HomeMade Pickles & Snacks: Taste the Best Resources

Project Description:

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

Scenario 1: Scalable Order Management for High Demand

A cloud-based system ensures seamless order processing during peak user activity. For instance, during a promotional event, hundreds of users simultaneously access the platform to place orders. The backend efficiently processes requests, updates inventory in real-time, and manages user sessions. The cloud infrastructure handles traffic spikes without performance degradation, ensuring smooth transactions and minimizing wait times.

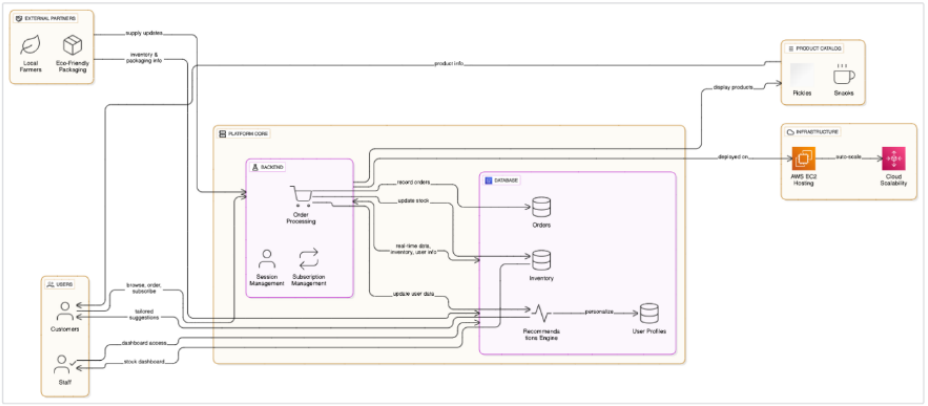
Scenario 2: Real-Time Inventory Tracking and Updates

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

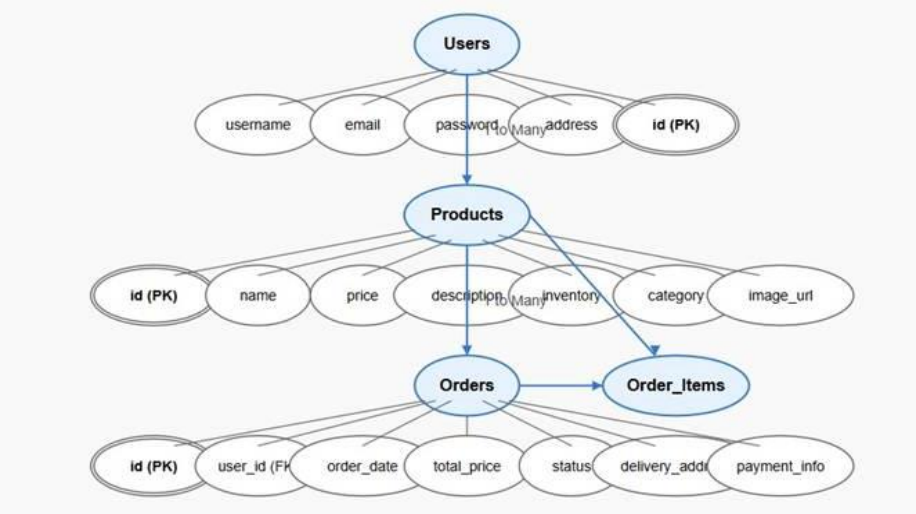
Scenario 3: Personalized User Experience and Recommendations

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

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Pre-requisites:

1. AWS Account Setup: AWS Account Setup
2. AWS IAM (Identity and Access Management): IAM Overview
3. AWS EC2 (Elastic Compute Cloud): EC2 Tutorial
4. AWS DynamoDB: DynamoDB Introduction
5. Git Documentation: GIT Documentation
6. VS Code Installation: Download Visual Studio Code

Project WorkFlow:

Milestone 1. Backend Development and Application Setup

- Develop the Backend Using Flask.
- Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login

- Set up an AWS account if not already done.
- Log in to the AWS Management Console

Milestone 3. DynamoDB Database Creation and Setup

- Create a DynamoDB Table.
- Configure Attributes for User Data and Book Requests.

Milestone 4. SNS Notification Setup

- Create SNS topics for book request notifications.
- Subscribe users and library staff to SNS email notifications.

Milestone 5. IAM Role Setup

- Create IAM Role
- Attach Policies

Milestone 6. EC2 Instance Setup

- Launch an EC2 instance to host the Flask application.
- Configure security groups for HTTP, and SSH access.

Milestone 7. Deployment on EC2

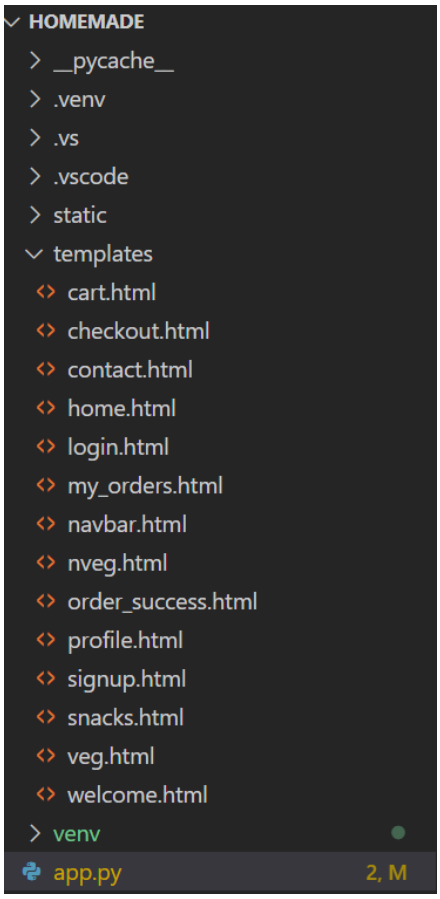
- Upload Flask Files
- Run the Flask App

Milestone 8. Testing and Deployment

- Conduct functional testing to verify user signup, login, buy/sell stocks and notifications.

Milestone 1:Backend Development and Application Setup

- **Activity 1.1: Develop the backend using Flask**
    - File Explorer Structure



Description: set up the Home Made pickles and snacks project with an app.py file, a static/ folder for assets, and a templates/ directory containing all required HTML pages like home, login, register, subject-specific pages (e.g.,veg_pickles.html, non_veg_pickles.html), and utility pages (e.g., checkout-form.html, profile.html).

Description of the code :

- **Flask App Initialization**

```
from flask import Flask, render_template, request, redirect, url_for
import boto3
from boto3.dynamodb.conditions import Key
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from bcrypt import hashpw, gensalt, checkpw
```

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 setup
AWS_REGION = os.environ.get('AWS_REGION', 'us-east-1')
dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION)
users_table = dynamodb.Table('users')
carts_table = dynamodb.Table('carts')
orders_table = dynamodb.Table('orders')
```

Description: initialize the DynamoDB resource for the ap-south-1 region and set up access to the Users and Requests tables for storing user details and order requests.

- **SNS Connection**

```
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587
SENDER_EMAIL = os.environ.get('SENDER_EMAIL', "dileep6@gmail.com")
SENDER_PASSWORD = os.environ.get('SENDER_PASSWORD', "vcbt qcsp yvgi dyhn")

def send_email(to_email, subject, body):
    msg = MIMEMultipart()
    msg['From'] = SENDER_EMAIL
    msg['To'] = to_email
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'plain'))
    try:
        import smtplib
        server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        server.starttls()
        server.login(SENDER_EMAIL, SENDER_PASSWORD)
        text = msg.as_string()
        server.sendmail(SENDER_EMAIL, to_email, text)
        server.quit()
        print("Email sent successfully")
    except Exception as e:
        print(f"Failed to send email: {e}")
```

Description: Configure **SNS** to send notifications when a book 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.) and enter the subscribed email in the SENDER_EMAIL section. 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():
    if 'user' in session:
        return render_template('home.html', username=session['user'])
    return redirect(url_for('login'))
```

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

- **Signup Route:**

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        hashed_password = generate_password_hash(password)
        try:
            c.execute('INSERT INTO users (username, email, password) VALUES (?, ?, ?)',
                      (username, email, hashed_password))
            conn.commit()
            flash('Signup successful! Please log in.')
            return redirect(url_for('login'))
        except sqlite3.IntegrityError:
            error = 'Username or email already exists.'
            return render_template('signup.html', error=error)
    return render_template('signup.html')
```

Description: define /register 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 routing:**

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        c.execute('SELECT password FROM users WHERE username = ?', (username,))
        user = c.fetchone()
        if user and check_password_hash(user[0], password):
            session['user'] = username
            return redirect(url_for('home'))
        else:
            return render_template('login.html', error='Invalid credentials')
    return render_template('login.html')
```

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

Add Cart routes

```
@app.route('/add_to_cart', methods=['POST'])
def add_to_cart():
    if 'user' not in session:
        flash("You need to login first")
        return redirect(url_for('login'))

    username = session['user']
    name = request.form['name']
    price = int(request.form['price'])
    image = request.form['image']

    # Check if item already in cart
    carts_c.execute('SELECT qty FROM carts WHERE username = ? AND name = ?', (username, name))
    row = carts_c.fetchone()
    if row:
        carts_c.execute('UPDATE carts SET qty = qty + 1 WHERE username = ? AND name = ?', (username, name))
    else:
        carts_c.execute('INSERT INTO carts (username, name, price, image, qty) VALUES (?, ?, ?, ?, ?)',
                        (username, name, price, image, 1))
    carts_conn.commit()
    flash(f"{name} added to cart!")
    return redirect(request.referrer or url_for('home'))
```

Checkout Routes:

```
@app.route('/checkout', methods=['GET', 'POST'])
def checkout():
    if 'user' not in session:
        flash("You must be logged in to checkout.")
        return redirect(url_for('login'))

    if request.method == 'POST':
        name = request.form['name']
        address = request.form['address']
        phone = request.form['phone']
        payment = request.form['payment']

        if not name or not address or not phone or not payment:
            error = "All fields are required."
            return render_template('checkout.html', error=error)

        username = session['user']
        carts_c.execute('SELECT name, price, image, qty FROM carts WHERE username = ?', (username,))
        cart = [
            {'name': row[0], 'price': row[1], 'image': row[2], 'qty': row[3]}
            for row in carts_c.fetchall()
        ]
        for item in cart:
            orders_c.execute('INSERT INTO orders (username, name, price, image, qty) VALUES (?, ?, ?, ?, ?)',
                             (username, item['name'], item['price'], item['image'], item['qty']))
        orders_conn.commit()
        # Clear user's cart after order
        carts_c.execute('DELETE FROM carts WHERE username = ?', (username,))
        carts_conn.commit()
        return redirect(url_for('order_success'))

    return render_template('checkout.html')
```

Description: These are the image routes are used to fall back in the local host for the static images in the static folder .

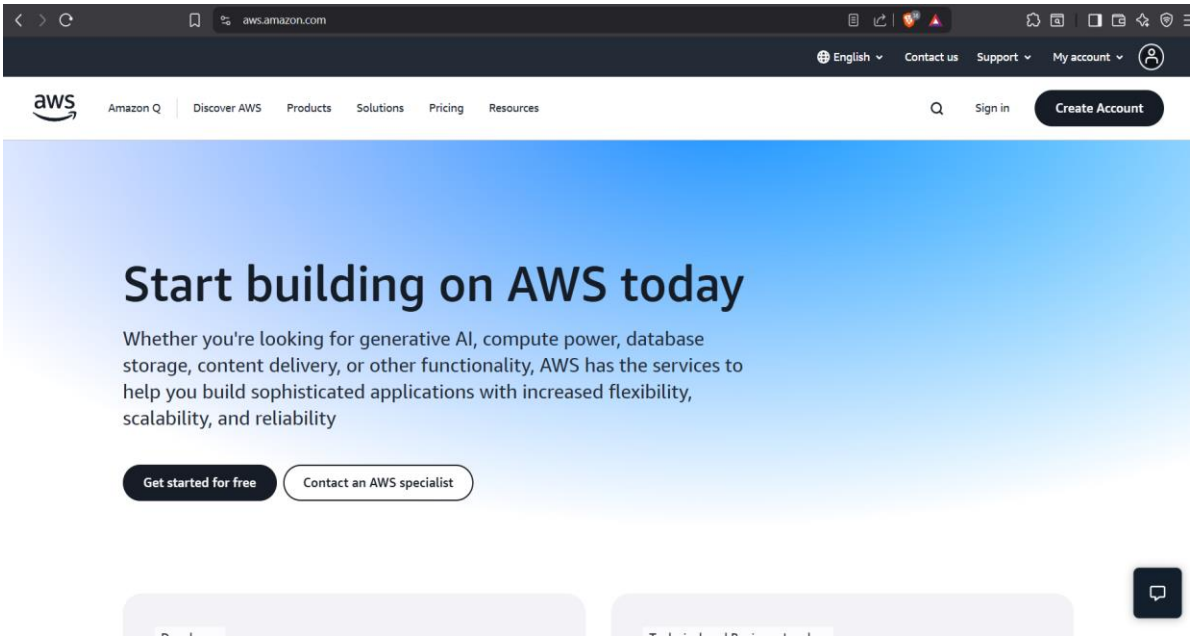
Deployment Code:

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

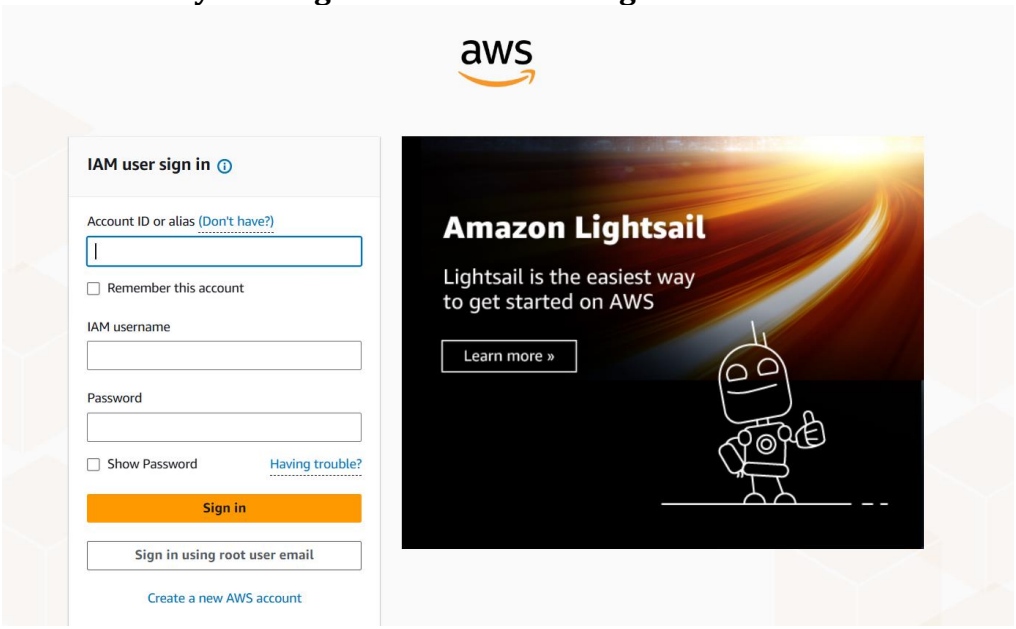
Description: start the Flask server to listen on all network interfaces (`0.0.0.0`) at port `80` with debug mode enabled for development and testing.

Milestone 2: AWS Account Setup and Login

- **Activity 2.1: Set up an AWS account if not already done.**
  - Sign up for an AWS account and configure billing settings.



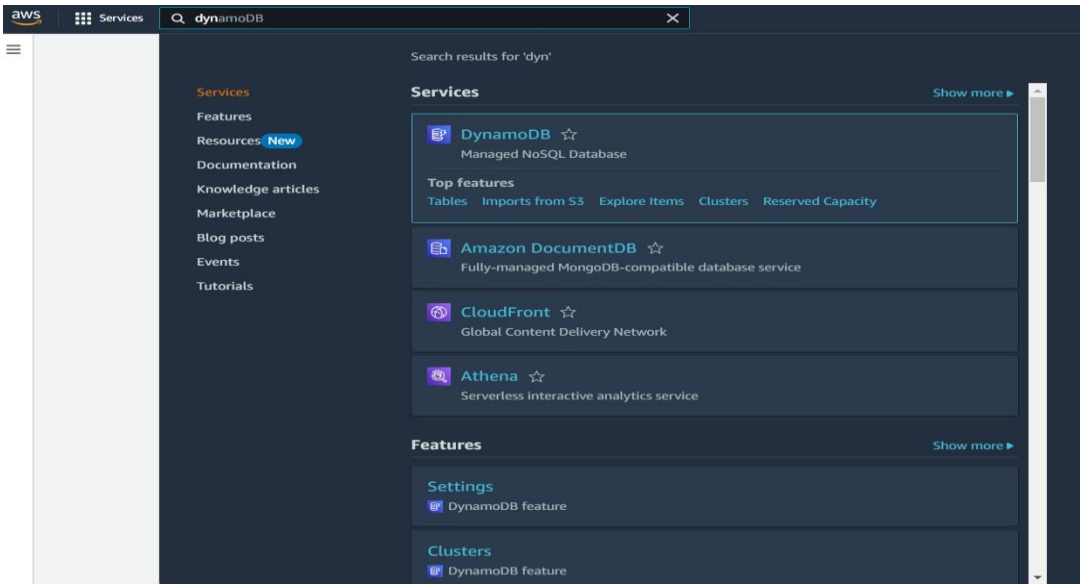
- **Activity 2.2: Log in to the AWS Management Console**

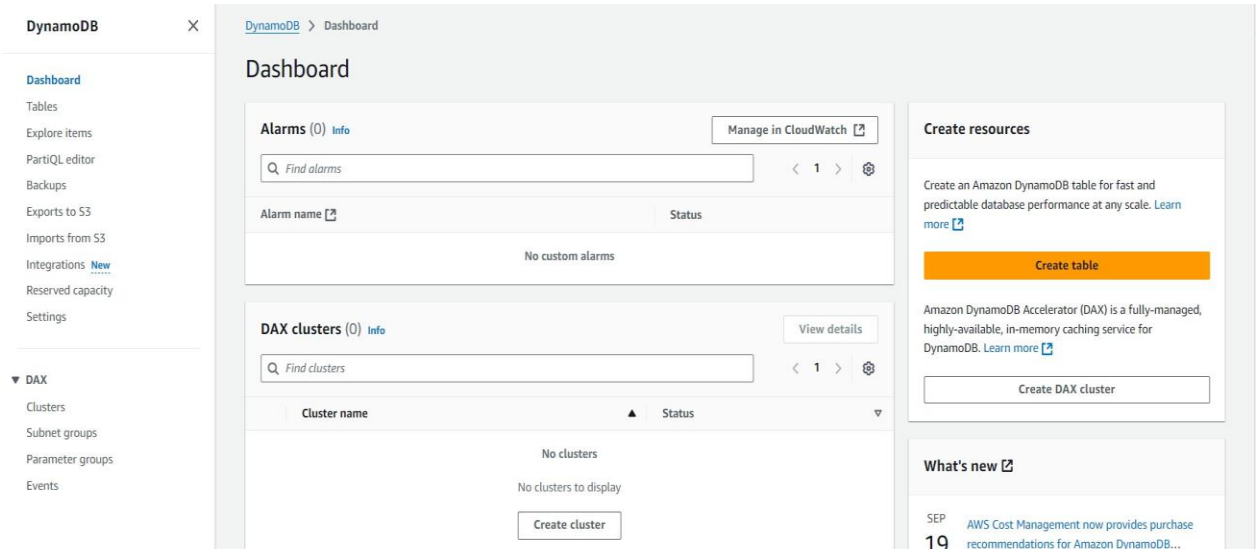


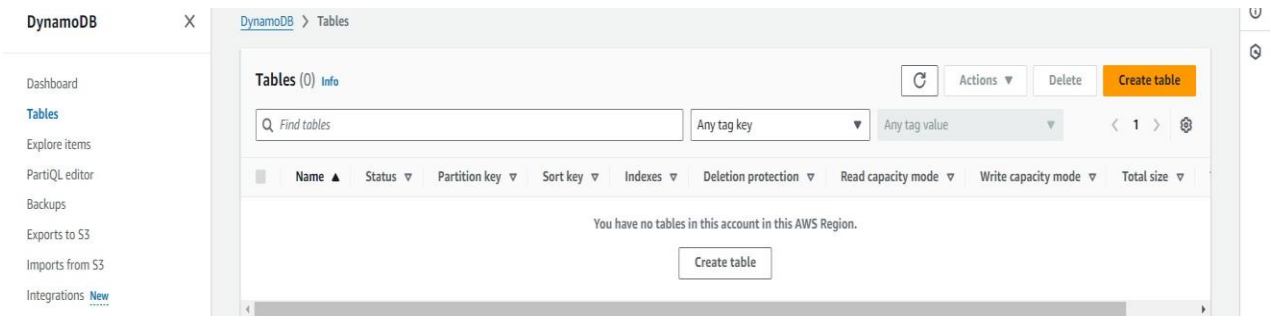
After setting up your account, log in to the AWS Management Console.

