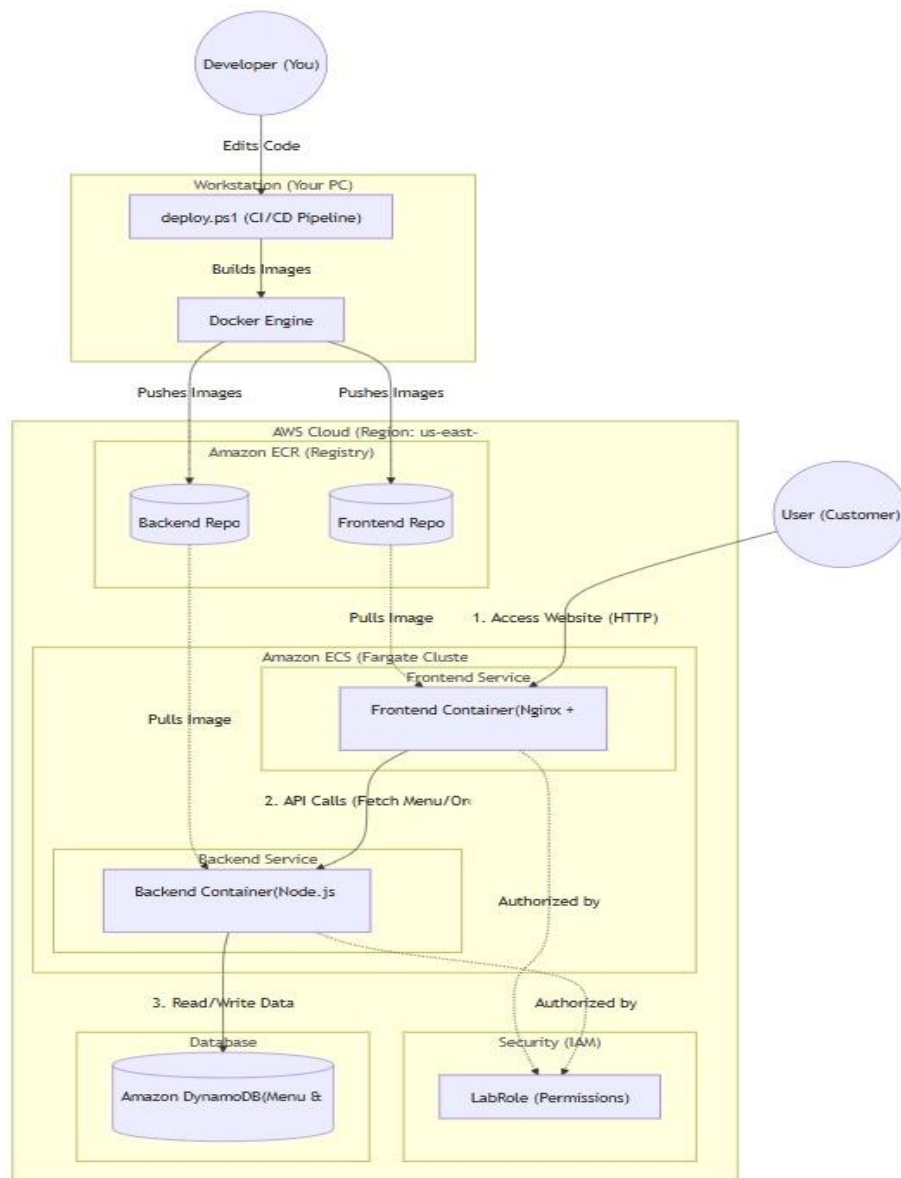


# AWS ARCHITECTURE DESIGN, IMPLEMENTATION STEPS & SCREENSHOTS

**Student Name:** Fatima Imran (BSSE23098)

**Project Title:** Serverless Food Ordering Platform

## 1. AWS ARCHITECTURE DIAGRAM



**Description:** The architecture utilizes a serverless approach. The User accesses the application via the **Frontend Service** running on **AWS ECS Fargate**. The frontend communicates with the **Backend Service** (also on Fargate) via REST API. The backend handles business logic and persists data to **Amazon DynamoDB**. All Docker images are stored in **Amazon ECR**.

