Homemade Pickles & Snacks: Taste The Best

Project Description:

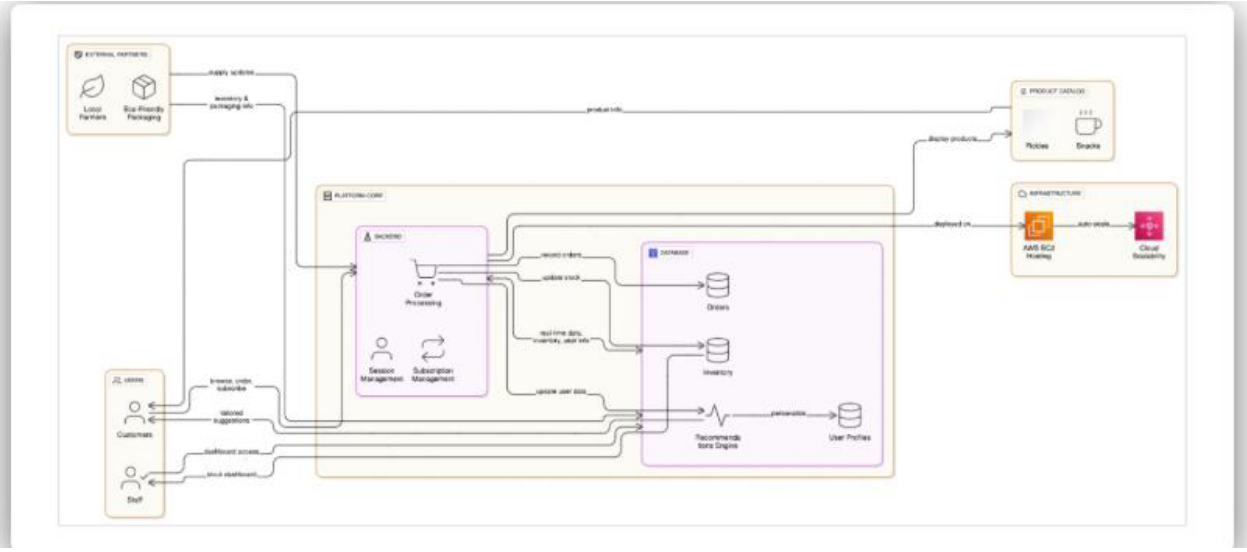
This project is a user-friendly website designed to showcase and sell a variety of homemade pickles and Snacks. The *Homemade Pickles & Snacks: Taste The Best* website is a flavorful e-commerce platform designed to bring traditional, homemade delights to customers nationwide. It showcases a vibrant catalog of pickles, snacks, and preserves inspired by authentic regional recipes. The project aims to connect modern audiences with the nostalgic taste of home-cooked treats while supporting small-batch production methods. Built with a user-friendly interface, the website ensures smooth navigation and a personalized shopping experience. Products are categorized by flavor profile, ingredients, and dietary preferences, allowing users to find their perfect match effortlessly. An integrated Flask backend handles order processing and dynamic product listings. Secure payment gateways and user authentication boost customer trust and transaction safety. AWS integration enables scalable hosting, real-time inventory updates, and responsive content delivery. The design prioritizes accessibility and mobile optimization for broader reach. A feedback and review section encourages community engagement and builds brand transparency. The blog section features recipes, tips, and stories behind the products to deepen customer connection. Admin dashboards allow easy product management and order tracking. SEO optimization and social media linking help improve visibility and drive organic traffic. The project reflects a harmonious blend of tradition and technology to promote homemade excellence. Overall, it delivers a warm, trustworthy, and appetizing user journey from homepage to checkout.

Scenario 1: Efficient Order Placement by Customers
In the "Flavour Cart" homemade pickles and snacks website, AWS EC2 powers a reliable and scalable infrastructure, allowing multiple customers to browse and order products simultaneously. For example,a user visits the site, explores the pickles section, and places an order for a mango pickle jar. Flaskhandles the backend workflow, processing user selections, updating inventory, and confirming orders in real time. The cloud-based architecture ensures a seamless and fast shopping experience, even during peak demand like festivals or seasonal sales.

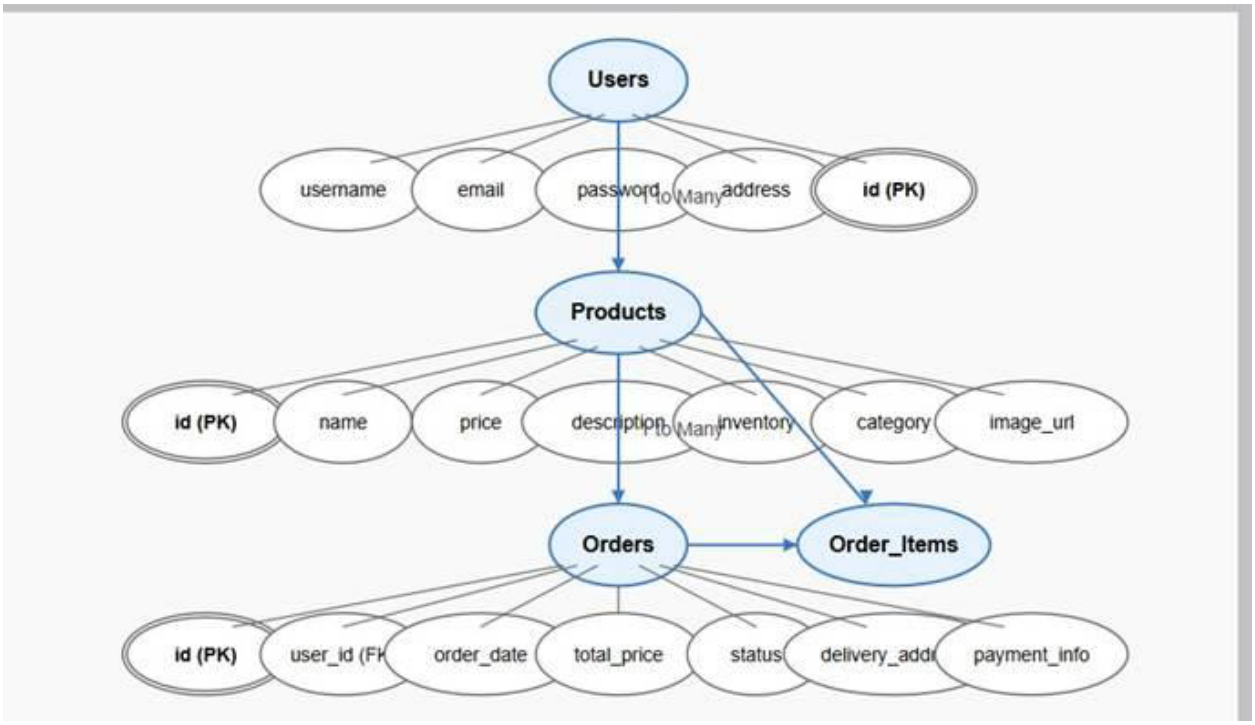
Scenario 2: Seamless Order Notifications for Kitchen Staff and Customers
When an order is placed, the system uses AWS SNS to instantly notify both the customer and the in-house kitchen team. Flask captures the order details and triggers an SNS email or SMS alert:
the customer receives a confirmation with order tracking info, while the kitchen staff is alerted with the order details to initiate preparation. All order information is securely stored in AWS DynamoDB, ensuring timely fulfilment and status monitoring.

Scenario 3: Real-Time Inventory and Product Availability
The website allows users to explore the entire product range—spicy pickles, sweet treats, crunchy snacks—with up-to-date availability status. Flask fetches real-time data from DynamoDB, showing users whether an item is in stock or on backorder. Thanks to EC2, the site delivers a consistently smooth and responsive experience, even when accessed by many users across mobile or desktop platforms, ensuring customers always have access to the latest listings.

AWS ARCHITECTURE :



Entity Relationship (ER)Diagram :



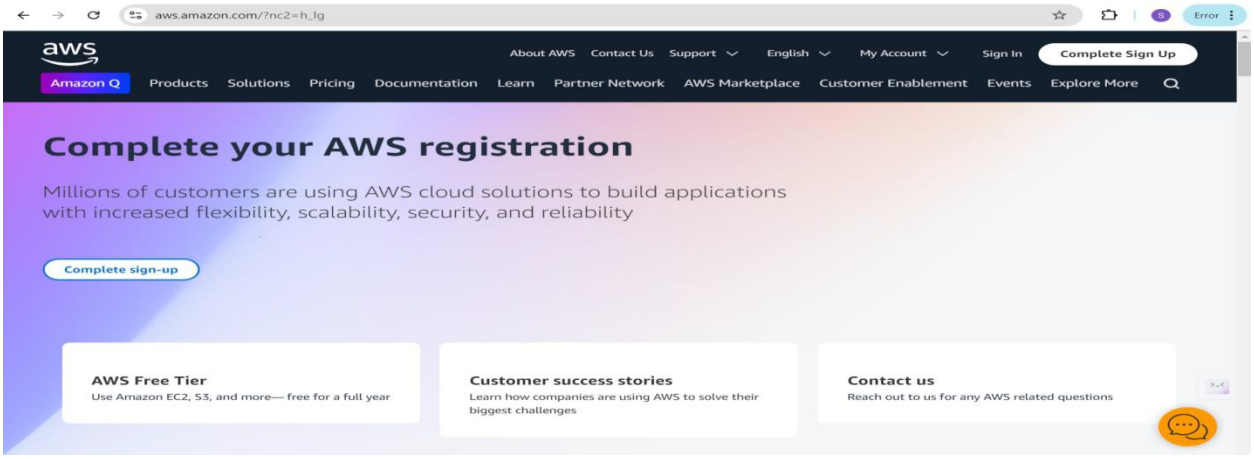
Pre-requisites:

1.AWS Account Setup: AWS Account Setup
2.Understanding IAM: IAM Overview
3.Amazon EC2 Basics: EC2 Tutorial
4.DynamoDB Basics: DynamoDB Introduction
5.VS Code Installation: VISUALSTUDIO.COM
6.Git Version Control: Git Documentation
Project Work Flow:
1. AWS Account Setup and Login
Activity 1.1: Set up an AWS account if not already done.
Activity 1.2: Log in to the AWS Management Console.
2. DynamoDB Database Creation and Setup
Activity 2.1: Create a DynamoDB Table.
Activity 2.2: Configure Attributes for User Data and Book Requests
3. SNS Notification Setup
Activity 3.1: Create SNS topics for book request notifications.
Activity 3.2: Subscribe users and library staff to SNS email notifications.
4. Backend Development and Application Setup
Activity 4.1: Develop the Backend Using Flask..
Activity 4.2: Integrate AWS Services Using boto3.
5. IAM Role Setup
Activity 5.1: Create IAM Role
Activity 5.2: Attach Policies
6. EC2 Instance Setup
Activity 6.1: Launch an EC2 instance to host the Flask application.
Activity 6.2: Configure security groups for HTTP, and SSH access.
7.Deployment on EC2
Activity 7.1: Upload Flask Files
Activity 7.2: Run the Flask App
8. Testing and Deployment
Activity 8.1: Conduct functional testing to verify user signup, login, orders and notifications.

Milestone 1: AWS Account Setup and Login

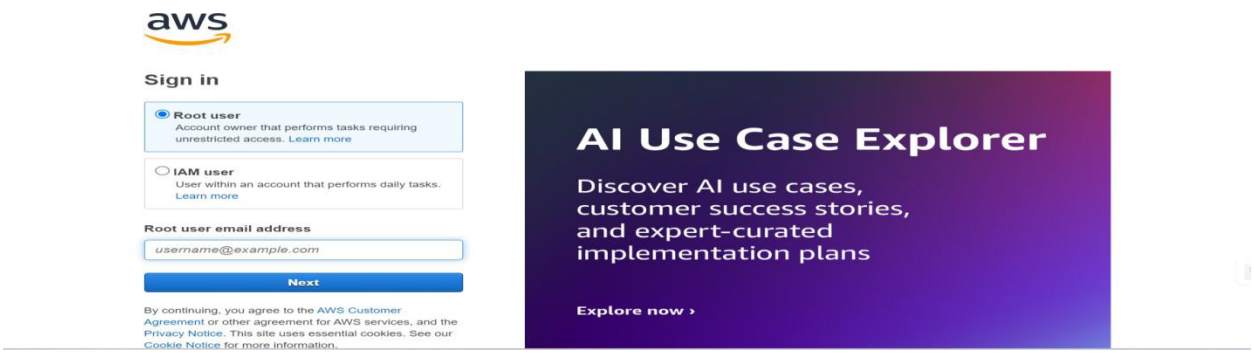
● **Activity 1.1:** Set up an AWS account if not already done.

○Sign up for an AWS account and configure billing settings.



● **Activity 1.2:** Log in to the AWS Management Console

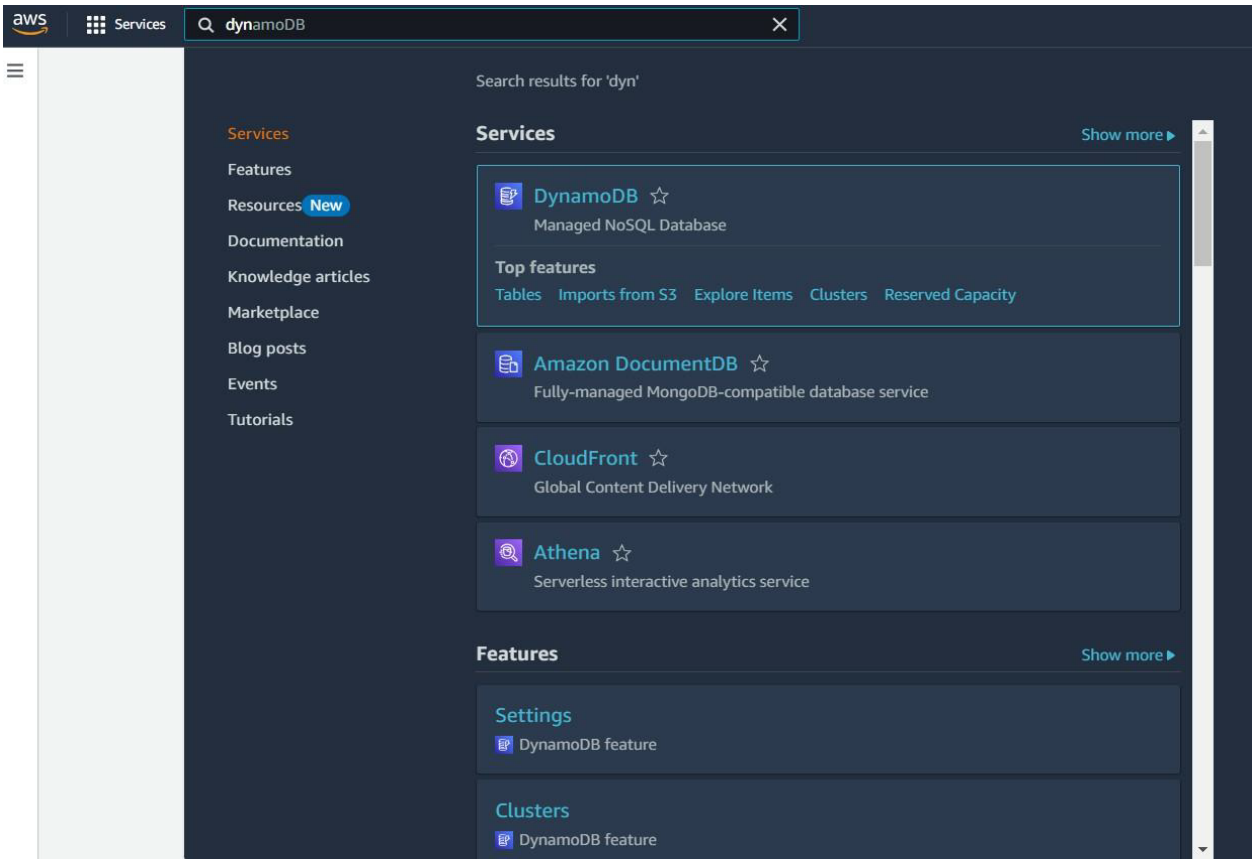
○After setting up your account, log in to the AWS Management Console.