Milestone 3: DynamoDB Database Creation and Setup

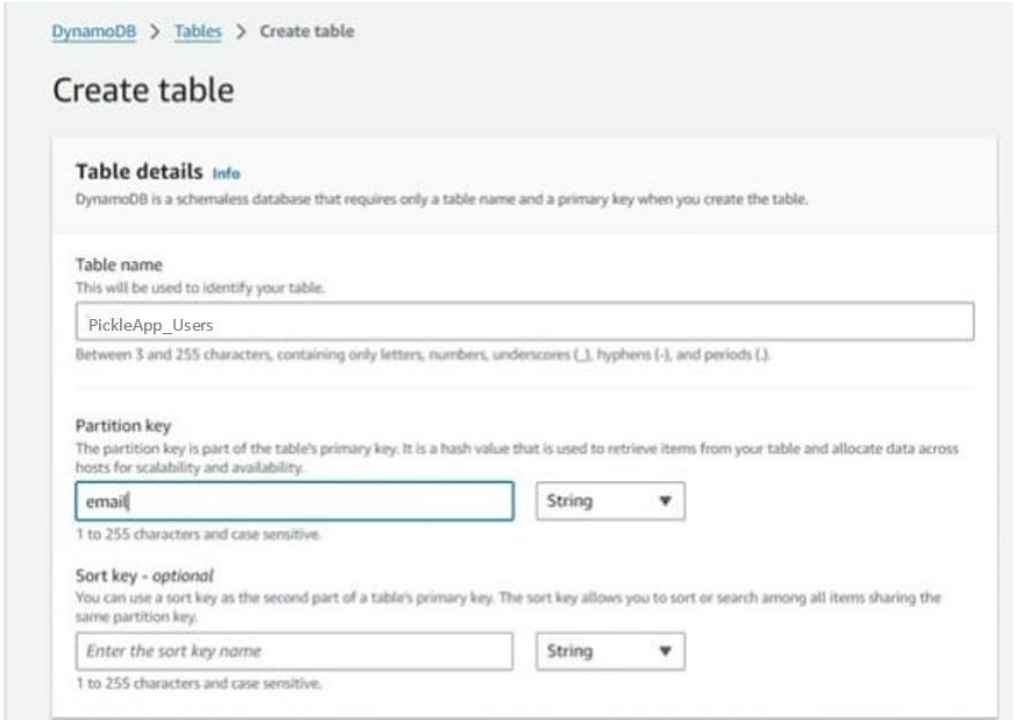
- **Activity 3.1:Navigate to the DynamoDB**
    - In the AWS Console, navigate to DynamoDB and click on create tables.

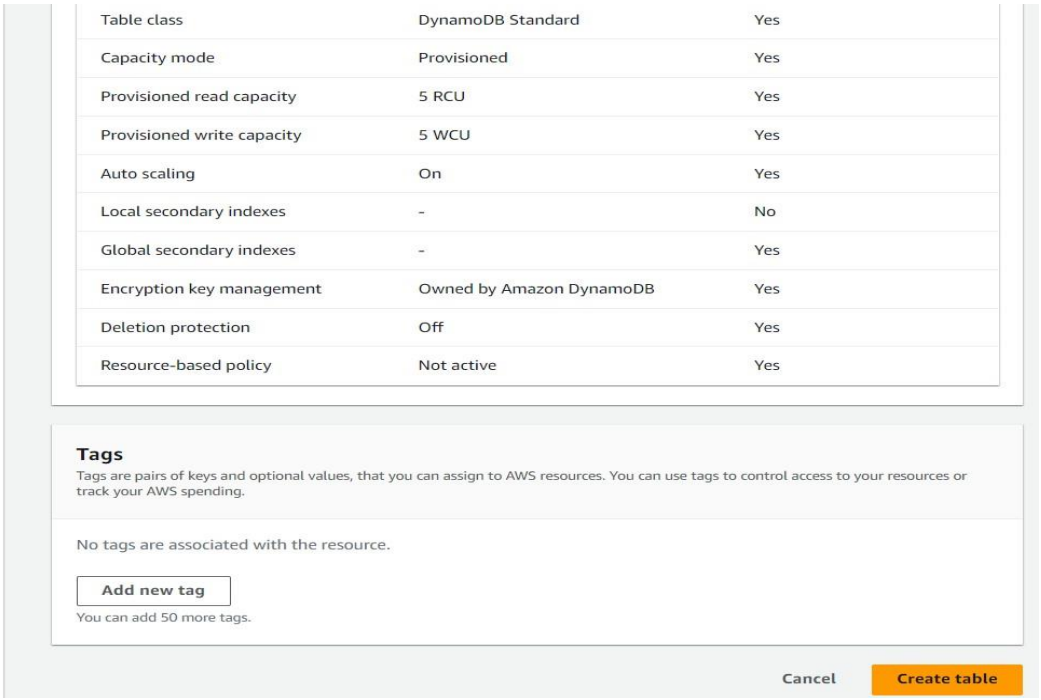


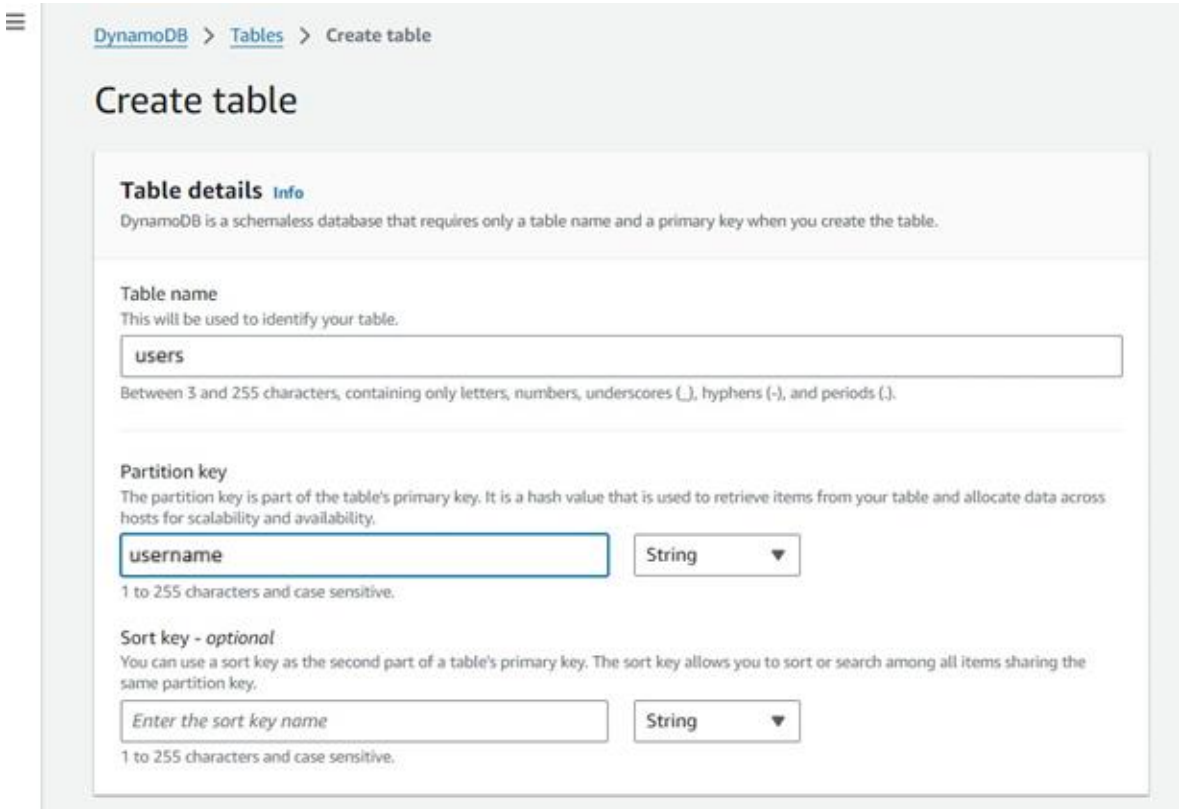


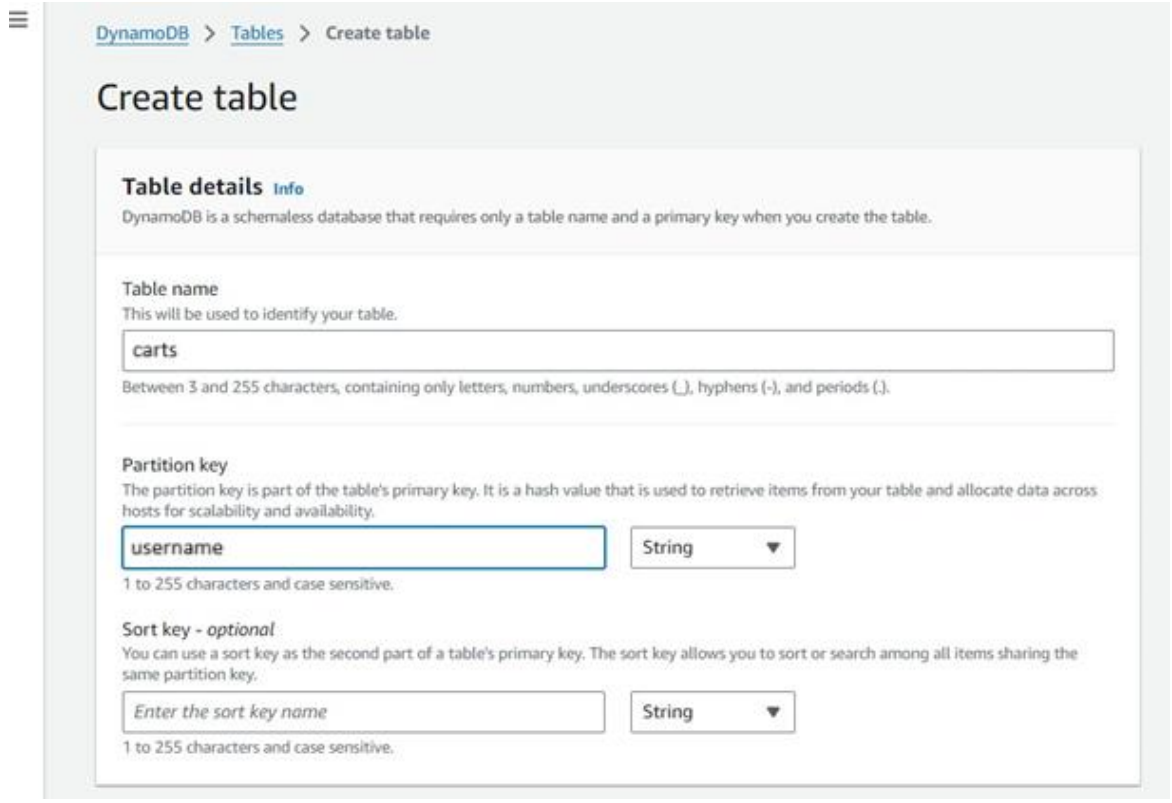


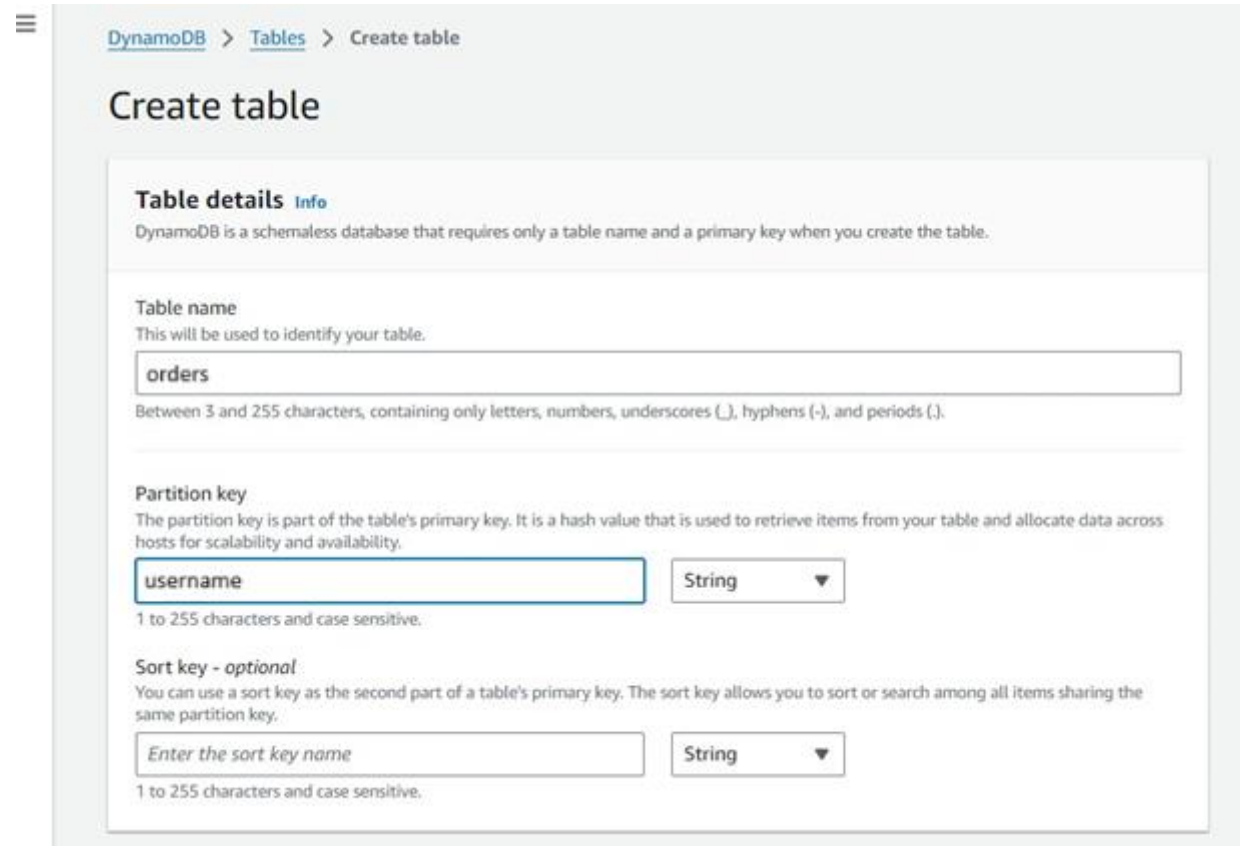
- **Activity 3.2:Create a DynamoDB table for storing registration details and book requests.**
    - Create Users table with partition key “Email” with type String and click on create tables.

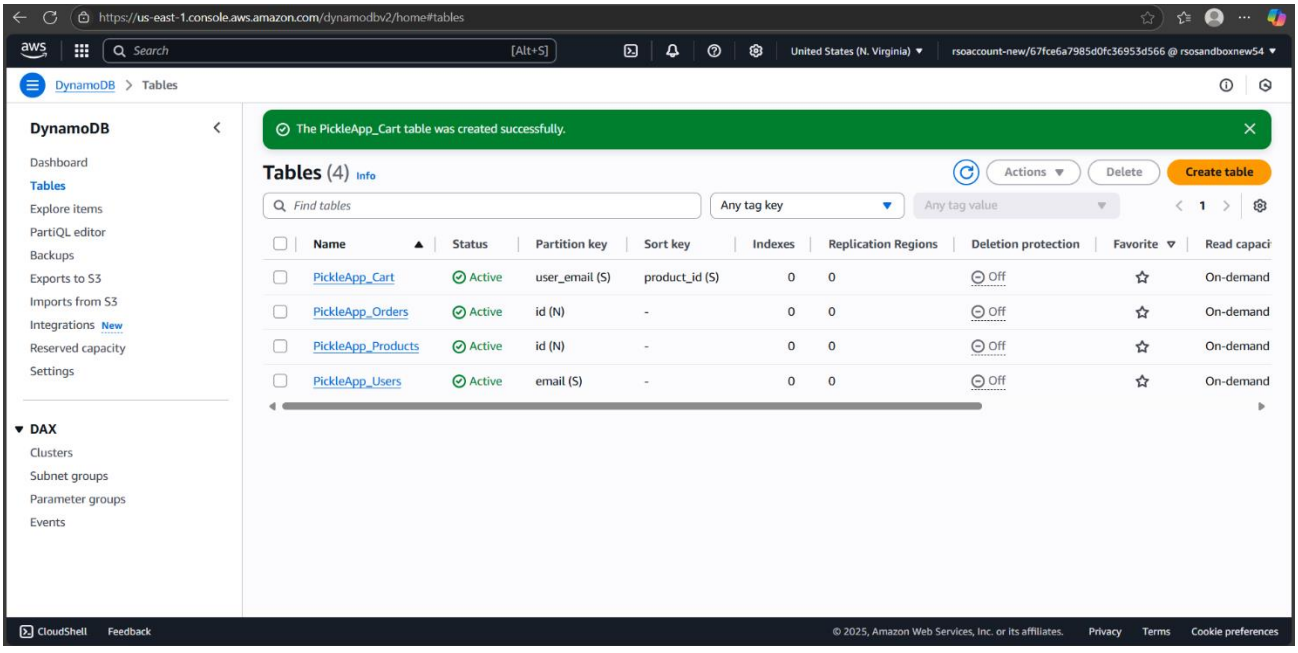





DynamoDB > Tables > Create table
Create table
Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.
Table name
This will be used to identify your table.
users
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).
Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
username
String
1 to 255 characters and case sensitive.
Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
Enter the sort key name
String
1 to 255 characters and case sensitive.