## 2. STEP-BY-STEP IMPLEMENTATION

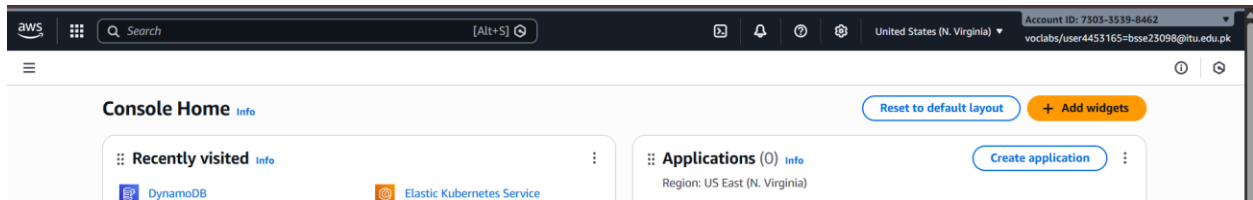
This section details the deployment process executed on the AWS Learner Lab environment.

### Phase 1: AWS Account & Security Setup

**Goal:** Configure the environment to allow resource creation.

1. **Access Learner Lab:** Logged into the AWS Academy Learner Lab portal and started the lab environment.
2. **IAM Role Identification:** Identified the LabRole

. This pre-configured role allows ECS and DynamoDB access without needing to create custom IAM policies (which is restricted in Learner Lab).

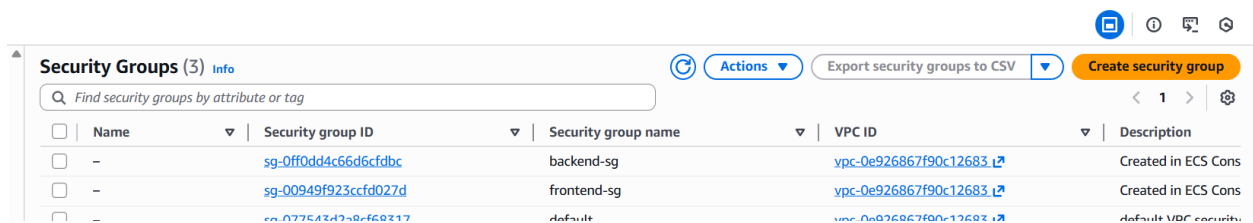


### AWS Management Console Dashboard

### Phase 2: Network & Security Configuration (VPC)

**Goal:** Ensure network isolation and traffic flow.

1. **VPC Selection:** Utilized the Default VPC (vpc-xxxxxxx) provided by the lab to ensure internet gateway access.
2. **Security Groups:** Created two security groups:
  - frontend-sg: Inbound Rule = HTTP (Port 80) from 0.0.0.0/0
  - backend-sg: Inbound Rule = HTTP (Port 80) from 0.0.0.0/0



**Security Groups configured for Frontend and Backend.**

Phase 3: Database Implementation (DynamoDB)

Goal: Create serverless tables for data storage.

- 1. **Menu Table:** Created table MenuTable with Partition Key id(String).
- 2. **Orders Table:** Created table OrdersTable with Partition Key orderId (String).
- 3. **Capacity Mode:** Selected "On-Demand" to minimize costs.

Tables (2) Info

Last updated January 2, 2026, 19:22 (UTC+5:00)

Actions

Delete

Create table

Find tables

Filter by tag Any tag key

Filter by tag value Any tag value

< 1 >

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity
<input type="checkbox"/>	<a href="#">MenuTable</a>	Active	id (S)	-	0	0	Off	☆	On-demand
<input type="checkbox"/>	<a href="#">OrdersTable</a>	Active	orderId (S)	-	0	0	Off	☆	On-demand

Serverless DynamoDB Tables created.

Phase 4: Container Registry (ECR)

Goal: Store application Docker images.

- 1. Created a private repository named food-ordering-backend
- 2. Created a private repository named food-ordering-frontend
- 3. Verified the URI (e.g., 123456789.dkr.ecr.us-east-1.amazonaws.com/...) for deployment scripts.

Private repositories (2)

View push commands

Delete

Actions

Create repository

Search by repository substring

<input type="radio"/>	Repository name	URI	Created at	Tag immutability	Encryption type
<input type="radio"/>	food-ordering-backend	730335398462.dkr.ecr.us-east-1.amazonaws.com/food-ordering-backend	December 30, 2025, 01:12:04 (UTC+05)	Mutable	AES-256
<input type="radio"/>	food-ordering-frontend	730335398462.dkr.ecr.us-east-1.amazonaws.com/food-ordering-frontend	December 30, 2025, 01:12:26 (UTC+05)	Mutable	AES-256

Amazon Elastic Container Registry repositories.