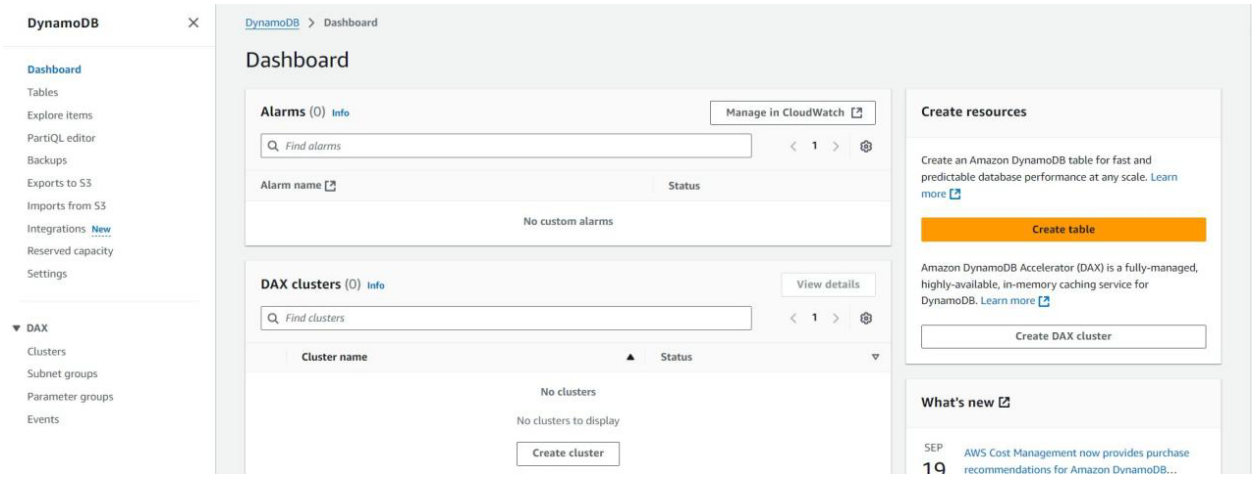


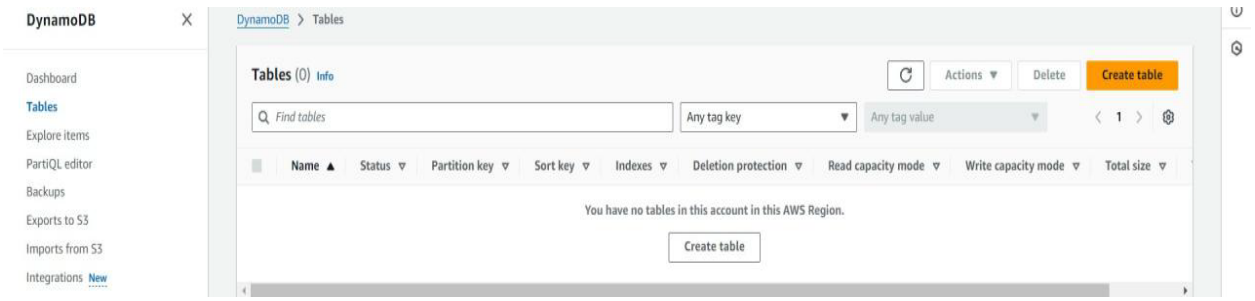
Milestone 2: DynamoDB Database Creation and Setup

● **Activity 2.1:**Navigate to the DynamoDB

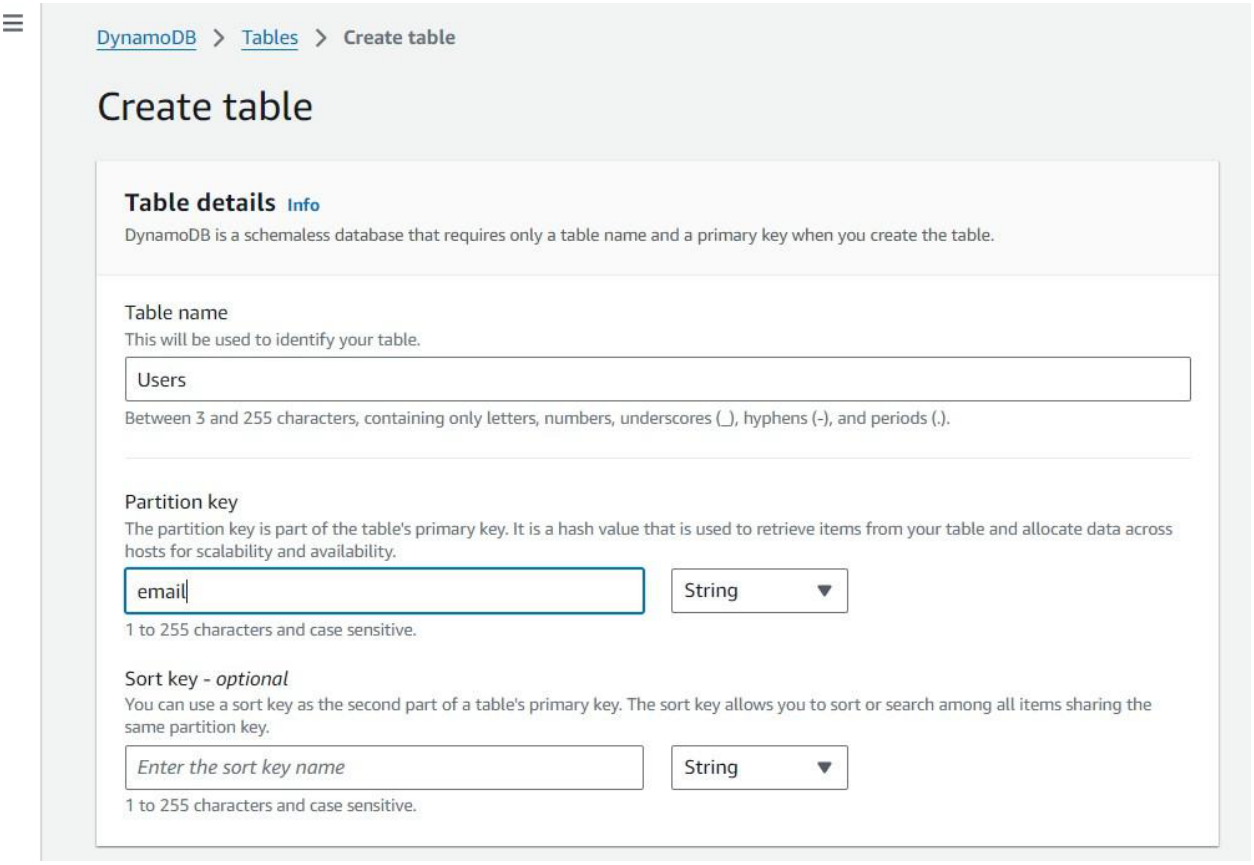
○In the AWS Console, navigate to DynamoDB and click on create tables.


aws
Services
dynamoDB
Search results for 'dyn'
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Services
Show more
DynamoDB
Managed NoSQL Database
Top features
Tables
Imports from S3
Explore Items
Clusters
Reserved Capacity
Amazon DocumentDB
Fully-managed MongoDB-compatible database service
CloudFront
Global Content Delivery Network
Athena
Serverless interactive analytics service
Features
Show more
Settings
DynamoDB feature
Clusters
DynamoDB feature


DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings
DAX
Clusters
Subnet groups
Parameter groups
Events
DynamoDB > Dashboard
Dashboard
Alarms (0) Info
Manage in CloudWatch
Find alarms
Alarm name
Status
No custom alarms
DAX clusters (0) Info
View details
Find clusters
Cluster name
Status
No clusters
No clusters to display
Create cluster
Create resources
Create an Amazon DynamoDB table for fast and predictable database performance at any scale. Learn more
Create table
Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. Learn more
Create DAX cluster
What's new
SEP
19
AWS Cost Management now provides purchase recommendations for Amazon DynamoDB...


DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
DynamoDB > Tables
Tables (0) Info
Actions
Delete
Create table
Find tables
Any tag key
Any tag value
Name
Status
Partition key
Sort key
Indexes
Deletion protection
Read capacity mode
Write capacity mode
Total size
You have no tables in this account in this AWS Region.
Create table

Activity 2.2: Create a DynamoDB table for storing Signup details and pickle requests.
○Create Users table with partition key “Email” with type String and click on create tables.



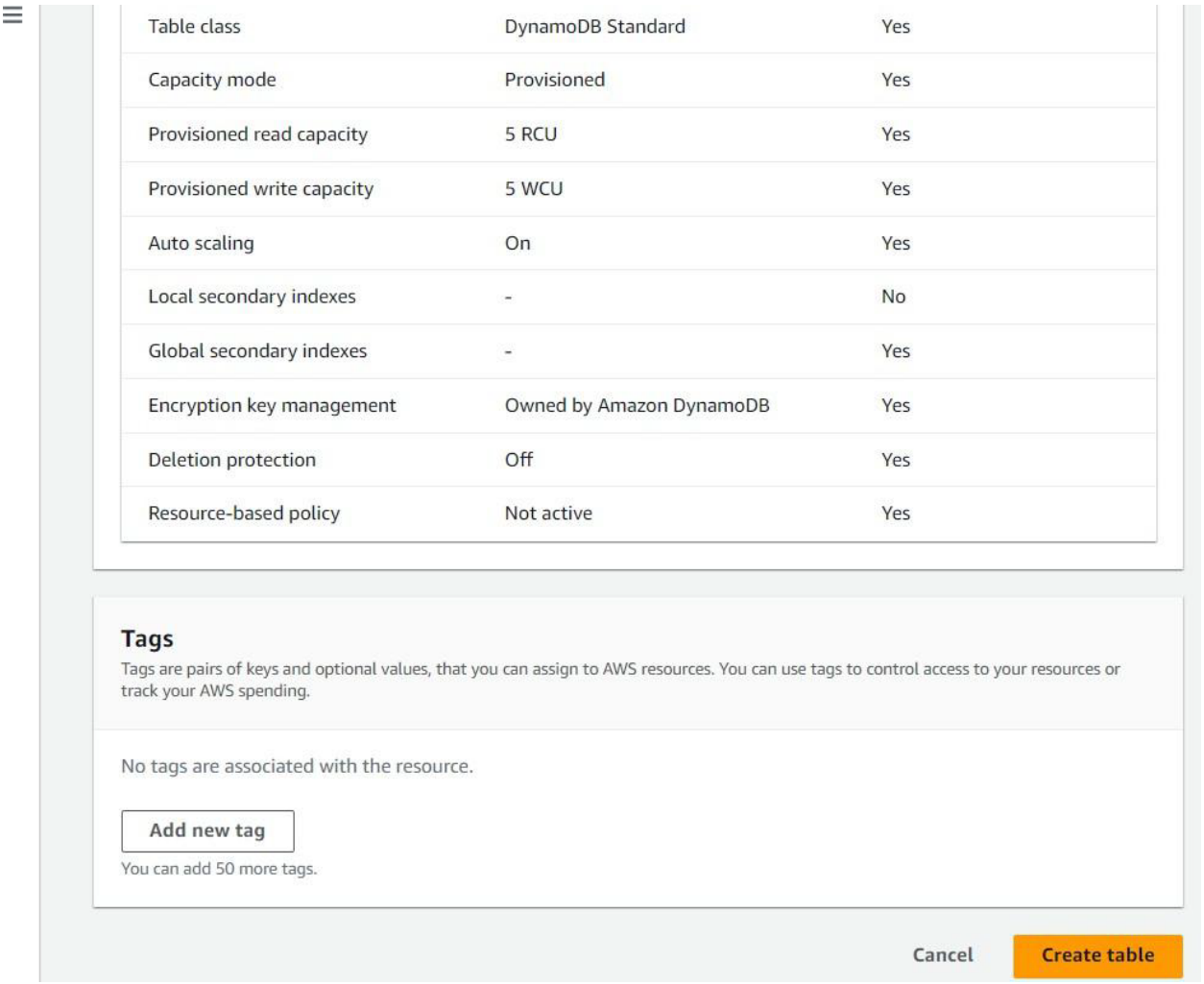
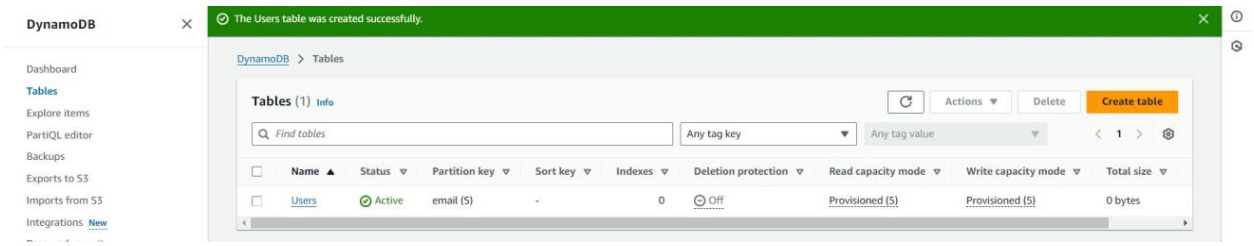


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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



Follow the same steps to create a requests table with Email as the primary key for pickles order, contact requests data.

ynanoDB > Tables > Create table

reate table

Table details Info

DynamoDB is a schemalesss dadatbase that requires a only a table name and primary key when you create the

Table name

This will be ussed to identify your table.

PicklesOrder

Between 3 and 255 characters, containing letterers, numderss‹(–), and perods ..

Partition key

The partition key part of the table's primary key. It is a hash value that is used to retrieve items from your table hosts for scalability and availability.

order_id | String

1 to 255 characcters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of table's primary key. Sort key allows you so sort or seach among il) same partition key.

Enter the sort key name | String

1 to 255 characcters and case sensitive.

- (•) Default settings
  This option optimizes you table based on AWS
- ( ) Customize settings
  Use this option to spetty or make Dynanol

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

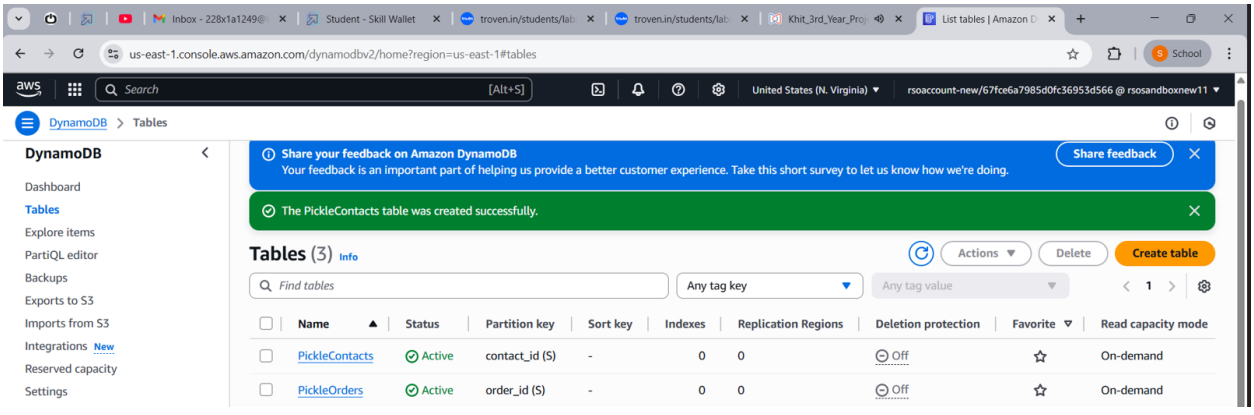
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

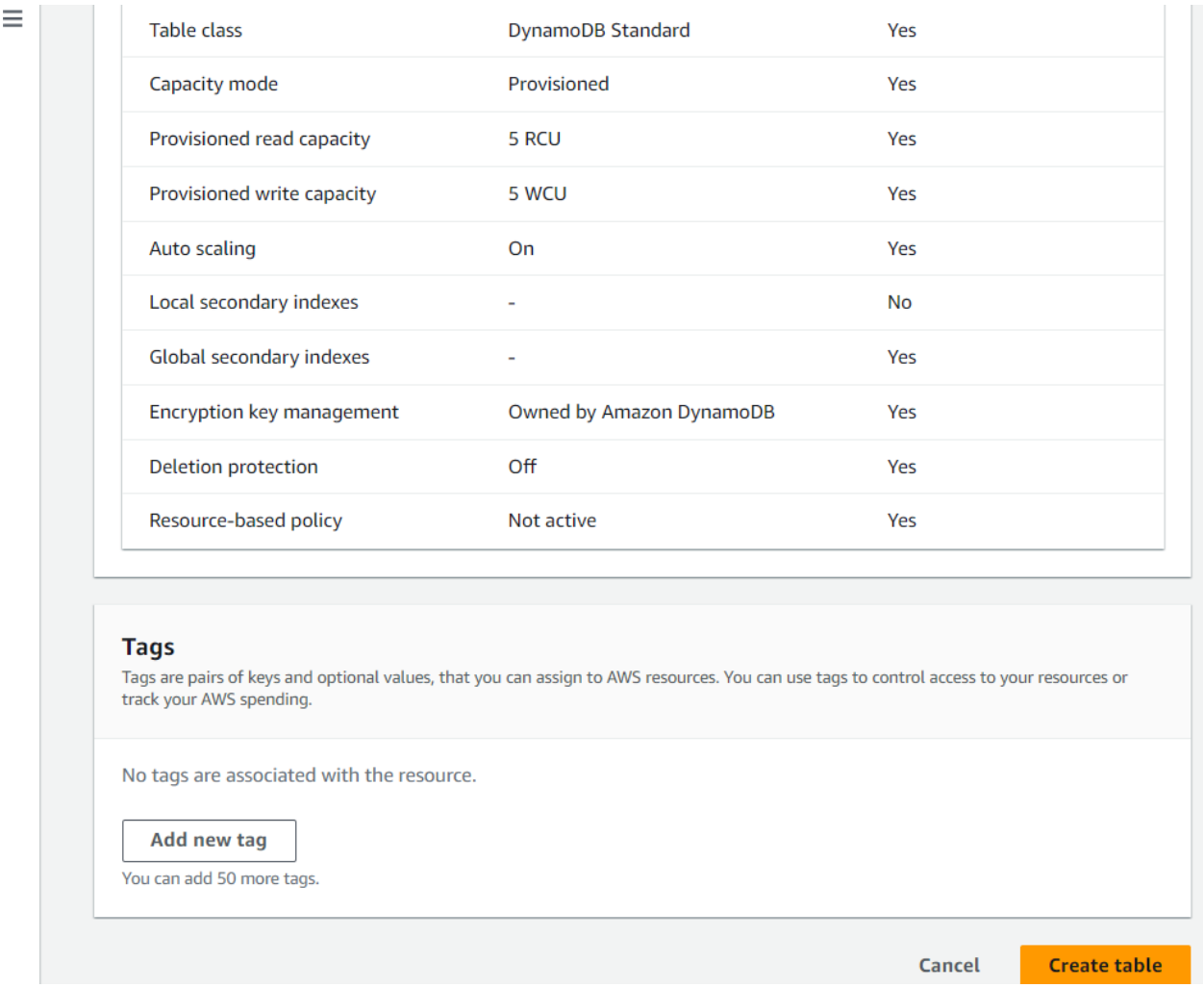
No tags are associated with the resource.

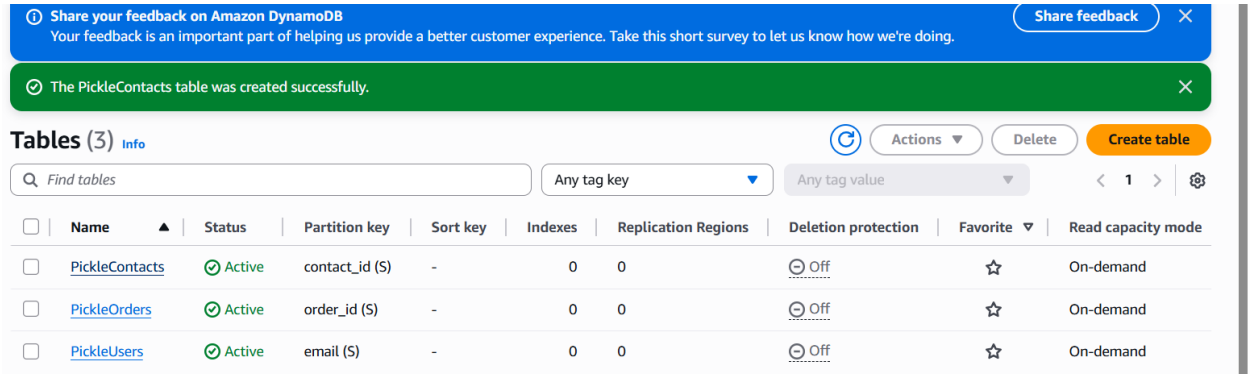
Add new tag

You can add 50 more tags.