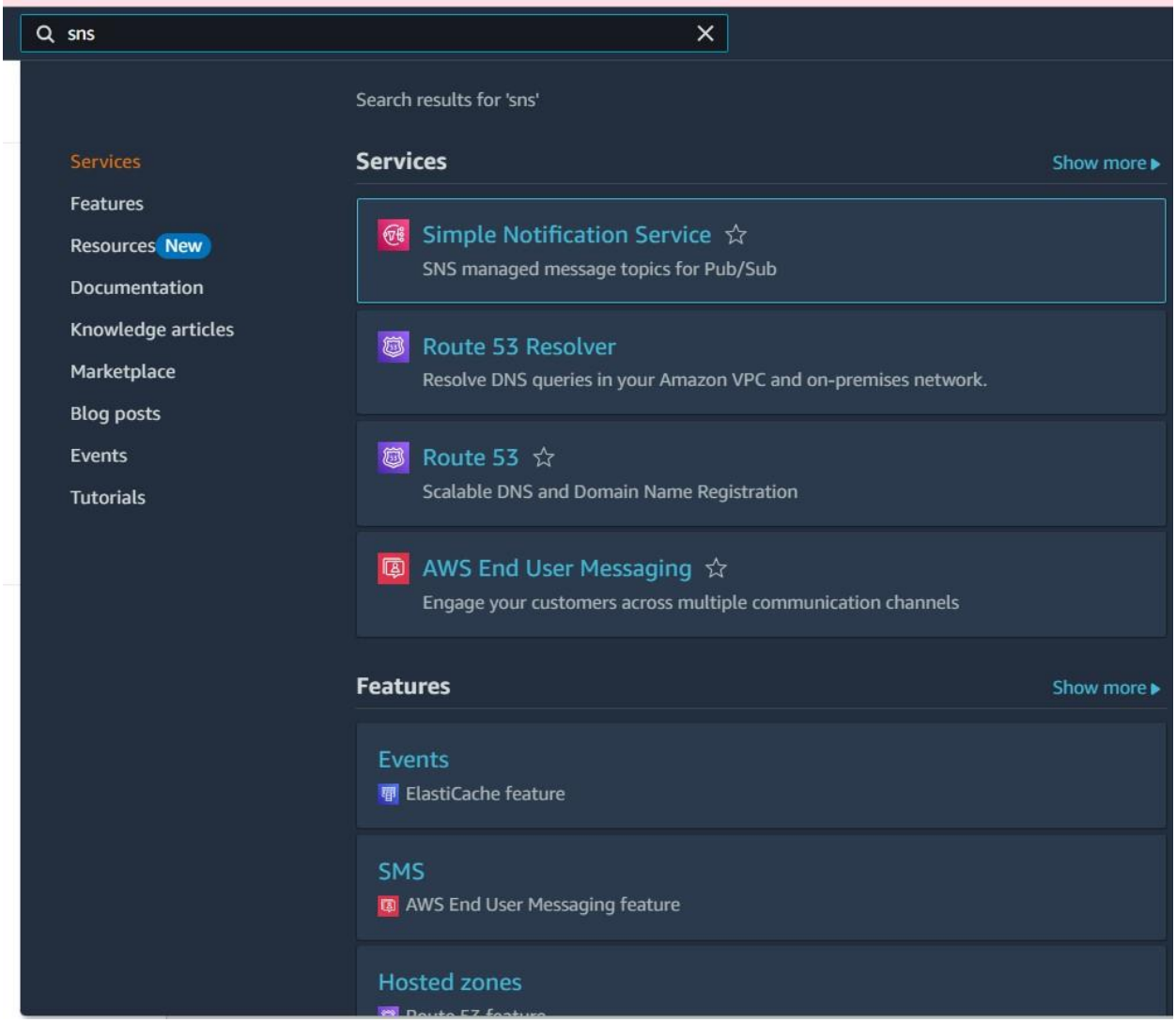

DynamoDB > Tables > Create table
Create table
Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.
Table name
This will be used to identify your table.
carts
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).
Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
username
String
1 to 255 characters and case sensitive.
Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
Enter the sort key name
String
1 to 255 characters and case sensitive.

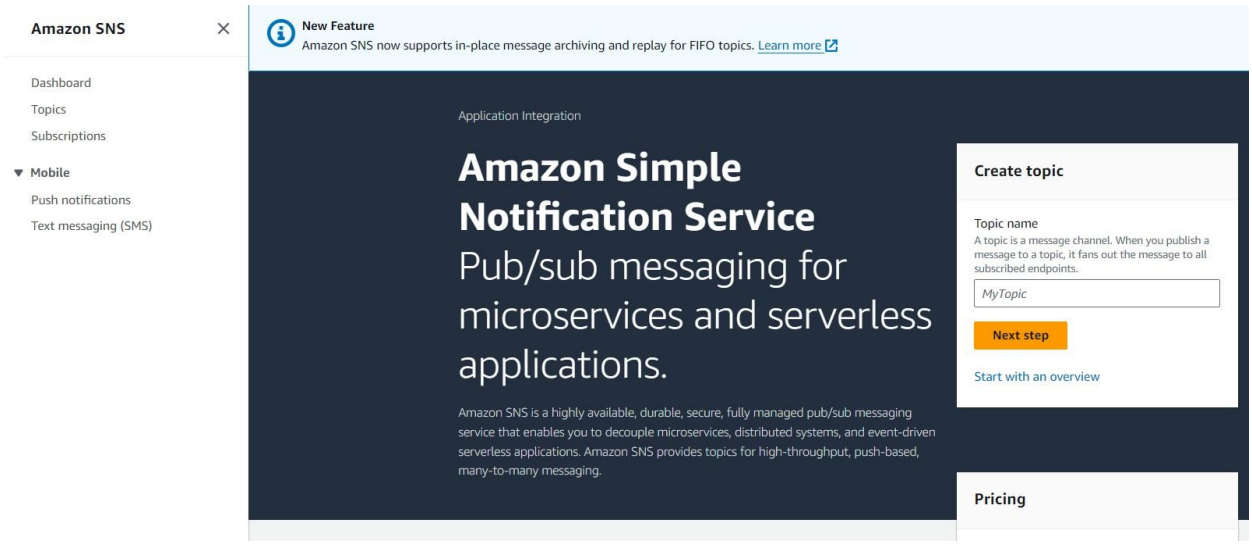

DynamoDB > Tables > Create table
Create table
Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.
Table name
This will be used to identify your table.
orders
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).
Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
username
String
1 to 255 characters and case sensitive.
Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
Enter the sort key name
String
1 to 255 characters and case sensitive.


https://us-east-1.console.aws.amazon.com/dynamodbv2/home#tables
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew54
DynamoDB > Tables
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
The PickleApp_Cart table was created successfully.
Tables (4) Info
Actions
Delete
Create table
Find tables
Any tag key
Any tag value
Name
Status
Partition key
Sort key
Indexes
Replication Regions
Deletion protection
Favorite
Read capaci
PickleApp_Cart
Active
user_email (S)
product_id (S)
0
0
Off
On-demand
PickleApp_Orders
Active
id (N)
-
0
0
Off
On-demand
PickleApp_Products
Active
id (N)
-
0
0
Off
On-demand
PickleApp_Users
Active
email (S)
-
0
0
Off
On-demand
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

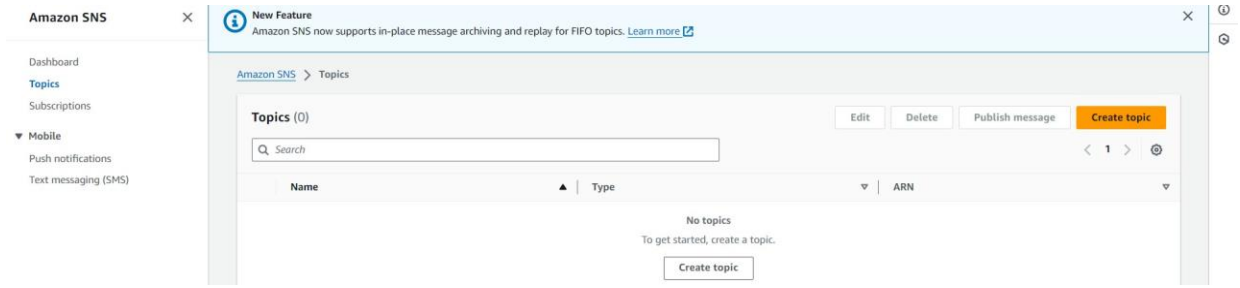
Milestone 4: SNS Notification Setup

- **Activity 4.1: Create SNS topics for sending email notifications for ordering alerts.**
    - In the AWS Console, search for SNS and navigate to the SNS Dashboard.

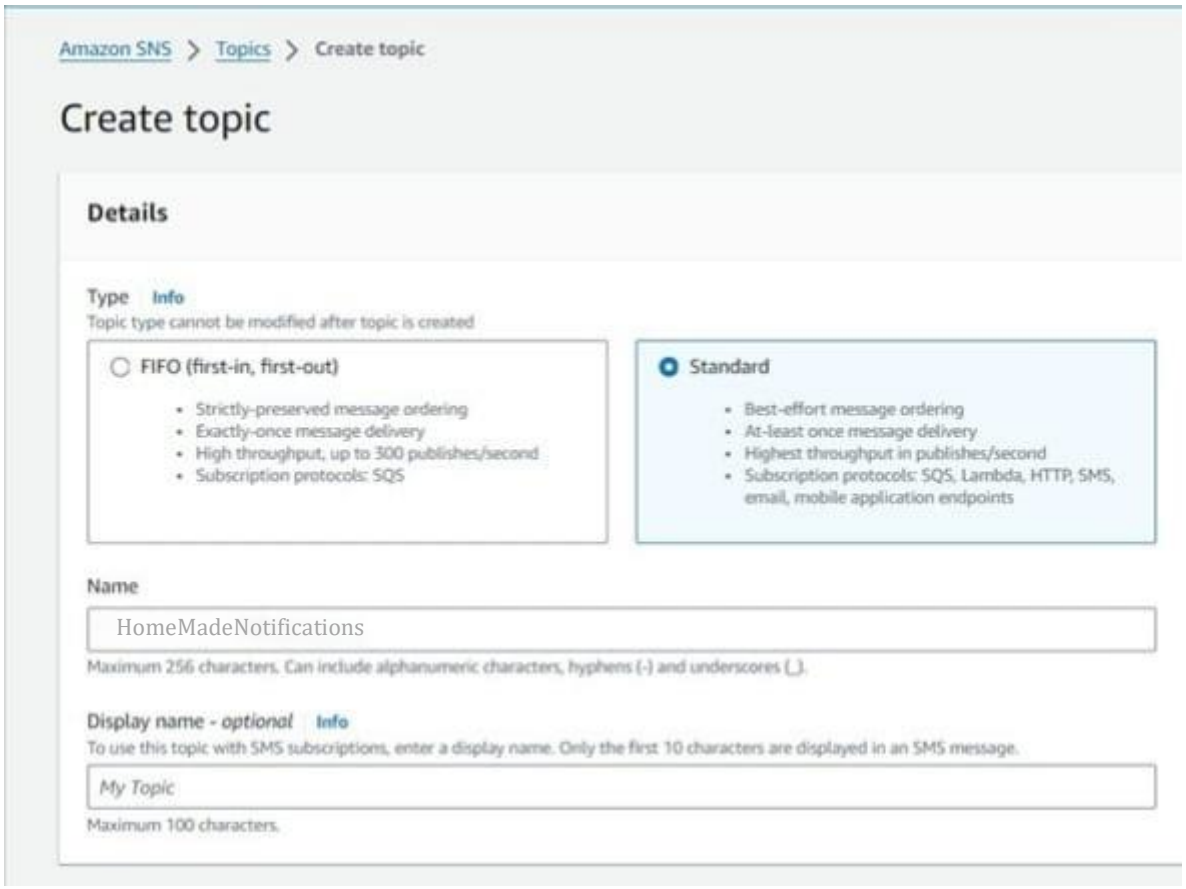


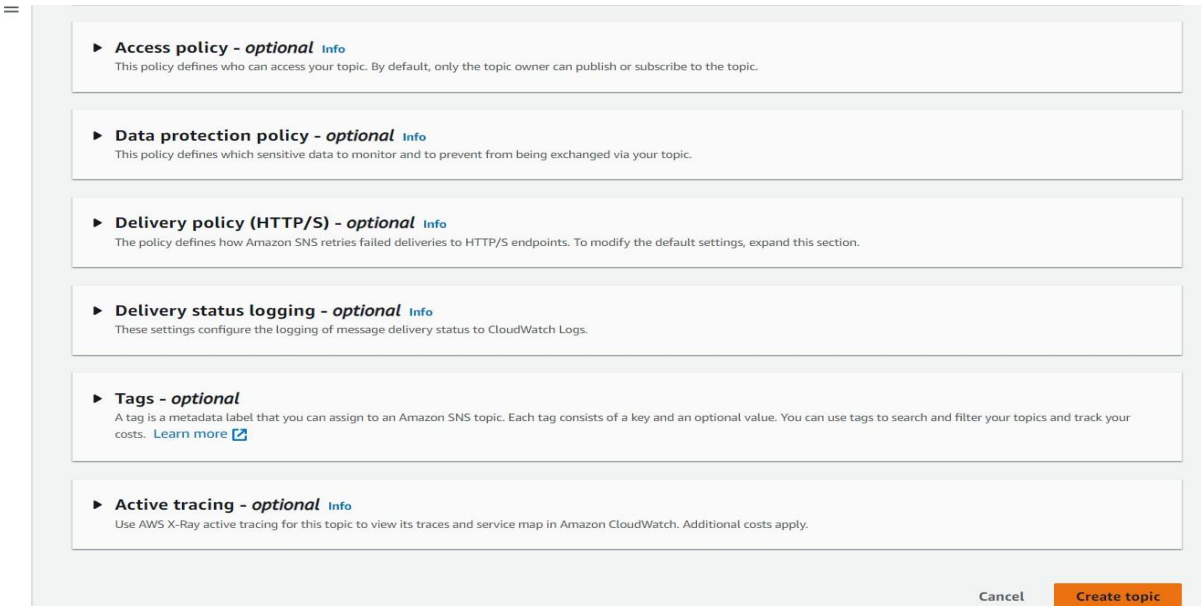


- Click on **Create Topic** and choose a name for the topic.

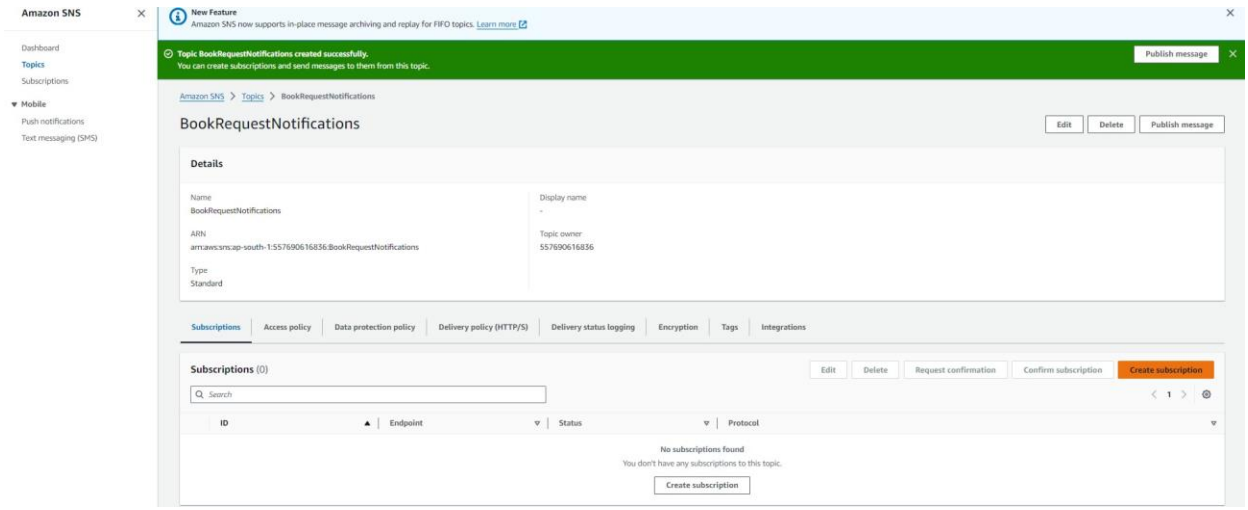


- Choose Standard type for general notification use cases and Click on Create Topic.


Amazon SNS > Topics > Create topic
Create topic
Details
Type Info
Topic type cannot be modified after topic is created
FIFO (first-in, first-out)
Strictly-preserved message ordering
Exactly-once message delivery
High throughput, up to 300 publishes/second
Subscription protocols: SQS
Standard
Best-effort message ordering
At-least once message delivery
Highest throughput in publishes/second
Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints
Name
HomeMadeNotifications
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).
Display name - optional Info
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.
My Topic
Maximum 100 characters.

