# **Project Proposal**

Comparative Evaluation of Serverless and Container-Based Microservice Architectures for a Simple Real-Time Messaging Application

**1. Introduction**

Modern software systems demand scalability, cost-efficiency, and reduced infrastructure management. Serverless architectures and container-based microservices have emerged as two leading cloud-native approaches to achieve these goals. However, organizations often struggle to decide which model is more suitable for small to medium-scale applications in terms of performance, operational cost, and system complexity. This project aims to develop a minimal real-time messaging application and deploy it using both Serverless Architecture (Firebase Cloud Functions) and Container-Based Microservices (Docker + Flask/Node.js). The purpose is to compare both architectures based on latency, scalability, operational complexity, and cost, while keeping the project simple, budget-friendly, and tool-efficient.

**2. Problem Statement**

Despite the popularity of serverless and containerized microservices, there is limited practical and simplified research comparing the two architectures using a real-world but lightweight application. Many existing comparisons require advanced tools such as Kubernetes, Terraform, Prometheus, or expensive cloud infrastructure, making them difficult for small teams or academic projects to replicate. There is a need for a simplified, low-cost, and practical comparison using only essential tools that do not require paid infrastructure or complex deployment pipelines. This project addresses that gap.

**3. Aim and Objectives**

### **Aim:**

To compare serverless and container-based microservice architectures by developing a simple messaging application and evaluating their performance, scalability, cost-efficiency, and operational complexity.

### **Objectives:**

1. To develop a simple real-time messaging API that allows user registration, sending messages, and retrieving messages.
2. To deploy the application using two different architectures: Firebase Serverless (Cloud Functions) and Docker-based microservices.
3. To measure and compare both architectures in terms of latency, scalability, operational complexity, and cost.
4. To provide recommendations and best practices for choosing the most suitable architecture for small-scale cloud applications.

## **4. Research Questions**

1. Which architecture Serverless or Docker-based offers lower latency for basic messaging operations?
2. How do both architectures handle scaling under 50 simultaneous user requests?
3. Which approach requires more development, deployment, and maintenance effort?

### **5. Scope of the Project**

The scope of this project is limited to the development and comparison of a simple real-time messaging backend application using two different architectural approaches: serverless computing through Firebase Cloud Functions and container-based microservices using Docker. The project will include core functionalities such as user registration, sending and receiving text messages, and storing messages in a cloud database like Firebase Firestore. The focus will be on backend development only, using minimal tools and free services wherever possible, and a full front-end mobile application will not be developed. The system’s performance will be evaluated based on latency, scalability under simulated user load, cost efficiency using free-tier cloud usage, and development and deployment complexity. Advanced technologies such as Kubernetes, CI/CD pipelines, Terraform, Kafka, and complex security implementations are intentionally excluded to maintain simplicity and cost-effectiveness. This study will provide a practical and affordable comparison of both architectures for small-scale applications.

## **6. Methodology**

Here in the below table the methodology is described;.

| **Step** | **Description** | **Tools Used** |
| --- | --- | --- |
| 1. System Design | Define the system requirements, API endpoints, data flow, and basic architecture for messaging functionality (register, send message, receive message). | API Specification Document |
| 2. Development – Serverless Version | Develop the messaging API using Firebase Cloud Functions and connect it to Firestore for storing user and message data. | Python |
| 3. Development – Docker-Based Version | Build the same API using Flask (Python) or Express.js (Node.js), containerize it using Docker, and connect to Firestore. | Docker Desktop, Flask |
| 4. Functional Testing | Test both versions of the API for correct functionality such as user registration, sending messages, and retrieving messages. | Python |
| 5. Performance Testing | Simulate multiple users by sending repeated API requests to measure latency and scalability under different load levels. | Python |
| 6. Cost and Resource Monitoring | Monitor API call usage, compute cost, and resource consumption using Firebase dashboard and local container resource usage. | Firebase Console, Docker Stats (for local testing) |
|  |  |  |
|  |  |  |

### **Testing & Evaluation**

* Use Postman to test API functionality.
* Use Postman Runner or simple Python script to simulate multiple users sending messages.
* A fully functional messaging API deployed using both Firebase Cloud Functions and Docker.
* Comparative results showing differences in latency, cost, and ease of development.  
  + A simplified decision-making guideline for choosing serverless or container-based architecture.

## **Risks & mitigations**

* Cloud cost spikes to set budgets/alerts; run tests in controlled windows.
* Provider limits (concurrency) and check quotas and request increases if needed.
* Comparability to keep API behaviour identical between implementations.

### **Conclusion**

This project aims to provide a practical and simplified comparison between serverless and container-based microservice architectures by developing a minimal real-time messaging application. By using only essential and cost-free tools such as Firebase Cloud Functions, Firestore, Docker, and Postman, the study remains accessible, budget-friendly, and replicable. Through performance testing, cost monitoring, and analysis of development complexity, the project will generate clear insights into the strengths, limitations, and suitability of each architecture for small-scale cloud applications. The expected outcome is a set of evidence-based recommendations and best practices that can guide developers and organizations in making informed architectural decisions without requiring advanced infrastructure or high financial investment. Overall, the project contributes to understanding modern cloud-native development while keeping implementation simple, affordable, and academically valuable.