Phase 5: Deployment Process (CI/CD)

Goal: Automate the Build-and-Push process.

- 1. **Scripting:** Developed a PowerShell script (deploy.ps1) to handle the Docker lifecycle.

- *Step A:* Authenticate Docker client to ECR (aws ecr get-login-password).
- *Step B:* docker build -t ... for both services.
- *Step C:* docker push ... to upload images to AWS.

2. **Execution:** Ran the script from the local workstation.

```
[4/5] Building FRONTEND Image...
[+] Building 3.2s (7/7) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 139B                             0.0s
=> [internal] load metadata for docker.io/library/nginx:alpine  2.4s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 237B                                  0.0s
=> [1/2] FROM docker.io/library/nginx:alpine@sha256:8491795299c8e739 0.1s
=> => resolve docker.io/library/nginx:alpine@sha256:8491795299c8e739 0.1s
=> CACHED [2/2] COPY . /usr/share/nginx/html                    0.0s
=> exporting to image                                           0.3s
=> => exporting layers                                           0.0s
=> => exporting manifest sha256:dd45a016751abbb6fbd4e7572d1eb3541ed6 0.0s
=> => exporting config sha256:07ac4cc9cb06f8e4722ffbe8de8712eca84502 0.0s
=> => exporting attestation manifest sha256:e59d644b7e1afe7a7108aa01 0.1s
=> => exporting manifest list sha256:a244a35b205b2795271a8528f41e3f8 0.0s
=> => naming to docker.io/library/food-ordering-frontend:latest 0.0s
=> => unpacking to docker.io/library/food-ordering-frontend:latest 0.0s
[5/5] Pushing FRONTEND to ECR...
The push refers to repository [730335398462.dkr.ecr.us-east-1.amazonaws.com/food-ordering-frontend]
734679ba2e0c: Pushed
1074353e0c0d: Layer already exists
085c5e5aaa8e: Layer already exists
567f84da6fbd: Layer already exists
cfc856a15d80: Layer already exists
da7c973d8b92: Layer already exists
33f95a0f3229: Layer already exists
0abf9e567266: Layer already exists
de54cb821236: Layer already exists
25f453064fd3: Layer already exists
latest: digest: sha256:a244a35b205b2795271a8528f41e3f8c702ebfd3909ffa27a1f4f2ef7e960580 size: 856
-----
BUILD & PUSH COMPLETE!
```

*Automated Deployment Script executing successfully.*

## Phase 6: Orchestration Setup (ECS Fargate)

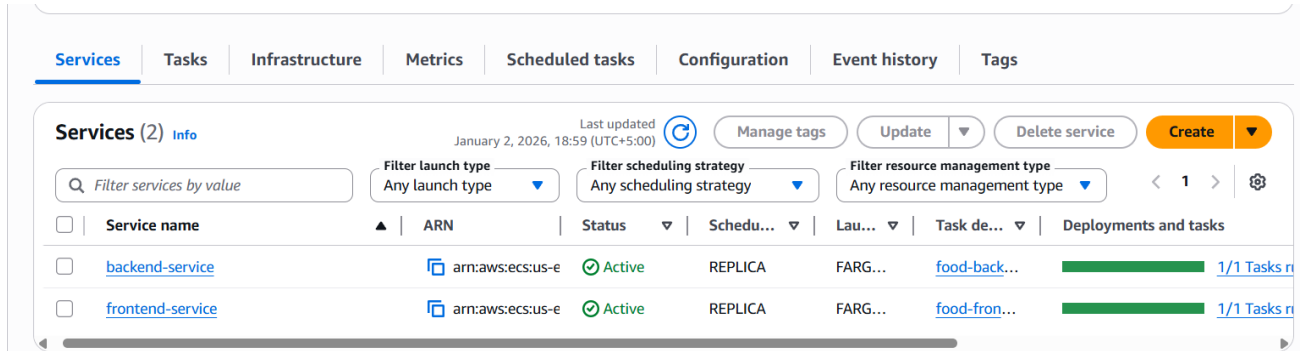
**Goal:** Run the containers serverlessly.

