

PROJECT PROPOSAL

Project Title: Serverless & Containerized Food Ordering Platform

Name: Fatima Imran (BSSE23098)

1. EXECUTIVE SUMMARY

The hospitality industry is undergoing a digital transformation, where speed and reliability are paramount. This project proposes the design and implementation of a **Serverless Food Ordering Platform** utilizing AWS cloud-native technologies.

- **The Challenge:** Traditional web hosting on Virtual Machines (VMs) imposes heavy operational burdens, high costs for idle resources, and difficulties in scaling during peak hours.
- **The Solution:** A microservices-based architecture where the application is containerized using **Docker** and orchestrated by **AWS ECS Fargate** (Serverless). Data persistence is managed by **Amazon DynamoDB**, ensuring high-speed access without database provision overhead.
- **Implementation:** The project utilizes a custom CI/CD pipeline built with **PowerShell** to automate the build-and-deploy lifecycle to **Amazon ECR**.
- **Outcome:** The final system demonstrates a production-ready environment that auto-scales with demand, reduces maintenance to near-zero, and offers a robust user experience for ordering food online.

2. INTRODUCTION

In the modern digital economy, customers expect seamless, fast, and always-available online services. For local restaurants and food chains, maintaining an online presence is no longer optional but a necessity for survival. However, the technical barrier to entry is high. Managing physical servers or configuring complex Virtual Private servers (VPS) requires specialized IT knowledge that most restaurant owners lack.

Moreover, traffic patterns in the food industry are highly bursty—lunch and dinner hours see massive spikes, while other times remain quiet. A traditional fixed-server infrastructure is either under-provisioned (leading to crashes during rush hour) or over-provisioned (leading to wasted money).

This project explores the "**Serverless First**" methodology. By abstracting the underlying infrastructure, we allow the business to focus solely on the application code and customer experience. We leverage the AWS ecosystem to provide enterprise-grade reliability to a common real-world problem.

3. PROBLEM STATEMENT

Context: Small-to-medium restaurants currently rely on manual phone orders or third-party aggregators that charge high commissions. Those attempting to host their own websites often face reliability issues.

The Core Problems:

5. **Scalability Bottlenecks:** Traditional hosting on a single VM cannot handle sudden spikes in traffic (e.g., a "Viral" promotion).
6. **Operational Overhead:** Servers require constant operating system updates, security patches, and monitoring.
7. **Deployment Complexity:** Updating a live website often involves downtime or complex manual file transfers (FTP), leading to errors.
8. **Cost Inefficiency:** Paying for a server 24/7, even when the restaurant is closed or has zero traffic.

Justification: There is a critical need for a solution that scales automatically, requires zero server management, and optimizes costs based on actual usage.

4. AIM & OBJECTIVES

4.1 Aim

The aim of this project is to develop and deploy a **Cloud-Native Food Ordering Web Application** that leverages Containerization and Serverless computing to achieve high availability and operational efficiency.

4.2 Objectives

To achieve this aim, the following specific, measurable objectives are defined:

1. **To Design a Microservices Architecture:** Decompose the application into a specific Frontend (User Interface) and Backend (API) to allow independent scaling.
2. **To Implement Containerization:** Use **Docker** to package both the frontend and backend services, ensuring consistent behavior across local development and cloud production environments.
3. **To Deploy on Serverless Infrastructure:** Utilize **AWS ECS Fargate** to run the containers without provisioning or managing EC2 instances.
4. **To Establish Scalable Storage:** Implement **Amazon DynamoDB** as a serverless key-value store to handle menu data and order transactions with single-digit millisecond latency.
5. **To Automate Software Delivery:** Construct a Continuous Integration/Continuous Deployment (CI/CD) pipeline using **AWS CLI** and **PowerShell** to drastically reduce deployment time and human error.