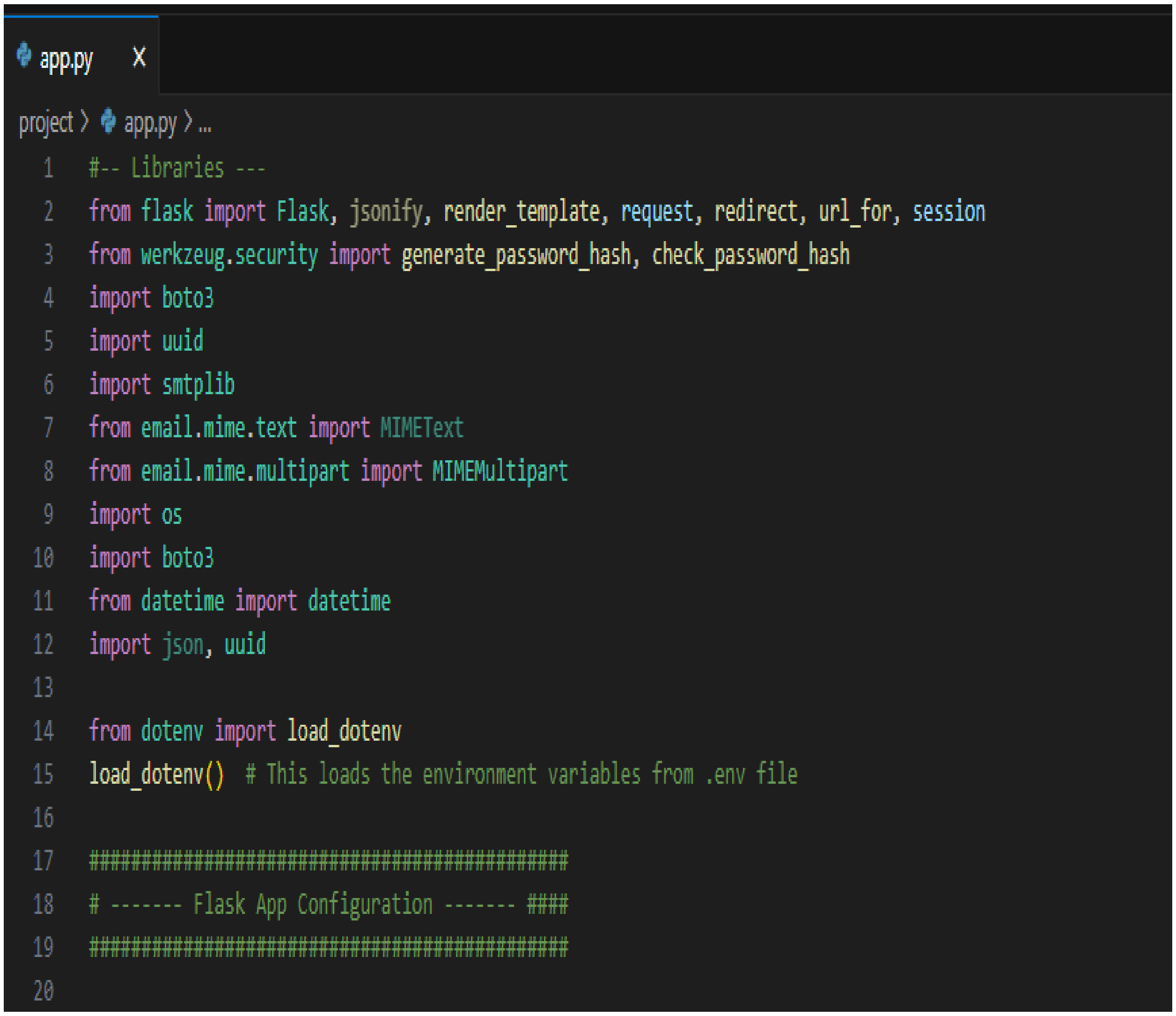
- At this point, modify your app.py and replace local dictionary/list operations with AWS services (such as DynamoDB, RDS, or others as per project requirements).

- Using the temporary credentials provided by Troven Labs, securely connect your application to AWS resources.

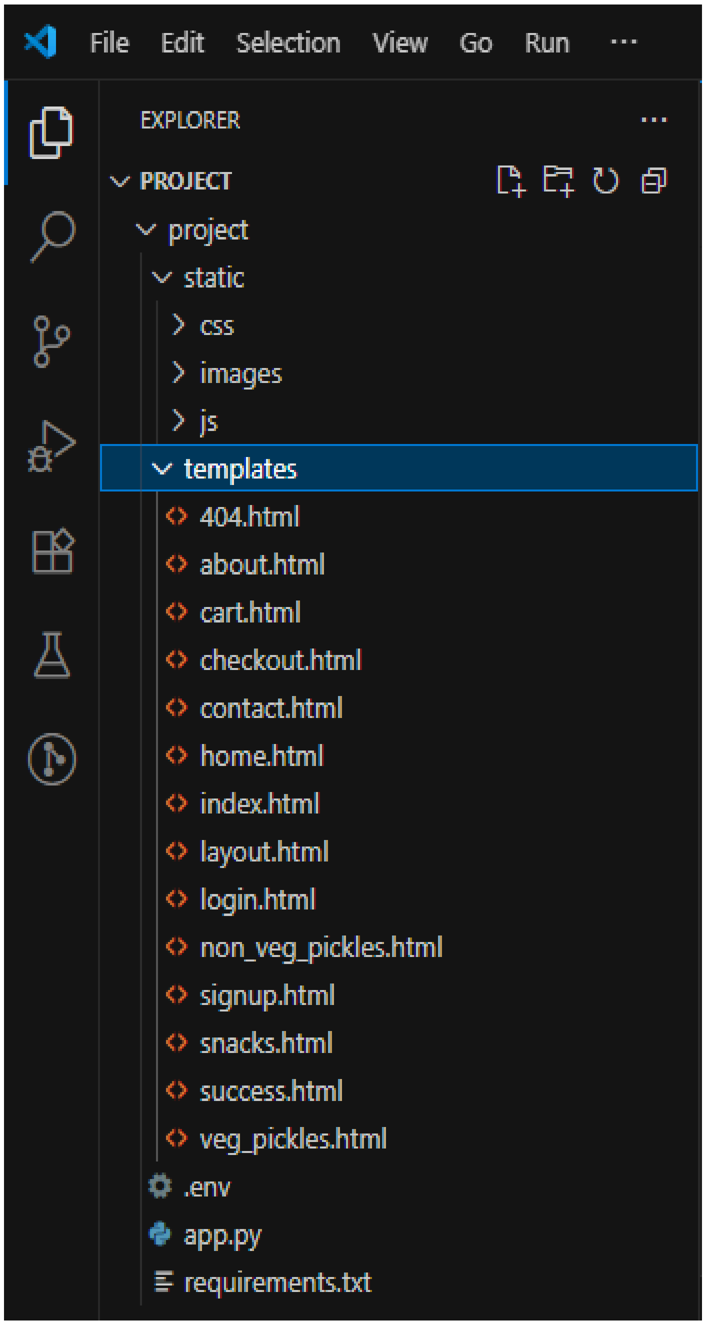
- Since the AWS configuration is lightweight and already instructed in the milestones, you should be able to complete the cloud integration efficiently within the allotted time.

LOCAL DEPLOYMENT

- Flask app initialisation and file explorer structure

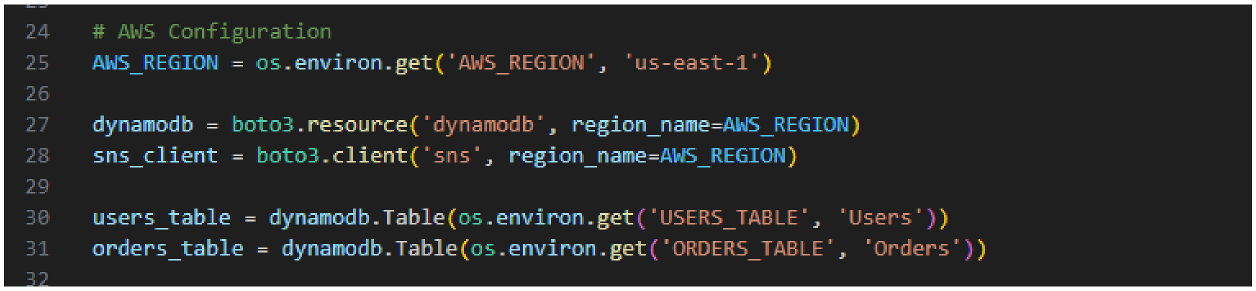


```python
#-- Libraries ---
from flask import Flask, jsonify, render_template, request, redirect, url_for, session
from werkzeug.security import generate_password_hash, check_password_hash
import boto3
import uuid
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import os
import boto3
from datetime import datetime
import json, uuid

from dotenv import load_dotenv
load_dotenv()  # This loads the environment variables from .env file

############################################
# ------- Flask App Configuration ------- ####
############################################