Cancel Create table



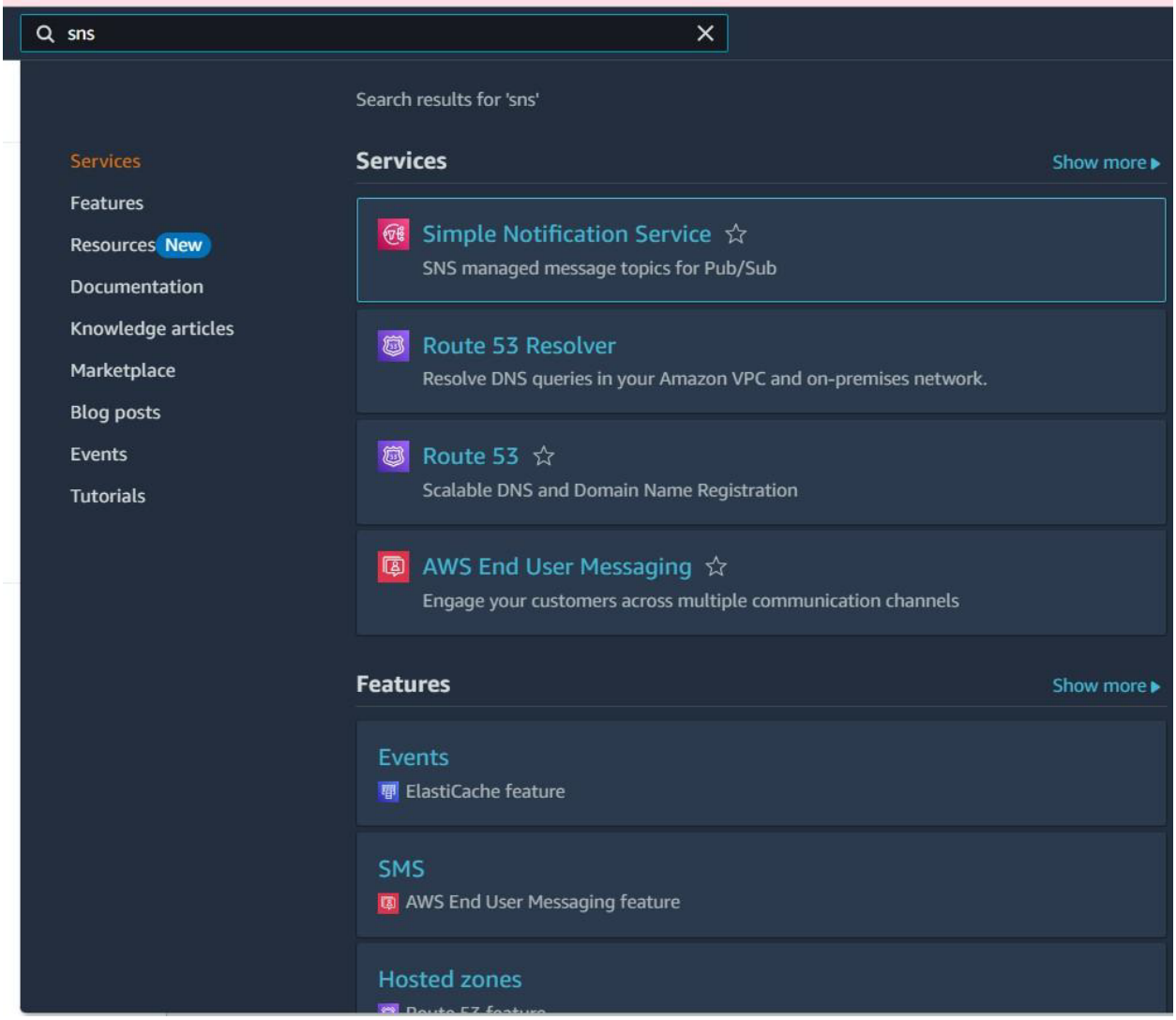


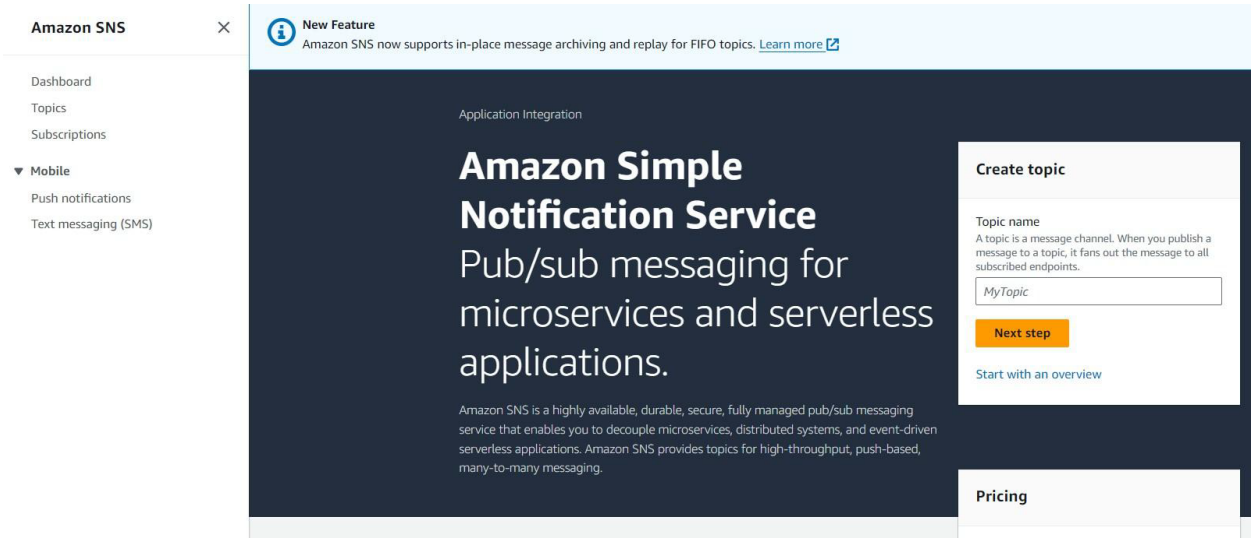


Milestone 3: SNS Notification Setup

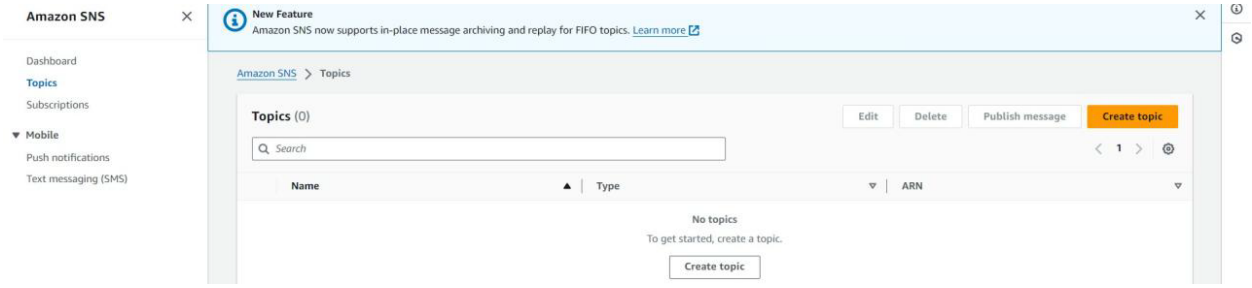
● **Activity 3.1:** Create SNS topics for sending email notifications to users and library staff.

In the AWS Console, search for SNS and navigate to the SNS Dashboard


sns
Search results for 'sns'
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Services
Show more
Simple Notification Service
SNS managed message topics for Pub/Sub
Route 53 Resolver
Resolve DNS queries in your Amazon VPC and on-premises network.
Route 53
Scalable DNS and Domain Name Registration
AWS End User Messaging
Engage your customers across multiple communication channels
Features
Show more
Events
ElastiCache feature
SMS
AWS End User Messaging feature
Hosted zones


Amazon SNS
Dashboard
Topics
Subscriptions
Mobile
Push notifications
Text messaging (SMS)
New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more
Application Integration
Amazon Simple Notification Service
Pub/sub messaging for microservices and serverless applications.
Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.
Create topic
Topic name
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.
MyTopic
Next step
Start with an overview
Pricing

- Click on Create Topic and choose a name for the topic.



- Choose Standard type for general notification use cases and Click on Create Topic.

Milestone 4:Backend Development and Application Setup

● **Activity 4.1:** Develop the backend using Flask

File Explorer Structure

Description: set up the INSTANT LIBRARY project with an app.py file, a static/ folder for images, and a templates/ directory containing all required HTML pages like home, login, signup, pickle-specific pages(e.g.,Veg_Pickles.html,Non_Veg_Pickles.html,Snacks.html), and utility pages (e.g., order.html, contact.html).

Description of the code :

● Flask App Initialization

```
from flask import Flask, request, render_template, redirect, url_for, session, flash,
jsonify
import boto3
import uuid
import os
import hashlib
from datetime import datetime
from botocore.exceptions import ClientError
```

Description: import essential libraries including Flask utilities for routing, Boto3 for DynamoDB

operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification

```
app = Flask(__name__)
```

Description: initialize the Flask application instance using Flask(name) to start building the web app.

● Dynamodb Setup:

```
# DynamoDB tables - Production ready
order_table = dynamodb.Table('PickleOrders')
user_table = dynamodb.Table('PickleUsers')
contact_table = dynamodb.Table('PickleContacts')
```

Description: initialize the DynamoDB resource for the us-east-1 region and set up access to the Users ,Orders, Contact tables for storing user details and pickles requests.

● SNS Connection

```
@app.route('/notify')
def notify():
    # Send a sample SNS message
    sns.publish(
        TopicArn=SNS_TOPIC_ARN,
        Message='A new order was received on Homemade Pickles & Snacks!',
        Subject='New Pickle Order Alert'
    )
    return "SNS notification sent!"
```

Description: Configure SNS to send notifications when a pickle request is submitted. Paste your stored ARN link in the sns_topic_arn space, along with the region, name where the SNS topic is created. Also, specify the chosen email service in SMTP_SERVER (e.g., Gmail, Yahoo, etc.) Create an 'App password' for the email ID and store it in the SENDER_PASSWORD section.

• Routes for Web Pages

● Home Route:

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

Description: define the home route / to automatically redirect users to the home page when they access the base URL.

• **Signup Route :**

```python
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        try:
            name = request.form['name']
            email = request.form['email']
            password = request.form['password']
            hashed_password = hash_password(password)
            timestamp = datetime.utcnow().isoformat()

            # Check if user already exists
            response = user_table.get_item(Key={'email': email})
            if 'Item' in response:
                flash('User already exists', 'error')
                return render_template('signup.html')

            # Save user to DynamoDB
            user_table.put_item(Item={
                'email': email,
                'name': name,
                'password': hashed_password,
                'created_at': timestamp,
                'status': 'active'
            })
```

```python
            # Send welcome email
            welcome_message = f"Dear {name},\n\nWelcome to Homemade Pickles &
            Snacks!\n\nYour account has been created successfully. You can now login and
            start ordering our delicious homemade pickles and snacks.\n\nThank you for
            joining us!\n\nBest regards,\nHomemade Pickles & Snacks Team"
            send_email_notification(email, 'Welcome to Homemade Pickles & Snacks!',
            welcome_message)

            flash('Account created successfully! Please login.', 'success')
            return redirect(url_for('index'))
        except Exception as e:
            flash(f'Signup failed: {str(e)}', 'error')
    return render_template('signup.html')
```

Description: define signup route to validate registration form fields, hash the user

password using Bcrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

• **login Route (GET/POST):**

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        try:
            email = request.form['email']
            password = request.form['password']
            hashed_password = hash_password(password)

            # Check user in DynamoDB
            response = user_table.get_item(Key={'email': email})
            if 'Item' in response:
                stored_password = response['Item'].get('password')
                if stored_password == hashed_password:
                    session['user_email'] = email
                    session['user_name'] = response['Item'].get('name')
                    return redirect(url_for('home'))

            flash('Invalid email or password', 'error')
        except Exception as e:
            flash(f'Login failed: {str(e)}', 'error')
    return render_template('login.html')
```

Description: define login route to validate user credentials against DynamoDB, check the password using encryption, update the login count on successful authentication, and redirect users to the home page.

• **Index Route :**

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

Description: The index page serves as the homepage, welcoming users with an overview of your brand and guiding them to key sections like products, contact, or login.

• **Order route:**

```python
@app.route('/order', methods=['GET', 'POST'])
def order():
    if request.method == 'POST':
        try:
            name = request.form['name']
            email = request.form['email']
            phone = request.form['phone']
            address = request.form['address']
            city = request.form['city']
            pincode = request.form['pincode']
            item = request.form['item']
            quantity = int(request.form['quantity'])
            notes = request.form.get('notes', '')
            order_id = str(uuid.uuid4())
            timestamp = datetime.utcnow().isoformat()

            # Save to DynamoDB with full order details
            order_table.put_item(Item={
                'order_id': order_id,
                'name': name,
                'email': email,
                'phone': phone,
                'address': address,
                'city': city,
                'pincode': pincode,
                'item': item,
                'quantity': quantity,
```

```
def order():
                'pincode': pincode,
                'item': item,
                'quantity': quantity,
                'notes': notes,
                'timestamp': timestamp,
                'status': 'pending',
                'total_amount': quantity * 100  # Sample pricing
            })

            # Send email notifications
            customer_message = f"Dear {name},\n\nYour order has been placed
            successfully!\n\nOrder ID: {order_id}\nItem: {item}\nQuantity: {quantity}
            \n\nWe'll contact you soon for delivery details.\n\nThank you for choosing
            Homemade Pickles & Snacks!"
            admin_message = f"New Order Received!\n\nOrder ID: {order_id}\nCustomer:
            {name}\nEmail: {email}\nPhone: {phone}\nItem: {item}\nQuantity: {quantity}
            \nAddress: {address}, {city} - {pincode}\nNotes: {notes}"

            send_email_notification(email, 'Order Confirmation - Homemade Pickles &
            Snacks', customer_message)
            send_email_notification(ADMIN_EMAIL, f'New Order - {order_id}', admin_message)

            session['last_order_id'] = order_id
            return redirect(url_for('sucess'))
        except Exception as e:
            flash(f'Order processing failed: {str(e)}', 'error')
            return render_template('order.html')
    return render_template('order.html')
```

Description: The order page allows customers to review their selected items, enter shipping details, and confirm payment to complete the purchase. It acts as the final step in the checkout process, ensuring a smooth and secure transaction.

• **Aws-info Route:**

```python
@app.route('/aws-info')
def aws_info():
    """Display AWS service information"""
    try:
        # Get EC2 instance info
        instance_id = get_instance_info()

        # Get IAM role info
        try:
            sts = boto3.client('sts', region_name='us-east-1')
            identity = sts.get_caller_identity()
            account_id = identity['Account']
            role_arn = identity.get('Arn', 'No role attached')
        except:
            account_id = 'Unknown'
            role_arn = 'No role attached'

        info = {
            'instance_id': instance_id,
            'account_id': account_id,
            'role_arn': role_arn,
            'region': 'us-east-1'
        }
        return jsonify(info)
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Description: The Aws-info route displays details about your website's integration with AWS, such as database status, storage usage, or service configurations. It helps admins monitor backend health and ensure smooth operation of the homemade pickles and snacks platform.

• **About route:**

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

Description: The about route displays a heartfelt introduction to your homemade pickles and snacks brand, sharing its origin, mission, and values. It builds trust and connection by telling your story and celebrating what makes your products unique.

• **Success route:**

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

Description: The success route displays a confirmation message after a successful order or form submission, reassuring users that their action was completed. It adds a friendly closing touch to the user journey with gratitude or next-step instructions.

• **Cart route:**

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

Description: The cart route displays all items a user has added for purchase, allowing them to review quantities and total cost before checkout. It acts as a shopping basket, giving users the option to update or remove products before finalizing their order.

• **Logout route:**

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

Description: The logout route securely ends the user's session, logging them out of their account. It helps protect personal information and ensures a fresh login is required for future activity.

• **Deployment Code:**

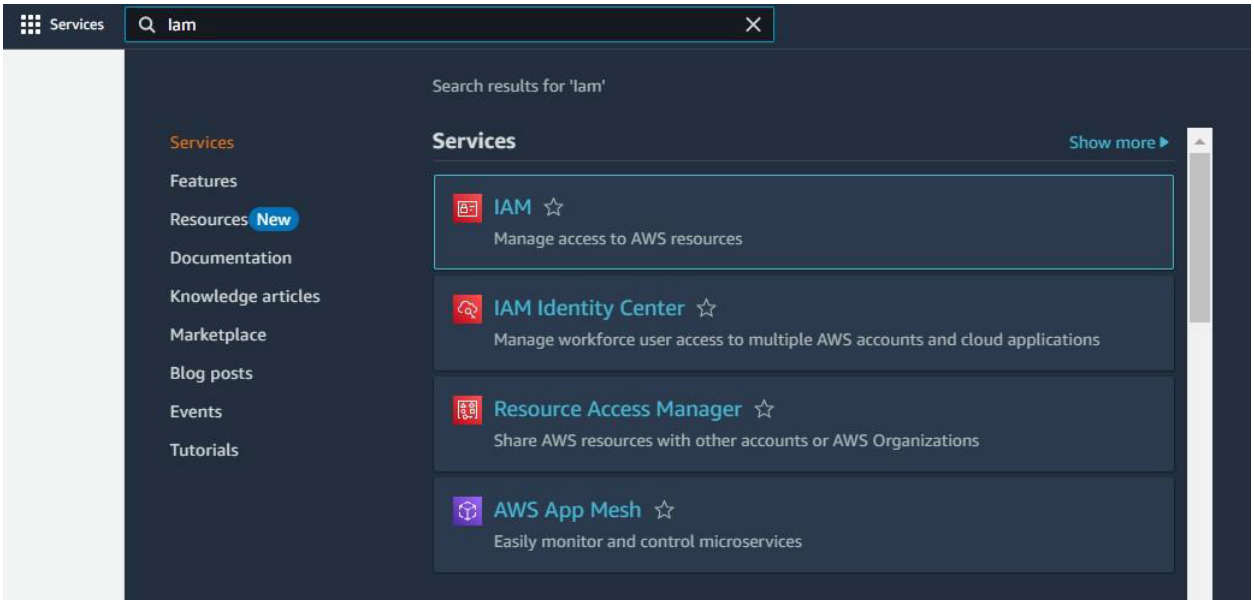
```python
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5000))
    debug = os.environ.get('DEBUG', 'False').lower() == 'true'
    app.run(host='0.0.0.0', port=port, debug=debug)
```

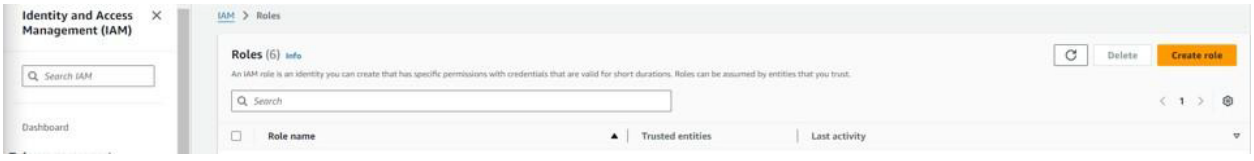
Description: The deployment code sets up and launches the homemade pickles and snacks website on a cloud platform like AWS or Heroku, ensuring public access. It typically includes configurations for web hosting, environment variables, and service integration to keep the app running smoothly.