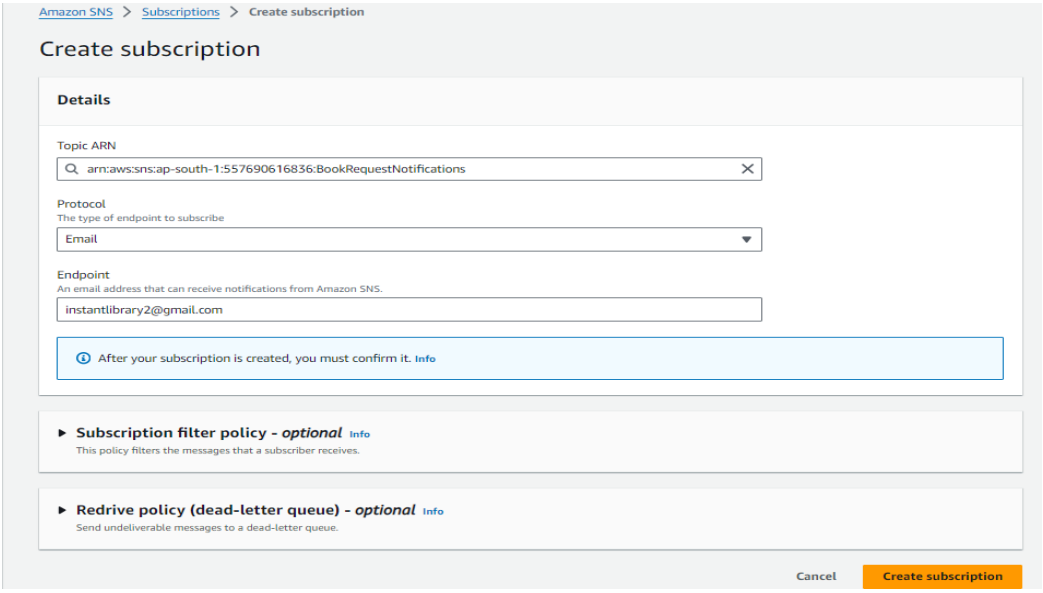

Access policy - optional Info
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.
Data protection policy - optional Info
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.
Delivery policy (HTTP/S) - optional Info
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.
Delivery status logging - optional Info
These settings configure the logging of message delivery status to CloudWatch Logs.
Tags - optional
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. Learn more
Active tracing - optional Info
Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.
Cancel
Create topic

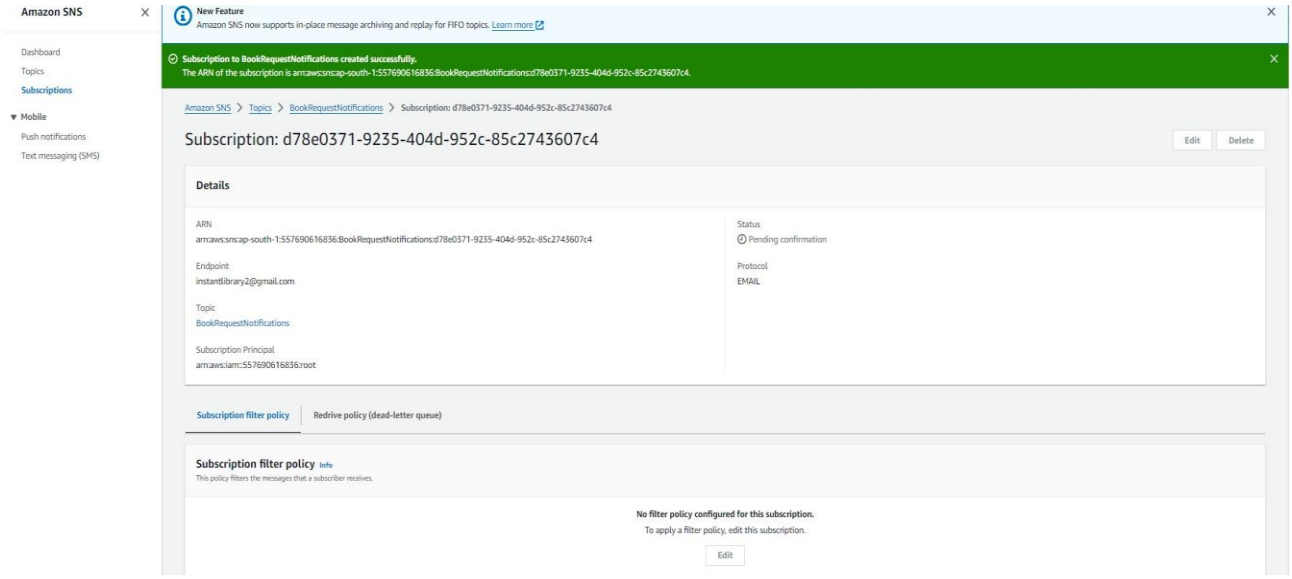
- Configure the SNS topic and note down the **Topic ARN**.



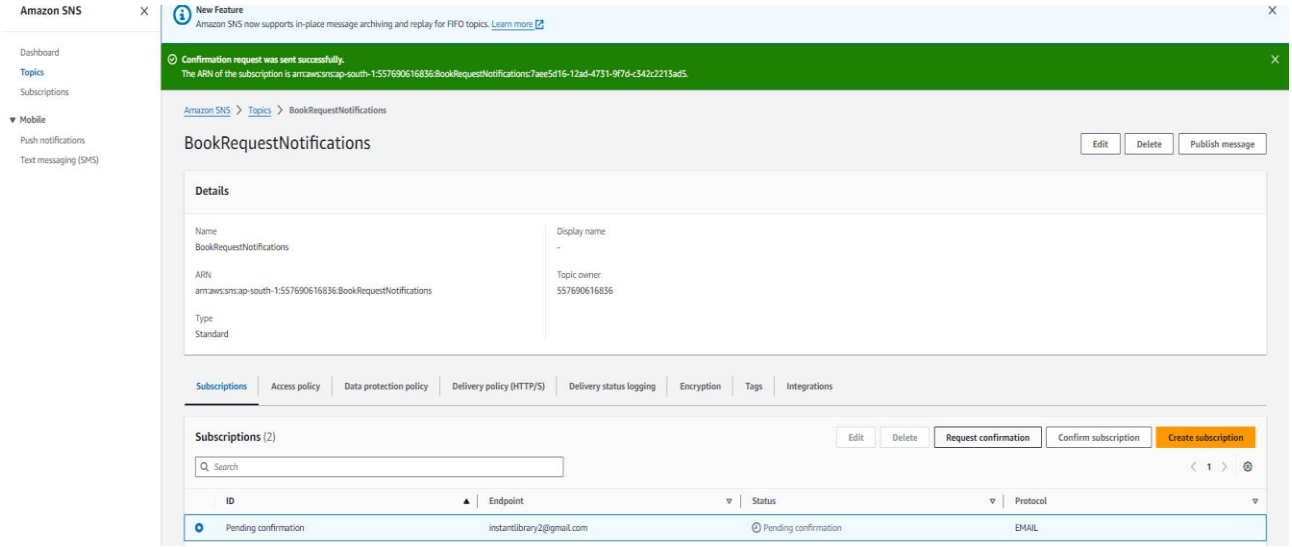
- **Activity 4.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

  - Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails**.**





- After subscription request for the mail confirmation



- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.

AWS Notification - Subscription Confirmation Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> 9
to me

You have chosen to subscribe to the topic:
arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to sns-opt-out

AWS Notifications <no-reply@sns.amazonaws.com>
to me
•••

You have chosen to subscribe to the topic:
arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to sns-opt-out

aws Simple Notification Service

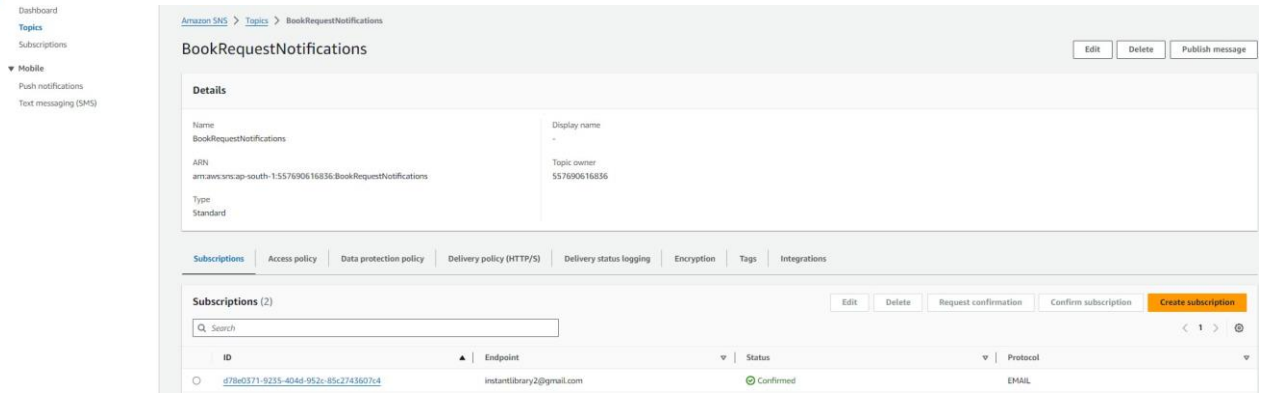
Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:
arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications:d78e0371-9235-404d-952c-85c2743607c4

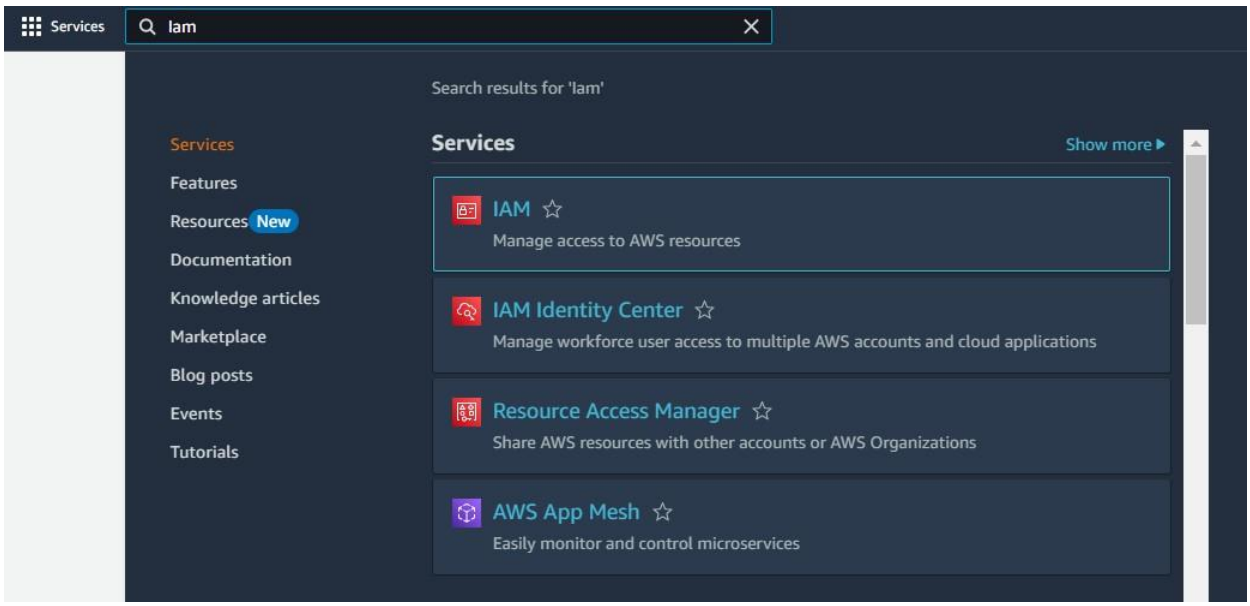
If it was not your intention to subscribe, click here to unsubscribe.

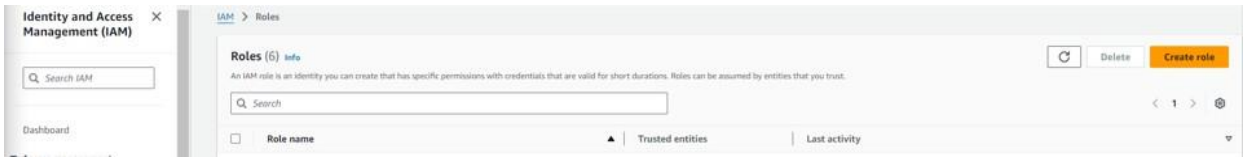
- Successfully done with the SNS mail subscription and setup, now store the ARN link.

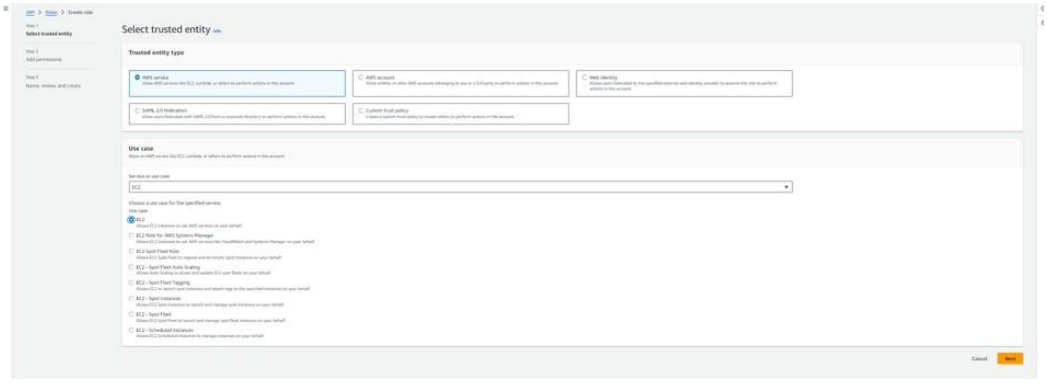


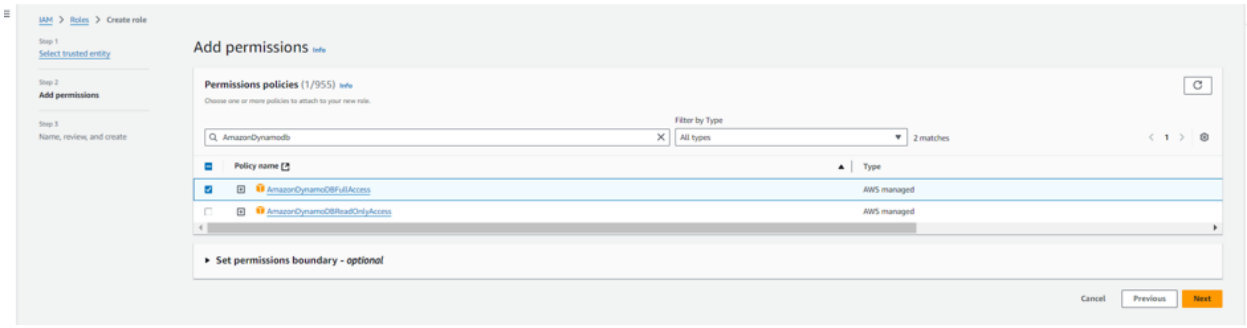
Milestone 5: IAM Role Setup

- **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.



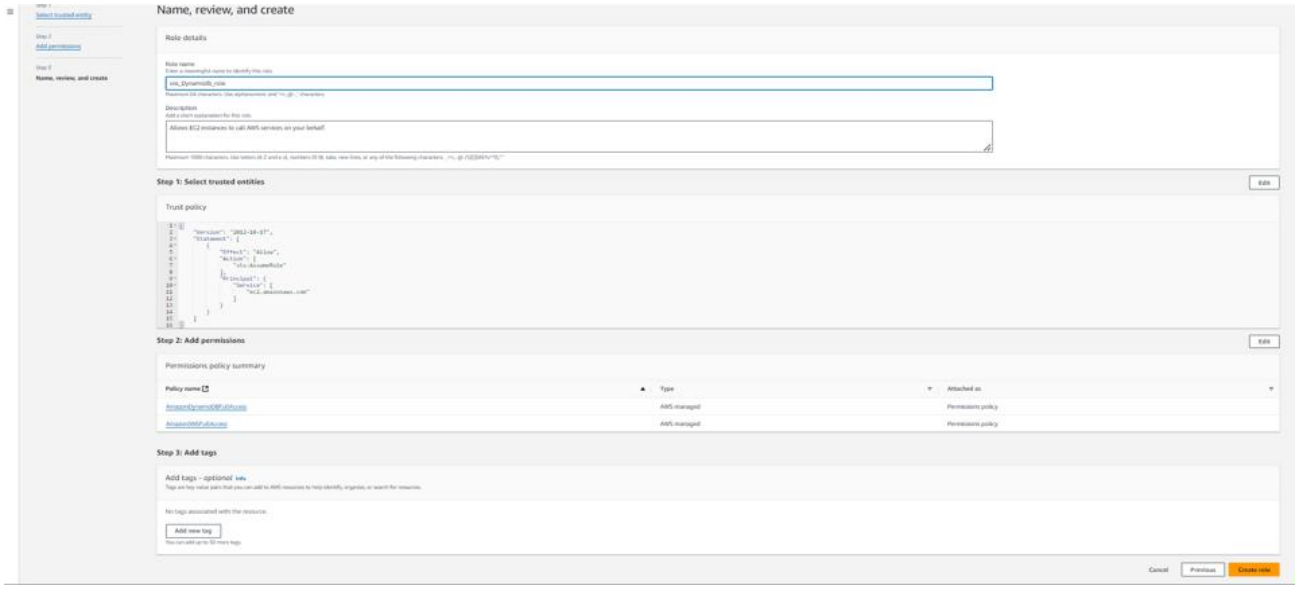


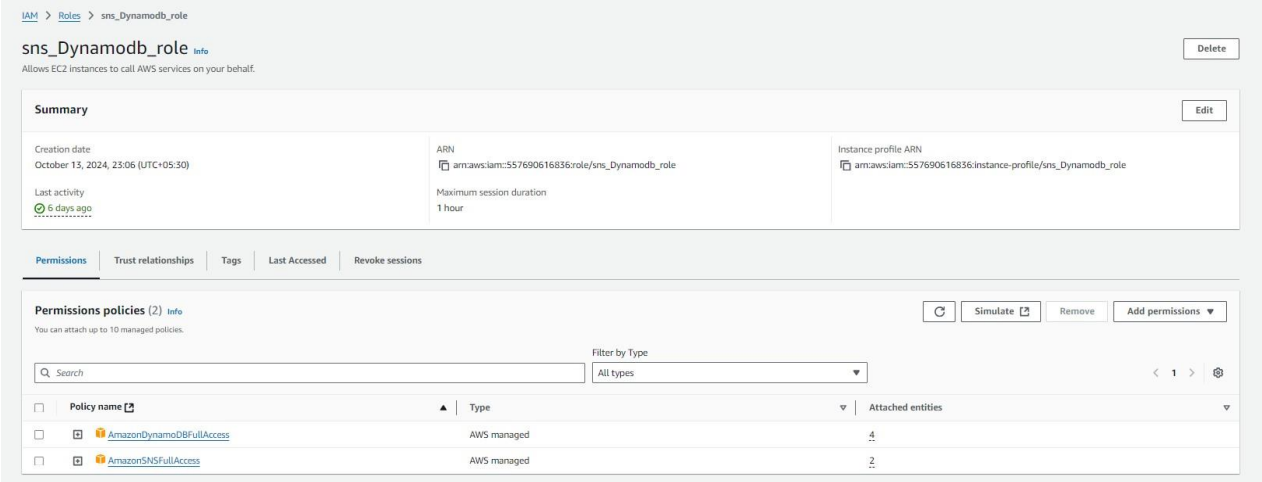




- **Activity 5.2: Attach Policies.**
    - Attach the following policies to the role:

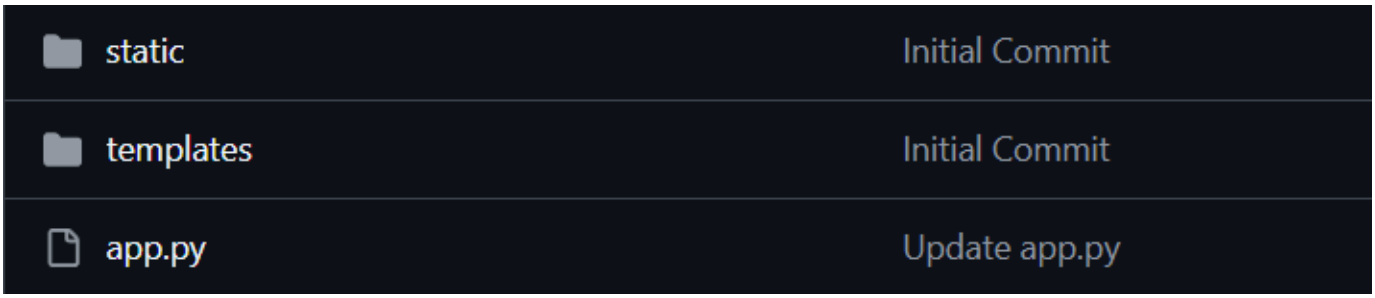
- AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
- AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.