```

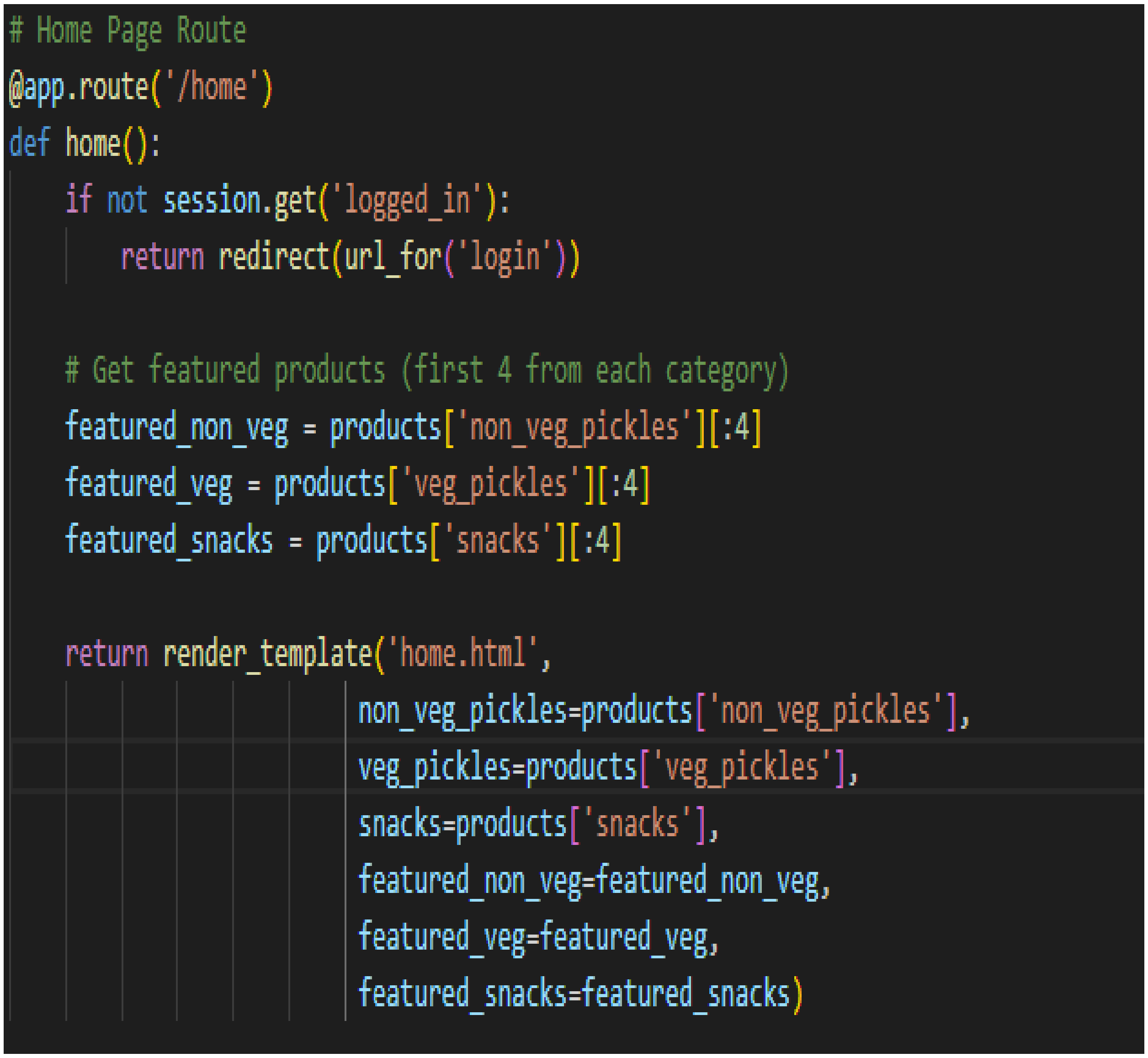


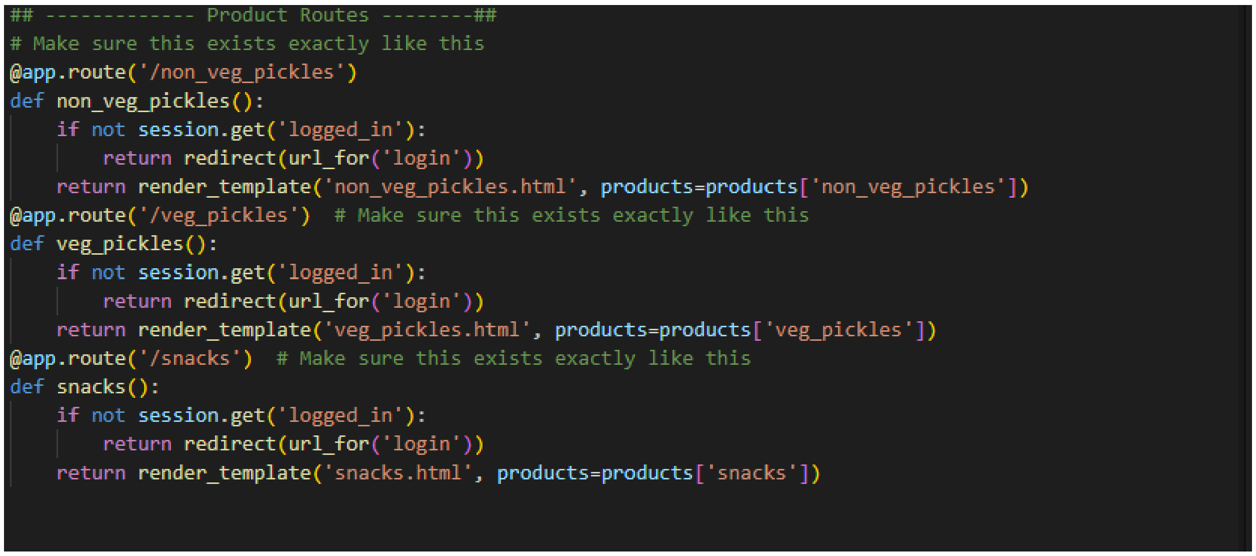
```
project
  static
    css
    images
    js
  templates
    404.html
    about.html
    cart.html
    checkout.html
    contact.html
    home.html
    index.html
    layout.html
    login.html
    non_veg_pickles.html
    signup.html
    snacks.html
    success.html
    veg_pickles.html
  .env
  app.py
  requirements.txt
```

use boto3 to connect to DynamoDB for handling user registration, order details databse operations and also mention region_name where DynamoDB tables are created


```python
# AWS Configuration
AWS_REGION = os.environ.get('AWS_REGION', 'us-east-1')

dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION)
sns_client = boto3.client('sns', region_name=AWS_REGION)

users_table = dynamodb.Table(os.environ.get('USERS_TABLE', 'Users'))
orders_table = dynamodb.Table(os.environ.get('ORDERS_TABLE', 'Orders'))
```

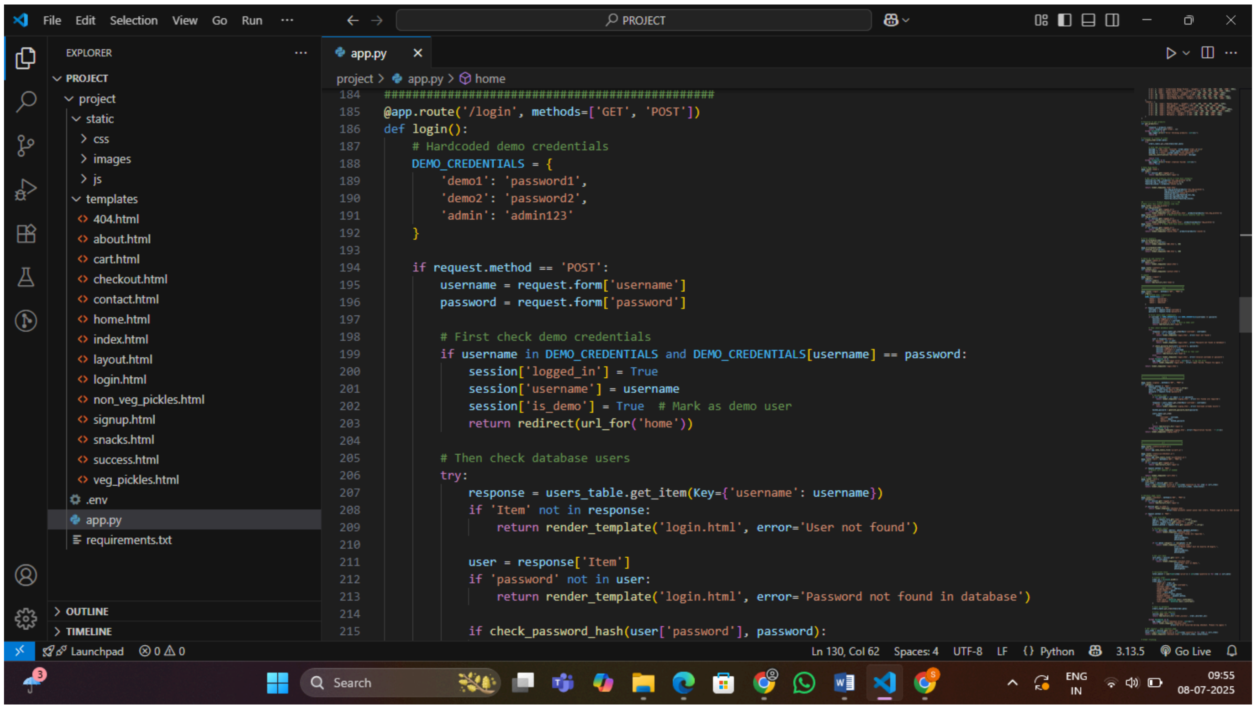
- Routes for Web Pages


```python
# Home Page Route
@app.route('/home')
def home():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    # Get featured products (first 4 from each category)
    featured_non_veg = products['non_veg_pickles'][:4]
    featured_veg = products['veg_pickles'][:4]
    featured_snacks = products['snacks'][:4]

    return render_template('home.html',
                          non_veg_pickles=products['non_veg_pickles'],
                          veg_pickles=products['veg_pickles'],
                          snacks=products['snacks'],
                          featured_non_veg=featured_non_veg,
                          featured_veg=featured_veg,
                          featured_snacks=featured_snacks)
```



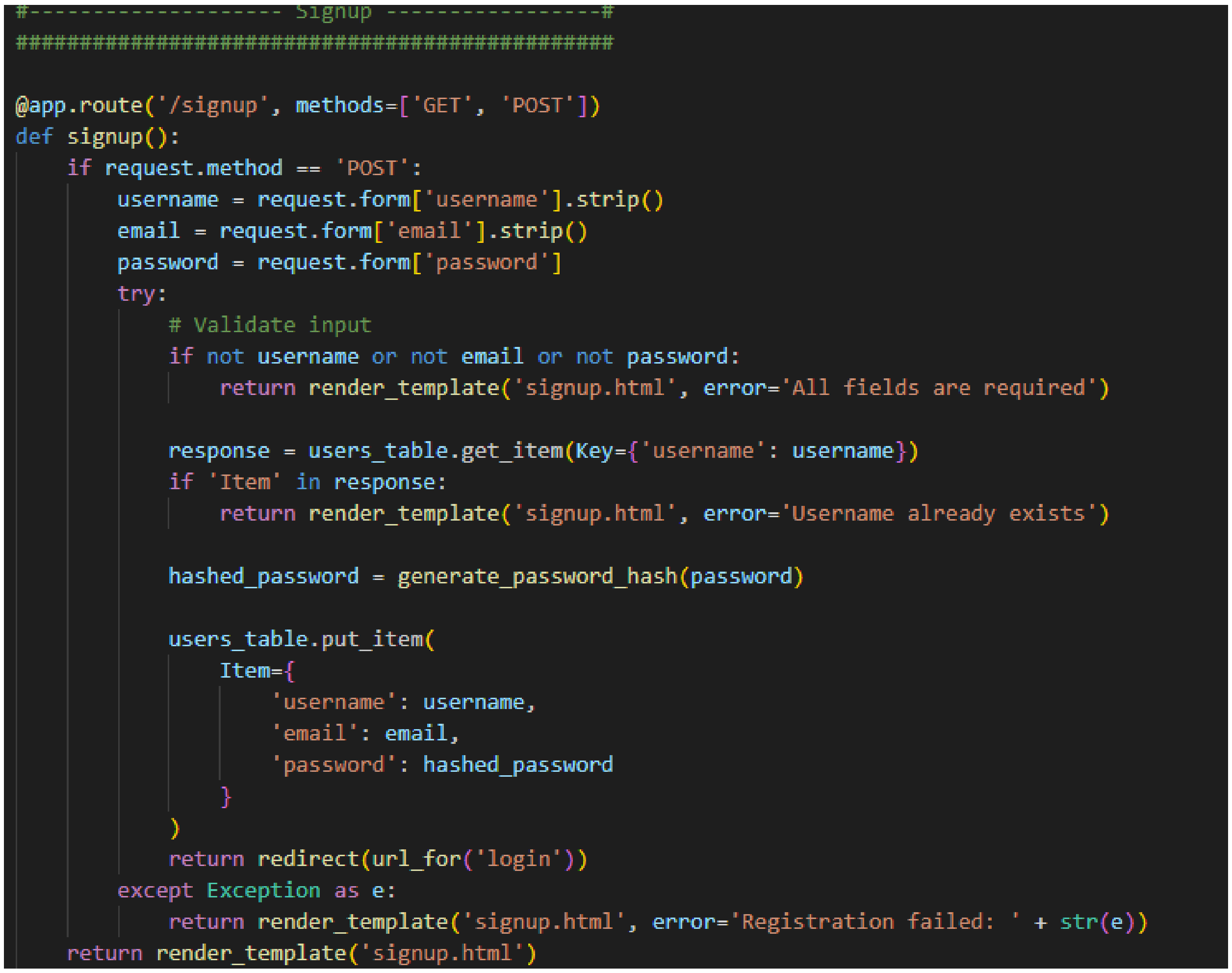
```python
## ------------- Product Routes --------##
# Make sure this exists exactly like this
@app.route('/non_veg_pickles')
def non_veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))
    return render_template('non_veg_pickles.html', products=products['non_veg_pickles'])
@app.route('/veg_pickles')  # Make sure this exists exactly like this
def veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))
    return render_template('veg_pickles.html', products=products['veg_pickles'])
@app.route('/snacks')  # Make sure this exists exactly like this
def snacks():
    if not session.get('logged_in'):
        return redirect(url_for('login'))
    return render_template('snacks.html', products=products['snacks'])
```