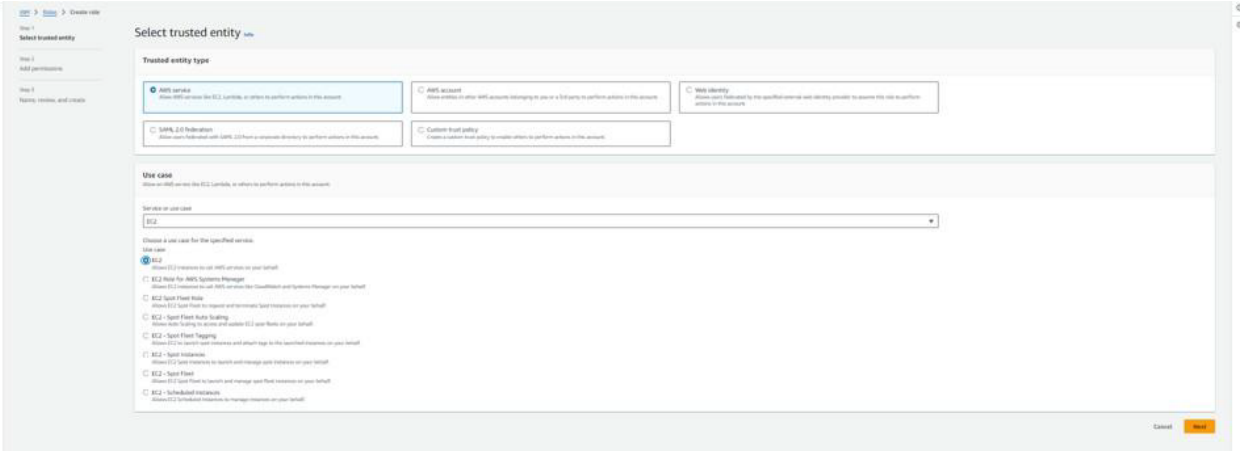
Milestone 5: IAM Role Setup

● **Activity 5.1:**Create IAM Role.

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





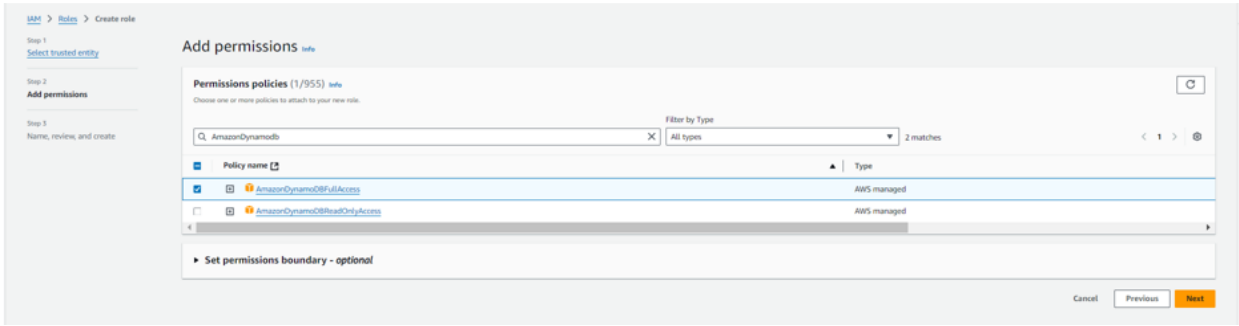


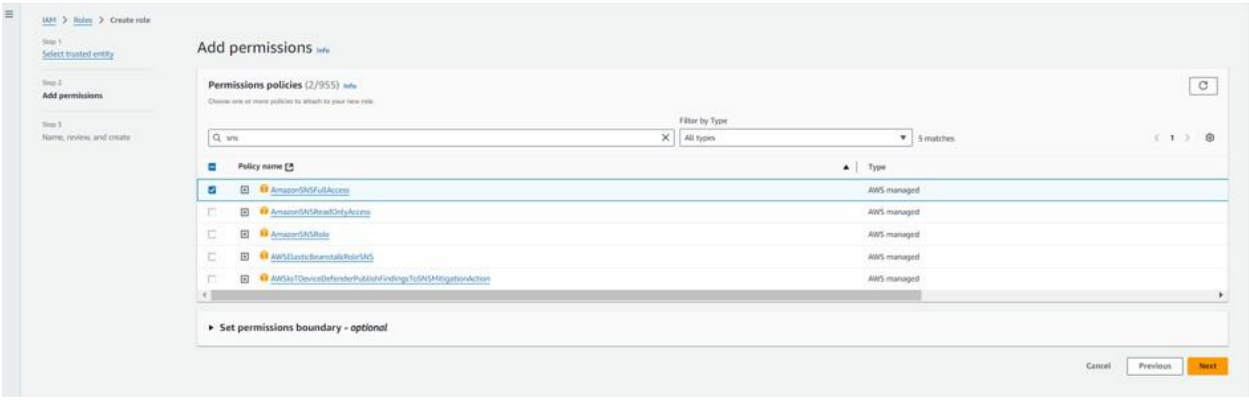
● **Activity 5.2:** Attach Policies.

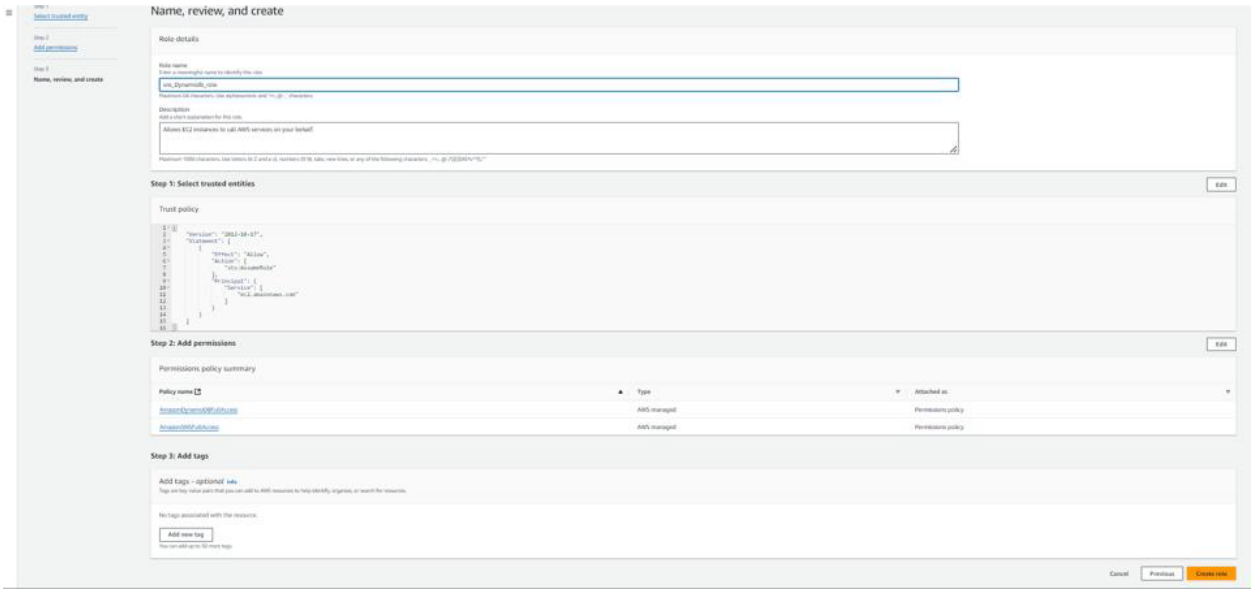
Attach the following policies to the role:

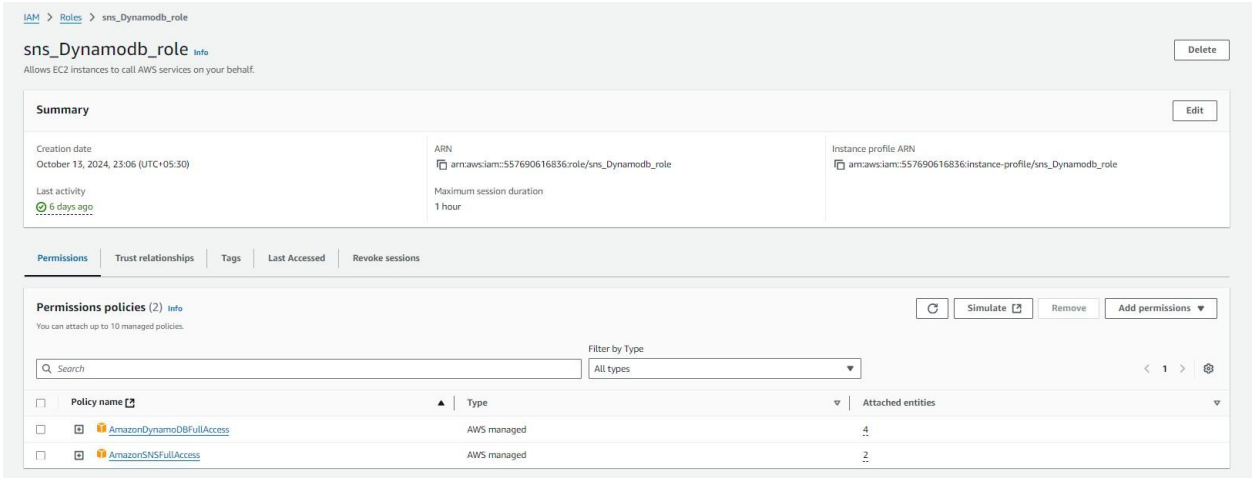
● AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.

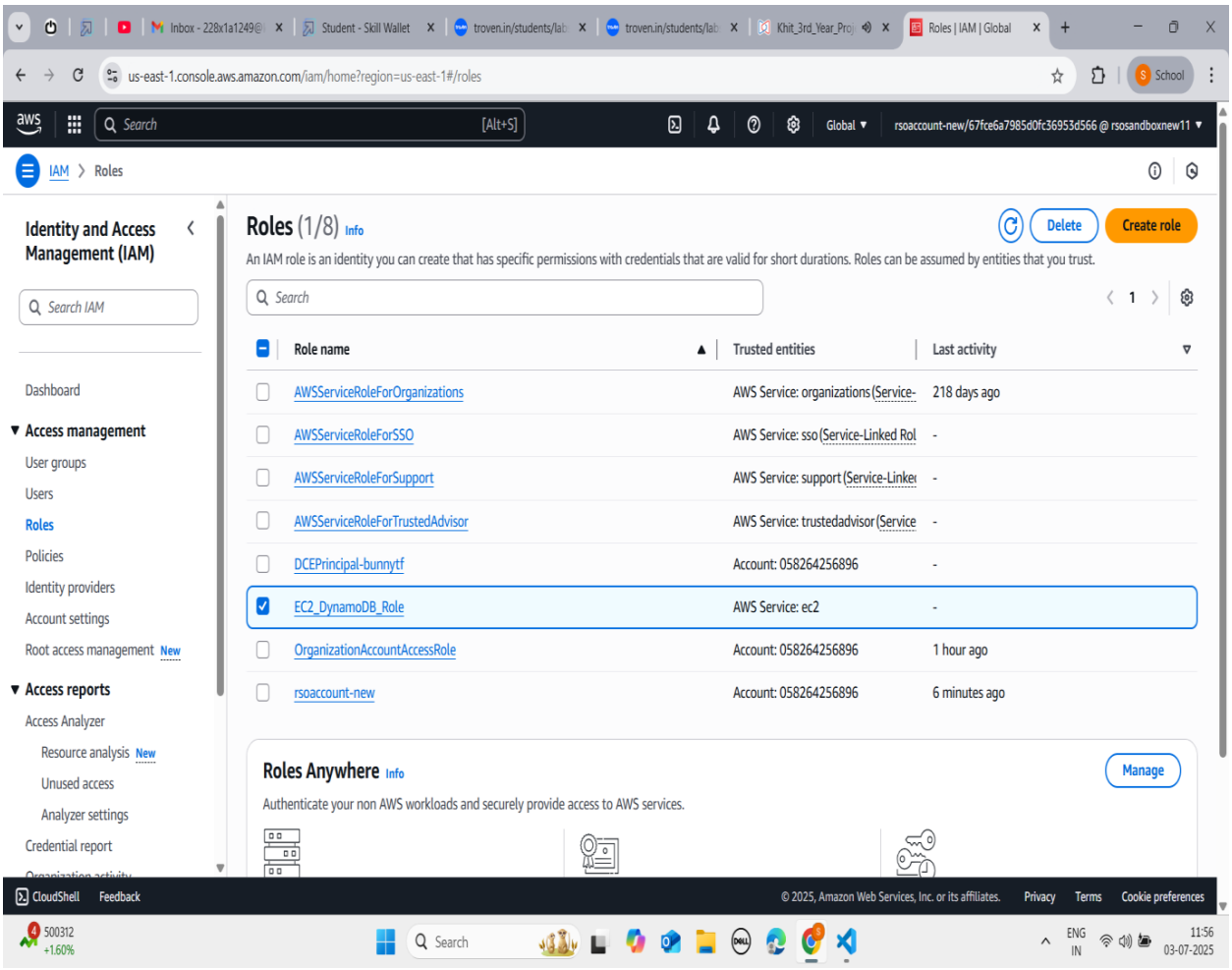
● AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.






Name, review, and create
Role details
Role name
sns_Dynamodb_role
Description
Allows EC2 instances to call AWS services on your behalf.
Step 1: Select trusted entities
Edit
Trust policy
Step 2: Add permissions
Edit
Permissions policy summary
Policy name
Type
Attached as
AWS managed
Permissions policy
Step 3: Add tags
Add tags - optional
No tags associated with the resource.
Add new tag
Cancel
Previous
Create role

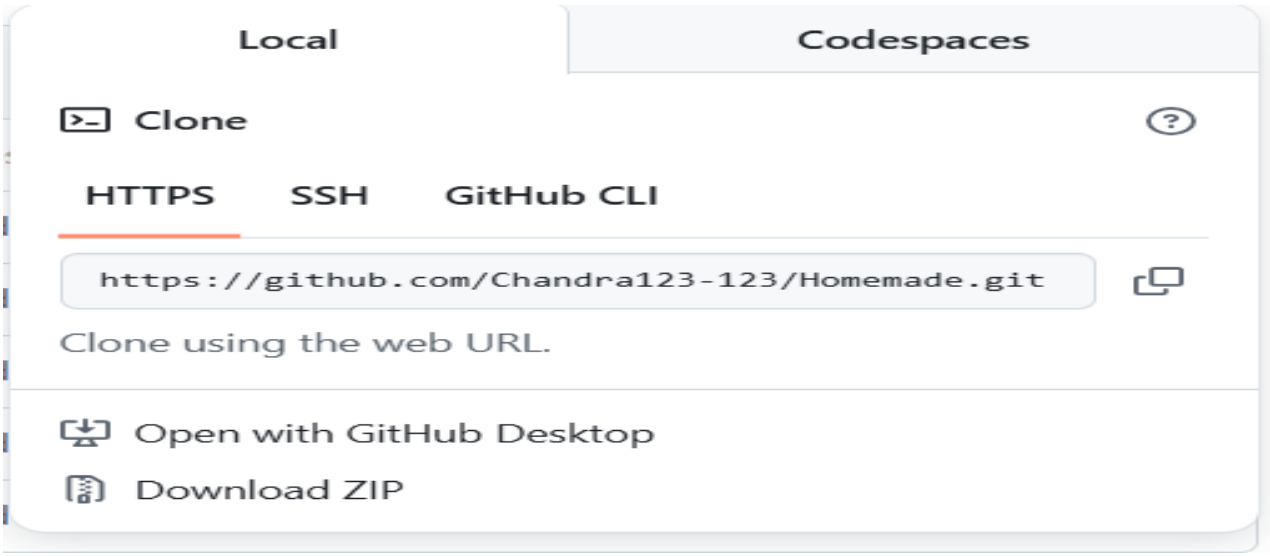

IAM > Roles > sns_Dynamodb_role
sns_Dynamodb_role Info
Allows EC2 instances to call AWS services on your behalf.
Delete
Summary
Edit
Creation date
October 13, 2024, 23:06 (UTC+05:30)
ARN
arn:aws:iam::557690616836:role/sns_Dynamodb_role
Instance profile ARN
arn:aws:iam::557690616836:instance-profile/sns_Dynamodb_role
Last activity
6 days ago
Maximum session duration
1 hour
Permissions
Trust relationships
Tags
Last Accessed
Revoke sessions
Permissions policies (2) Info
You can attach up to 10 managed policies.
Simulate
Remove
Add permissions
Filter by Type
Search
All types
Policy name
Type
Attached entities
AmazonDynamoDBFullAccess
AWS managed
4
AmazonSNSFullAccess
AWS managed
2