1. **Cluster:** Created a Fargate Cluster named FoodPlatformCluster
2. **Task Definitions:**
  - defined food-backend-task(Node.js container, Port 80, Environment Variables for DynamoDB).
  - defined food-frontend-task (Nginx container, Port 80).
  - **IAM Integration:** Assigned LabRole to "Task Role" and "Execution Role" to allow DynamoDB access.

### 3. Services:

- Launched backend-service (1 Desired Task).
- Launched frontend-service (1 Desired Task, Public IP ENABLED).

4.



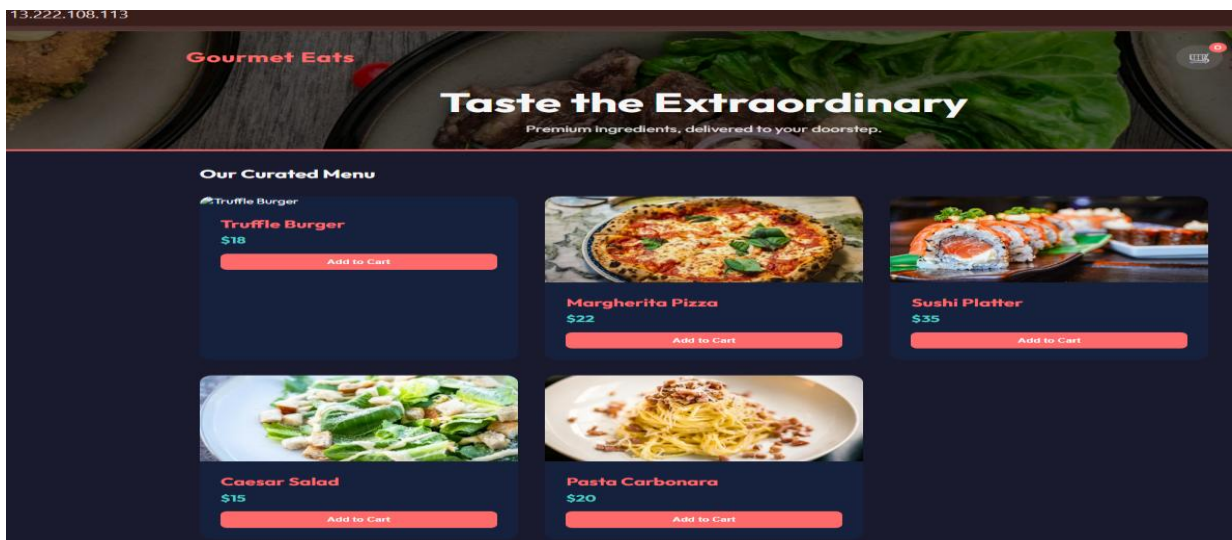
Services (2) <a href="#">Info</a>							
January 2, 2026, 18:59 (UTC+5:00) <a href="#">Manage tags</a> <a href="#">Update</a> <a href="#">Delete service</a> <a href="#">Create</a>							
<input type="text" value="Filter services by value"/> <input type="button" value="Filter launch type"/> Any launch type <input type="button" value="Filter scheduling strategy"/> Any scheduling strategy <input type="button" value="Filter resource management type"/> Any resource management type							
<input type="checkbox"/>	Service name	ARN	Status	Schedu...	Lau...	Task de...	Deployments and tasks
<input type="checkbox"/>	<a href="#">backend-service</a>	arn:aws:ecs:us-e	Active	REPLICA	FARG...	<a href="#">food-back...</a>	<div></div> <a href="#">1/1 Tasks r</a>
<input type="checkbox"/>	<a href="#">frontend-service</a>	arn:aws:ecs:us-e	Active	REPLICA	FARG...	<a href="#">food-fron...</a>	<div></div> <a href="#">1/1 Tasks r</a>

*ECS Cluster services running in Fargate.*

## Phase 7: Testing Workflow

**Goal:** Verify functionality.

1. **Backend Verification:** Accessed <http://<BACKEND-IP>/menu> in the browser.
  - *Result:* Received JSON response of menu items.
2. **Frontend Verification:** Accessed the Frontend Public IP.
  - *Result:* Web Application loaded.
3. **End-to-End Test:** Added items to cart and clicked "Checkout".
  - *Result:* Order ID generated and saved to DynamoDB.



Functional User Interface.