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



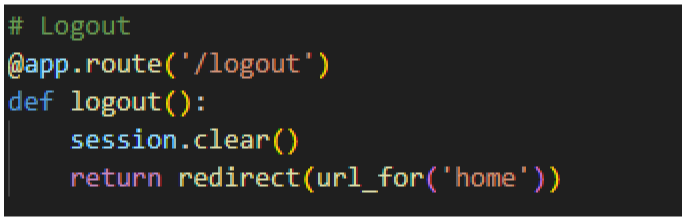
```python
##################################################
@app.route('/login', methods=['GET', 'POST'])
def login():
    # Hardcoded demo credentials
    DEMO_CREDENTIALS = {
        'demo1': 'password1',
        'demo2': 'password2',
        'admin': 'admin123'
    }

    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # First check demo credentials
        if username in DEMO_CREDENTIALS and DEMO_CREDENTIALS[username] == password:
            session['logged_in'] = True
            session['username'] = username
            session['is_demo'] = True  # Mark as demo user
            return redirect(url_for('home'))

        # Then check database users
        try:
            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']
            if 'password' not in user:
                return render_template('login.html', error='Password not found in database')

            if check_password_hash(user['password'], password):
```

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



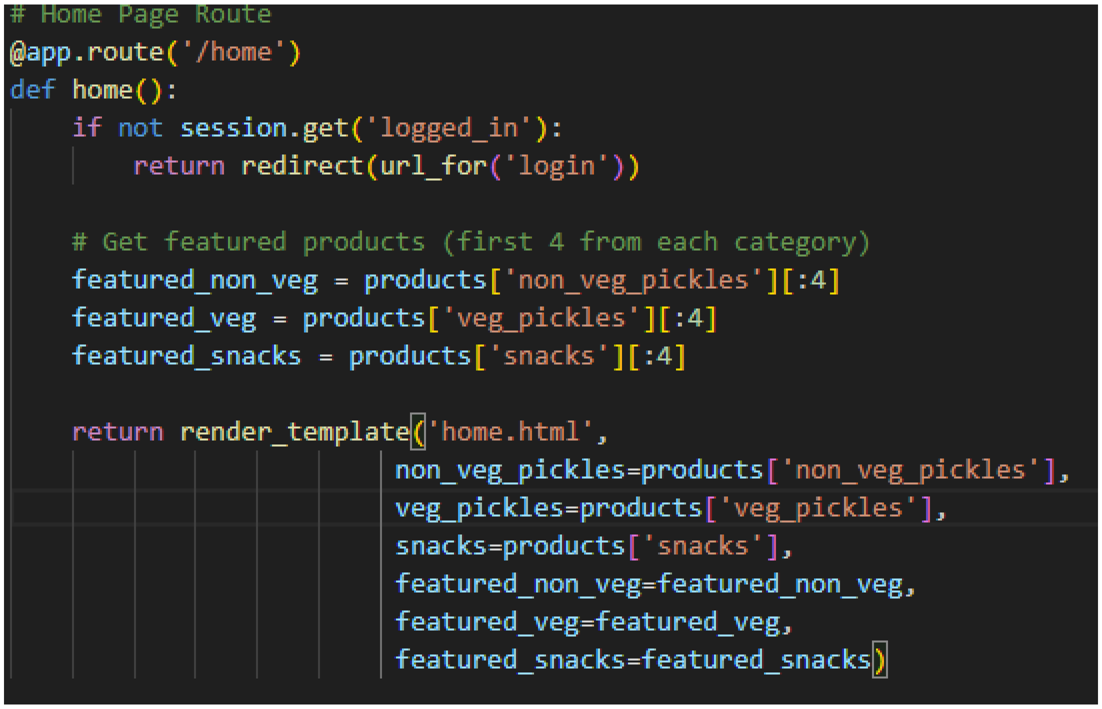
```
#------------------- Signup ----------------#
###############################################

@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:
            # Validate input
            if not username or not email or not password:
                return render_template('signup.html', error='All fields are required')

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

            hashed_password = generate_password_hash(password)

            users_table.put_item(
                Item={
                    'username': username,
                    'email': email,
                    'password': hashed_password
                }
            )
            return redirect(url_for('login'))
        except Exception as e:
            return render_template('signup.html', error='Registration failed: ' + str(e))
    return render_template('signup.html')
```

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



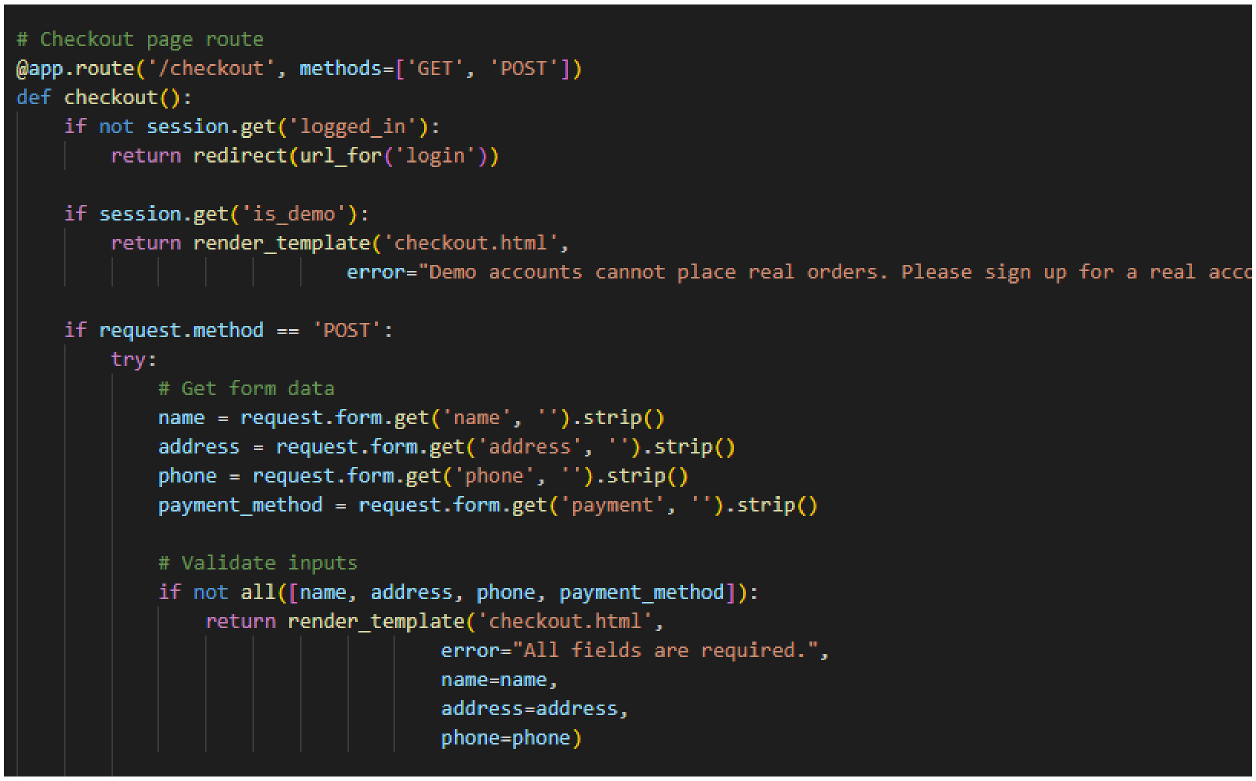
```
# Logout
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('home'))
```

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



```
# Home Page Route
@app.route('/home')
def home():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    # Get featured products (first 4 from each category)
    featured_non_veg = products['non_veg_pickles'][:4]
    featured_veg = products['veg_pickles'][:4]
    featured_snacks = products['snacks'][:4]

    return render_template('home.html',
                          non_veg_pickles=products['non_veg_pickles'],
                          veg_pickles=products['veg_pickles'],
                          snacks=products['snacks'],
                          featured_non_veg=featured_non_veg,
                          featured_veg=featured_veg,
                          featured_snacks=featured_snacks)
```

Check out Route:



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

    if session.get('is_demo'):
        return render_template('checkout.html',
                              error="Demo accounts cannot place real orders. Please sign up for a real acco