Milestone 6: EC2 Instance Setup

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

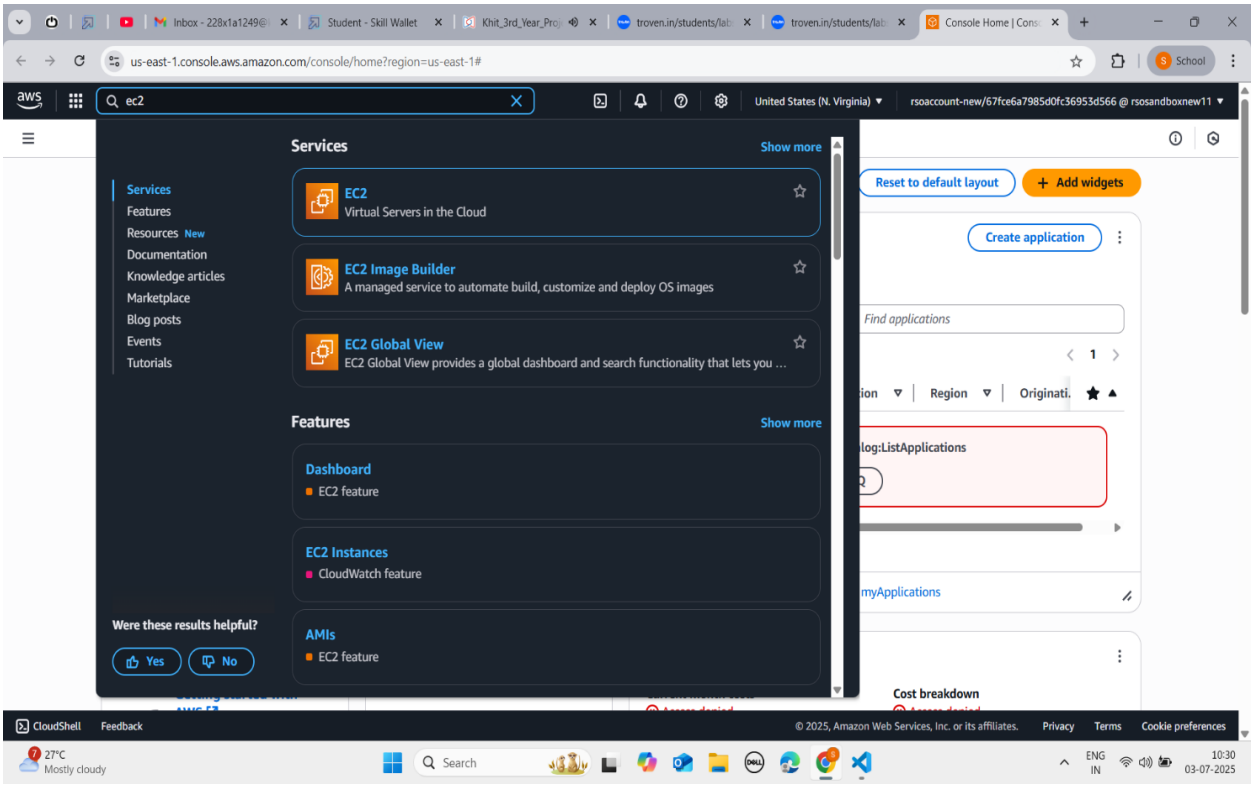
static/images	Delete static/images/images - Copy.jpg
templates	Add files via upload
README-AWS.md	Add files via upload
app.py	Add files via upload
aws-setup.py	Add files via upload
deploy.py	Add files via upload



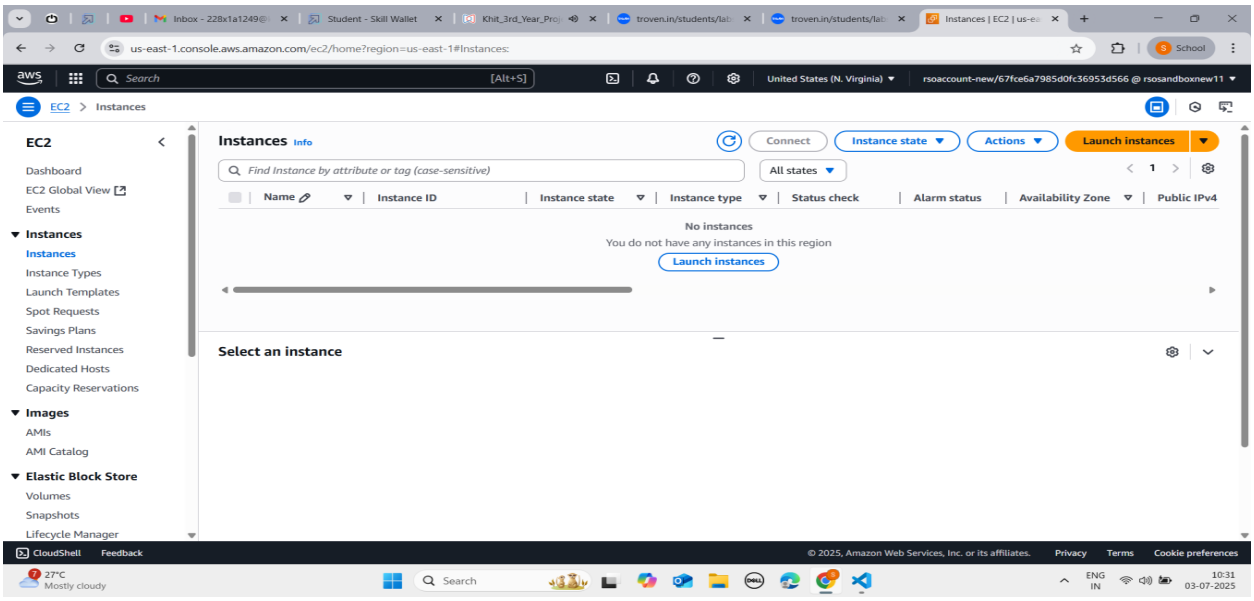
● **Activity 6.1:** Launch an EC2 instance to host the Flask application.

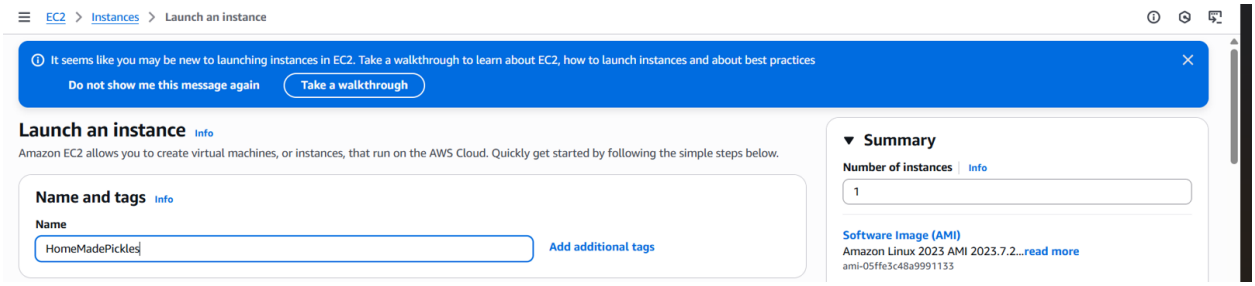
●Launch EC2 Instance

○In the AWS Console, navigate to EC2 and launch a new instance.

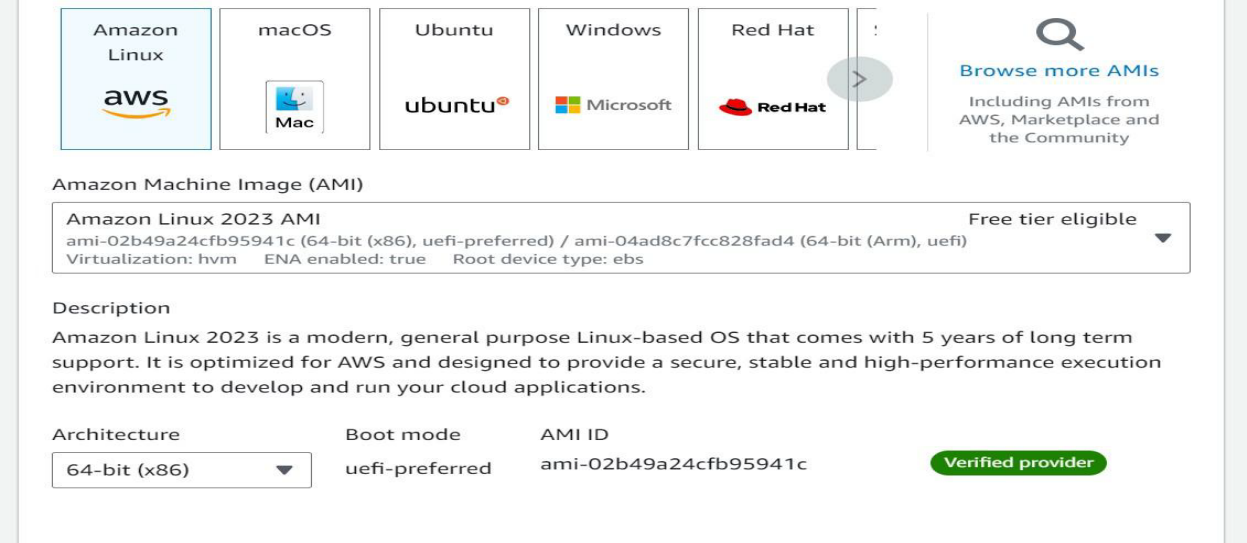


● Click on Launch instance to launch EC2 instance

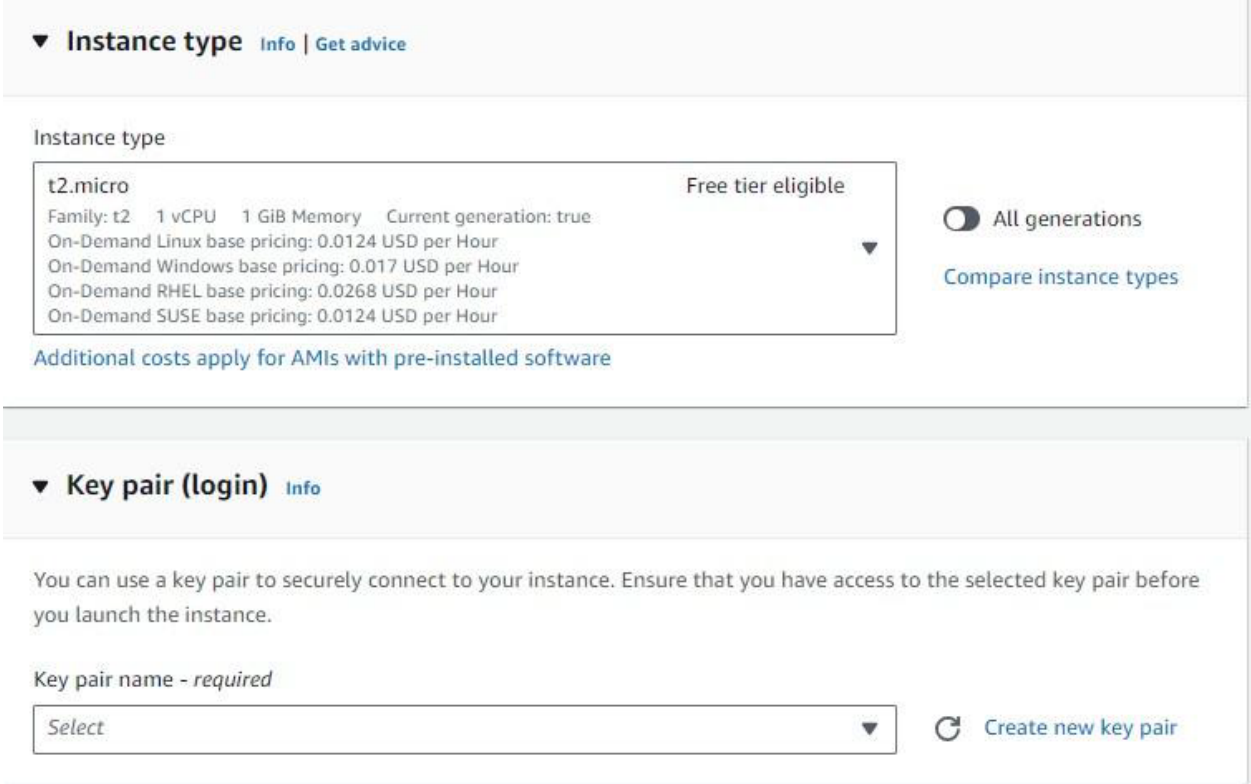


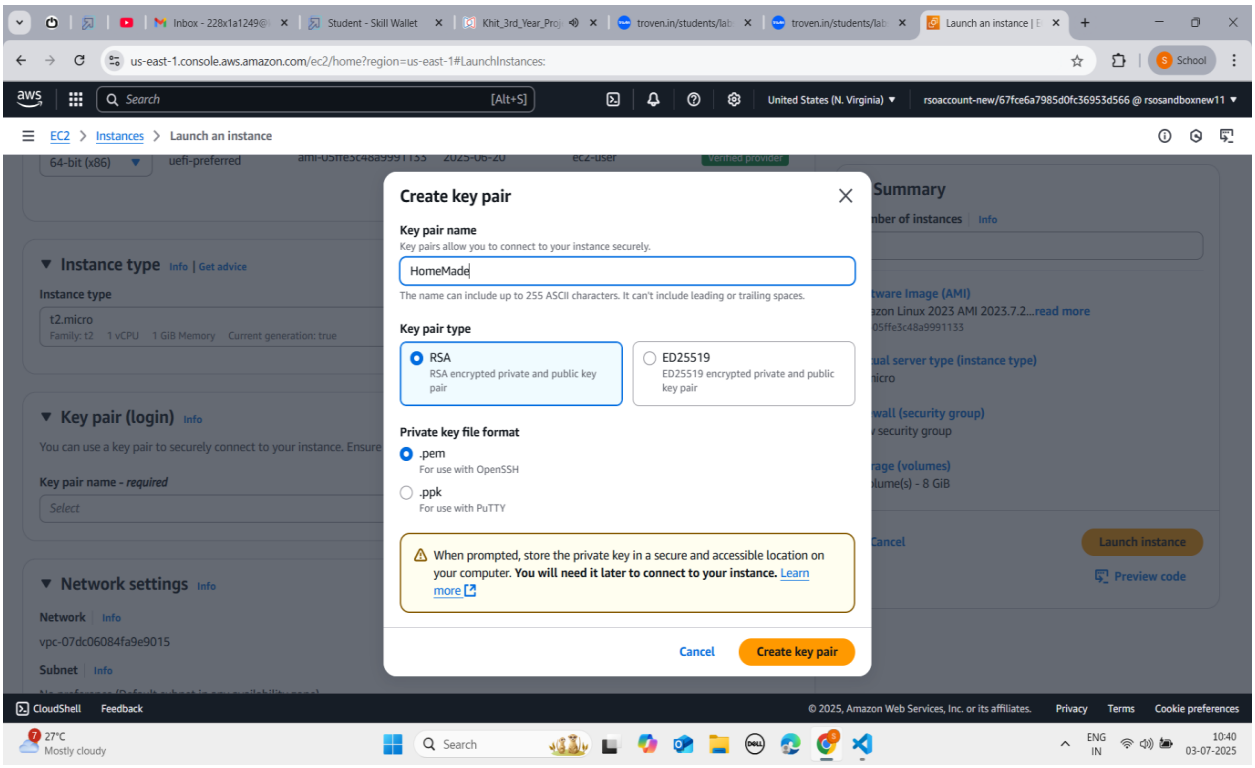


● Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



- Create and download the key pair for Server access.





- **Activity 6.2:**Configure security groups for HTTP, and SSH access.