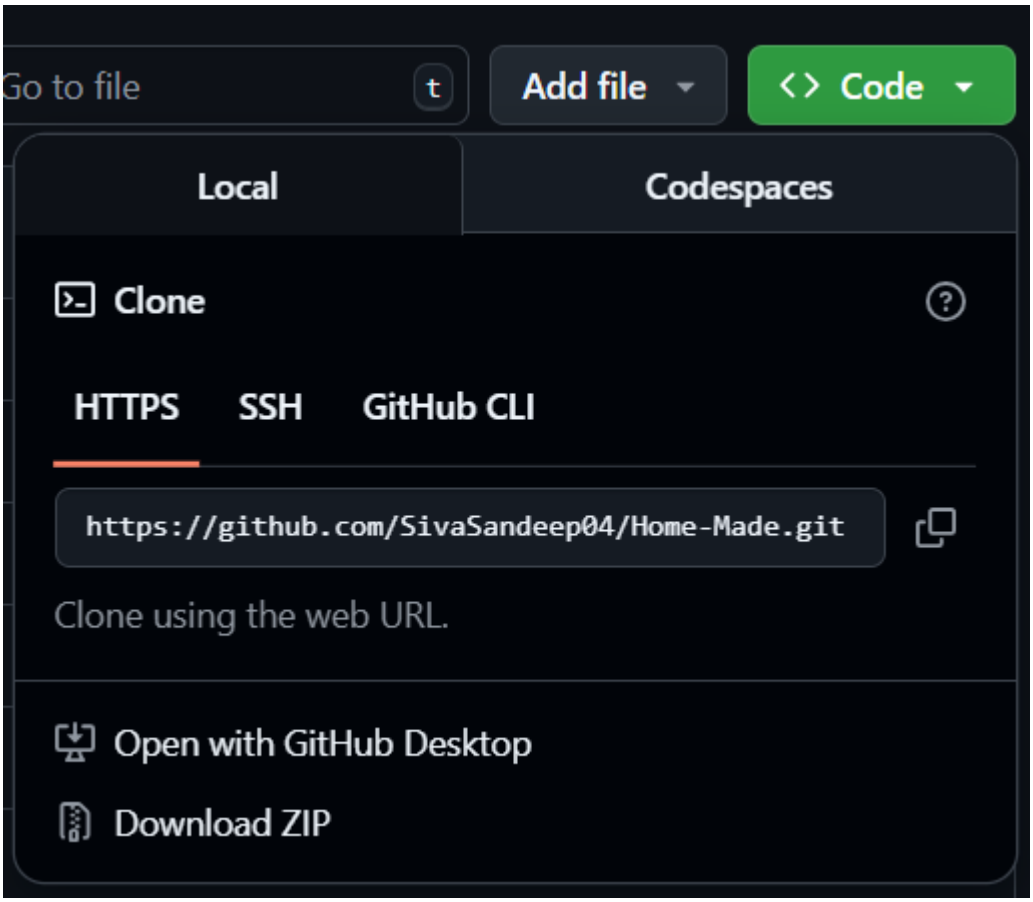




Milestone 6: EC2 Instance Setup

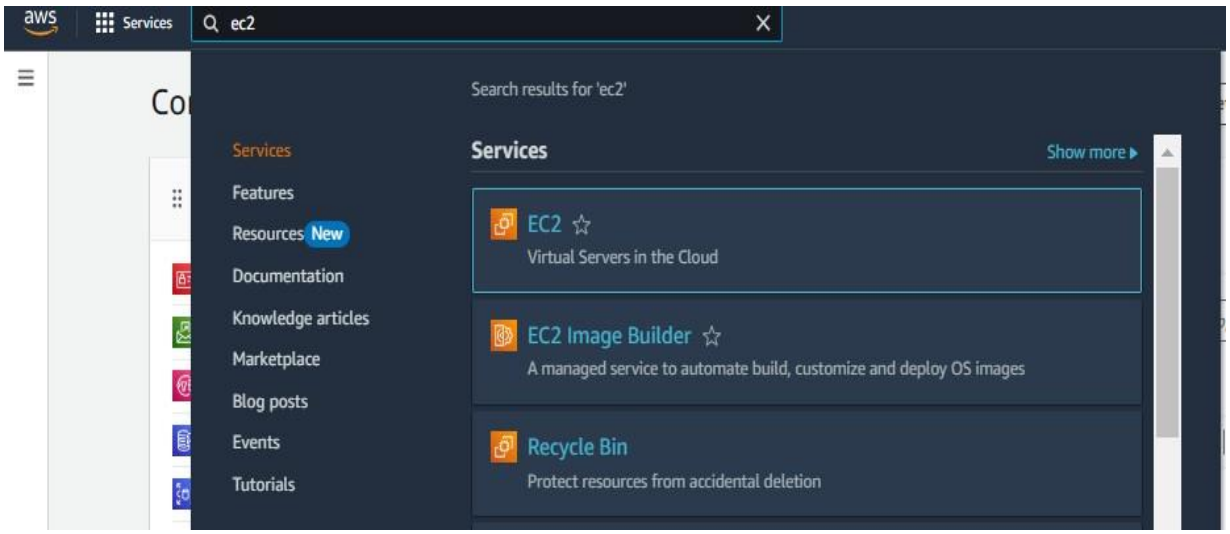
- **Note: Load your Flask app and Html files into GitHub repository.**



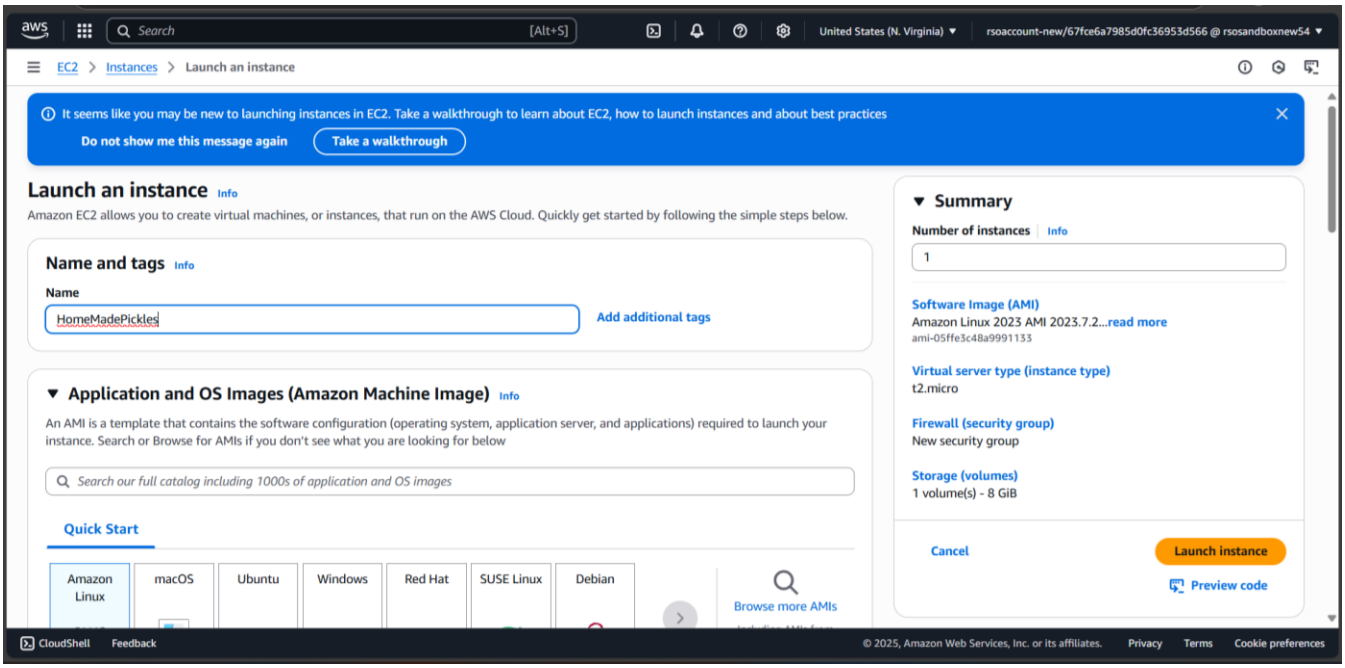


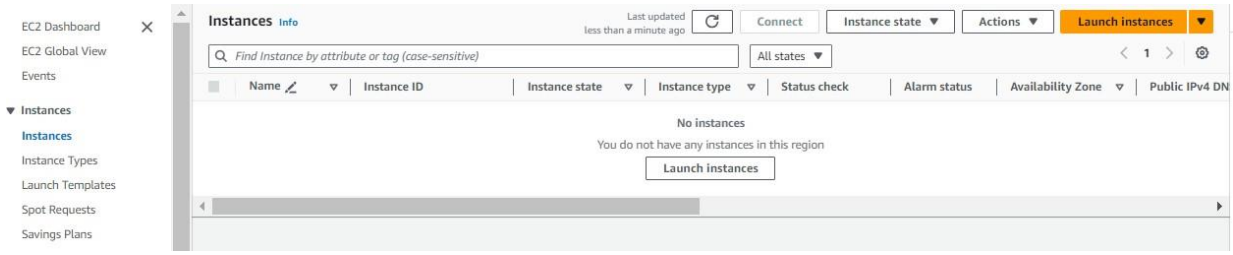
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

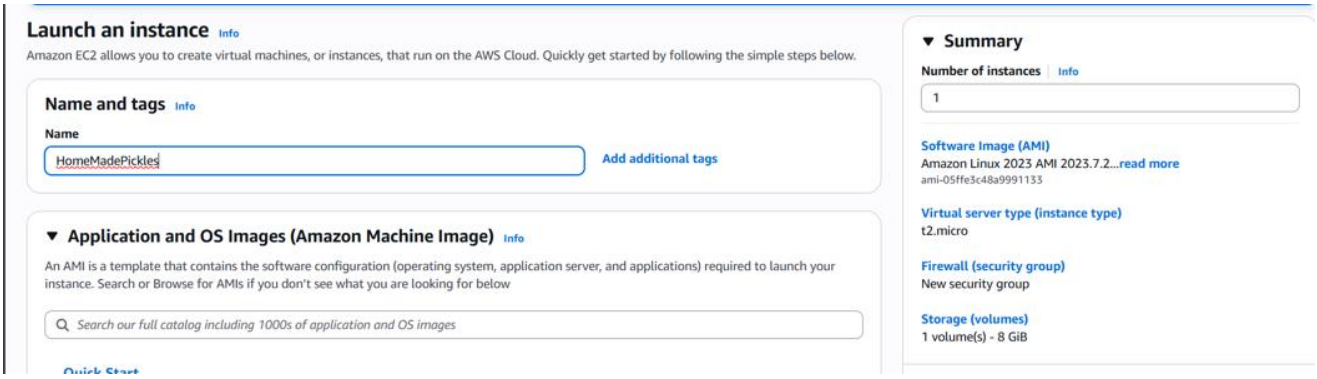
  - **Launch EC2 Instance**
    - In the AWS Console, navigate to EC2 and launch a new instance.



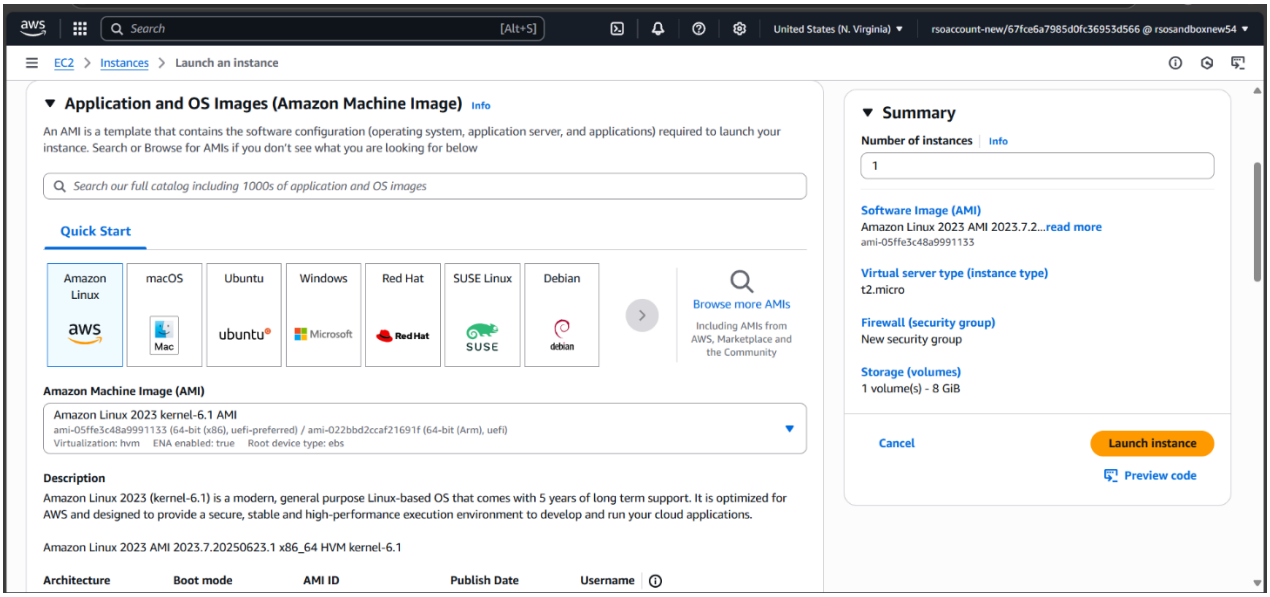
- Click on Launch instance to launch EC2 instance



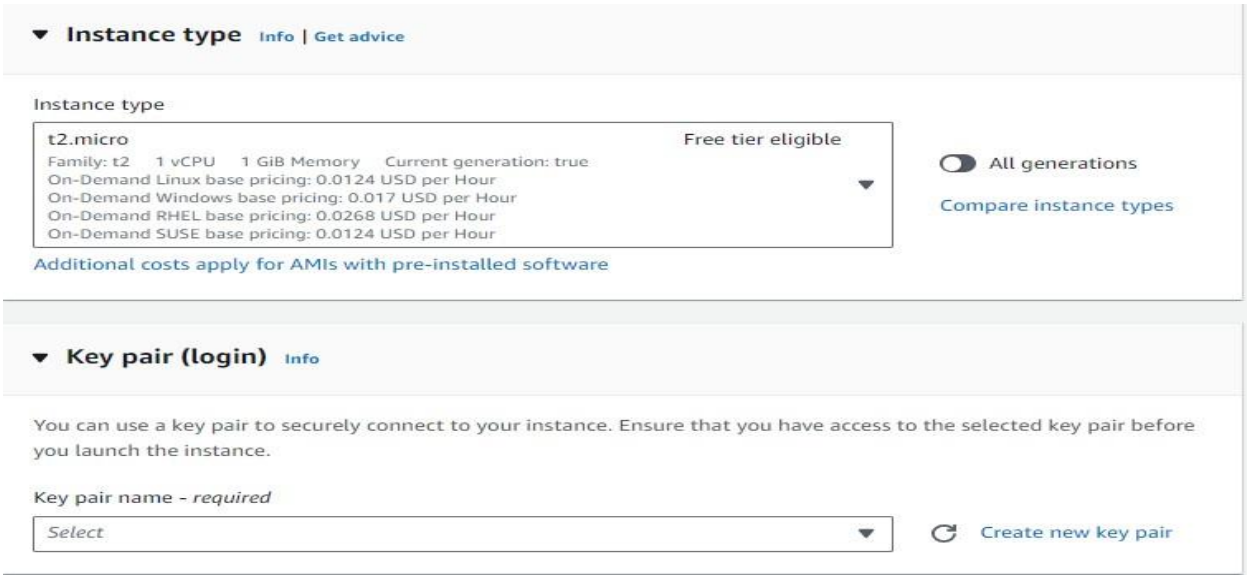


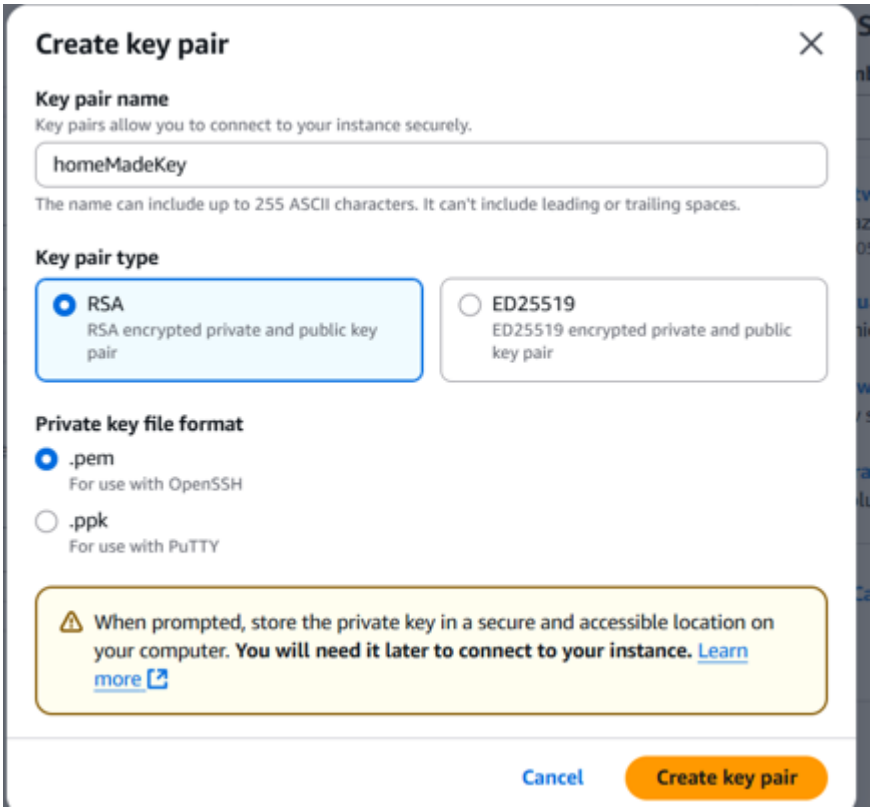


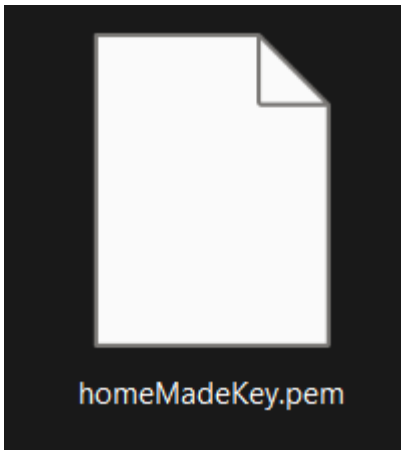
- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

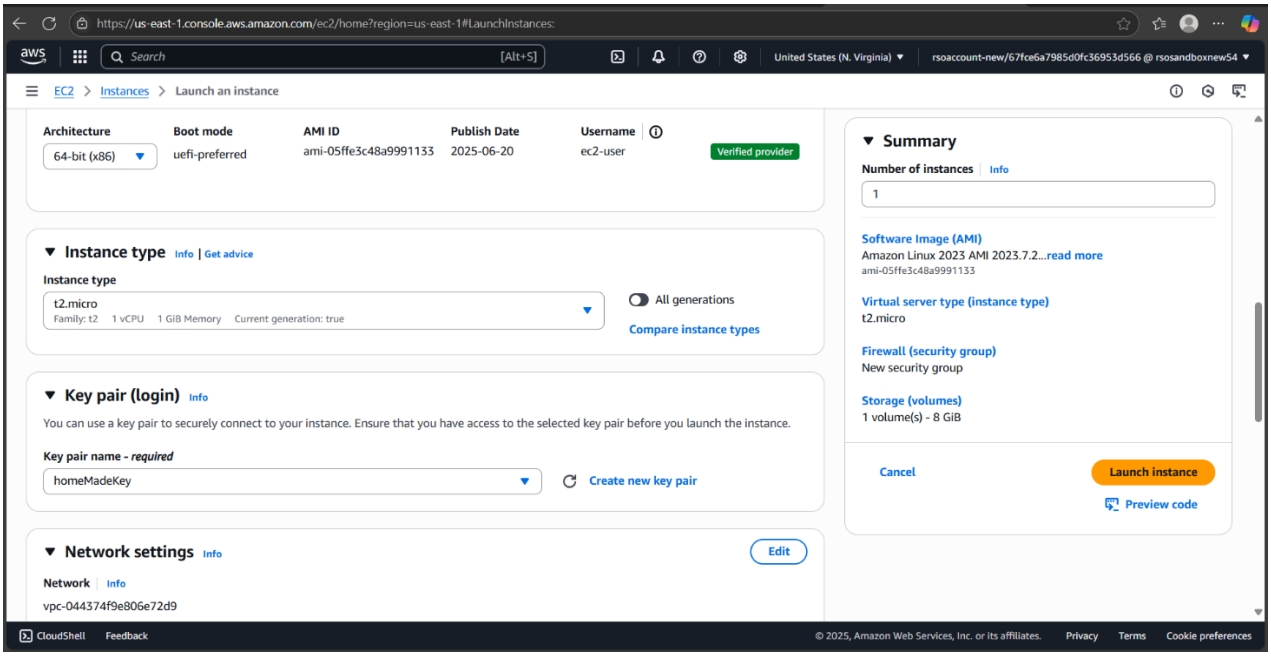


- Create and download the key pair for Server access.