    if request.method == 'POST':
        try:
            # Get form data
            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.",
                                      name=name,
                                      address=address,
                                      phone=phone)
```

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 -

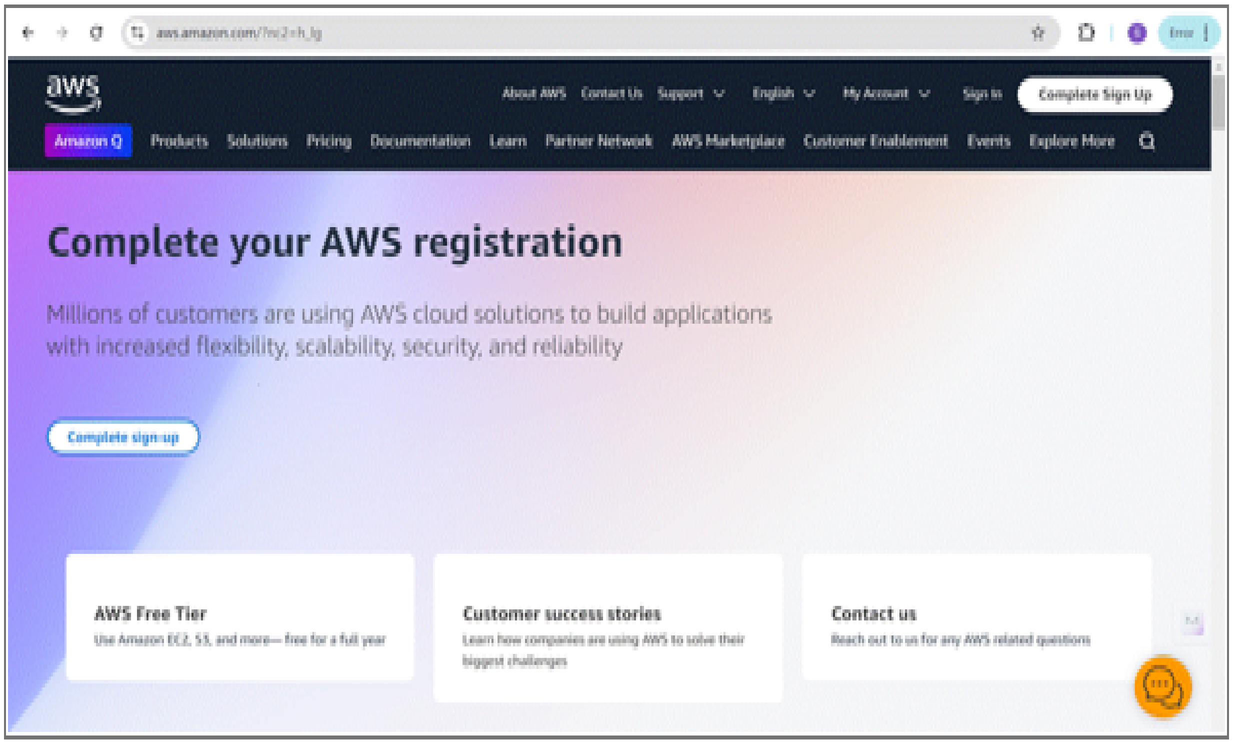
https://drive.google.com/file/d/1HzWc7AMJ2BrxhV-uaw5s0vWtcd-28qgI/view?usp=sharing

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 accounts.



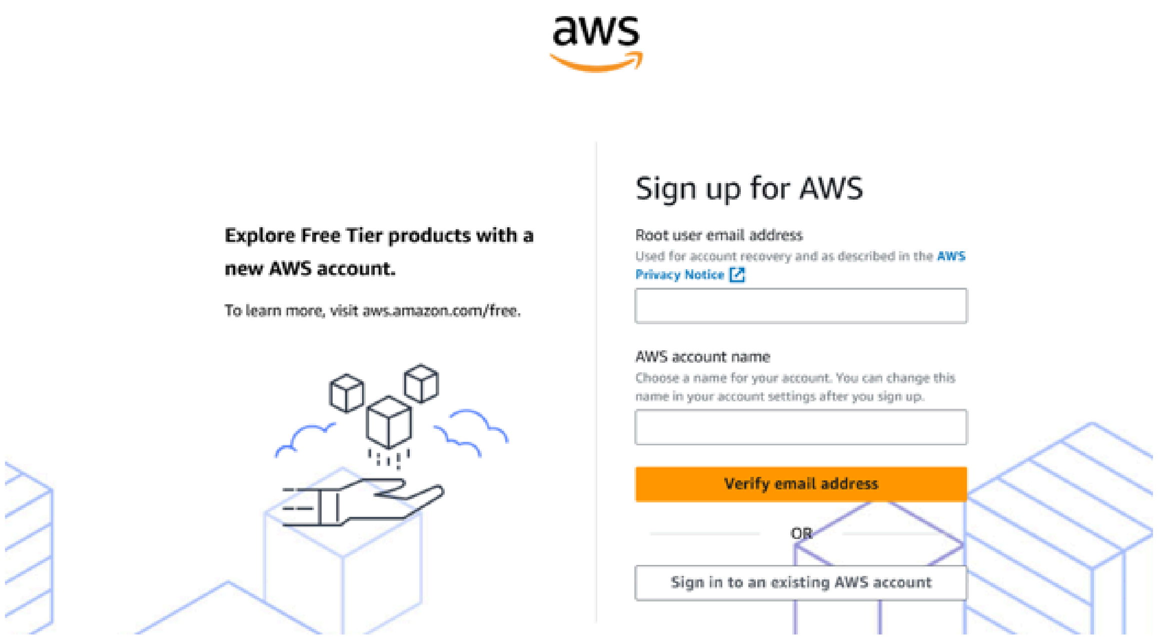
Complete your AWS registration

Millions of customers are using AWS cloud solutions to build applications with increased flexibility, scalability, security, and reliability

AWS Free Tier
Use Amazon EC2, S3, and more— free for a full year

Customer success stories
Learn how companies are using AWS to solve their biggest challenges

Contact us
Reach out to us for any AWS related questions


Explore Free Tier products with a new AWS account.

To learn more, visit aws.amazon.com/free.

Sign up for AWS

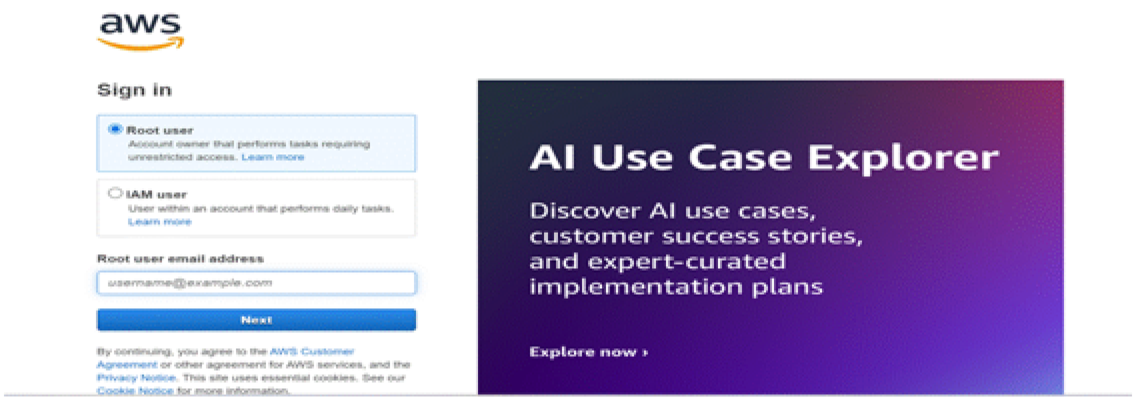
Root user email address

Used for account recovery and as described in the AWS Privacy Notice

AWS account name

Choose a name for your account. You can change this name in your account settings after you sign up.

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


Sign in

[✓ Root user] Account owner that performs tasks requiring unrestricted access. Learn more

IAM user User within an account that performs daily tasks. Learn more

Root user email address

username@example.com

By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our Cookie Notice for more information.

AI Use Case Explorer

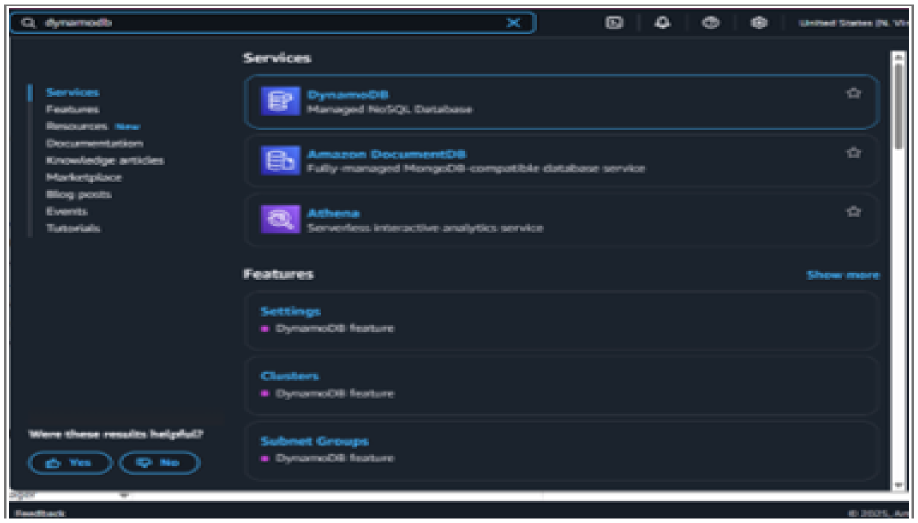
Discover AI use cases, customer success stories, and expert-curated implementation plans

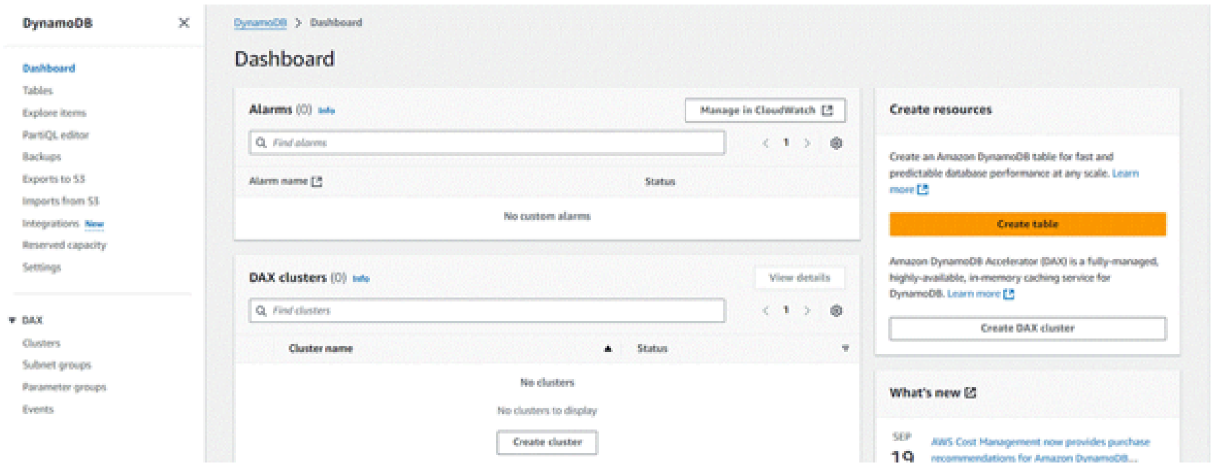
Milestone 3 : DynamoDB Database Creation and Setup

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.

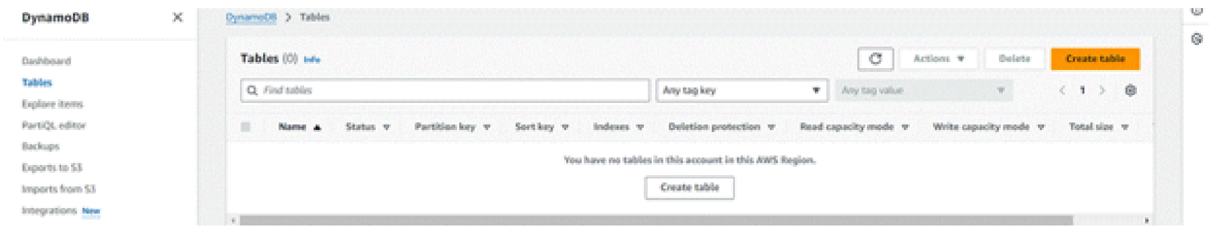




Create an Amazon DynamoDB table for fast and predictable database performance at any scale. Learn more

Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. Learn more

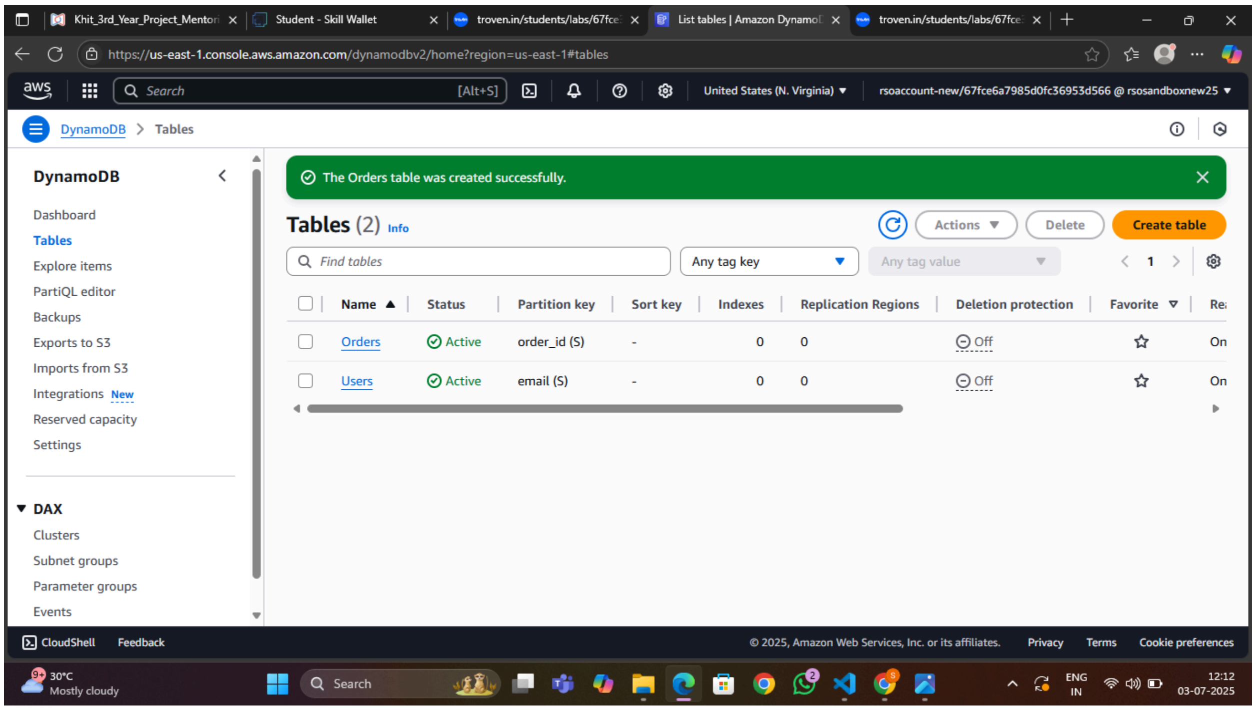
AWS Cost Management now provides purchase recommendations for Amazon DynamoDB...



You have no tables in this account in this AWS Region.

Create a DynamoDB table for storing data

- Create Users tableand orders table with partition keys “email” and "order_id" with type String and click on create tables.



https://us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables

United States (N. Virginia) ▼ rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew25 ▼

DynamoDB > Tables

The Orders table was created successfully.

Tables (2) Info

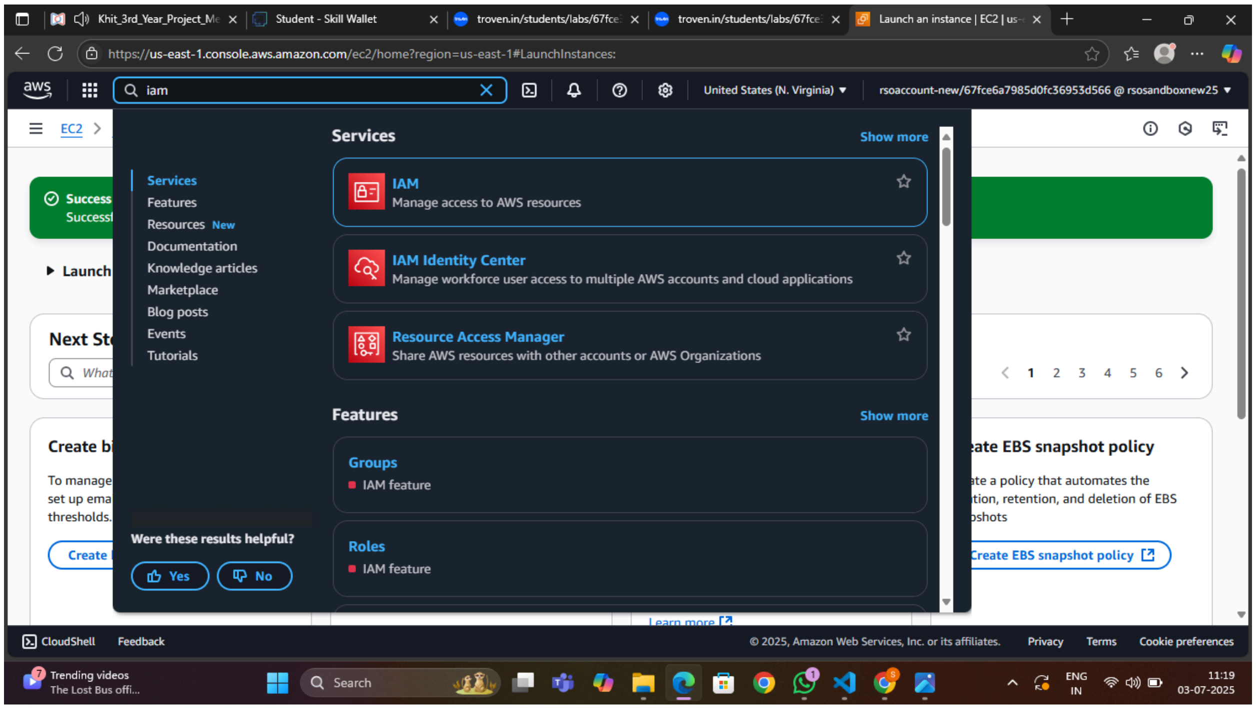
	Name ▲	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Re
☐	Orders	✓ Active	order_id (S)	-	0	0	Off	☆	On
☐	Users	✓ Active	email (S)	-	0	0	Off	☆	On

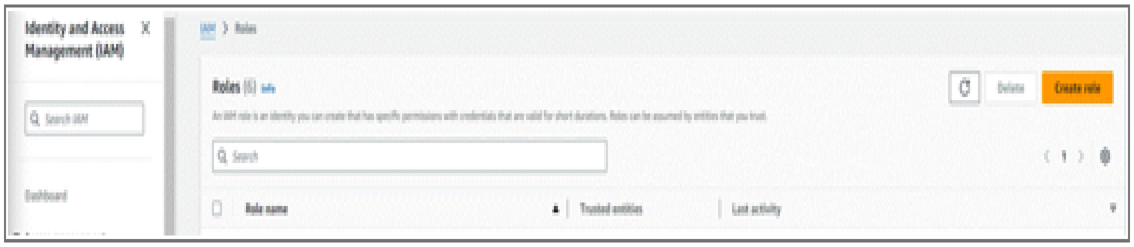
Milestone 4 : IAM Role Setup

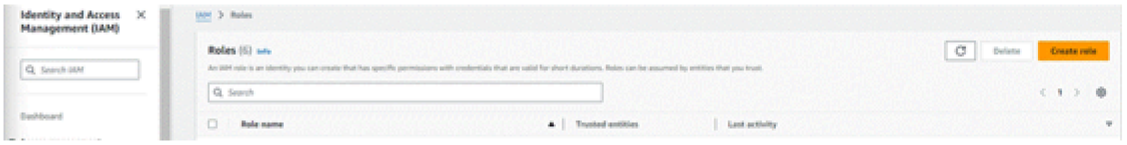
IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows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