Your Order

Margherita Pizza

\$22

Sushi Platter

\$35

Margherita Pizza

\$22

Margherita Pizza

\$22

Margherita Pizza

\$22

Margherita Pizza

\$22

Sushi Platter

\$35

Sushi Platter

\$35

Sushi Platter

\$35

Sushi Platter

\$35

Truffle Burger

\$18

Truffle Burger

\$18

Total: \$321.00

Enter your Name

Confirm Order

Table: OrdersTable - Items returned (11)

Scan started on December 30, 2025, 20:06:38

Actions

Create item

<div><div>&lt;</div><div>1</div><div>&gt;</div></div> <div></div>						
<input type="checkbox"/>	orderId (String)	<input type="checkbox"/> customerNa...	<input type="checkbox"/> items	<input type="checkbox"/> status	<input type="checkbox"/> timestamp	total
<input type="checkbox"/>	<a href="#">1767105318213</a>	fatima	[{"M": {"n...	Pending	2025-12-3...	79
<input type="checkbox"/>	<a href="#">1767101919853</a>	fatima	[{"M": {"n...	Pending	2025-12-3...	40
<input type="checkbox"/>	<a href="#">1767106578783</a>	fatima	[{"M": {"n...	Pending	2025-12-3...	174
<input type="checkbox"/>	<a href="#">1767044497334</a>	fatima	[{"M": {"n...	Pending	2025-12-2...	25
<input type="checkbox"/>	<a href="#">1767106168116</a>	fatima	[{"M": {"n...	Pending	2025-12-3...	79
<input type="checkbox"/>	<a href="#">1767102357762</a>	fatima	[{"M": {"n...	Pending	2025-12-3...	57
<input type="checkbox"/>	<a href="#">1767105874696</a>	fatima	[{"M": {"n...	Pending	2025-12-3...	77
<input type="checkbox"/>	<a href="#">1767107176644</a>	<div>fatima<div></div></div>	[{"M": {"n...	Pending	2025-12-3...	321
<input type="checkbox"/>	<a href="#">1767106264879</a>	fatima	[{"M": {"n...	Pending	2025-12-3...	22
<input type="checkbox"/>	<a href="#">1767106900931</a>	fatima	[{"M": {"n...	Pending	2025-12-3...	187
<input type="checkbox"/>	<a href="#">1767043604531</a>	fatima	[{"M": {"n...	Pending	2025-12-2...	10

Proof of Data Persistence after an order.

Phase 8: Monitoring & Logging (CloudWatch)

Goal: Observe application health.

- 1. Accessed **AWS CloudWatch** via the ECS Console "Logs" tab.
- 2. Verified backend-service logs to see "Server running on port 80" and "Connected to DynamoDB".

Log streams (7)

Delete

Create

By default, we only load the most recent log streams.

Q

Filter log streams or try prefix search

☐ Exact match

☐ Show ex

<input type="checkbox"/>	Log stream	Last event time
<input type="checkbox"/>	<a href="#">ecs/backend/433abf7887a149509857e6d6094c9724</a>	2025-12-29 22:56:03 (UTC)
<input type="checkbox"/>	<a href="#">ecs/backend/bf79e1c71e4f4069a5df9a3cf5735bab</a>	2025-12-29 22:28:09 (UTC)
<input type="checkbox"/>	<a href="#">ecs/backend/e92584085b4149569cef072c47b66b3a</a>	2025-12-29 22:18:26 (UTC)
<input type="checkbox"/>	<a href="#">ecs/backend/abd9fb901b284641b623901d9208ce95</a>	2025-12-29 22:05:10 (UTC)
<input type="checkbox"/>	<a href="#">ecs/backend/db2bdfc51d2942c1959d2c7456d798bd</a>	2025-12-29 21:52:00 (UTC)
<input type="checkbox"/>	<a href="#">ecs/backend/7eca3012e5944e6ebf0dec2b853c0b75</a>	2025-12-29 21:46:44 (UTC)
<input type="checkbox"/>	<a href="#">ecs/backend/bea536b00a984342ada59c015023d417</a>	2025-12-29 20:47:55 (UTC)

CloudWatch Logs verifying backend connectivity.

FINAL OUTPUT

The project is fully deployed. The architecture is robust, utilizing **AWS Fargate** for zero-maintenance compute and **DynamoDB** for limitless storage scale. The security is managed via **IAM Roles** and **Security Groups**, adhering to the principle of least privilege within the Learner Lab constraints.



Successful Transaction flow.