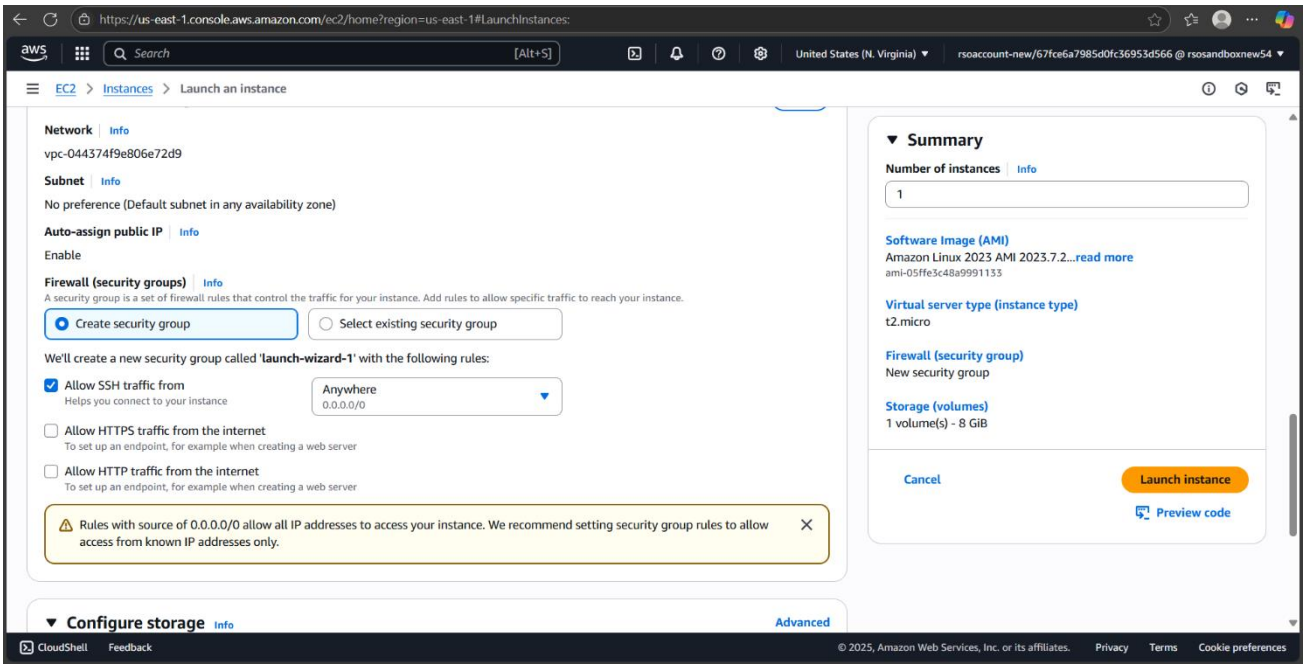


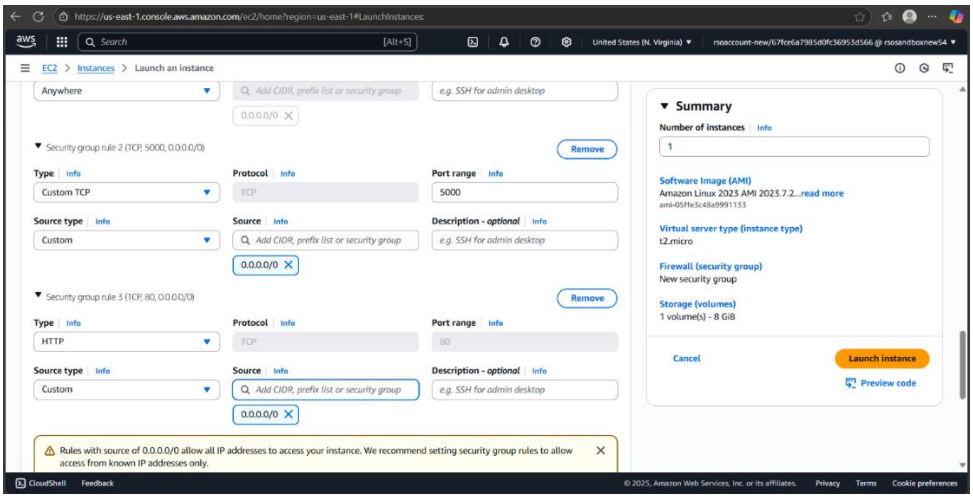

Create key pair
Key pair name
Key pairs allow you to connect to your instance securely.
homeMadeKey
The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.
Key pair type
RSA
RSA encrypted private and public key pair
ED25519
ED25519 encrypted private and public key 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

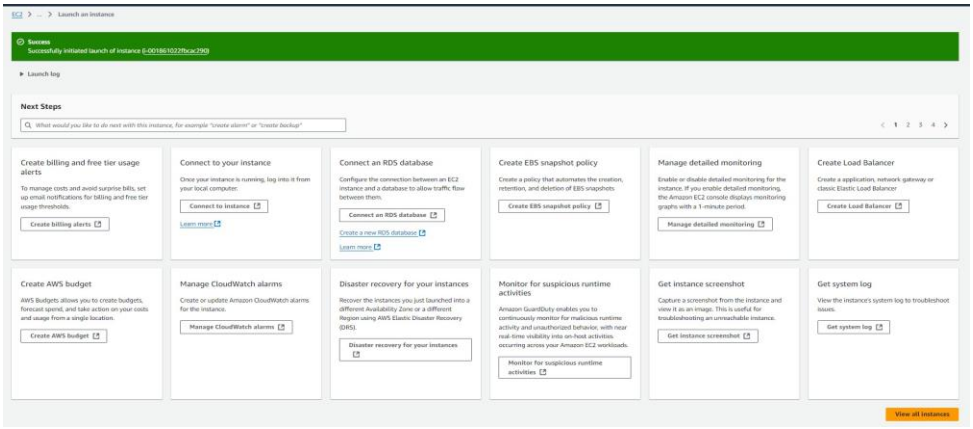

homeMadeKey.pem



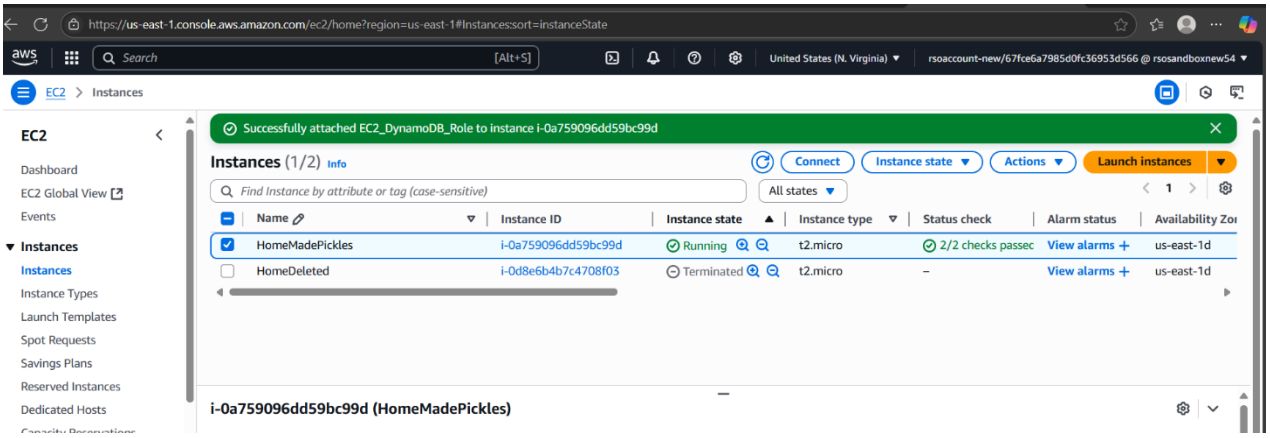
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**

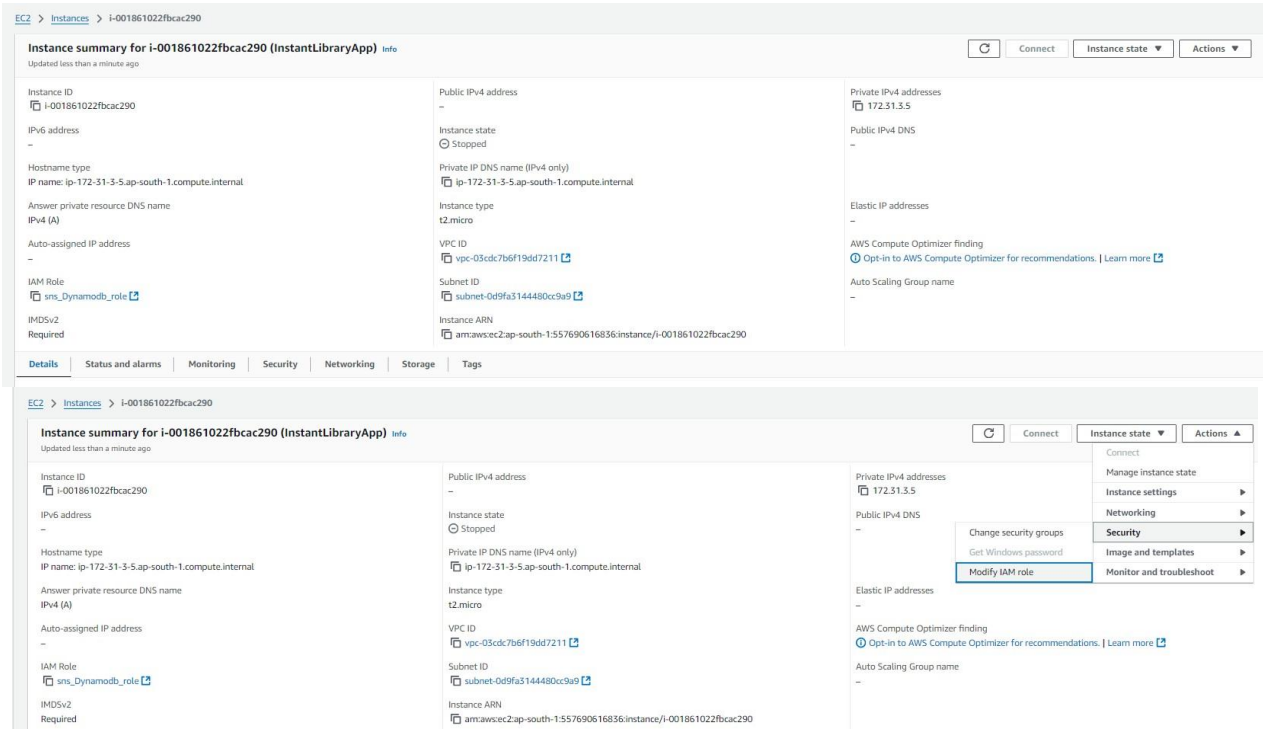


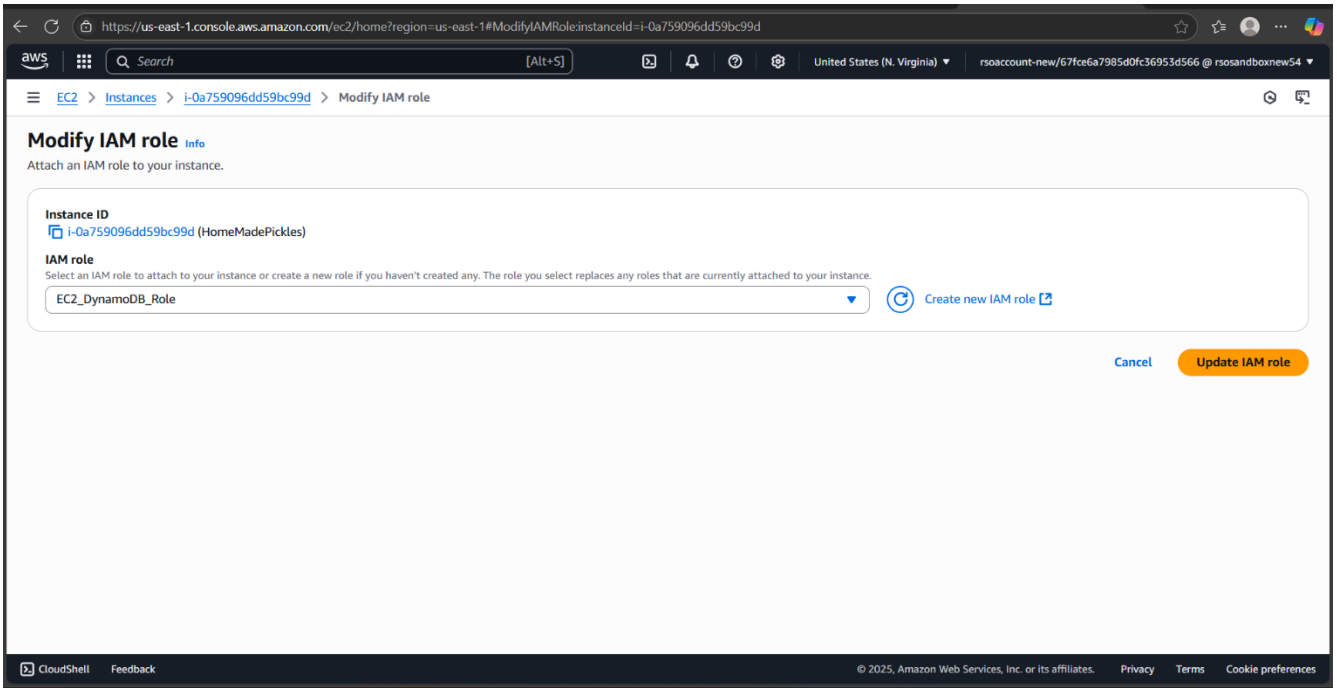




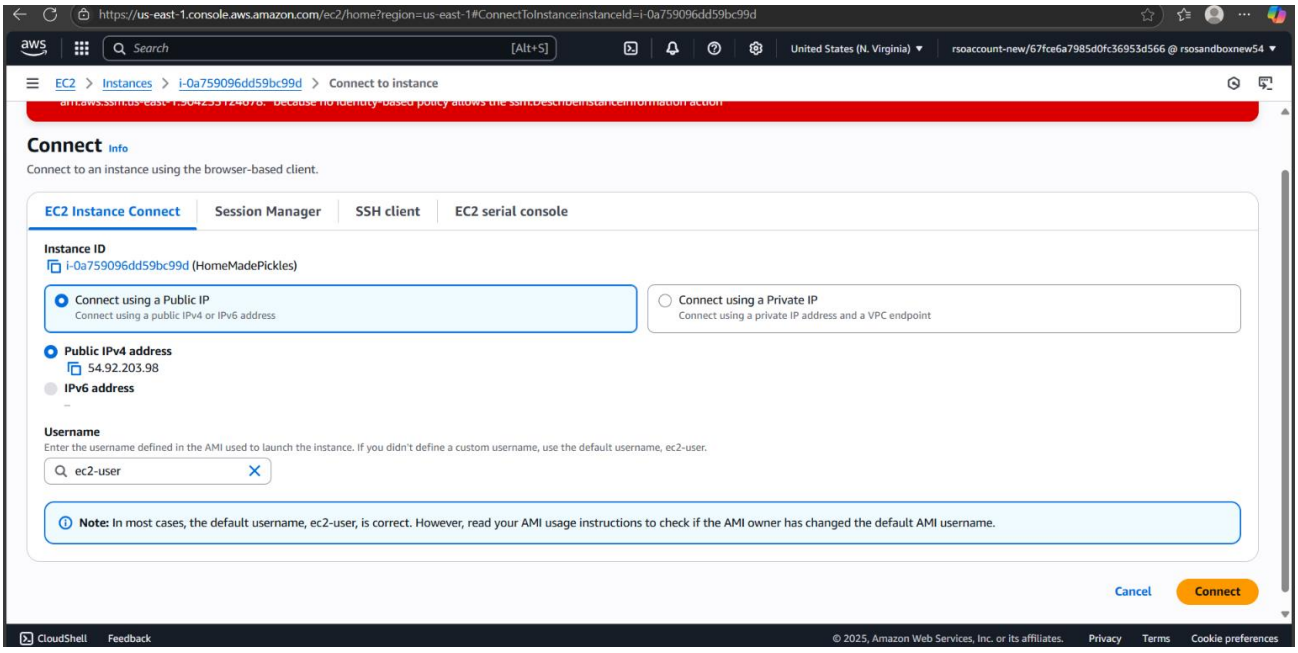
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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

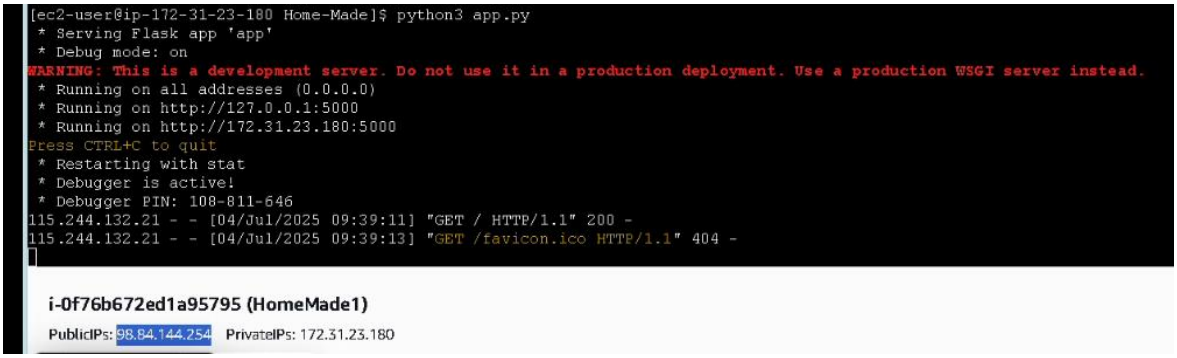

https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:sort=instanceState
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew54
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Successfully attached EC2_DynamoDB_Role to instance i-0a759096dd59bc99d
Instances (1/2) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zo
HomeMadePickles
i-0a759096dd59bc99d
Running
t2.micro
2/2 checks passec
View alarms +
us-east-1d
HomeDeleted
i-0d8e6b4b7c4708f03
Terminated
t2.micro
–
View alarms +
us-east-1d
i-0a759096dd59bc99d (HomeMadePickles)


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



- Now connect the EC2 with the files.