iam
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Were these results helpful?
Yes
No
Services
Show more
IAM
Manage access to AWS resources
IAM Identity Center
Manage workforce user access to multiple AWS accounts and cloud applications
Resource Access Manager
Share AWS resources with other accounts or AWS Organizations
Features
Show more
Groups
IAM feature
Roles
IAM feature
United States (N. Virginia)
EC2
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


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



Identity and Access Management (IAM)

Search IAM

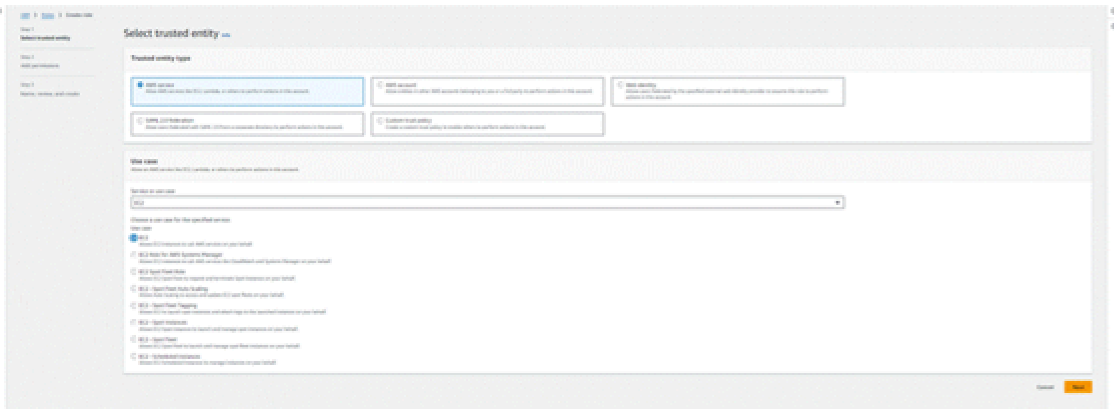
Dashboard

IAM > Roles

Roles (6) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity



IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity Info

Trusted entity type

- [✓ AWS service]
- AWS account
- Web identity
- SAML 2.0 federation
- Custom trust policy

Use case

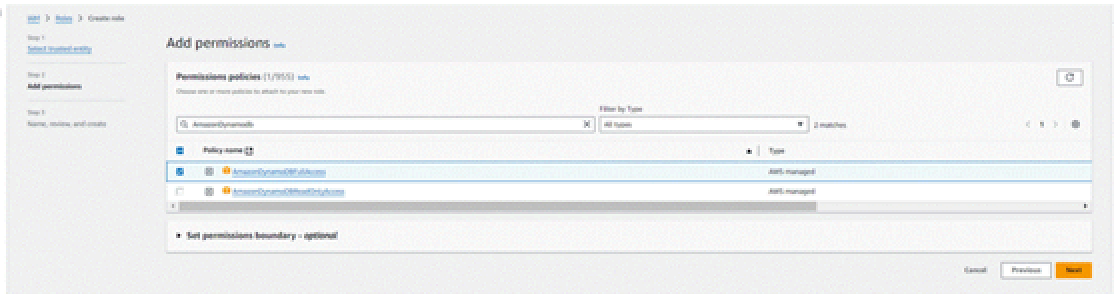
Service or use case: EC2

Choose a use case for the specified service.

Use case

- [✓ EC2]
- EC2 Role for AWS Systems Manager
- EC2 Spot Fleet Role
- EC2 - Spot Fleet Auto Scaling
- EC2 - Spot Fleet Tagging
- EC2 - Spot Instances
- EC2 - Spot Fleet
- EC2 - Scheduled Instances

Cancel Next



IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions Info

Permissions policies (1/955) Info

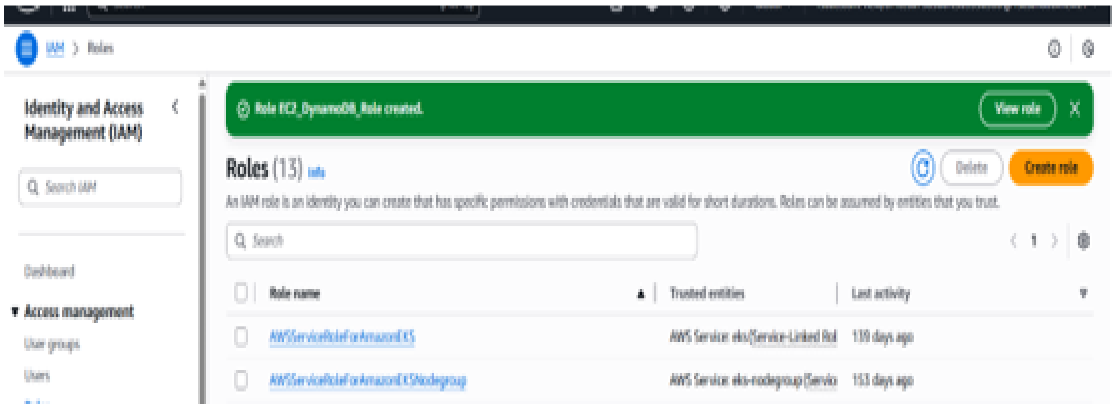
Choose one or more policies to attach to your new role.

Search: AmazonDynamodb — Filter by Type: All types — 2 matches

	Policy name	Type
✓	AmazonDynamoDBFullAccess	AWS managed
	AmazonDynamoDBReadOnlyAccess	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

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.

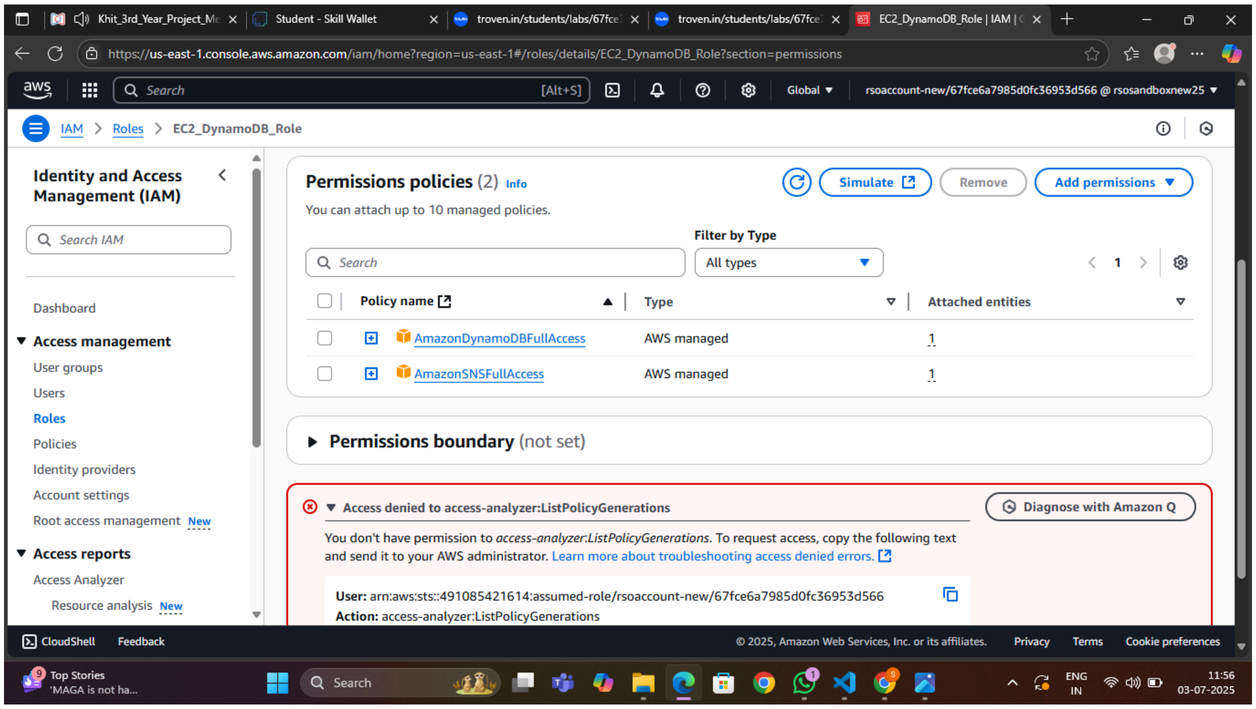
Delete Create role

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.



https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/details/EC2_DynamoDB_Role?section=permissions

IAM > Roles > EC2_DynamoDB_Role

Permissions policies (2)

You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	1
AmazonSNSFullAccess	AWS managed	1

Permissions boundary (not set)

Access denied to access-analyzer:ListPolicyGenerations

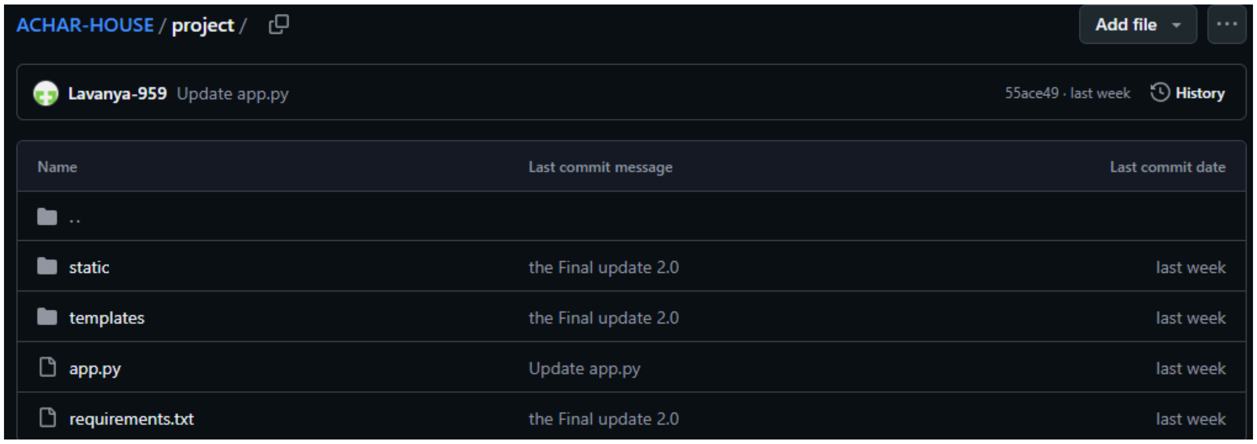
You don't have permission to *access-analyzer:ListPolicyGenerations*. To request access, copy the following text and send it to your AWS administrator. Learn more about troubleshooting access denied errors.

User: arn:aws:sts::491085421614:assumed-role/rsoaccount-new/67fce6a7985d0fc36953d566
Action: access-analyzer:ListPolicyGenerations

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.

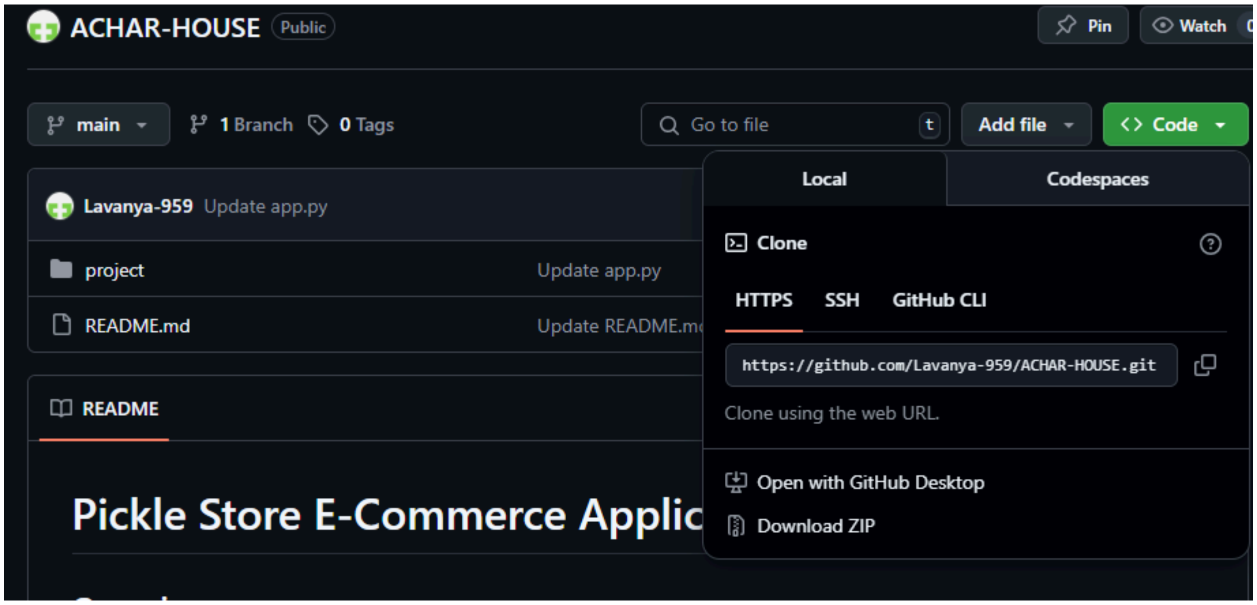
- Note: Load your Flask app and Html files into GitHub repository.


ACHAR-HOUSE / project /

Add file

Lavanya-959 Update app.py 55ace49 · last week History

Name	Last commit message	Last commit date
..		
static	the Final update 2.0	last week
templates	the Final update 2.0	last week
app.py	Update app.py	last week
requirements.txt	the Final update 2.0	last week


ACHAR-HOUSE Public

Pin Watch

main 1 Branch 0 Tags Go to file Add file Code

Lavanya-959 Update app.py

project	Update app.py
README.md	Update README.m

README

Pickle Store E-Commerce Applic

Local Codespaces

Clone

HTTPS SSH GitHub CLI