Network settings Info

VPC - *required* Info

vpc-03cdc7b6f19dd7211 (default)
172.31.0.0/16

Subnet Info

No preference

Create new subnet

Auto-assign public IP Info

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

- (●) Create security group
- ( ) Select existing security group

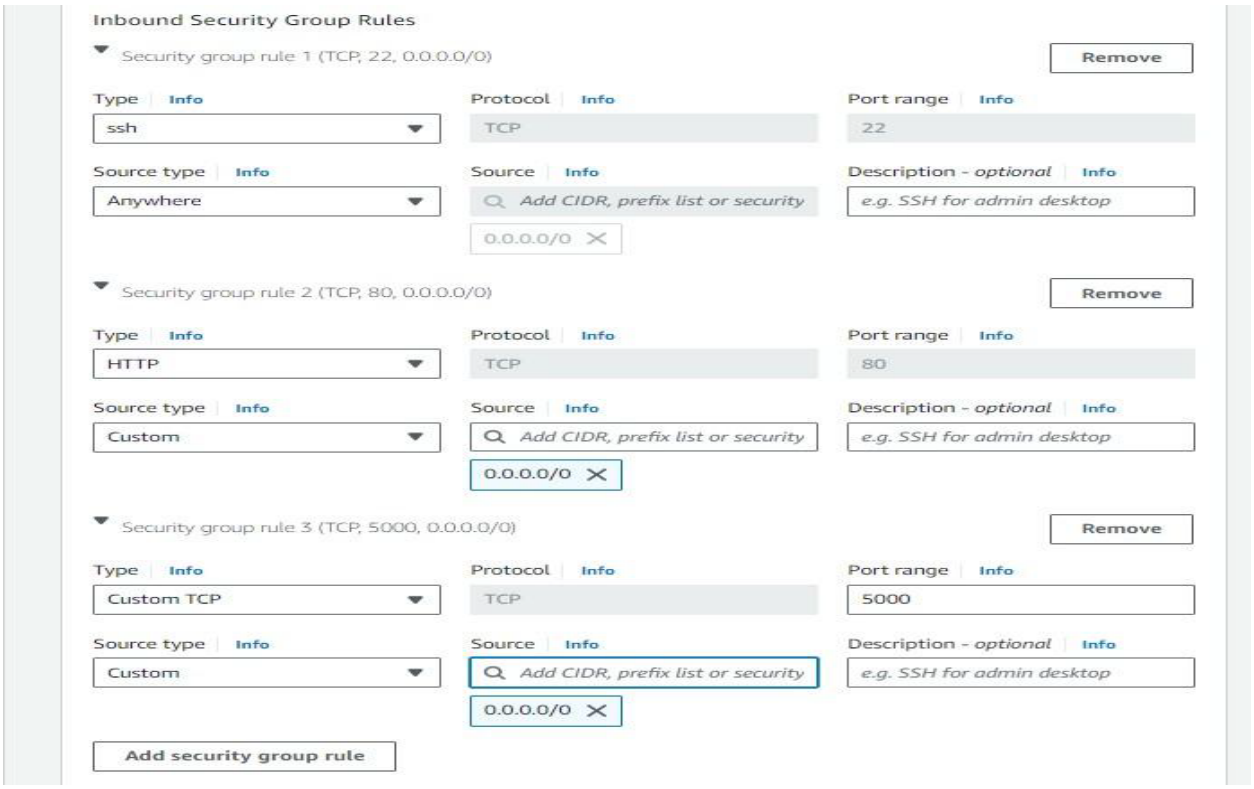
Security group name - *required*

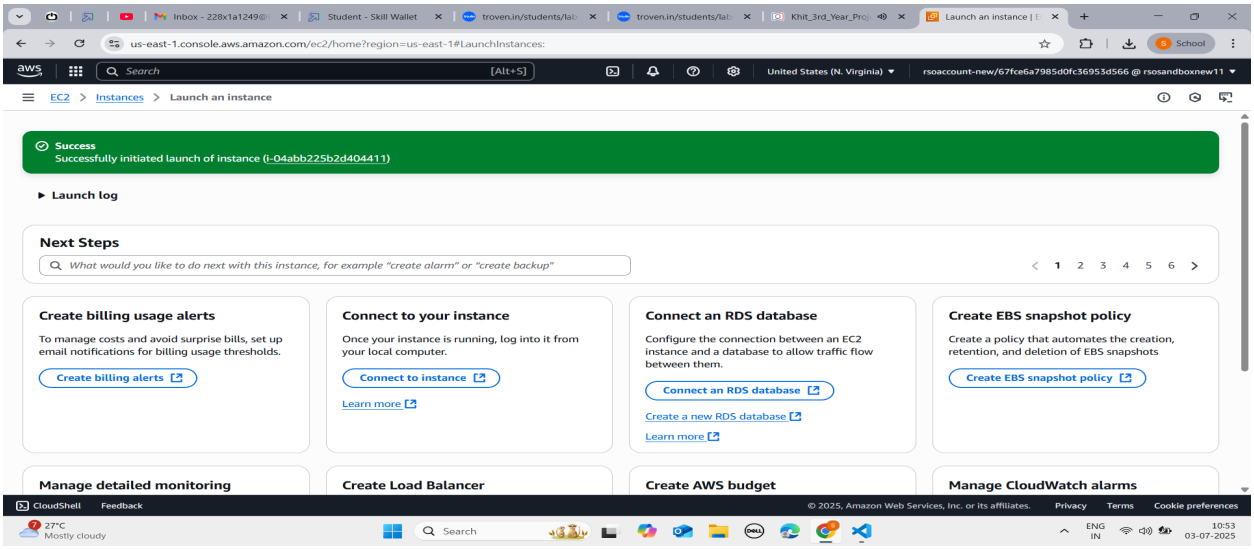
launch-wizard

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and ._-:/()#,@[]+=&;{}!$*

Description - *required* Info

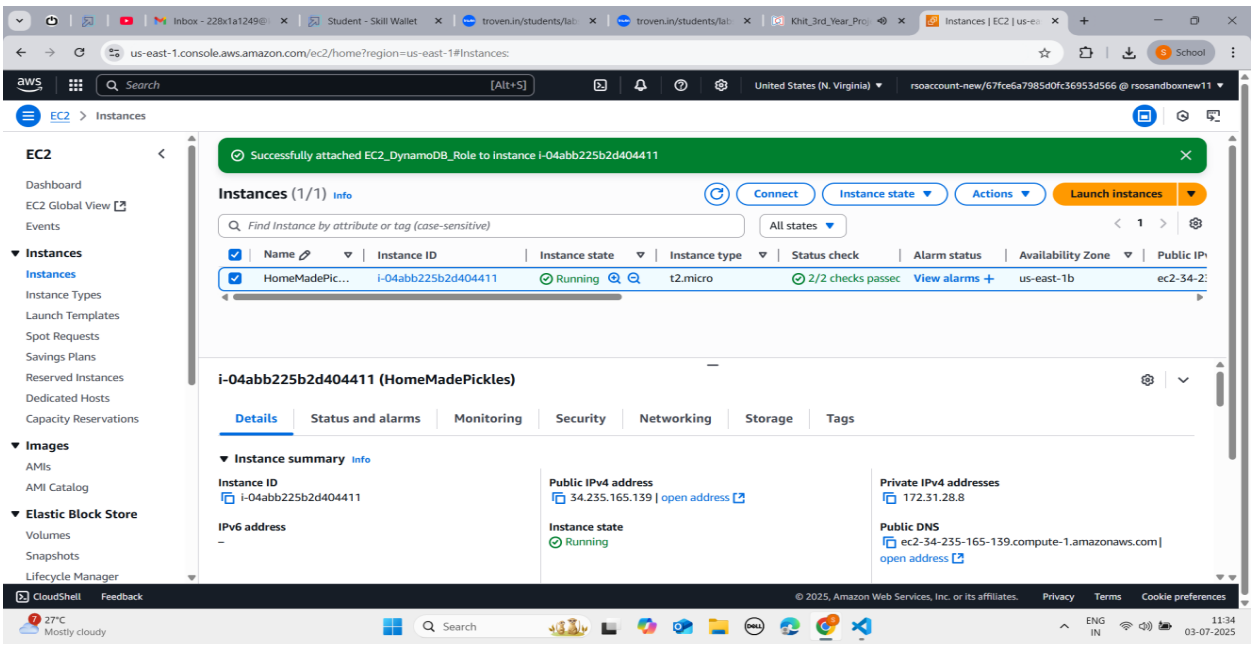
launch-wizard created 2024-10-13T17:49:56.622Z



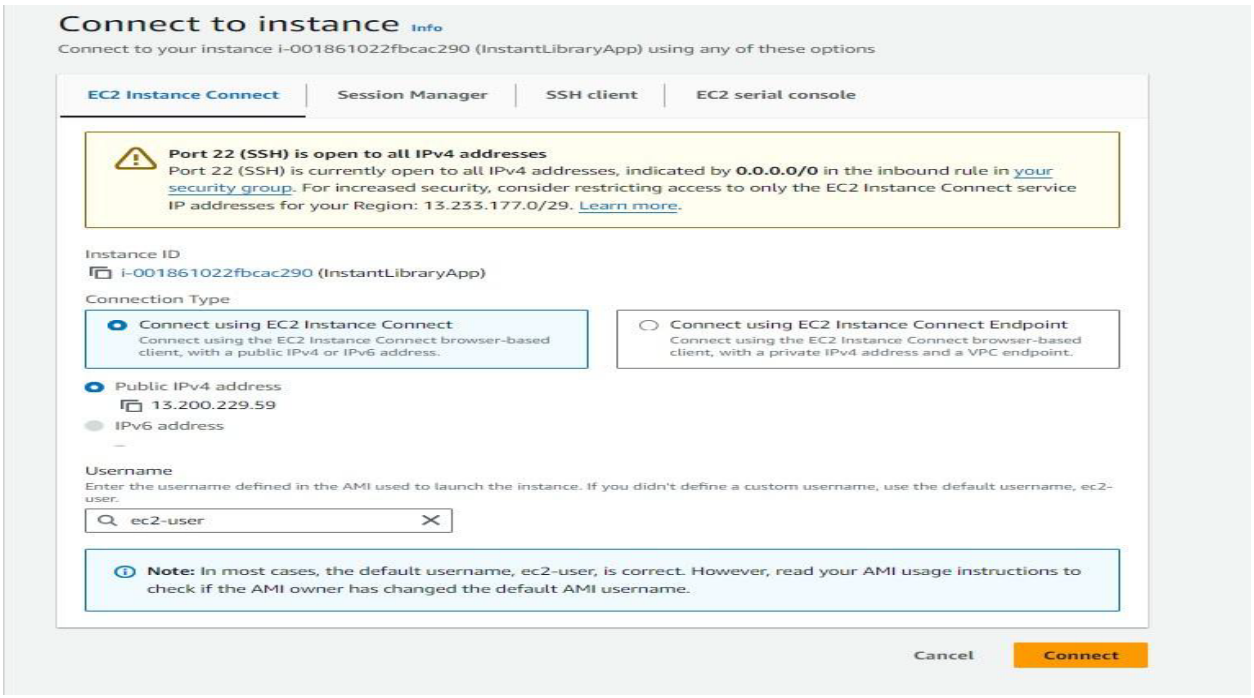


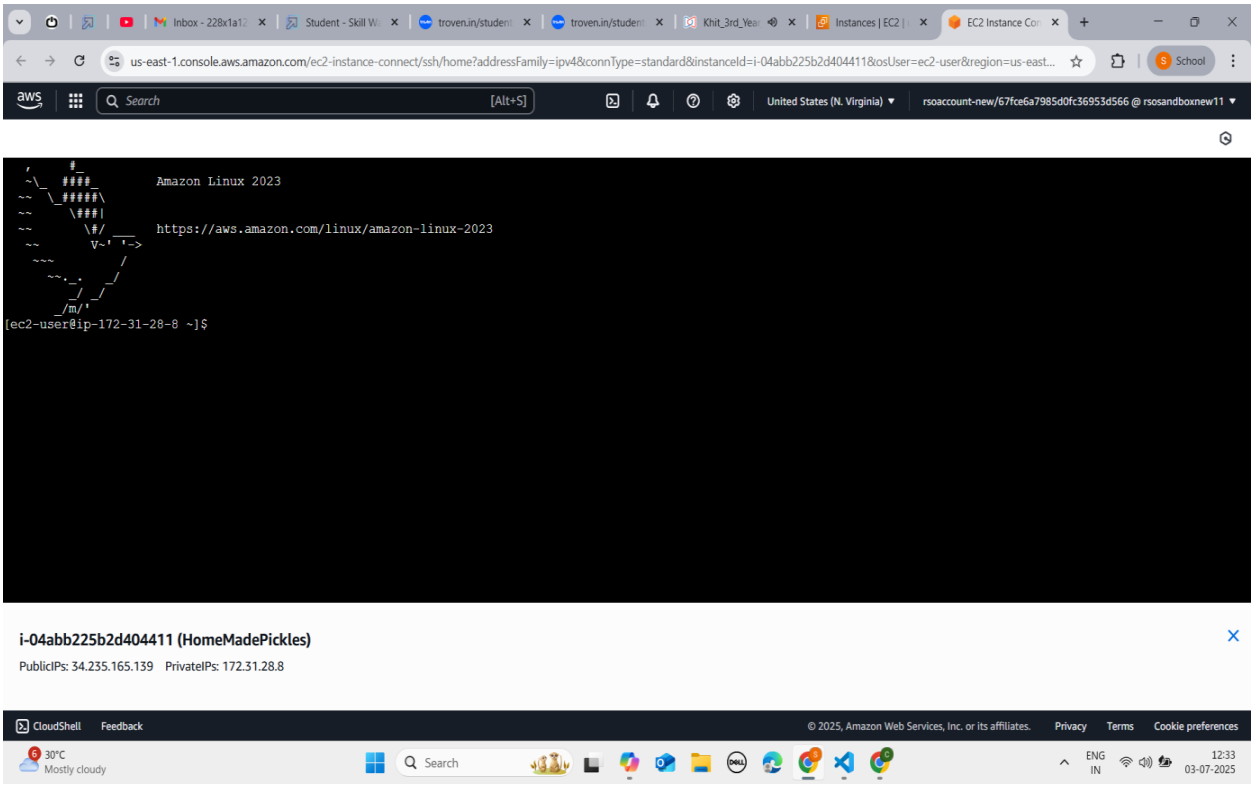
To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the

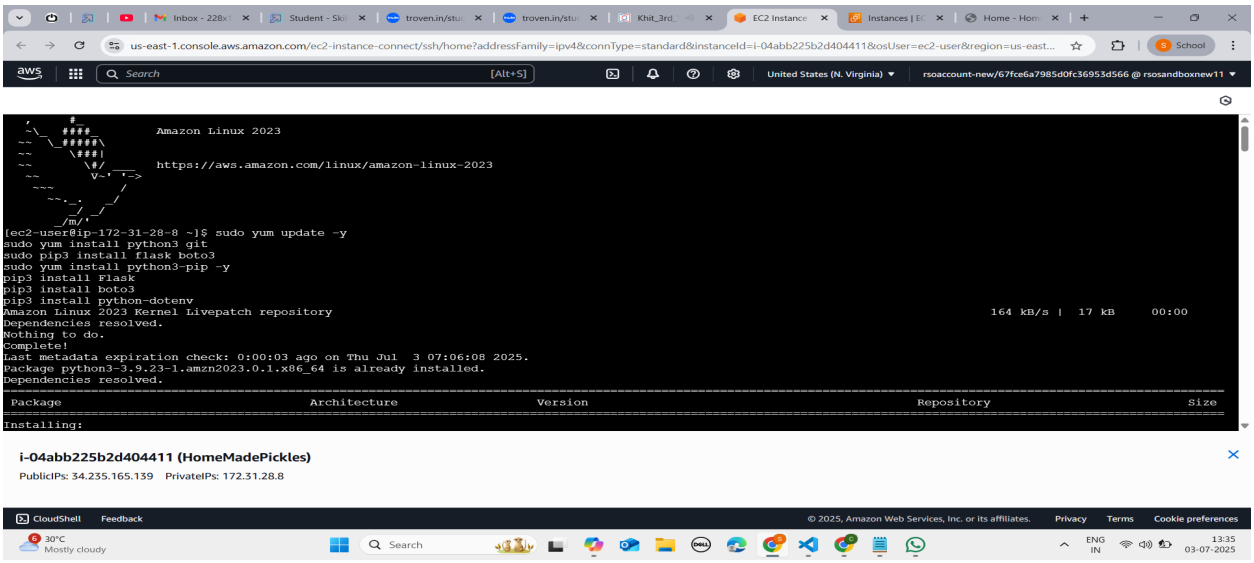
appropriate role. After the IAM role is connected, navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the EC2 Dashboard, click the Connect button. From the connection methods presented, choose EC2 Instance Connect. Finally, click Connect again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

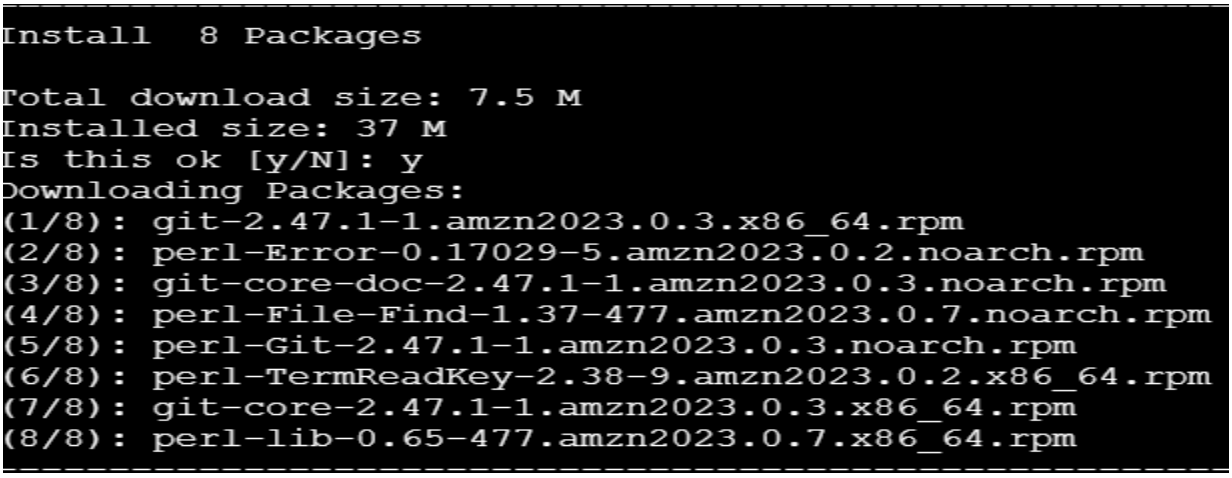


- Now connect the EC2 with the files


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


us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-04abb225b2d404411&osUser=ec2-user&region=us-east...
Search
[Alt+S]
United States (N. Virginia)
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
[ec2-user@ip-172-31-28-8 ~]$
i-04abb225b2d404411 (HomeMadePickles)
PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences





Milestone 7: Deployment on EC2

Activity 7.1: 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

sudo yum install python3-pip -y

pip3 install Flask
pip3 install boto3
pip3 install python-dotenv

Verify Installations:

flask --version git --version

Activity 7.2: Clone Your Flask Project from GitHub

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

Run: 'git clone https://github.com/Chandra123-123/Homemade.git'

● This will download your project to the EC2 instance.

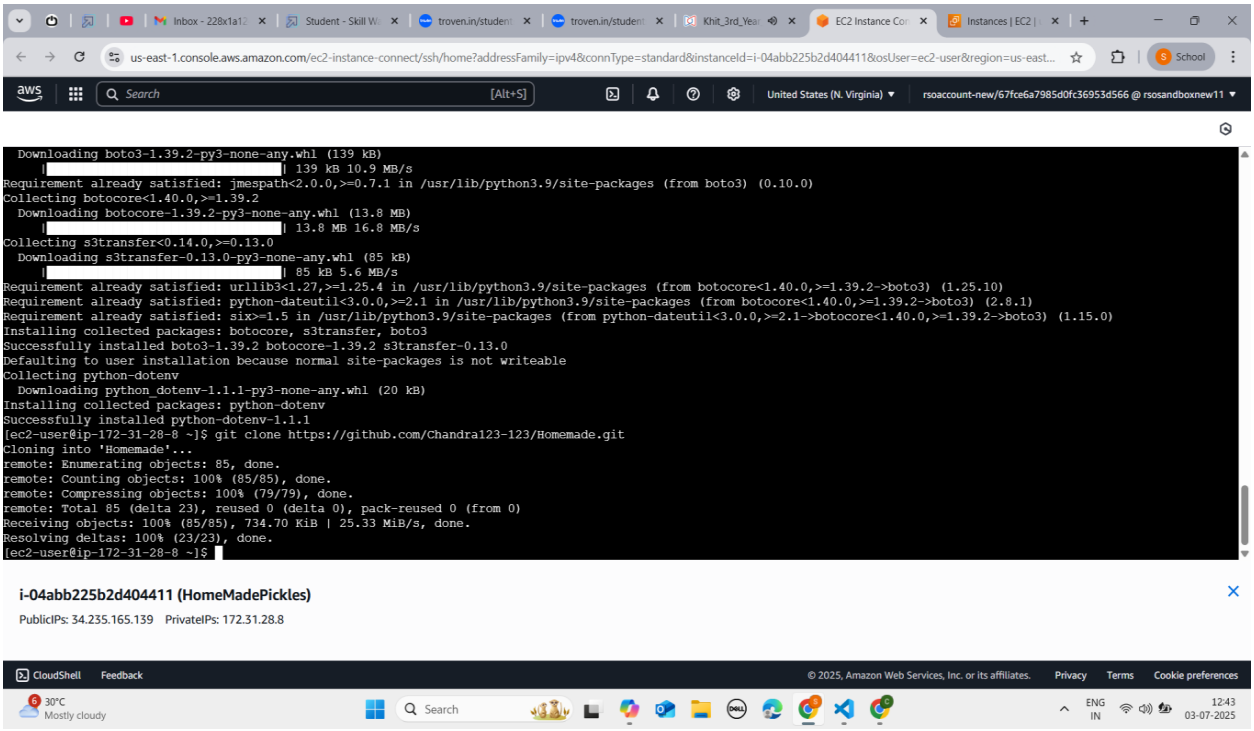
To navigate to the project directory, run the following command:

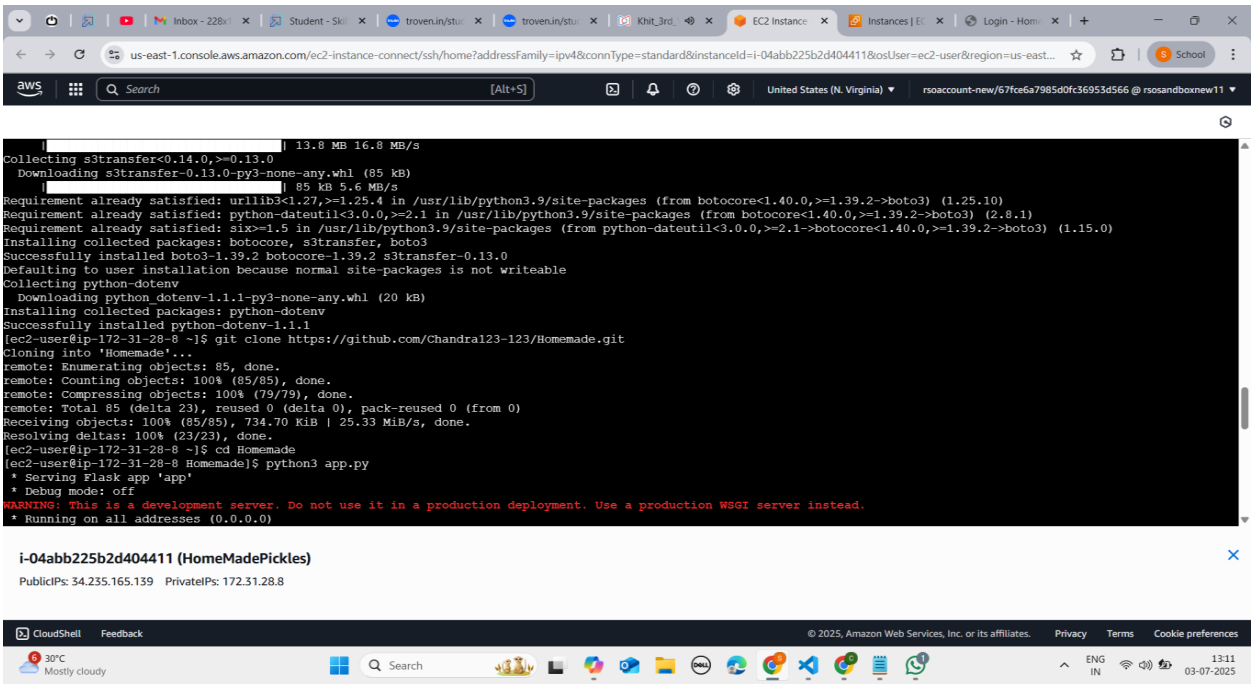
cd Homemade

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

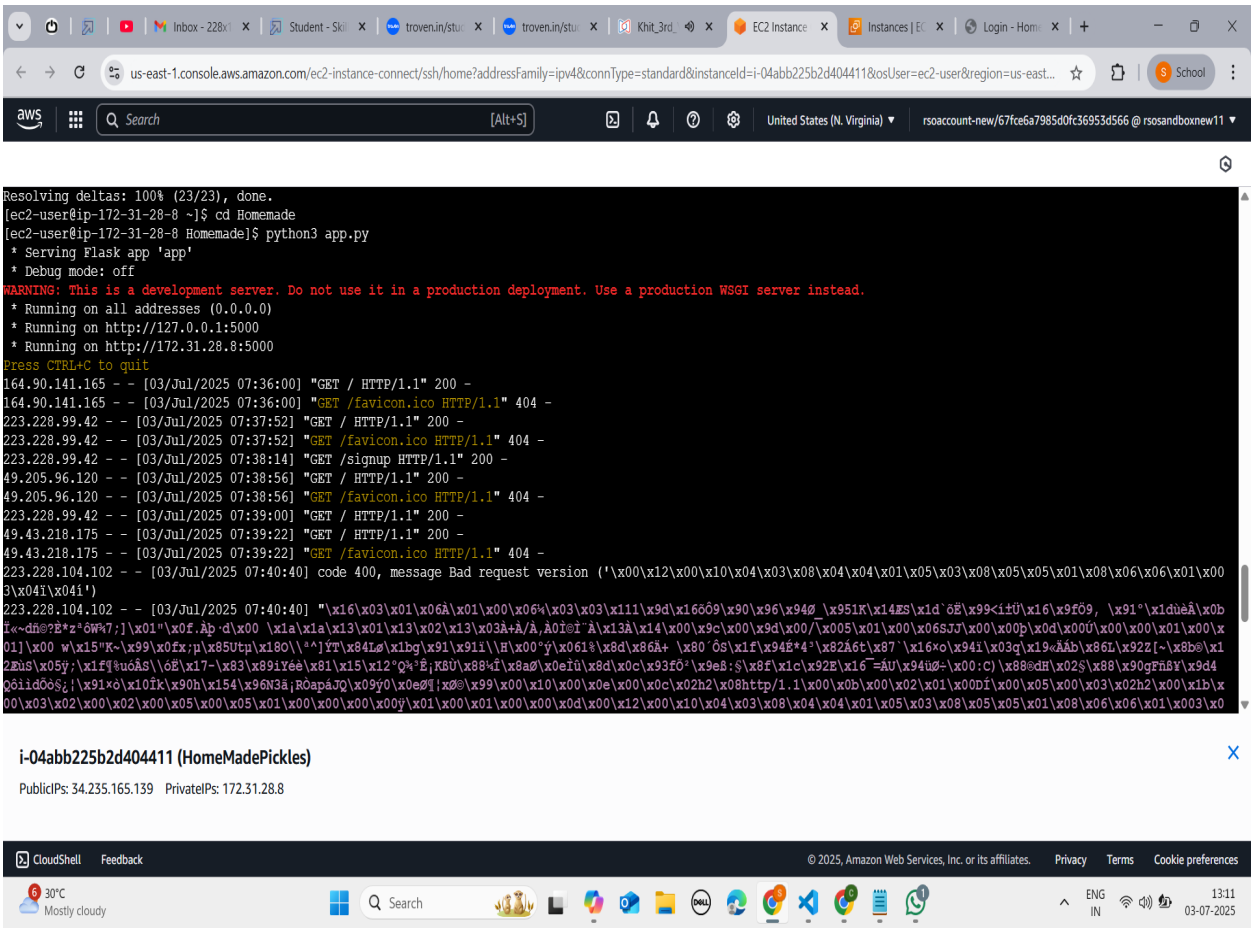


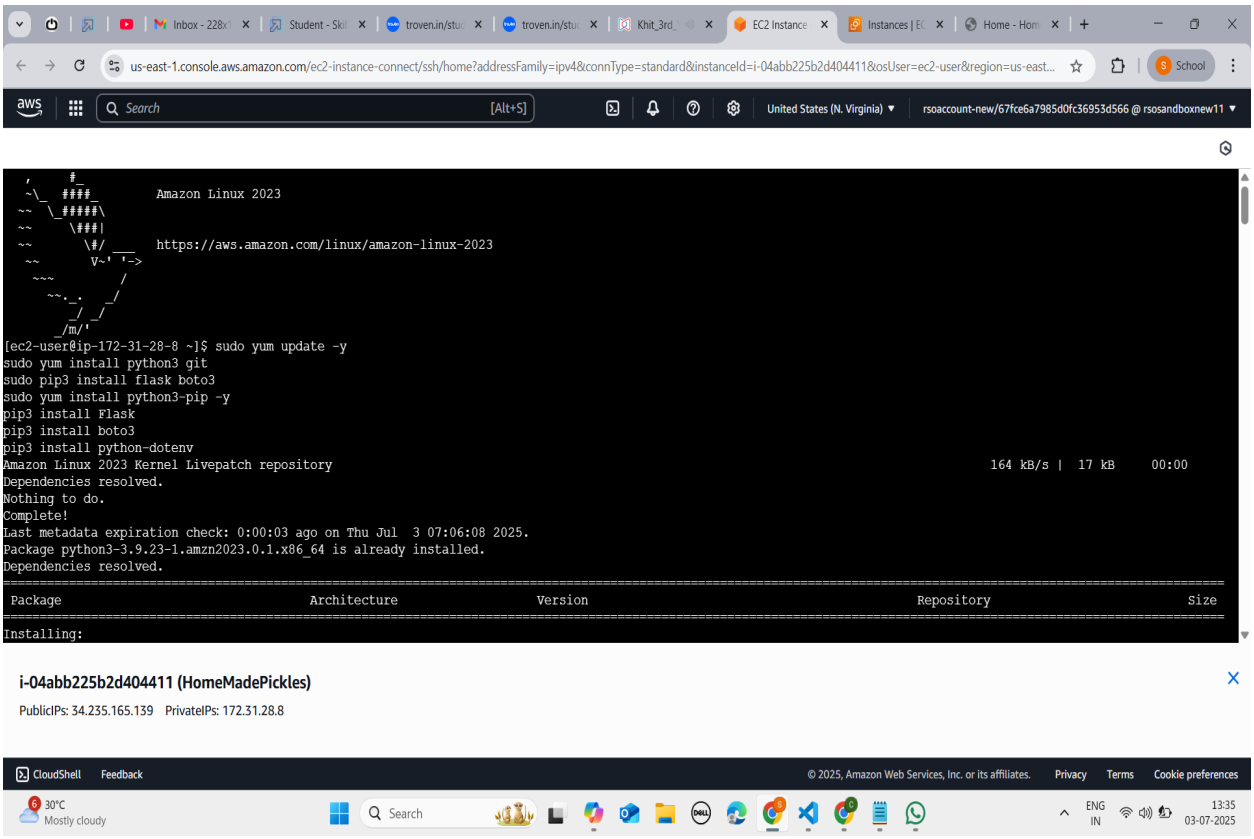



```
[ec2-user@ip-172-31-28-8 ~]$ cd Homemade
[ec2-user@ip-172-31-28-8 Homemade]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.28.8:5000
Press CTRL+C to quit
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET / HTTP/1.1" 200 -
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET /favicon.ico HTTP/1.1" 404 -
```

i-04abb225b2d404411 (HomeMadePickles)

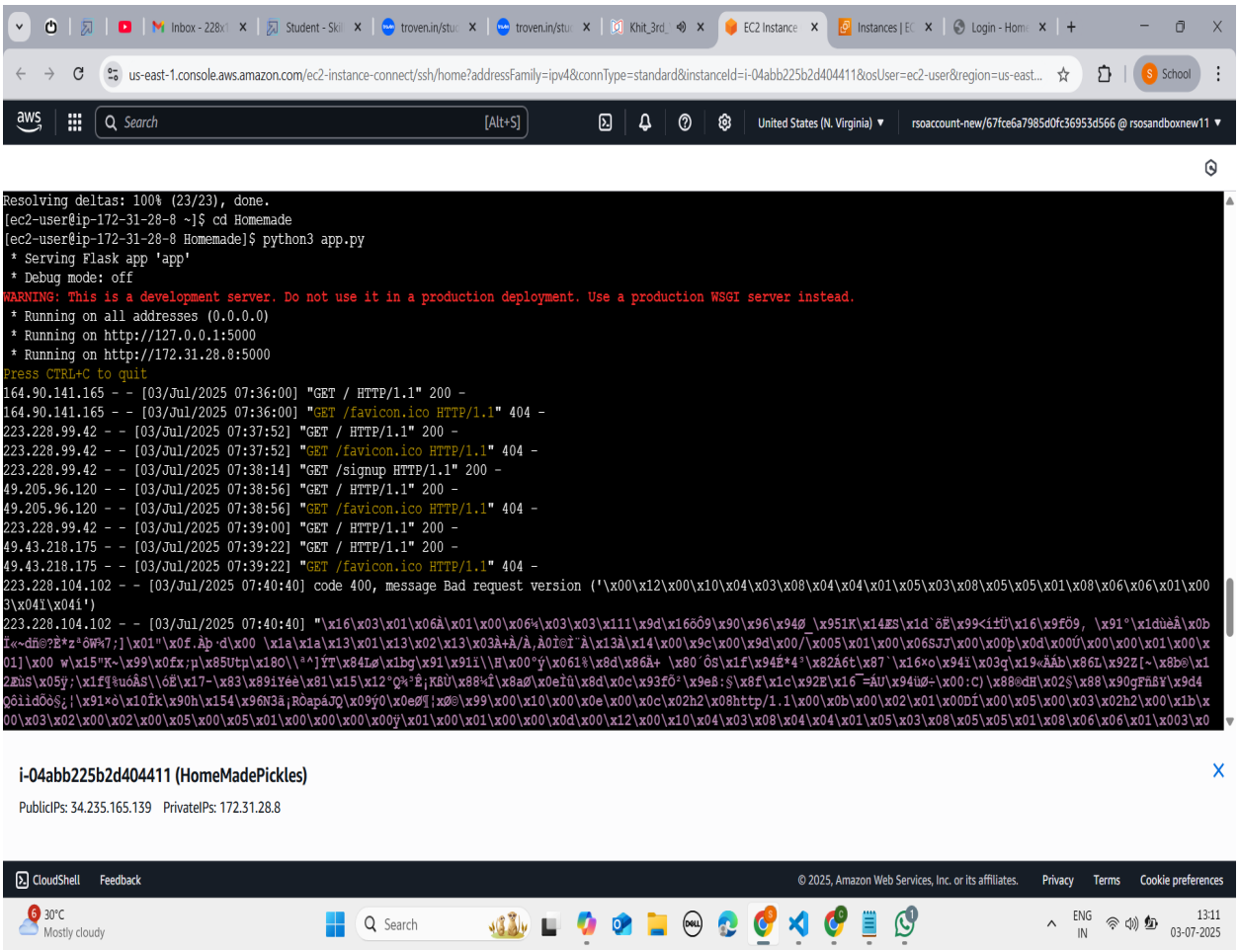
PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8


Inbox - 228x
Student - Skill
troven.in/stud
troven.in/stud
Khit_3rd_
EC2 Instance
Instances | EC
Login - Home
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-04abb225b2d404411&osUser=ec2-user&region=us-east...
School
aws
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew11
Resolving deltas: 100% (23/23), done.
[ec2-user@ip-172-31-28-8 ~]$ cd Homemade
[ec2-user@ip-172-31-28-8 Homemade]$ python3 app.py
* Serving Flask app 'app'
* 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://172.31.28.8:5000
Press CTRL+C to quit
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET / HTTP/1.1" 200 -
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET /favicon.ico HTTP/1.1" 404 -
223.228.99.42 - - [03/Jul/2025 07:37:52] "GET / HTTP/1.1" 200 -
223.228.99.42 - - [03/Jul/2025 07:37:52] "GET /favicon.ico HTTP/1.1" 404 -
223.228.99.42 - - [03/Jul/2025 07:38:14] "GET /signup HTTP/1.1" 200 -
49.205.96.120 - - [03/Jul/2025 07:38:56] "GET / HTTP/1.1" 200 -
49.205.96.120 - - [03/Jul/2025 07:38:56] "GET /favicon.ico HTTP/1.1" 404 -
223.228.99.42 - - [03/Jul/2025 07:39:00] "GET / HTTP/1.1" 200 -
49.43.218.175 - - [03/Jul/2025 07:39:22] "GET / HTTP/1.1" 200 -
49.43.218.175 - - [03/Jul/2025 07:39:22] "GET /favicon.ico HTTP/1.1" 404 -
223.228.104.102 - - [03/Jul/2025 07:40:40] code 400, message Bad request version ('\x00\x12\x00\x10\x04\x03\x08\x04\x04\x01\x05\x03\x08\x05\x05\x01\x08\x06\x06\x01\x003\x04ï\x04í')
223.228.104.102 - - [03/Jul/2025 07:40:40] "\x16\x03\x01\x06À\x01\x00\x06¾\x03\x03\x111\x9d\x166Ô9\x90\x96\x94Ø _\x951K\x14ŒS\x1d`öË\x99<i‡Ú\x16\x9fÓ9, \x91°\x1dùèÂ\x0bÍ«~dñ@²Ë*z³ôW℀7;]\x01"\x0f.Àþ·d\x00 \x1a\x1a\x13\x01\x13\x02\x13\x03À+À/À,ÀÒÌ©Ì¯À\x13À\x14\x00\x9c\x00\x9d\x00/¯\x005\x01\x00\x06SJJ\x00\x00þ\x0d\x00Ú\x00\x00\x01\x00\x01\x00 w\x15"K~\x99\x0fx;µ\x85Utµ\x18O\\ª^]ŸT\x84Lø\x1bg\x91\x91i\\H\x00°ý\x061¾\x8d\x86Ã+ \x80´Ôš\x1f\x94É*4³\x82Á6t\x87`\x16×o\x94ï\x03q\x19«ÃÁb\x86L\x92Z[~\x8b®\x12ŒùS\x05ÿ;\x1f¶¾uóÂS\\óË\x17-\x83\x89iÝéè\x81\x15\x12°Q¾³Ê¡K§Ù\x88¼Î\x8aØ\x0eÌù\x8d\x0c\x93fÔ²\x9eß:§\x8f\x1c\x92E\x16¯=ÁU\x94üØ÷\x00:C)\x88®dH\x02§\x88\x90gFñ¥\x9d4ÇôiidÔò§¿¦\x91×ò\x10ÎÌk\x90h\x154\x96N3ã¡RÒapâJQ\x09ý0\x0eØ¶¦xØ®\x99\x00\x10\x00\x0e\x00\x0c\x02h2\x08http/1.1\x00\x0b\x00\x02\x01\x00DÍ\x00\x05\x00\x03\x02h2\x00\x1b\x00\x03\x02\x00\x02\x00\x05\x00\x05\x01\x00\x00\x00\x00ÿ\x01\x00\x01\x00\x00\x0d\x00\x12\x00\x10\x04\x03\x08\x04\x04\x01\x05\x03\x08\x05\x05\x01\x08\x06\x06\x01\x003\x0
i-04abb225b2d404411 (HomeMadePickles)
PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
30°C
Mostly cloudy
Search
ENG
IN
13:11
03-07-2025



