WAN Pizza – Cloud-Based E-Commerce Application

Module: Cloud Computing (ADDS241F)
Team Members:006,012,020
Submission Date: 08/04/2025

Abstract

Users can access WAN Pizza through a scalable cloud-based e-commerce platform that operates on AWS to choose pizzas while making online orders and personalizing their selections. This system utilizes EC2, RDS along with S3 and Lambda along with API Gateway to guarantee performance scalability as well as cost-efficiency. The main objective is to showcase a robust and resilient deployment system for a commercial web application which utilizes cloud infrastructure.

Introduction

WAN Pizza provides contemporary e-commerce services that support the digital pizza enjoyment of users. Users can access this web application through AWS because its design incorporates secure user access with seamless performance and high availability features. RDS manages structured database activities (users and orders and the menu) and S3 retains various image and static file types. The application runs its backend on EC2 while using API Gateway and Lambda as a scalable API component.

Cloud Architecture Design

Architecture Overview

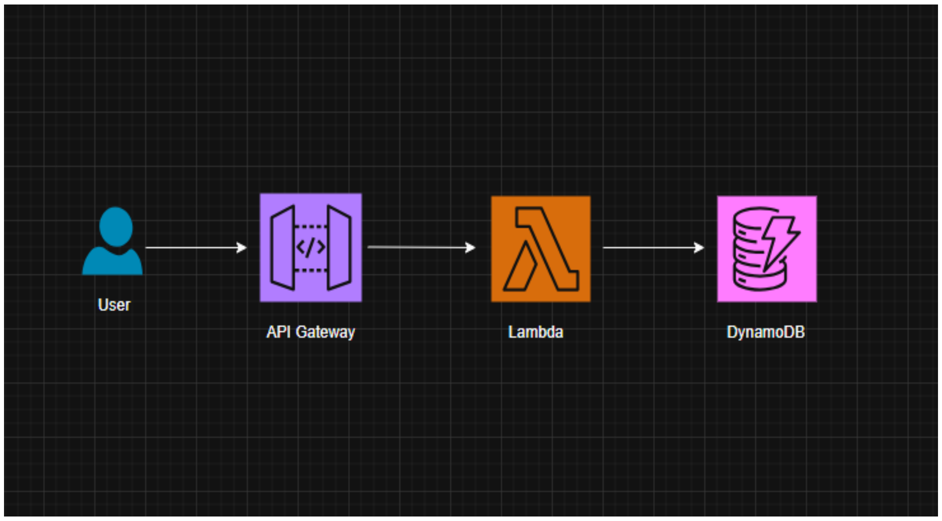
The architecture consists of:

- Frontend**:** Static HTML pages served via Lambda (can optionally be hosted on S3).
- Backend**:** AWS Lambda function handles GET/POST HTTP methods.

• Database**:** Amazon DynamoDB stores order data.

• API **Exposure:** API Gateway connects clients to Lambda endpoints.

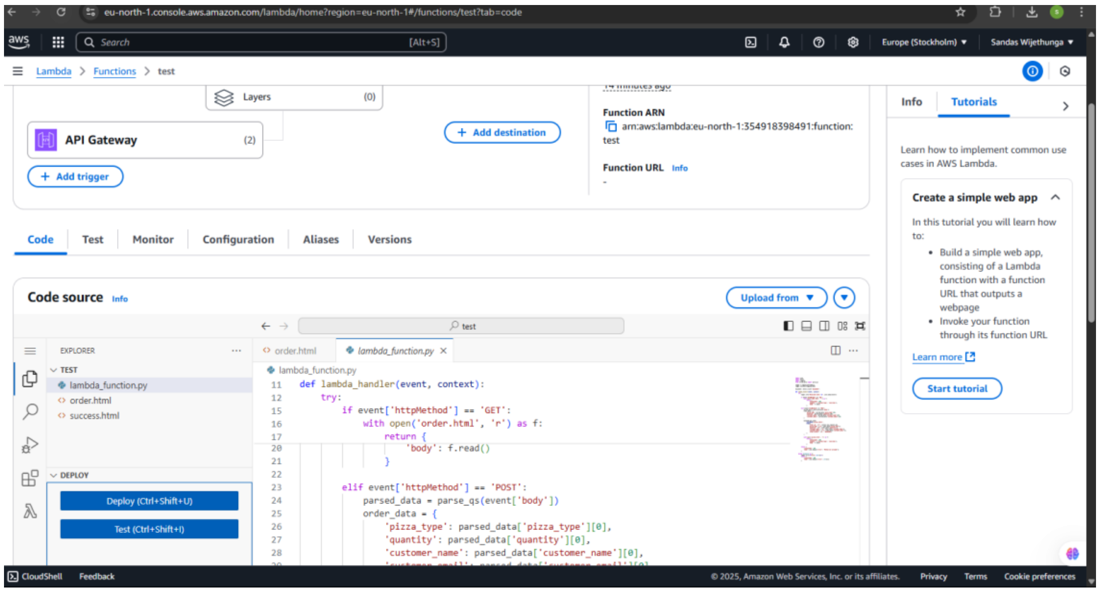
• Monitoring**:** AWS CloudWatch for logs and diagnostics.