```
https://github.com/Lavanya-959/ACHAR-HOUSE.git
```

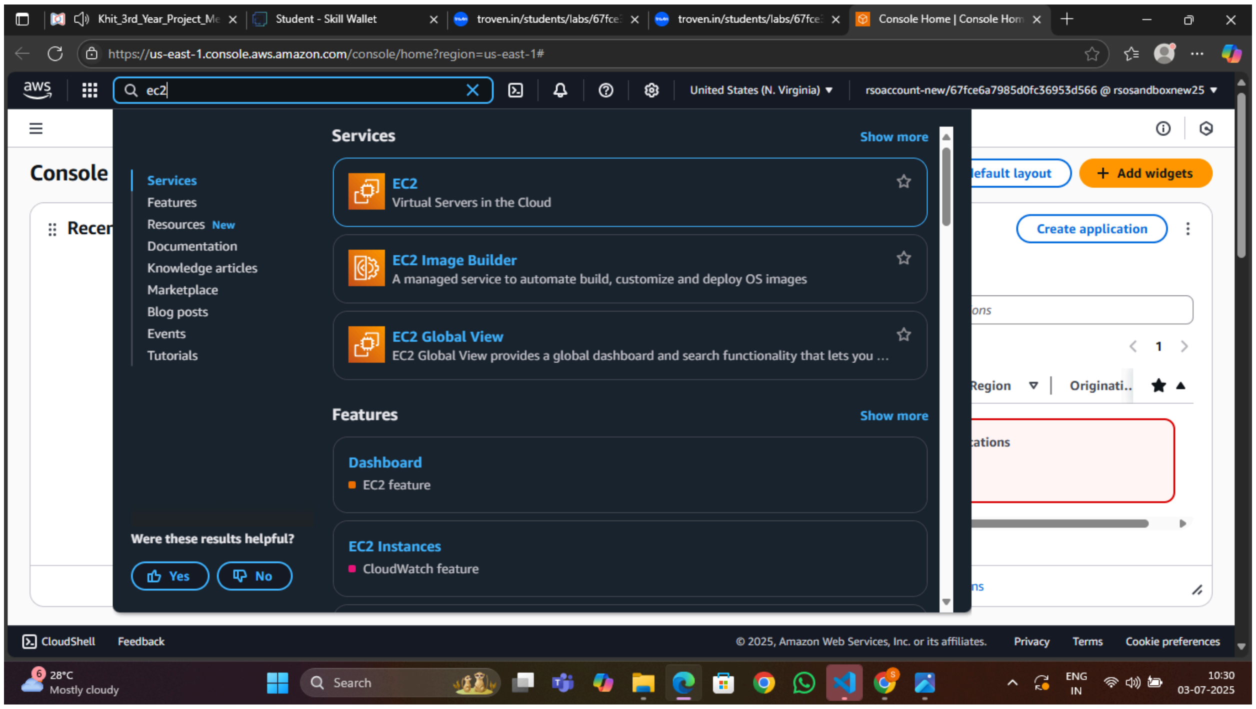
Clone using the web URL.

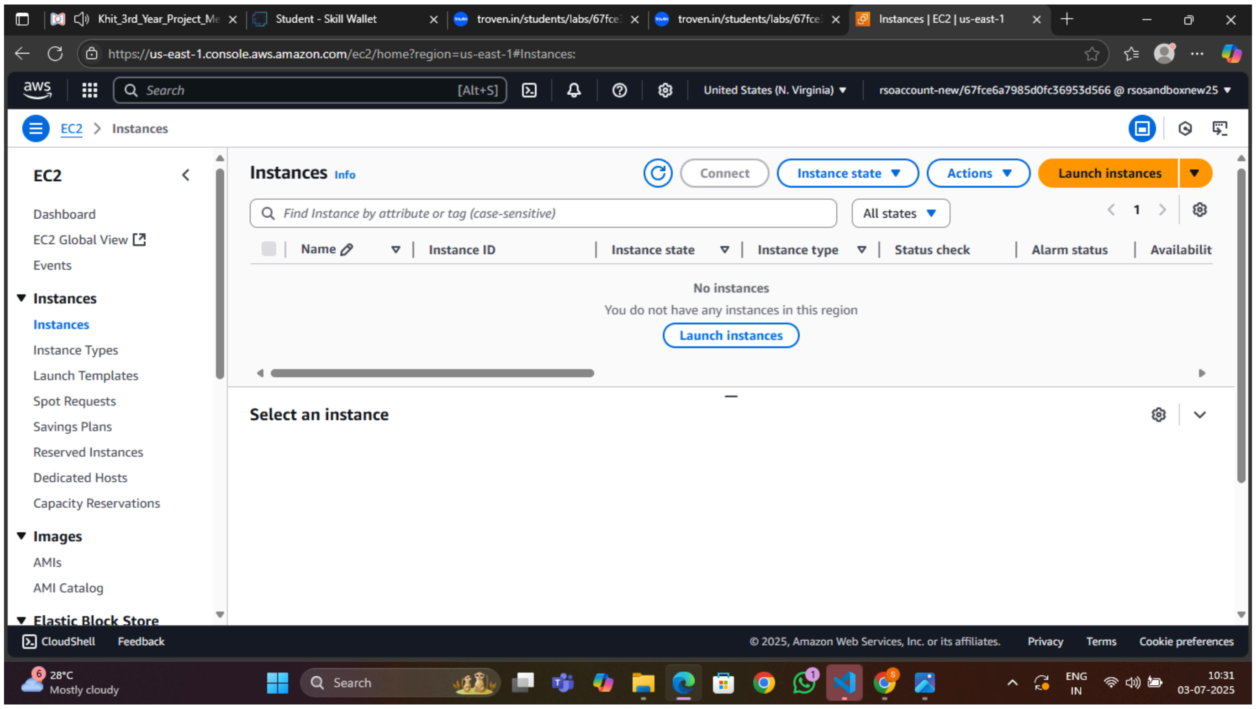
Open with GitHub Desktop

Download ZIP

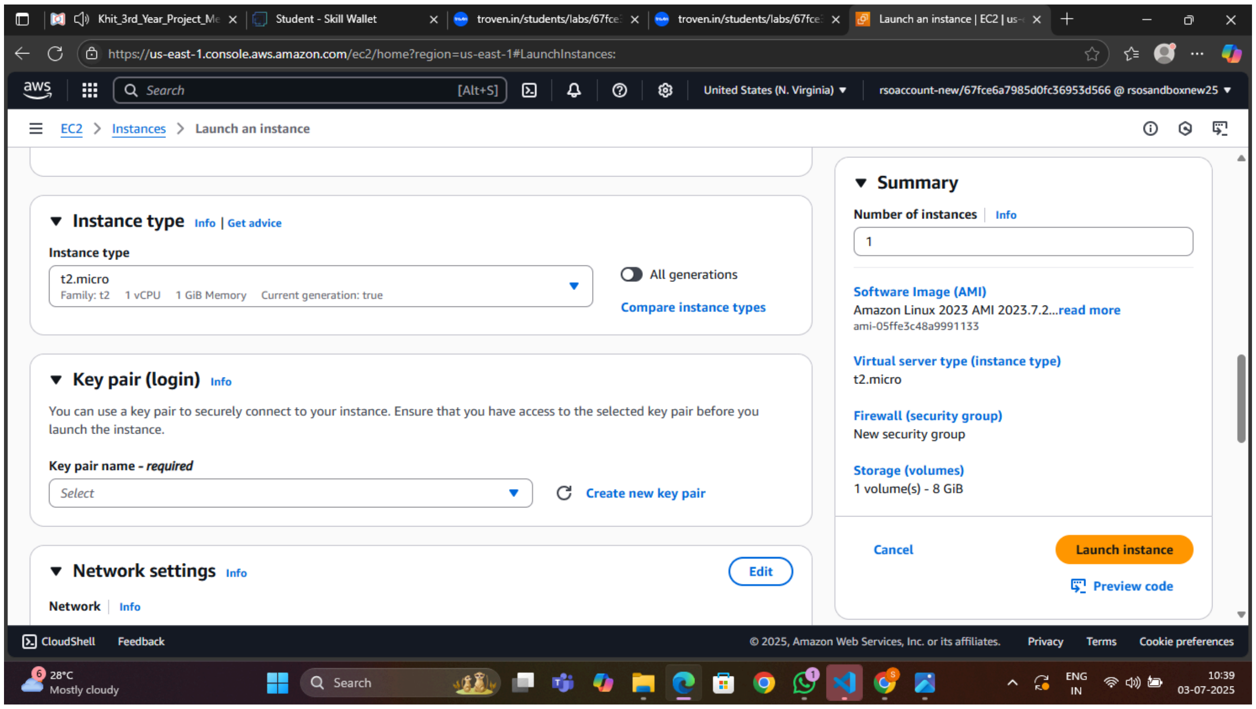
Launch an EC2 instance to host the Flask

- Launch EC2 Instance
- In the AWS Console, navigate to EC2
- After navigating into EC2 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).


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

EC2 > Instances > Launch an instance

Instance type Info | Get advice

Instance type

t2.micro
Family: t2 1 vCPU 1 GiB Memory Current generation: true

All generations

Compare instance types

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

Network settings Info

Edit

Network Info

Summary

Number of instances Info

1

Software Image (AMI)
Amazon Linux 2023 AMI 2023.7.2...read more
ami-05ffe3c48a9991133

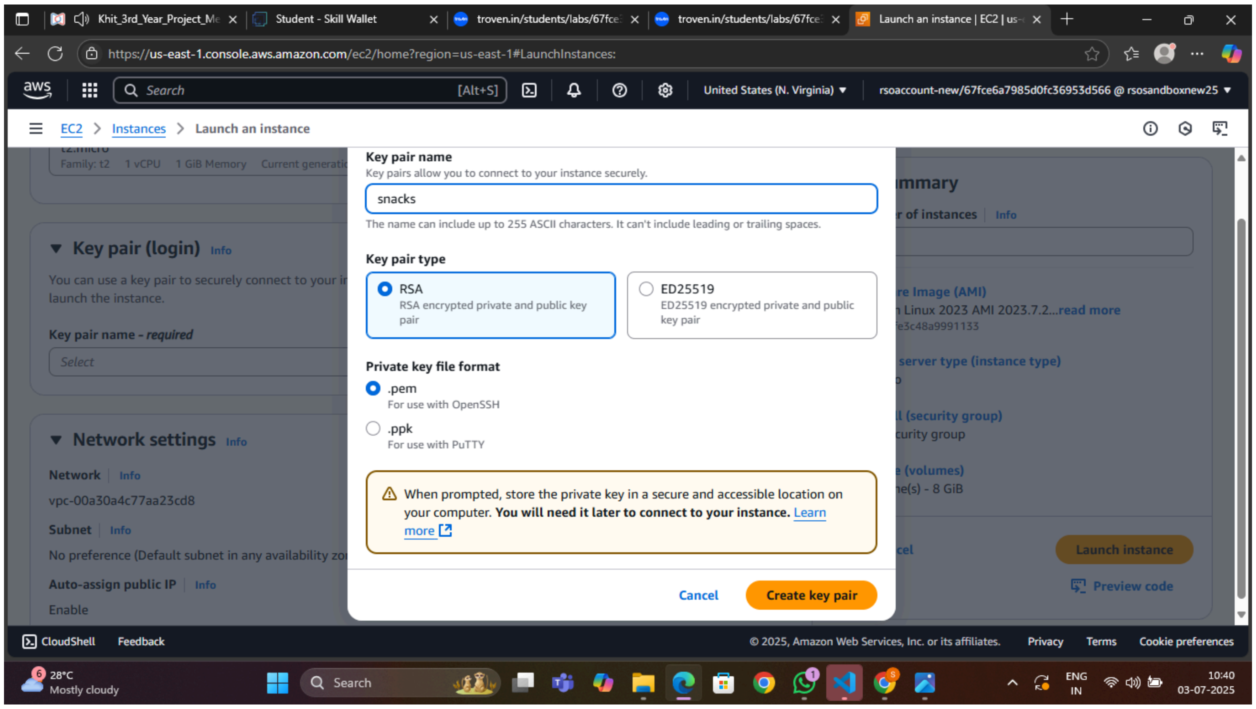
Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

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

Cancel Launch instance

Preview code


Key pair name
Key pairs allow you to connect to your instance securely.

snacks

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

● RSA
RSA encrypted private and public key pair

○ ED25519
ED25519 encrypted private and public key pair

Private key file format

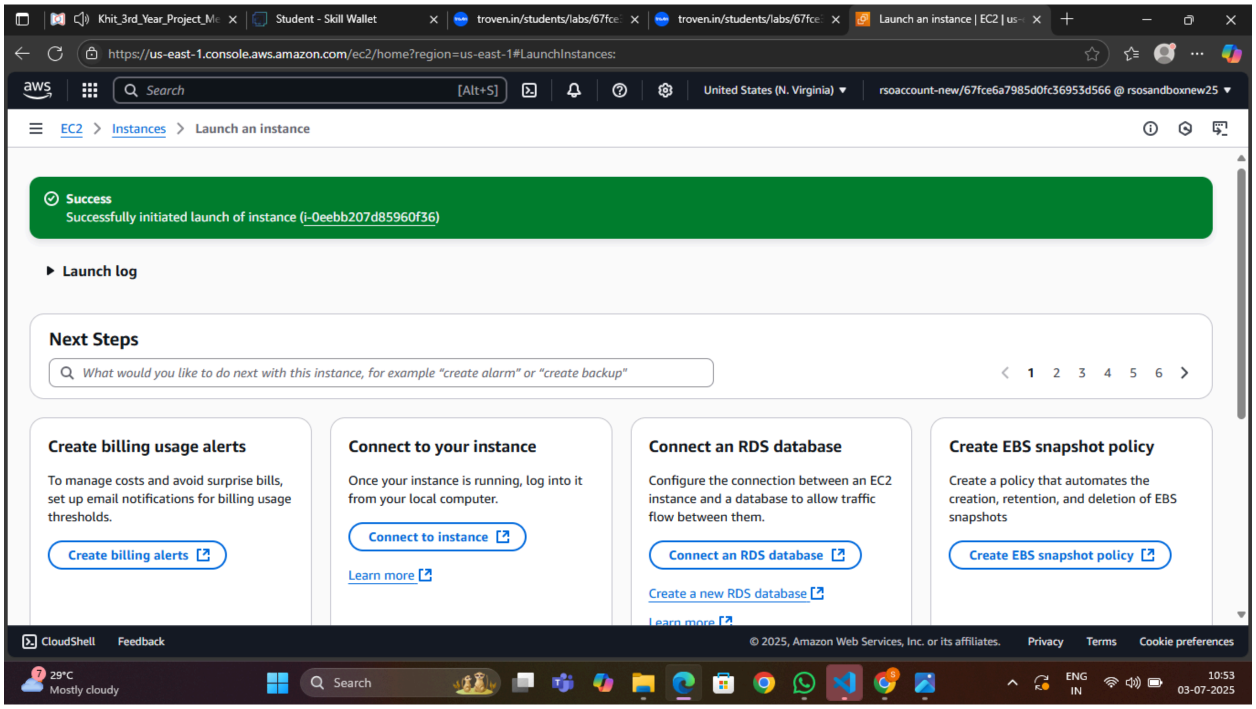
● .pem
For use with OpenSSH

○ .ppk
For use with PuTTY

When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** Learn more

Cancel Create key pair

- Create and download the key pair for Server access.


Khit_3rd_Year_Project_M
Student - Skill Wallet
troven.in/students/labs/67fce
troven.in/students/labs/67fce
