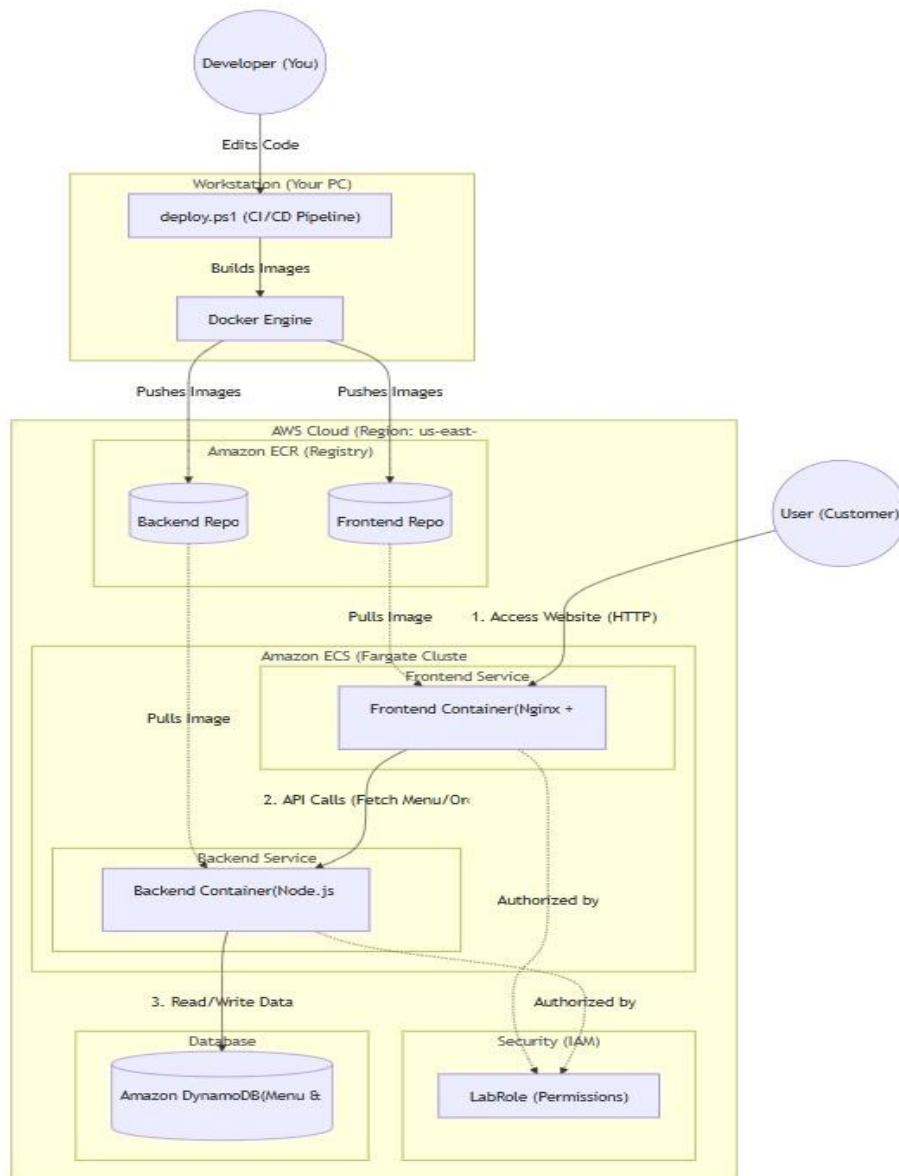


# AWS ARCHITECTURE DESIGN, IMPLEMENTATION STEPS & SCREENSHOTS

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

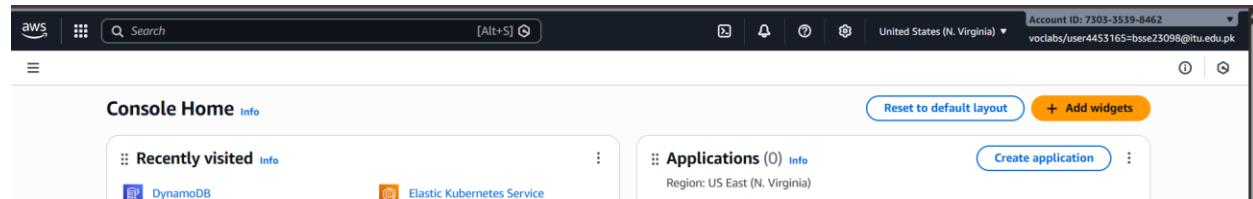
This section details the deployment process executed on the AWS Learner Lab environment.