Implementation Details

Step 1: Create Lambda Function – Test

- Click **"Create function"**
- Choose runtime: python
- Handles GET and POST methods.
- Returns HTML form (GET) and processes orders (POST).
- Saves order data to DynamoDB.
- Successfully created as shown below



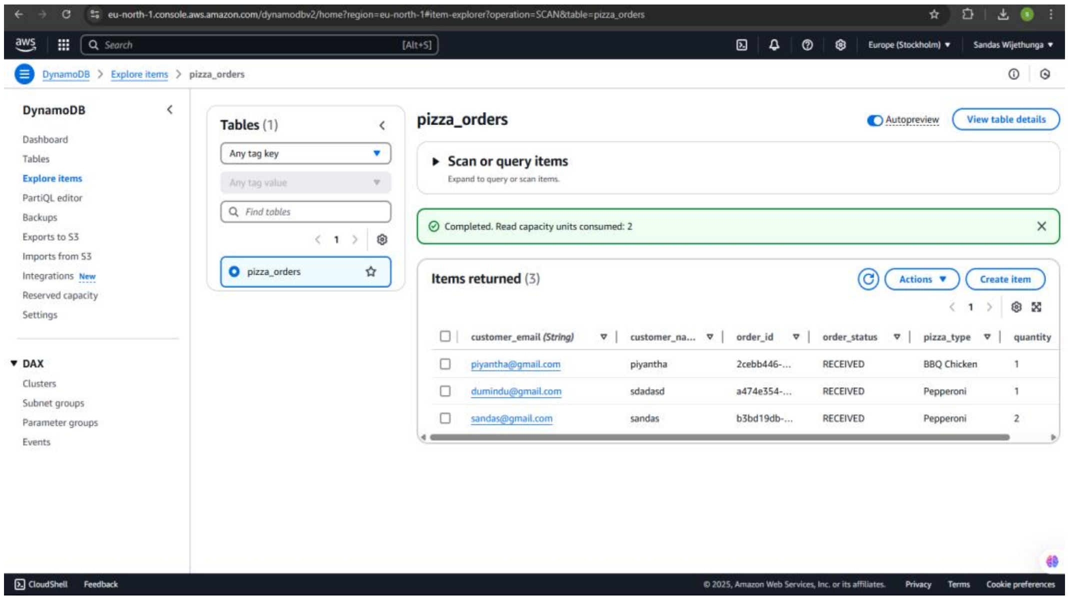
Function ARN
arn:aws:lambda:eu-north-1:354918398491:function:test

```
def lambda_handler(event, context):
    try:
        if event['httpMethod'] == 'GET':
            with open('order.html', 'r') as f:
                return {
                    'body': f.read()
                }

        elif event['httpMethod'] == 'POST':
            parsed_data = parse_qs(event['body'])
            order_data = {
                'pizza_type': parsed_data['pizza_type'][0],
                'quantity': parsed_data['quantity'][0],
                'customer_name': parsed_data['customer_name'][0],
```

Step 2: Create DynamoDB Table – Pizza_ orders

- Navigate to DynamoDB
- Click **"Create table"**
- Table Name: pizza_orders
- Partition Key: email (String)
- Provisioned mode: On-Demand
- Successfully initiated table creation as shown:



Completed. Read capacity units consumed: 2

Items returned (3)

customer_email (String)	customer_na...	order_id	order_status	pizza_type	quantity
piyantha@gmail.com	piyantha	2cebb446-...	RECEIVED	BBQ Chicken	1
dumindu@gmail.com	sdadasd	a474e354-...	RECEIVED	Pepperoni	1
sandas@gmail.com	sandas	b3bd19db-...	RECEIVED	Pepperoni	2

Step 3: Connect Lambda to API Gateway

To allow users or your frontend to access the backend logic (Lambda), we need to expose an HTTP endpoint using **Amazon API Gateway**.