Launch an instance | EC2 | us-
https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances:
Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew25
EC2 > Instances > Launch an instance

Success
Successfully initiated launch of instance (i-0eebb207d85960f36)

Launch log

Next Steps

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

Create billing usage alerts
To manage costs and avoid surprise bills, set up email notifications for billing usage thresholds.
Create billing alerts

Connect to your instance
Once your instance is running, log into it from your local computer.
Connect to instance
Learn more

Connect an RDS database
Configure the connection between an EC2 instance and a database to allow traffic flow between them.
Connect an RDS database
Create a new RDS database
Learn more

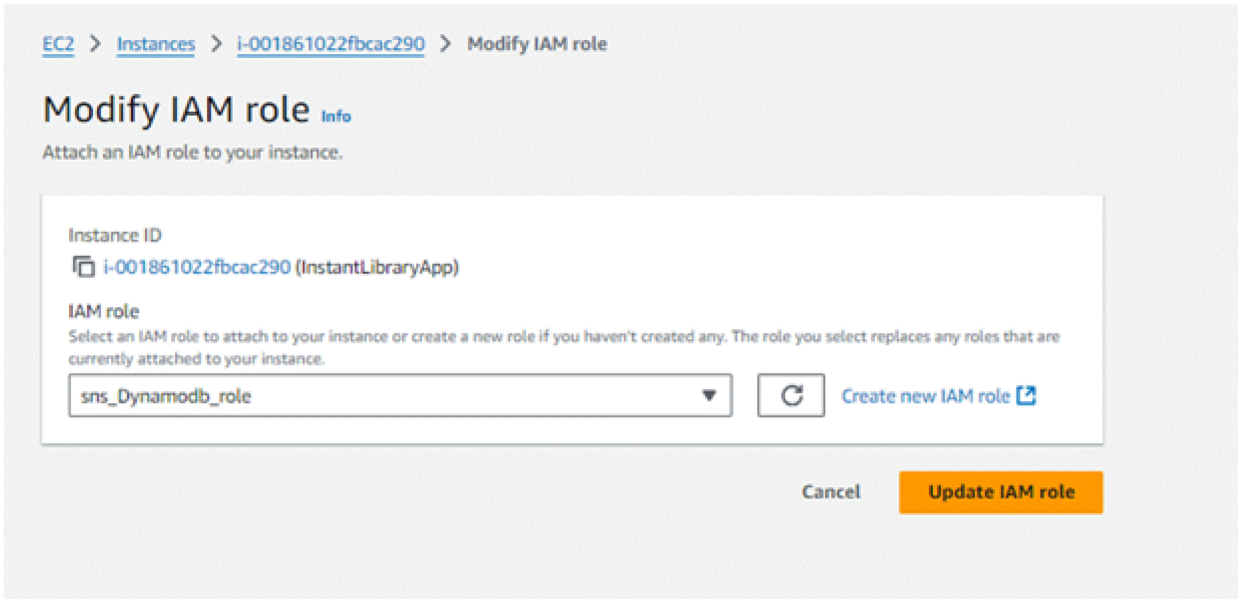
Create EBS snapshot policy
Create a policy that automates the creation, retention, and deletion of EBS snapshots
Create EBS snapshot policy

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 > Modify IAM role

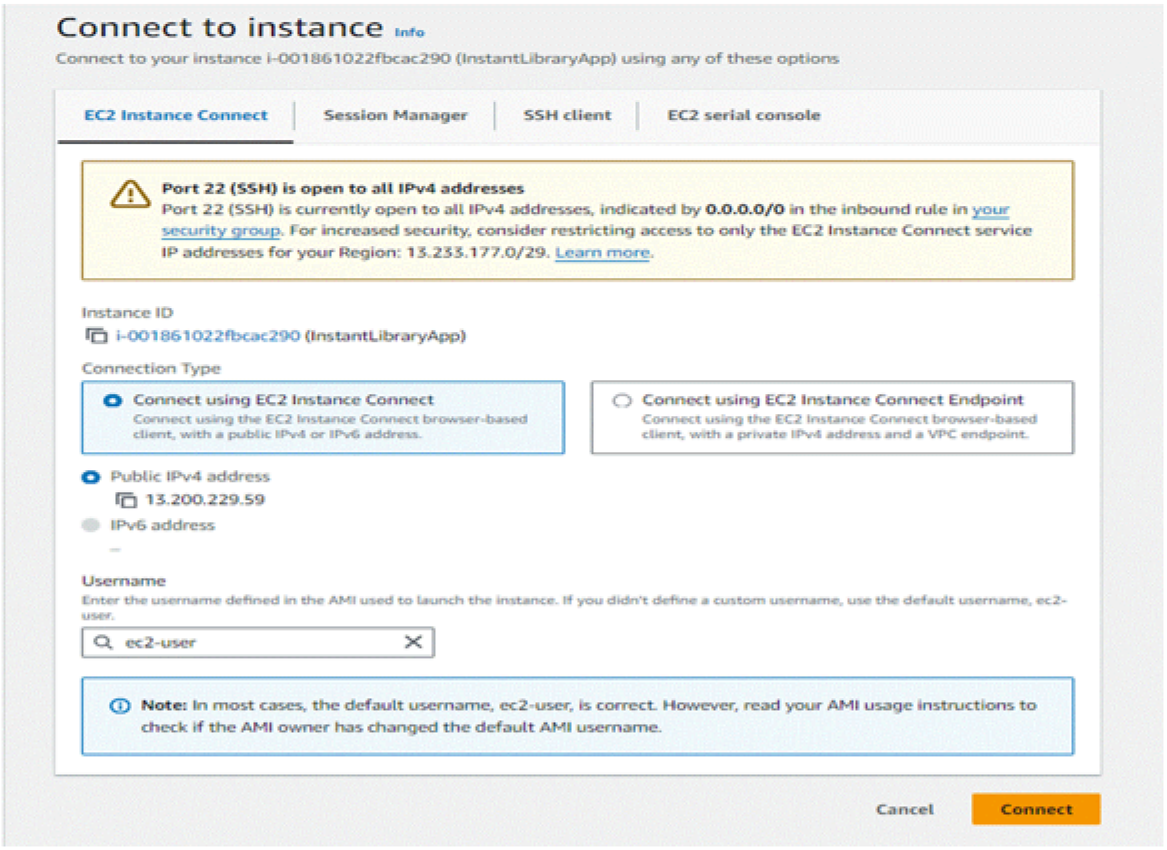
Modify IAM role Info

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



Connect to instance Info

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

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)

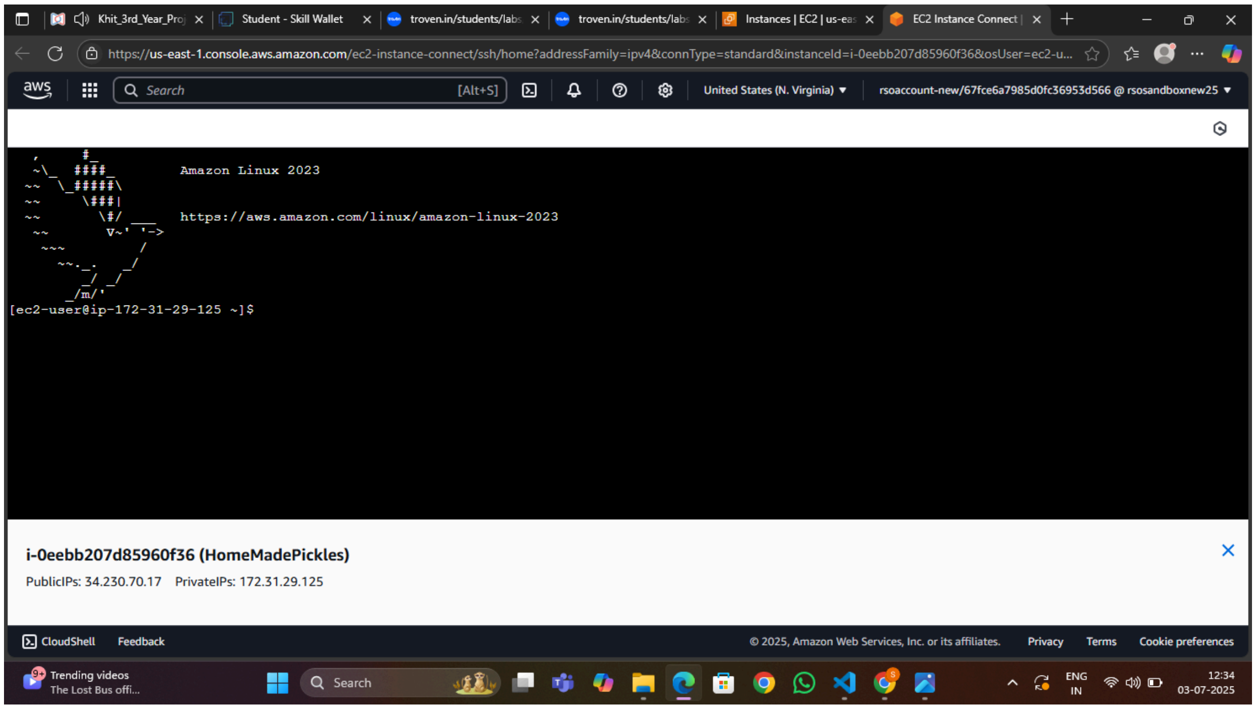
Connection Type
[✓ Connect using EC2 Instance Connect] Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.
Connect using EC2 Instance Connect Endpoint — Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

[✓ 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

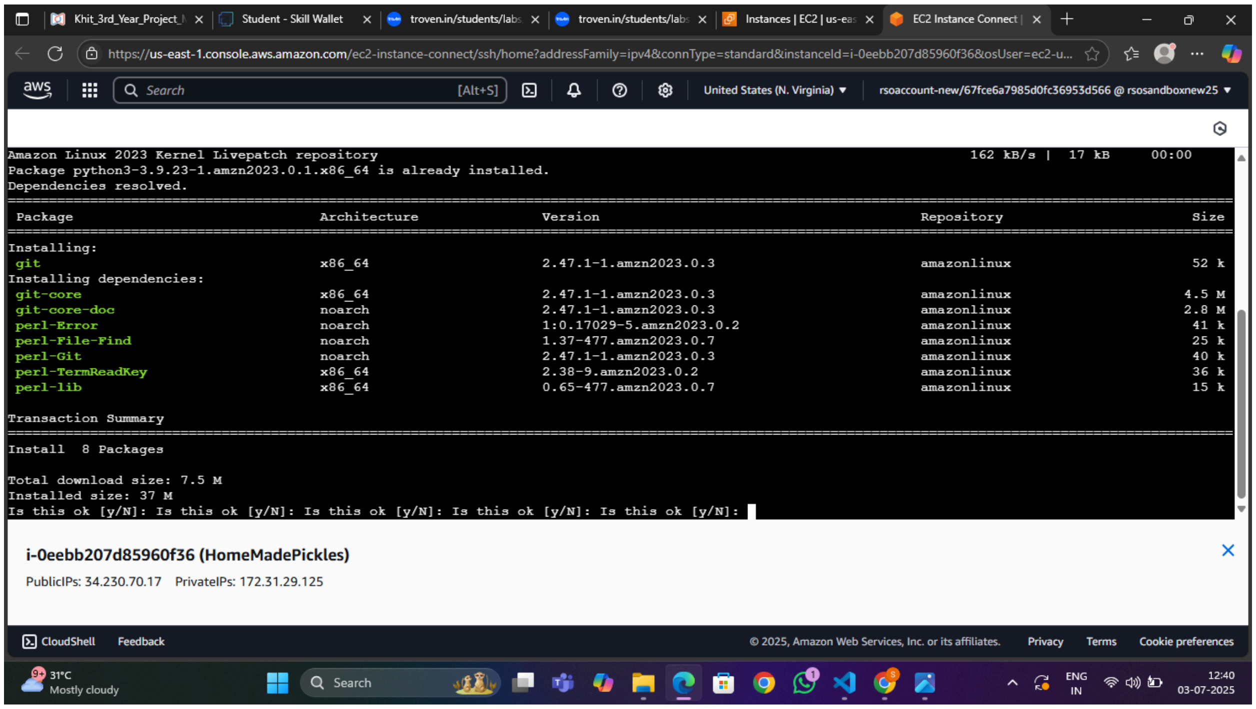
Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.