Verify the Flask app is running: http://34.235.165.139

○ Run the Flask app on the EC2 instance

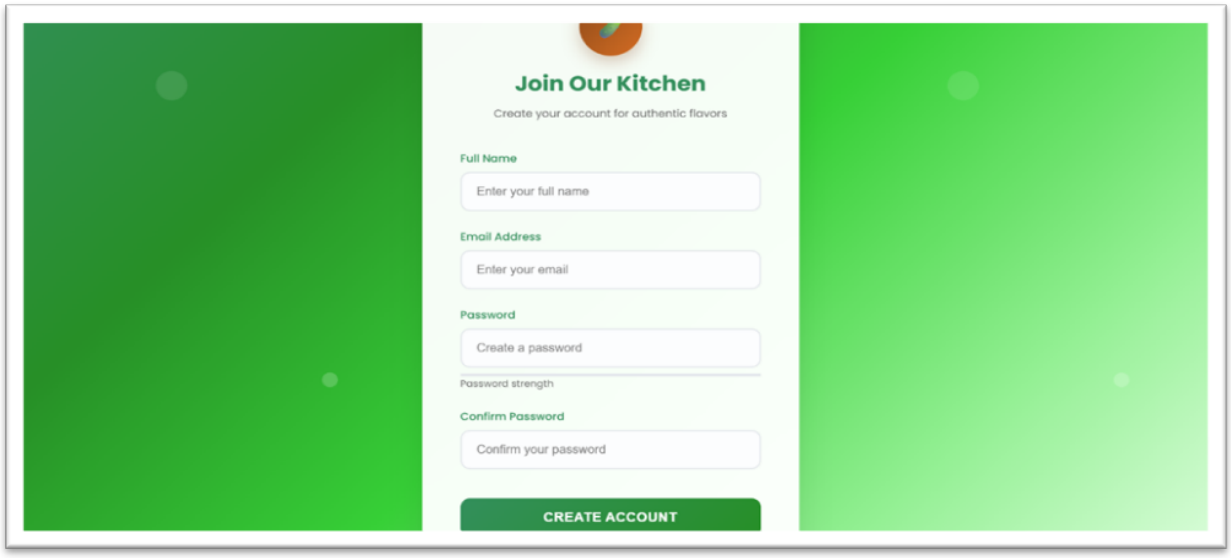


Access the website through:
Public IPs: http://34.235.165.139

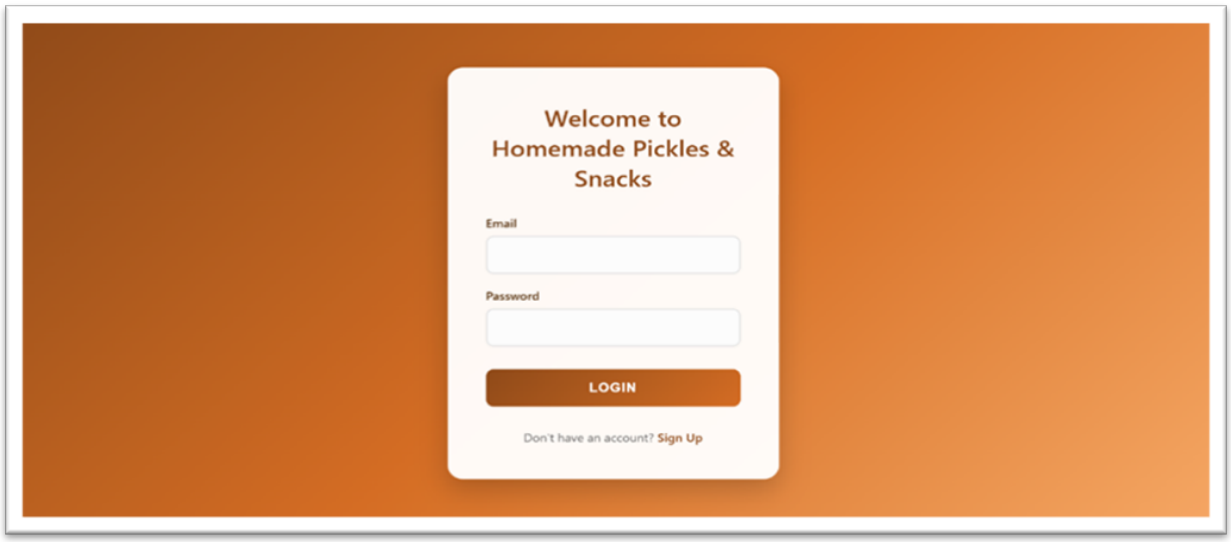
Milestone 8: Testing and Deployment

● **Activity 8.1:** Conduct functional testing to verify user signup, login, pickles requests, and notifications.

Signup Page :

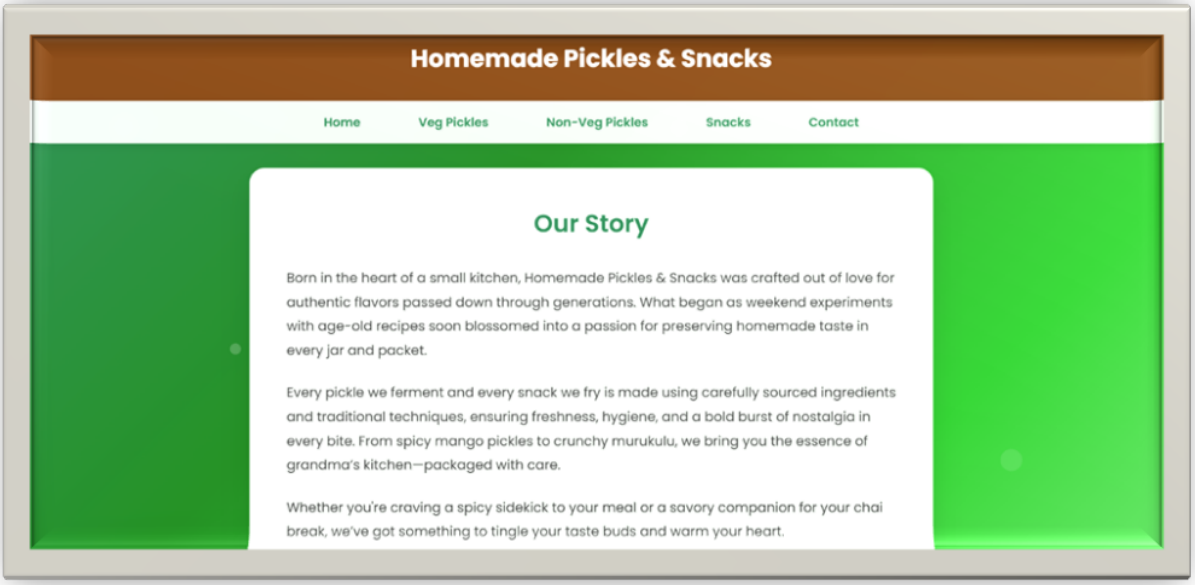


Login Page :

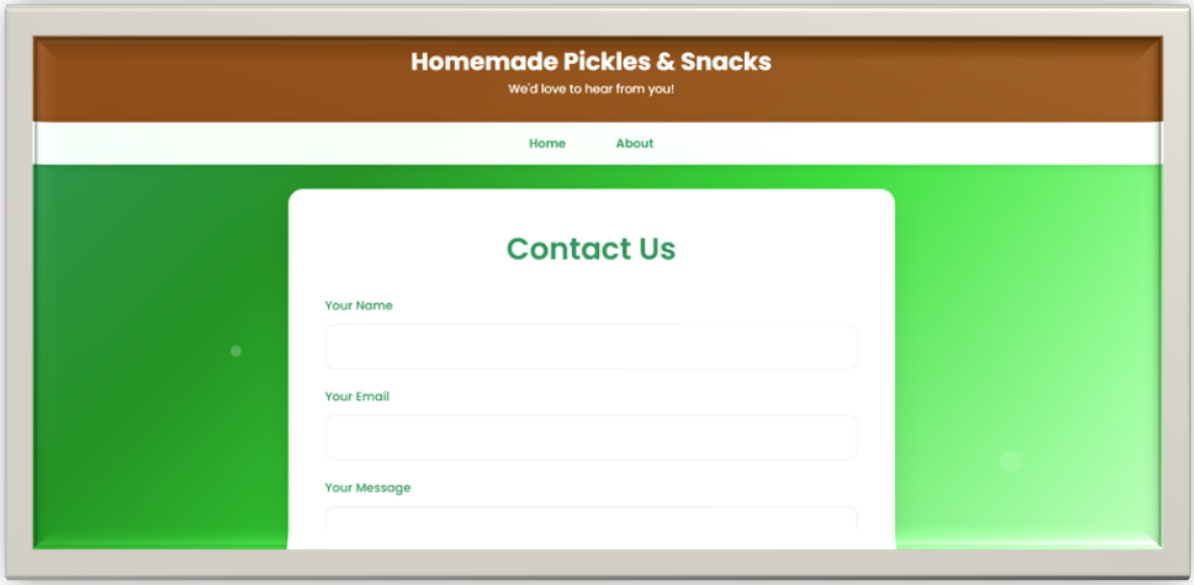


Home page :

About_Us page:

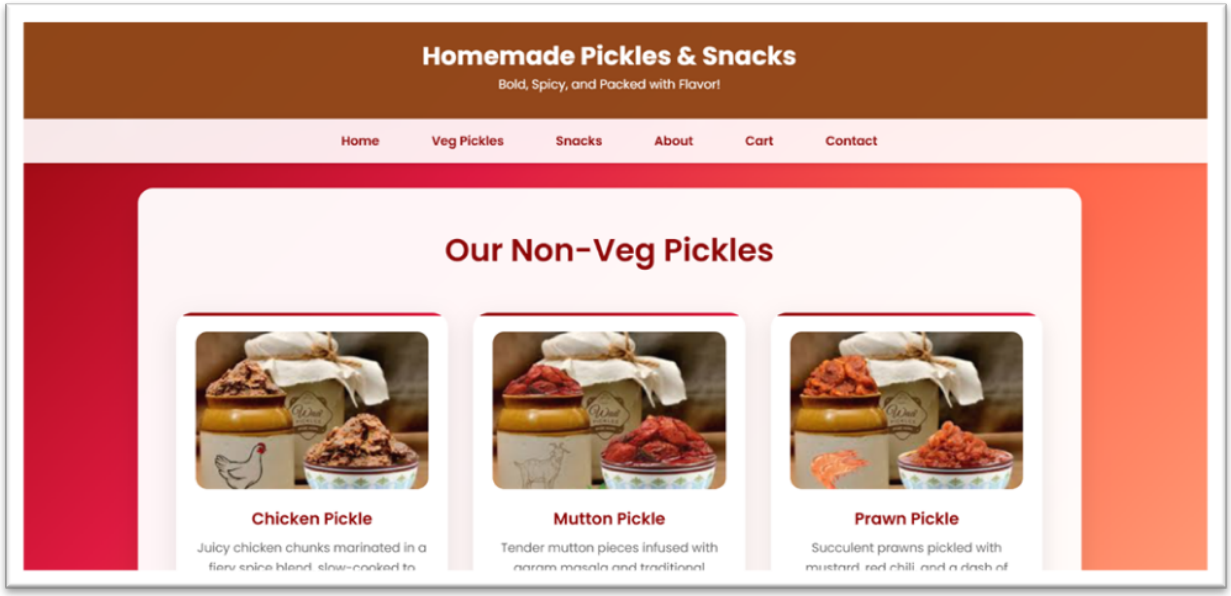


Contact Page:

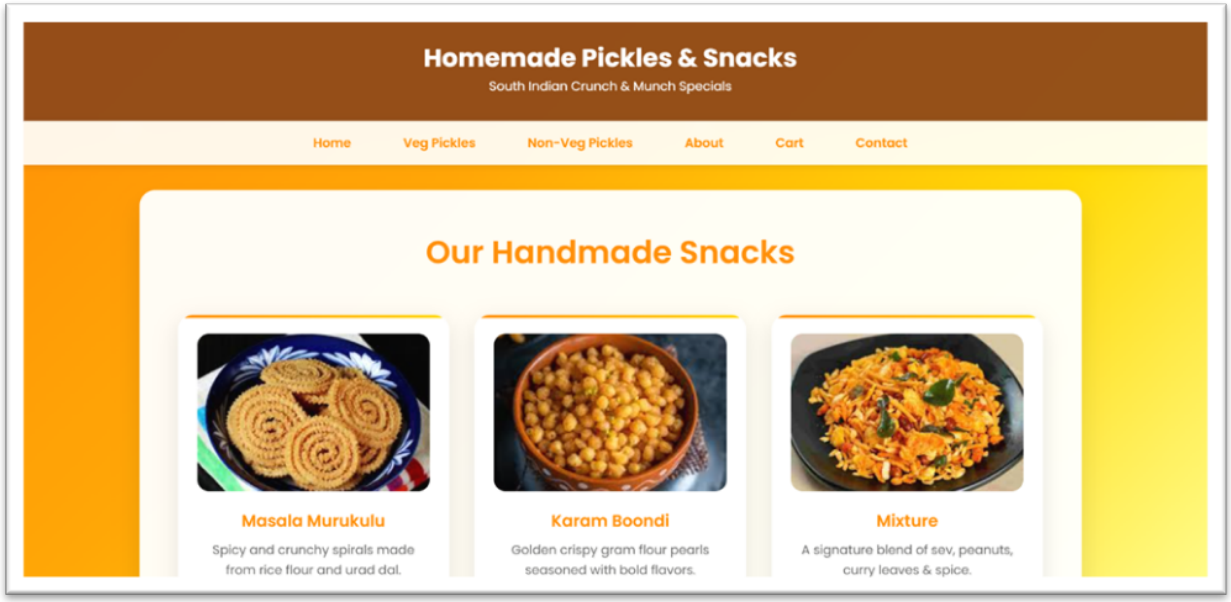


Veg-Pickles page :

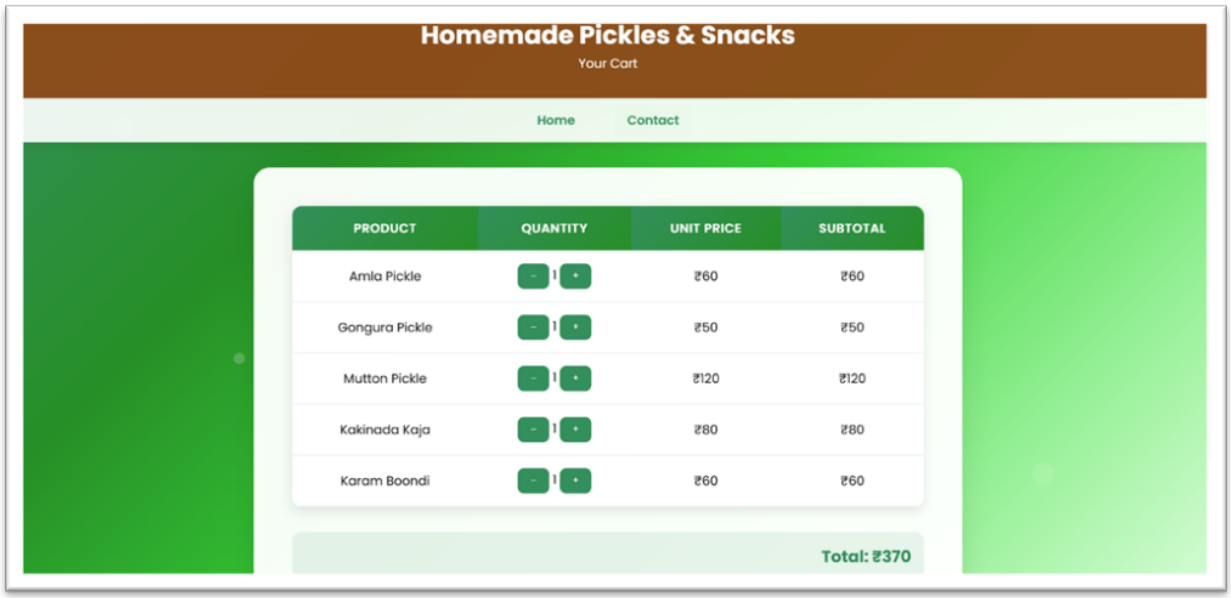
Non-Veg-Pickles page:



Snacks Page :

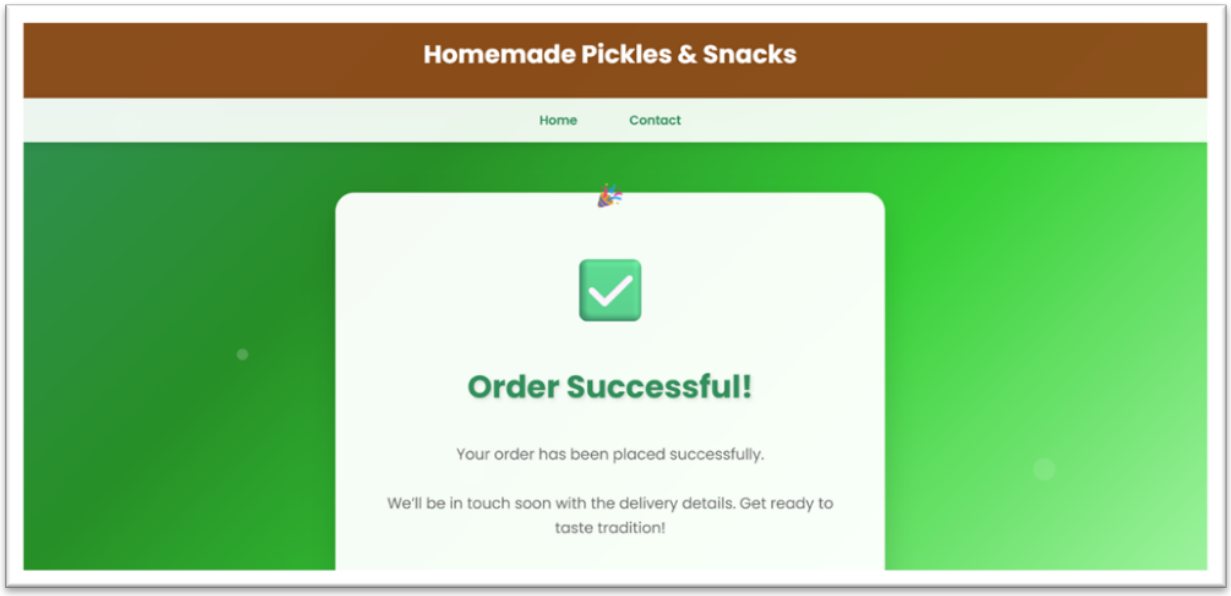


Cart Page:



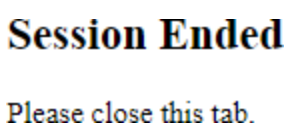
Check-out Page:

Success Page:



Your order has been placed successfully! We Will get backto you soon

Exit:



Conclusion:

The Homemade Pickles and Snacks Website has been successfully developed and deployed using a robust cloud-native architecture to ensure high performance, scalability, and customer satisfaction. By leveraging **AWS EC2** for reliable hosting, **DynamoDB** for secure product and order data management, and **SNS** for real-time customer and staff notifications, the platform delivers a seamless end-to-end experience for home made food lovers. This system addresses the need for efficient online access to traditional food offerings,allowing customers to conveniently browse, order, and track their favourite pickles and snacks. The cloud infrastructure enables the site to scale effortlessly during high-demand periods like festive seasons or promotions, without compromising responsiveness or user experience.

The integration of **Flask with AWS services** ensures that backend operations—such as order processing, inventory updates, and customer messaging—function smoothly in real time. Thorough testing has validated the stability of all core features, from product browsing and secure checkout to order confirmation notifications. In conclusion, this website serves as a modern platform that blends tradition with technology, providing a delightful shopping experience while streamlining business operations. It stands as a strong example of how cloud-based solutions can elevate local, handmade products to a broader digital audience