Steps Taken:

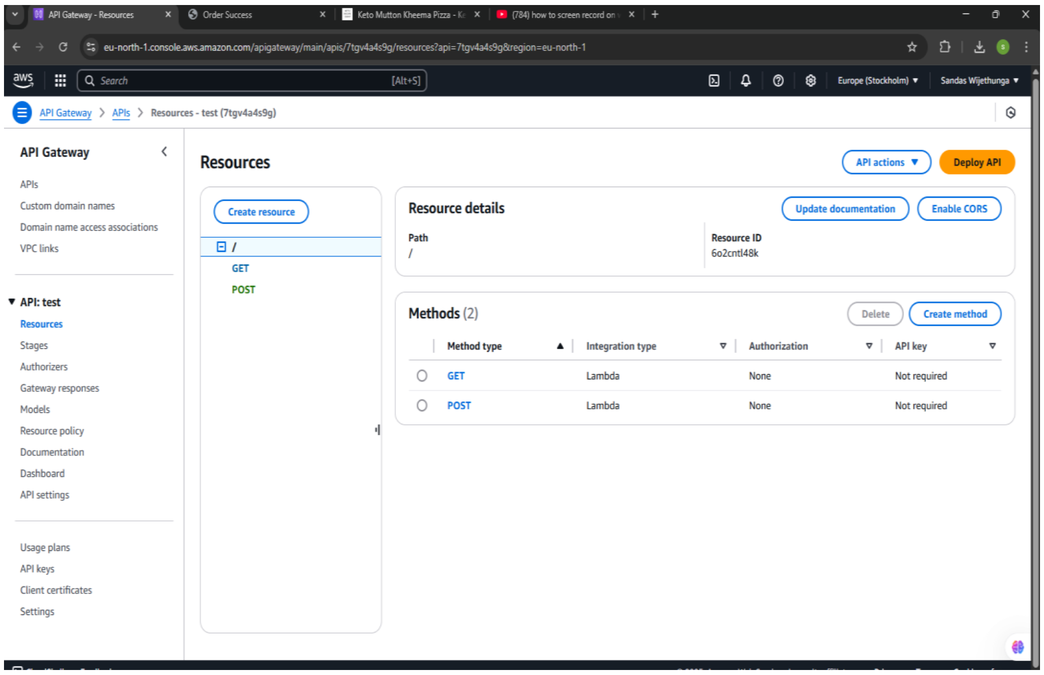
1. Go to **API Gateway** in AWS Console.
2. Create a new **HTTP API**.
3. Select **"Add integration"** → **Lambda function**.
4. Choose your Lambda function - test
5. Two methods created:

   GET → Returns order form.

   POST → Invokes Lambda to store order data.

6. Deploy the API and note the **public URL**.

   https://7tgv4a4s9g.execute-api.eu-north-1.amazonaws.com/test



Resource details

Path: /

Resource ID: 6o2cntl48k

Methods (2)

Method type	Integration type	Authorization	API key
GET	Lambda	None	Not required
POST	Lambda	None	Not required

Cost Analysis and Optimization

Using serverless and on-demand services like Lambda and DynamoDB keeps costs minimal:

- **Lambda** only charges per invocation
- **DynamoDB** on-demand mode helps avoid over-provisioning
- **Cost Optimization Tactics**:
    - Use AWS Free Tier where possible
    - Monitor usage with **Billing Dashboard** and **CloudWatch**
    - Set alarms for unexpected costs

Security & IAM

- IAM roles created automatically for Lambda to access other services securely
- DynamoDB permissions restricted via IAM policies
- Lambda runs inside secured AWS environment
- CORS can be enabled if hosted frontend requires cross-origin requests.