### 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-xxxxxxxx) 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 (3) <small>Info</small>					
<small>Find security groups by attribute or tag</small>					
Name	Security group ID	Security group name	VPC ID	Description	Actions
-	sg-0ff0dd4c66d6cfdb	backend-sg	vpc-0e926867f90c12683	Created in ECS Cons	
-	sg-00949f923ccfd027d	frontend-sg	vpc-0e926867f90c12683	Created in ECS Cons	
-	sg-0775124798fc0217	default	vpc-0e926867f90c12683	default VPC security	

**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.

The screenshot shows the AWS DynamoDB 'Tables' page. At the top, it displays 'Tables (2)' and the last update time as 'January 2, 2026, 19:22 (UTC+5:00)'. There are filters for 'Filter by tag' (Any tag key) and 'Filter by tag value' (Any tag value). Below the filters is a search bar with placeholder 'Find tables'. To the right are buttons for 'Actions', 'Delete', and 'Create table'. The main table lists two tables: 'MenuTable' and 'OrdersTable'. Both tables have an 'Active' status, 'id (\$)' as the partition key, and 'orderId (\$)' as the sort key. They both have 0 items and 0 bytes. Deletion protection is set to 'Off' for both. The 'Read capacity' is listed as 'On-demand' for both. The table has columns for Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read capacity.

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

The screenshot shows the AWS ECR 'Private repositories' page. At the top, it displays 'Private repositories (2)'. There is a search bar with placeholder 'Search by repository substring'. To the right are buttons for 'View push commands', 'Delete', 'Actions', and 'Create repository'. The main table lists two repositories: 'food-ordering-backend' and 'food-ordering-frontend'. Both were created on December 30, 2025, at 01:12:04 (UTC+05). Both are mutable and use AES-256 encryption. The table has columns for Repository name, URI, Created at, Tag immutability, and Encryption type.

Repository name	URI	Created at	Tag immutability	Encryption type
<a href="#">food-ordering-backend</a>	<a href="#">730335398462.dkr.ecr.us-east-1.amazonaws.com/food-ordering-backend</a>	December 30, 2025, 01:12:04 (UTC+05)	Mutable	AES-256
<a href="#">food-ordering-frontend</a>	<a href="#">730335398462.dkr.ecr.us-east-1.amazonaws.com/food-ordering-frontend</a>	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
=> [internal] load build definition from Dockerfile               docker:desktop-linux
=> => 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 sha256:07ac4cc9cb06fbe4722ffbe8de8712eca84502 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
1074353eecd0d: Layer already exists
085c5e5aaa8e: Layer already exists
567f84da6fb0d: 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).

The screenshot shows the AWS CloudWatch Metrics interface with the 'Services' tab selected. There are two services listed: 'backend-service' and 'frontend-service'. Both services are active and have a status of 'REPLICA'. They are running on the 'arn:aws:ecs:us-east-1' account. The 'backend-service' has 1/1 tasks running, and the 'frontend-service' also has 1/1 tasks running. The interface includes filters for launch type, scheduling strategy, and resource management type, along with buttons for 'Update', 'Delete service', and 'Create'.

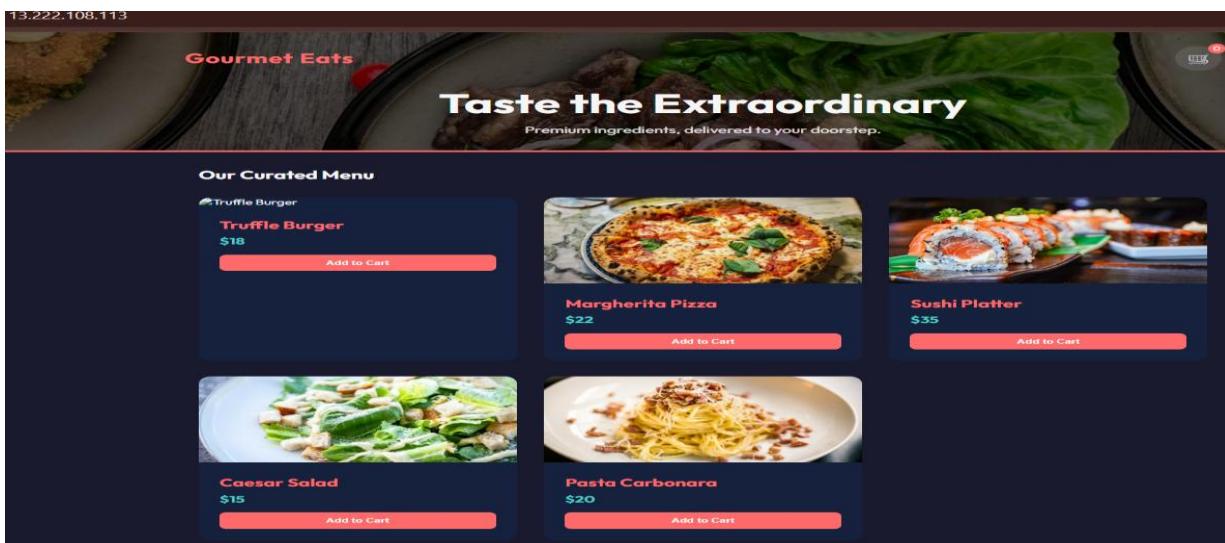
4.

**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
Sushi Platter	\$35
Sushi Platter	\$35
Sushi Platter	\$35
Truffle Burger	\$18
Truffle Burger	\$18
<b>Total: \$321.00</b>	

Enter your Name

**Confirm Order**

**Table: OrdersTable - Items returned (11)**

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

( Actions ▾ Create item)

	orderId (String)	customerNa...	items	status	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>	fatima	[ {"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

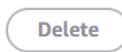
**Proof of Data Persistence after an order.**

## 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)

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

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