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

Verify Installations:

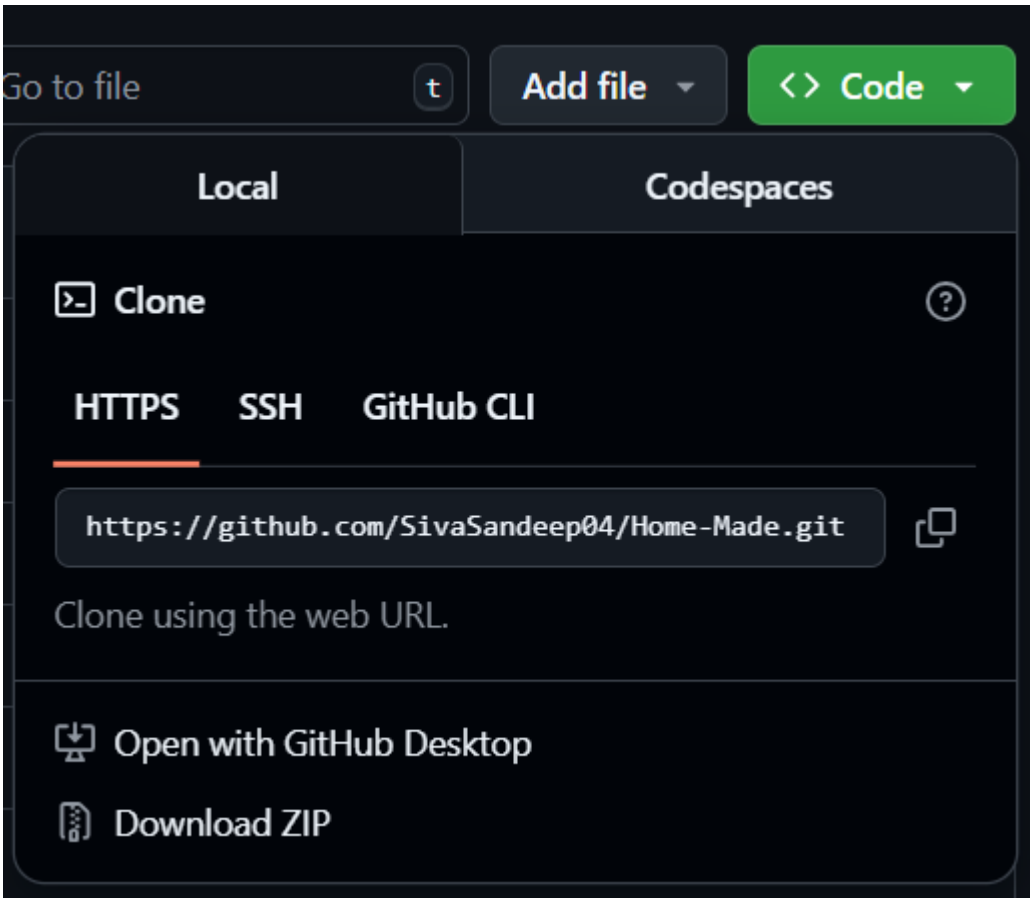
flask --

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/Dileep-3dev/home_made.git'

- This will download your project to the EC2 instance.

Here : git clone https://github.com/Dileep-3dev/home_made.git'

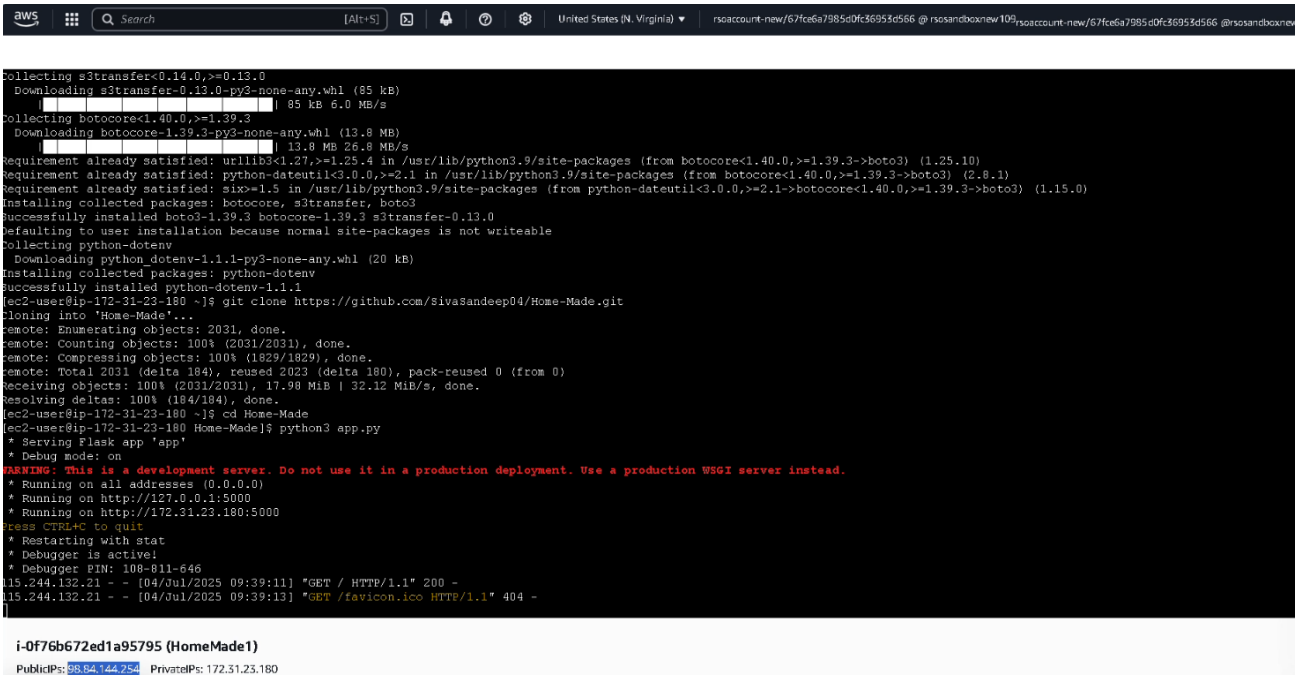
To navigate to the project directory, run the following command:

cd home_made

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



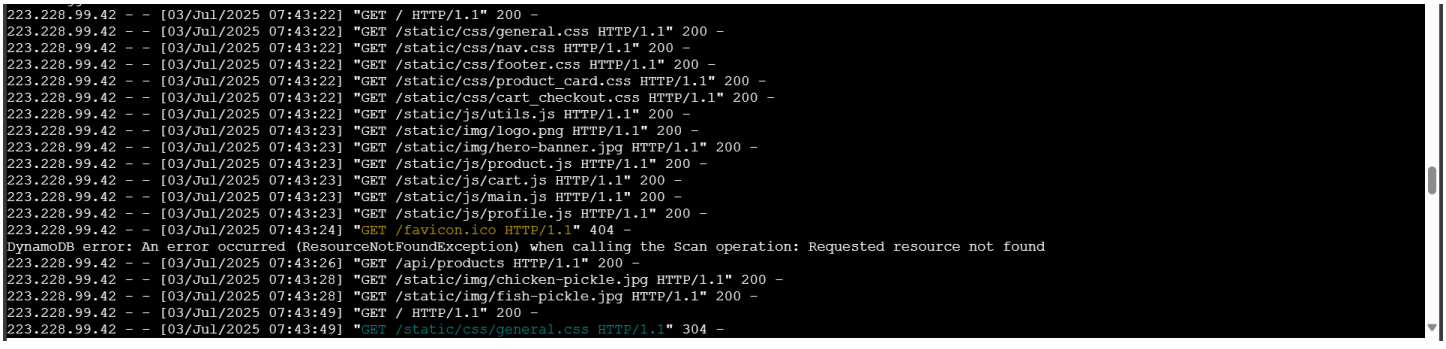
Verify the Flask app is

running:

http://your-ec2-

public-ip

- Run the Flask app on the EC2 instance



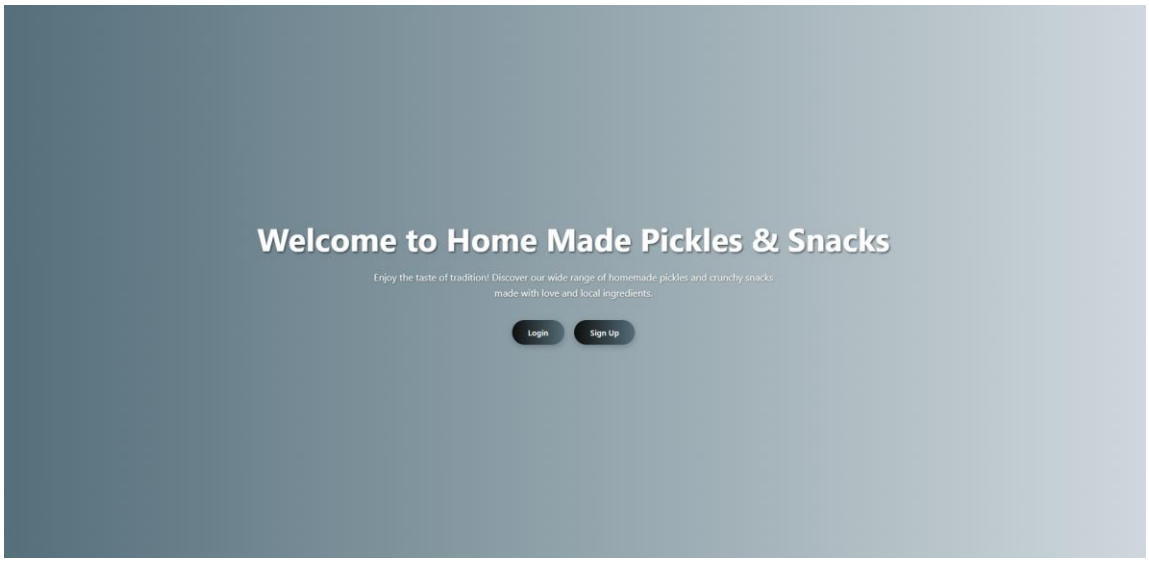
Access the website through:

PublicIPs: http://98.84.144.254:5000/

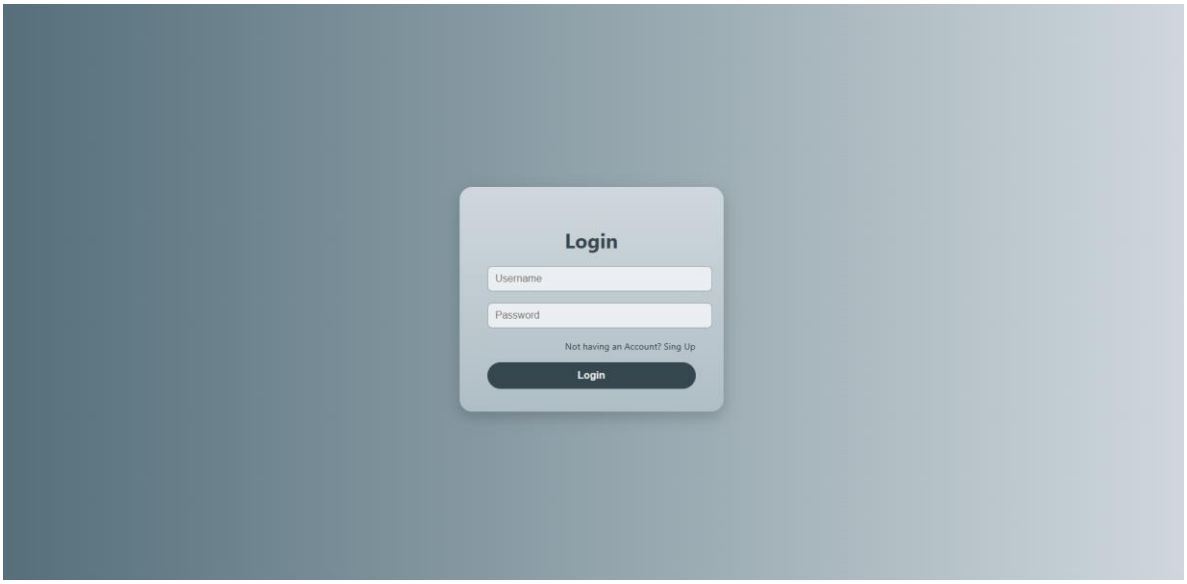
Milestone 8: Testing and Deployment

- **Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.**

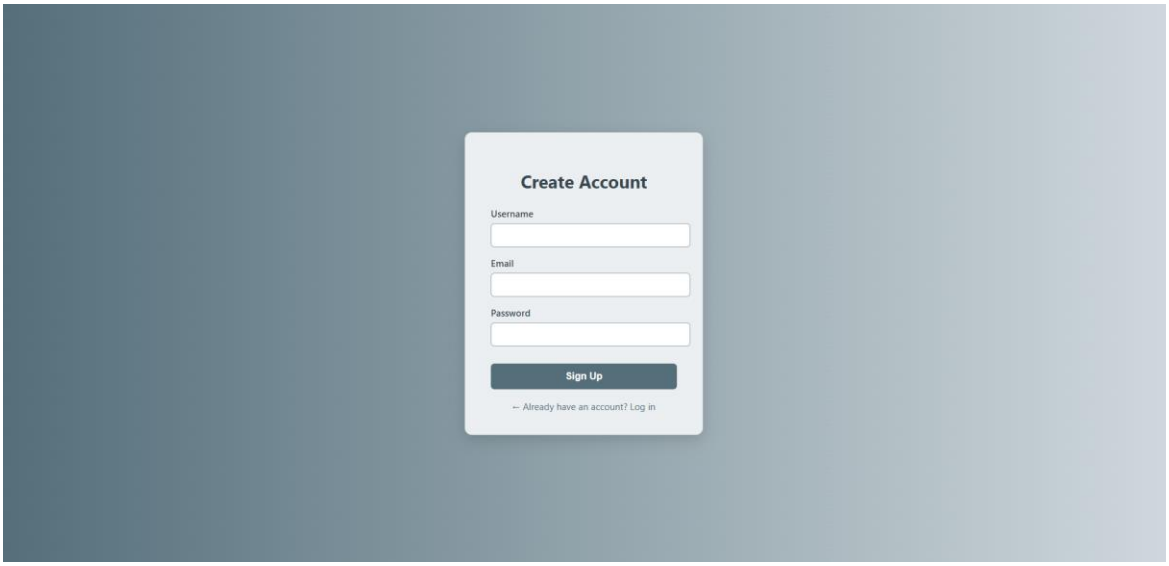
Welcome Page:



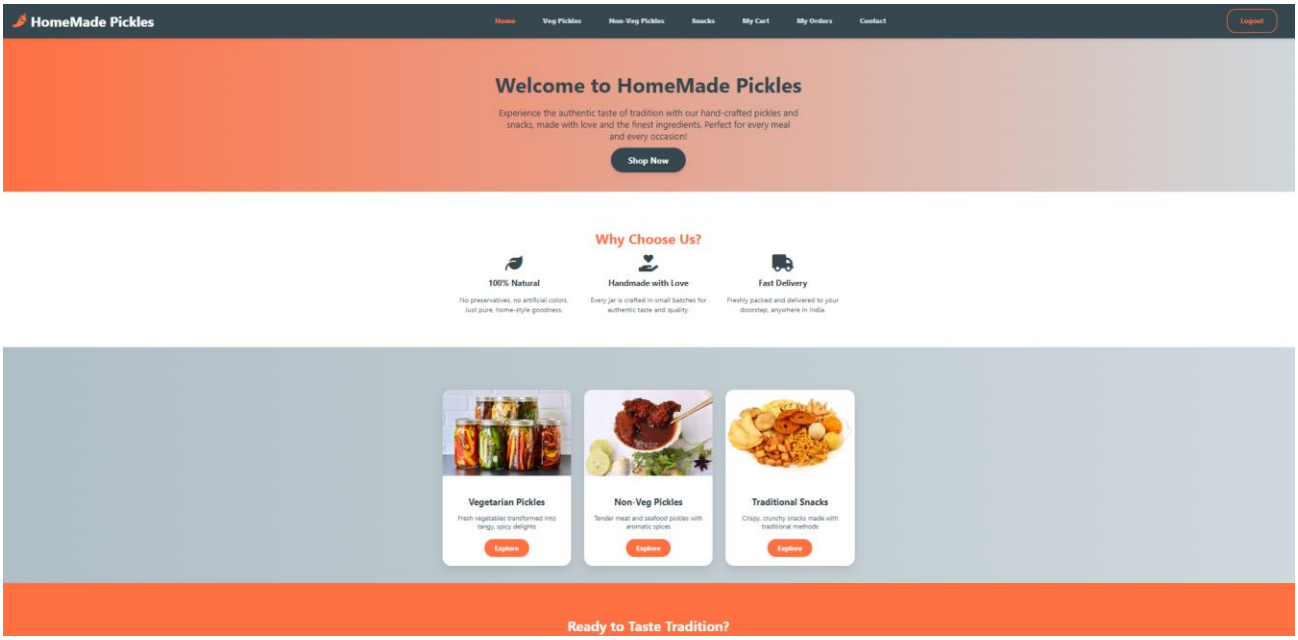
Login Page:



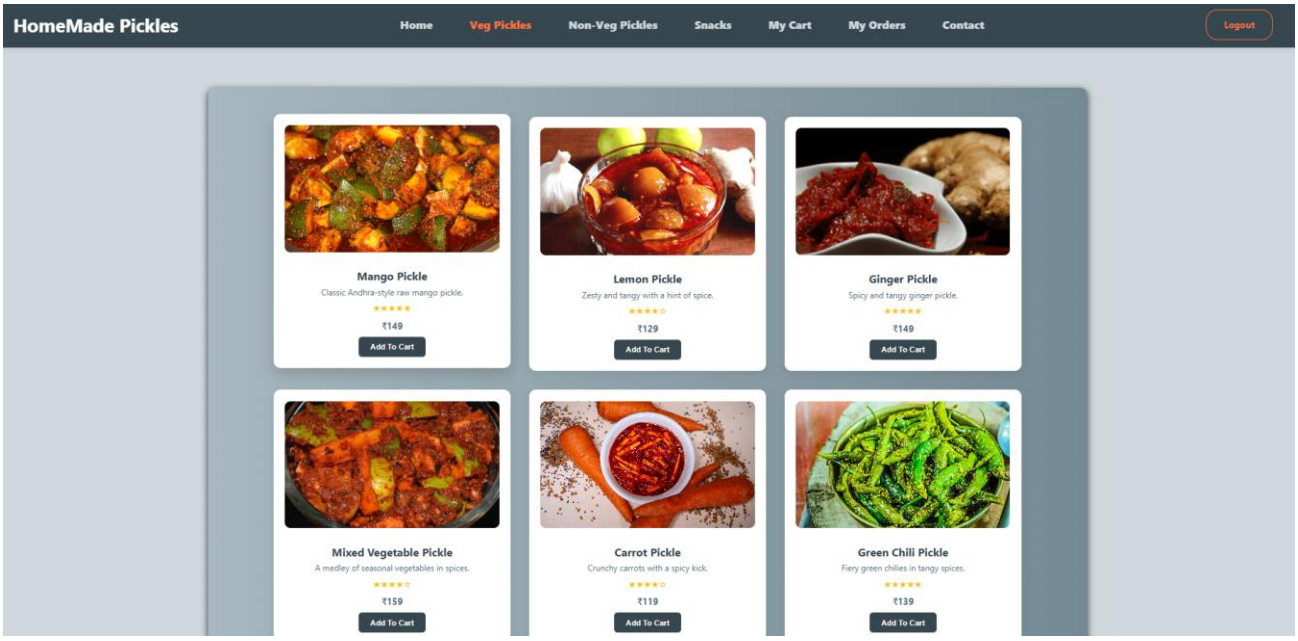
Register Page:



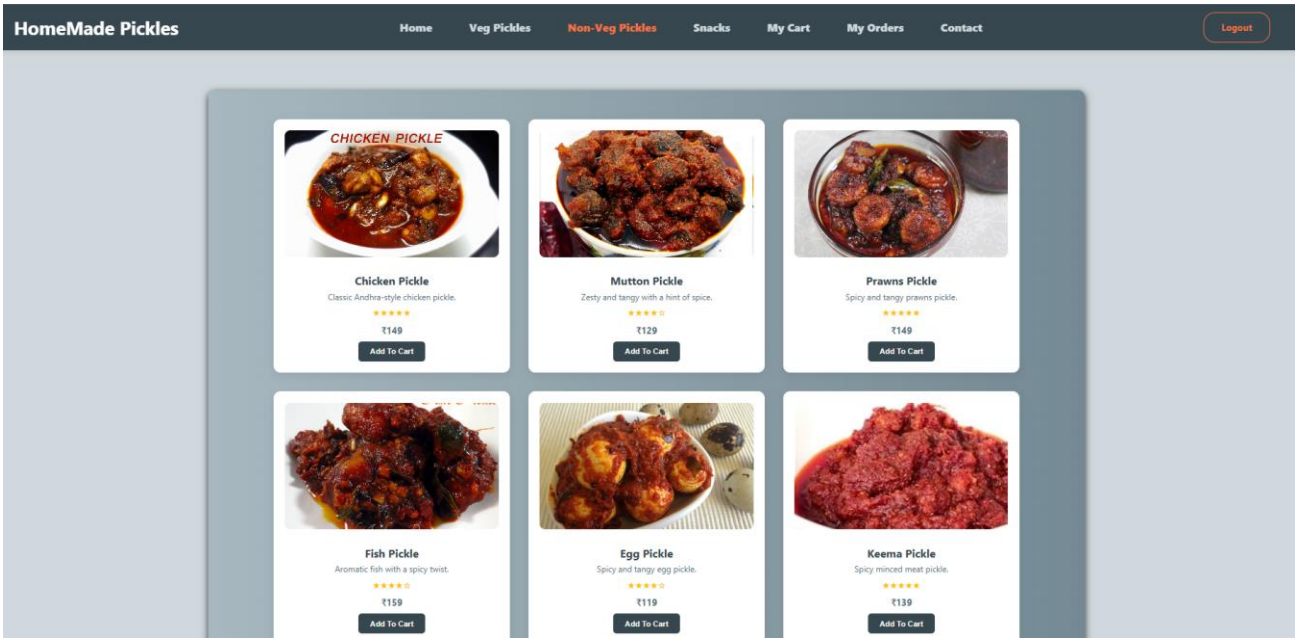
Home page:



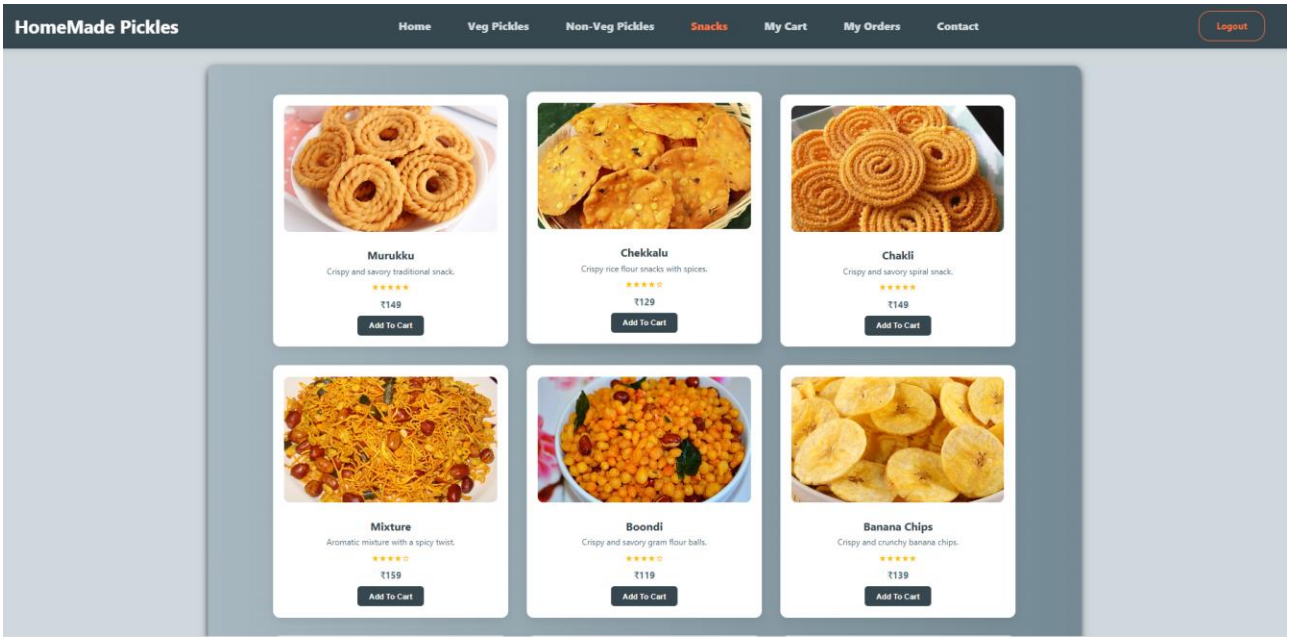
Veg Pickles Page :



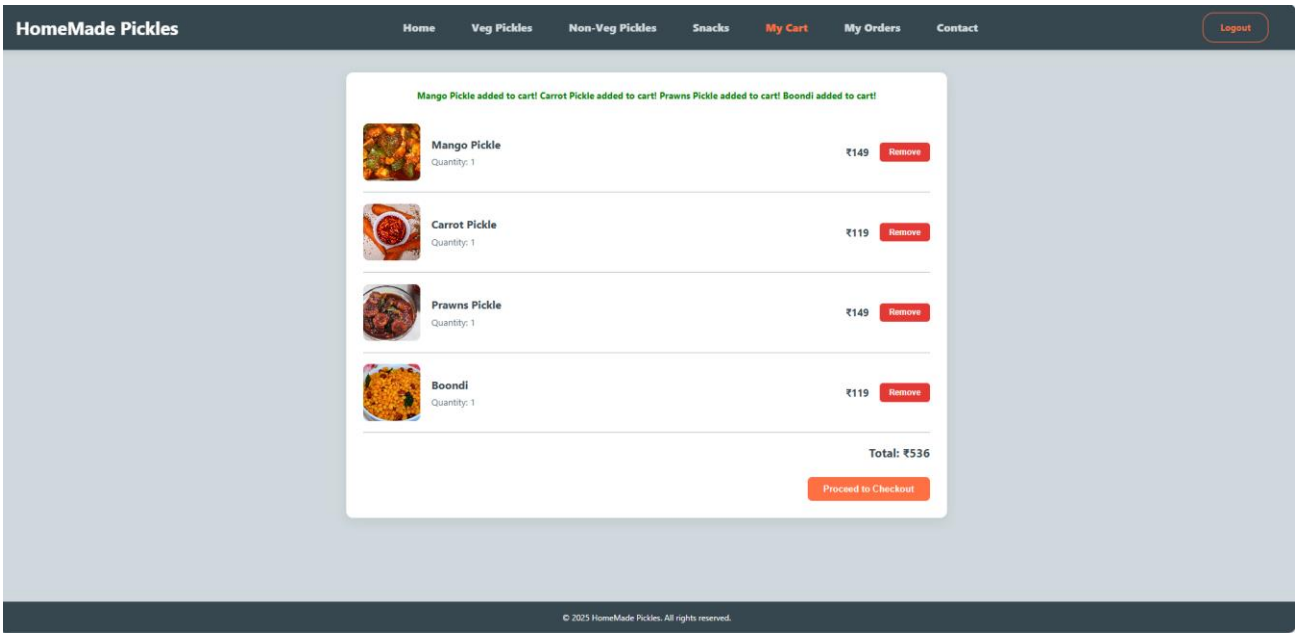
Non-Veg pickles Page:



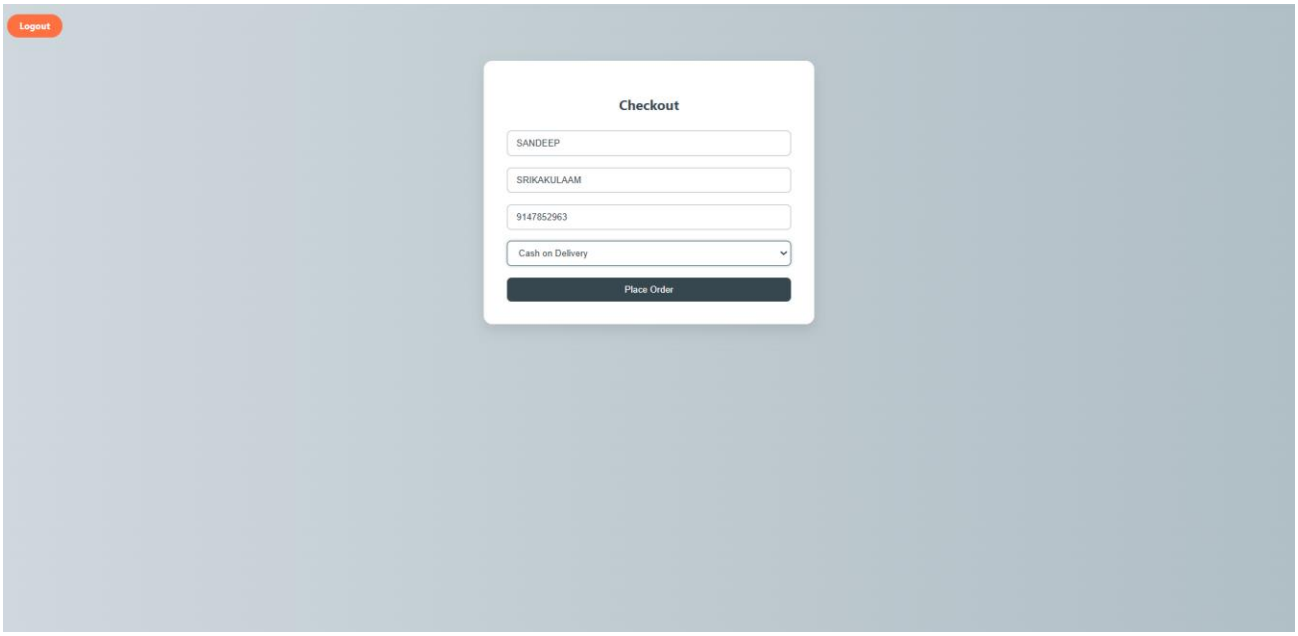
Snacks Page:



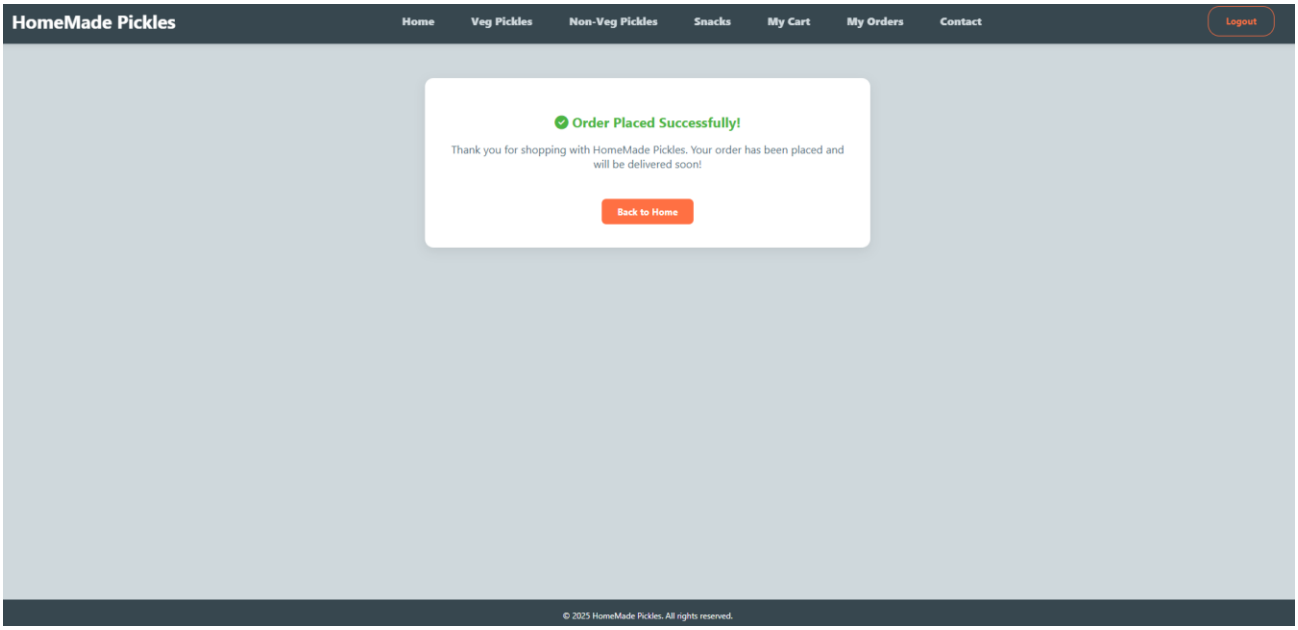
My Cart:



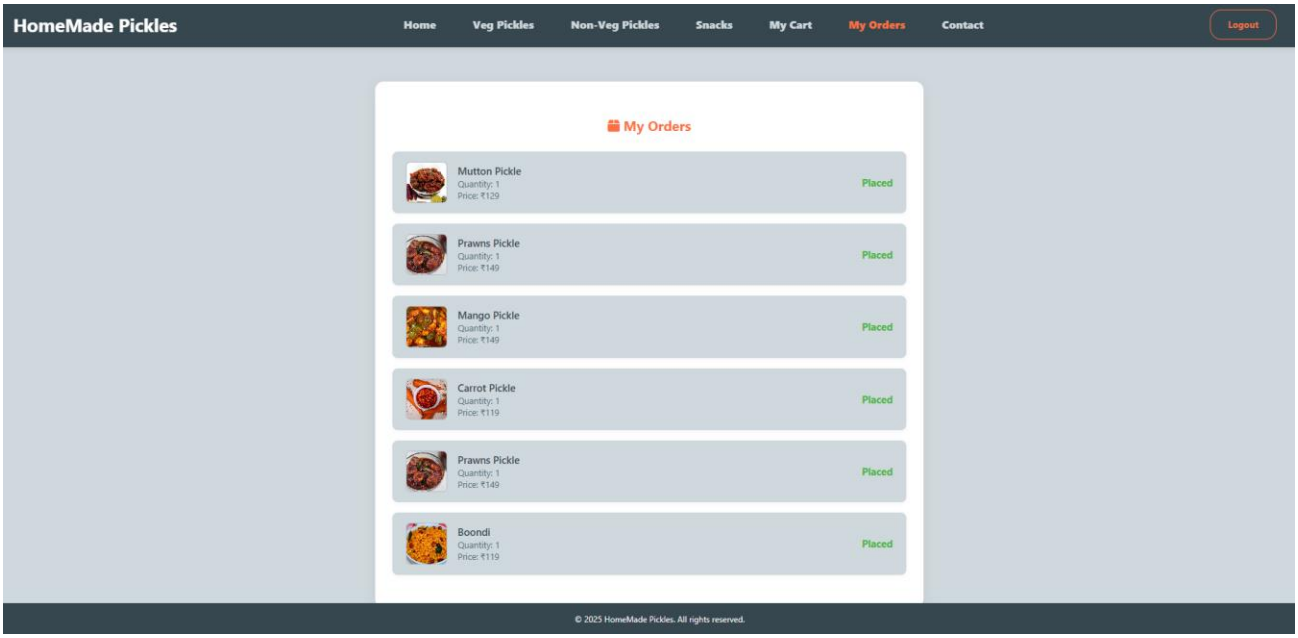
Checkout:



Conformation Page :



Orders Page:



Conclusion:

The Home Made Pickles platform has been successfully developed and deployed using a robust, cloud-native architecture. By leveraging AWS EC2 for hosting, DynamoDB for data management, and SNS for real-time order and registration notifications, the application ensures reliable and scalable access to artisan pickle products. This system overcomes the limitations of traditional sales channels by providing customers with a seamless way to browse offerings, manage carts, and place orders—while makers can efficiently track and fulfill requests.

The cloud-native approach guarantees that as customer demand grows, the platform scales automatically without compromising performance. Integration of Flask with AWS services ensures that backend processes—including user authentication, product browsing, and order processing—run smoothly. Rigorous testing has validated every workflow, from signup through checkout notifications, confirming end-to-end reliability.

In summary, the Home Made Pickles platform offers a modern, efficient solution for small-scale food artisans, enhancing customer experience and operational efficiency. This project showcases the power of cloud-based architectures to transform niche businesses and drive growth.