Disaster Recovery and Backup

- Enable **DynamoDB point-in-time recovery** for data protection
- Use **CloudWatch Logs** to trace Lambda errors

Challenges & Solutions

Challenges	Solutions
Lambda to DynamoDB integration	Used `boto3` SDK and IAM role permissions
Testing form data input	Built POST handler and tested via API Gateway
Visualizing backend data	Used DynamoDB Console to confirm order entry

Conclusion & Future Work

WAN Pizza effectively demonstrates how to build and deploy a scalable e-commerce app using AWS serverless technologies. It reduces operational complexity while offering flexibility for future expansion.

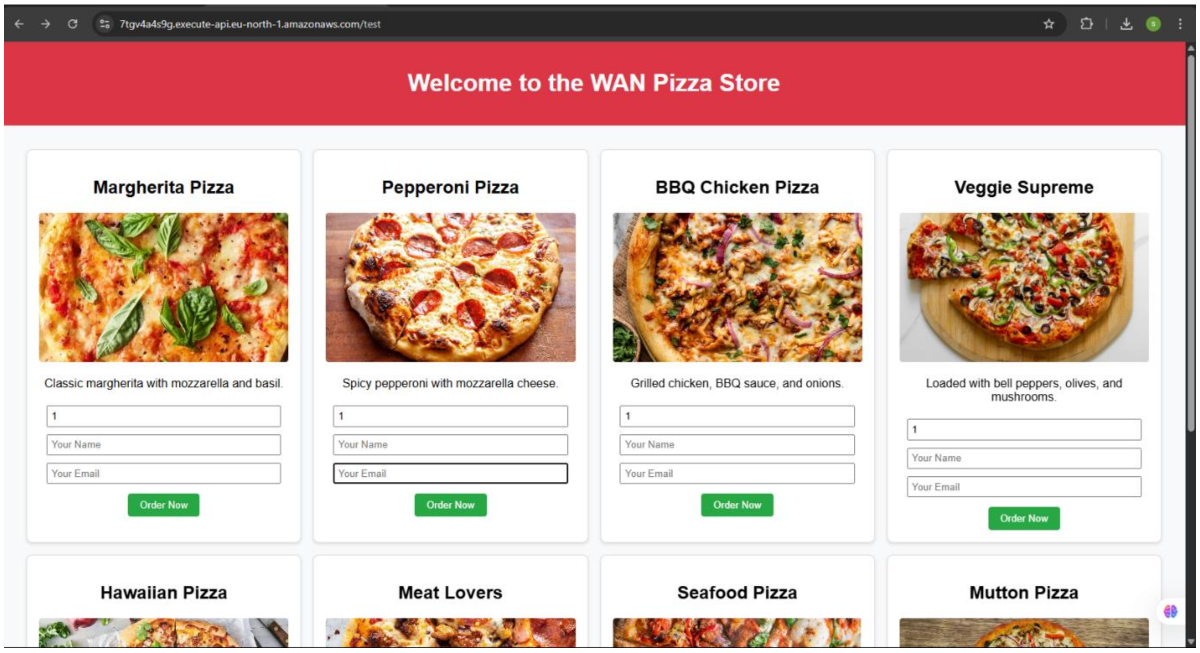
Future Improvements

- Integrate authentication via **Amazon Cognito**
- Build a dynamic frontend using **React** or **Vue.js**
- Add admin dashboard for analytics
- Implement email notifications upon order receipt

Conclusion & Future Work

WAN Pizza demonstrates a fully serverless, cost-effective e-commerce platform using AWS. It shows the flexibility and scalability cloud computing offers. Future improvements could include:

- User authentication with Cognito
- Full frontend integration with React
- Order history and admin dashboard



7tgv4a4s9g.execute-api.eu-north-1.amazonaws.com/test

Welcome to the WAN Pizza Store

Margherita Pizza

Classic margherita with mozzarella and basil.

Pepperoni Pizza

Spicy pepperoni with mozzarella cheese.

BBQ Chicken Pizza

Grilled chicken, BBQ sauce, and onions.

Veggie Supreme

Loaded with bell peppers, olives, and mushrooms.

Hawaiian Pizza

Meat Lovers

Seafood Pizza

Mutton Pizza