```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-29-125 ~]$
```

i-0eebb207d85960f36 (HomeMadePickles)

PublicIPs: 34.230.70.17   PrivateIPs: 172.31.29.125



```
Amazon Linux 2023 Kernel Livepatch repository                                    162 kB/s |  17 kB     00:00
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
==========================================================================================================================
 Package                     Architecture            Version                              Repository              Size
==========================================================================================================================
Installing:
 git                         x86_64                  2.47.1-1.amzn2023.0.3                amazonlinux             52 k
Installing dependencies:
 git-core                    x86_64                  2.47.1-1.amzn2023.0.3                amazonlinux            4.5 M
 git-core-doc                noarch                  2.47.1-1.amzn2023.0.3                amazonlinux            2.8 M
 perl-Error                  noarch                  1:0.17029-5.amzn2023.0.2             amazonlinux             41 k
 perl-File-Find              noarch                  1.37-477.amzn2023.0.7                amazonlinux             25 k
 perl-Git                    noarch                  2.47.1-1.amzn2023.0.3                amazonlinux             40 k
 perl-TermReadKey            x86_64                  2.38-9.amzn2023.0.2                  amazonlinux             36 k
 perl-lib                    x86_64                  0.65-477.amzn2023.0.7                amazonlinux             15 k

Transaction Summary
==========================================================================================================================
Install  8 Packages

Total download size: 7.5 M
Installed size: 37 M
Is this ok [y/N]: Is this ok [y/N]: Is this ok [y/N]: Is this ok [y/N]: Is this ok [y/N]:
```

i-0eebb207d85960f36 (HomeMadePickles)

PublicIPs: 34.230.70.17   PrivateIPs: 172.31.29.125

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 https://github.com/Lavanya-959/ACHAR-HOUSE.git
- Note: change your-github-username and your-repository-name with your credentials,
- This will download your project to the EC2 instance.
- To navigate to the project directory, run the following command

```
cd ACHAR-HOUSE
cd project
```

Create a Virtual Environment:

- python3 -m venv venv
- source venv/bin/activate

- sudo yum install python3 git
- sudo pip3 install flask boto3

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

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

Verify the Flask app is running:

http://your-ec2-public-ip

- Run the Flask app on the EC2 instance

Access the website through:

public ips: http://34.230.70.17:5000/

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.

Conclusion

ACHAAR HOUSE 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.