COMP 4651 Project Report

Comparative Analysis of Serverless and Server-Based Architectures for Image Classification Deployment

Group 8

Authors

HUANG, I-wei - 20824074 - ihuangaa@connect.ust.hk

LI, Yu-hsi - 20819823 - ylils@connect.ust.hk

LIN, Yen-po - 20819782 - ylindr@connect.ust.hk

November 28, 2024

1 Introduction

1.1 Background and Context

Machine learning (ML) has become integral to modern applications across industries, enabling tasks such as image recognition, natural language processing, and predictive analytics. However, deploying ML models at scale presents significant challenges. Traditional deployments often require maintaining dedicated servers, which can lead to high resource usage, increased operational complexity, and significant costs.

Serverless architectures offer a contemporary solution to these challenges. By adopting a function-as-a-service (FaaS) paradigm, serverless platforms dynamically allocate compute resources based on demand. This approach eliminates the need for constant server maintenance, reduces costs, and enhances scalability, making it a compelling alternative for ML model deployment.

1.2 Objective

This project aims to deploy a ML model using both serverless and traditional server-based architectures. The goal is to compare and evaluate the practicality and effectiveness of each approach, highlighting their respective advantages and challenges.

1.3 Scope

The project explores open-source serverless solutions, specifically OpenFaaS [1], deployed on Kubernetes [2] and Docker, with Ngrok [3] used for external accessibility. While the study is limited to OpenFaaS, the findings provide insights into the broader applicability of serverless architectures for ML model deployment.

2 Methodology

2.1 Machine Learning Model

The project utilizes MobileNetV2 [4], a lightweight convolutional neural network pre-trained on the ImageNet dataset [5], for image classification tasks. MobileNetV2 is chosen for its efficiency and suitability for environments with constrained resources. The model is packaged into a pipeline that processes image input and returns classification results.

2.2 Serverless Architecture

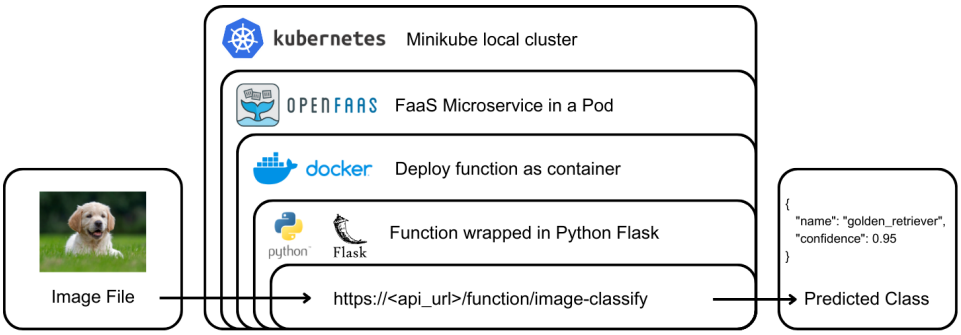

Figure 1: Serverless Architecture

The serverless architecture is shown in Figure 1, the deployment leverages OpenFaaS Community Edition (CE), an open-source serverless framework designed to run on Kubernetes.

Setup

A local, single-node Kubernetes cluster is created using Minikube. OpenFaaS is installed into this cluster via its official Helm chart, ensuring organized deployment of components into dedicated pods. These pods handle requests dynamically, enabling efficient resource utilization.

Deployment Steps

1. A new function is created using the OpenFaaS Python Flask template.
2. The Python function is implemented to load the pre-trained MobileNetV2 model using TensorFlow and create an inference pipeline.
3. The function is containerized and deployed using the `faas-cli up` command, which builds, pushes, and deploys the function to the OpenFaaS server.
4. The deployed function runs as a containerized service within the OpenFaaS framework, enabling remote inference capabilities.

2.3 Traditional Server-Based Deployment

The traditional server-based deployment follows a similar approach but does not utilize serverless platforms. Instead, the pre-trained MobileNetV2 model is wrapped in a Python Flask application and directly executed on the local machine. This approach relies on a persistent server running the Flask app to handle requests.

2.4 Exposing Local Server to the Internet

To make both serverless and traditional deployments accessible from external networks, **Ngrok** [3] is used. Ngrok establishes secure tunnels to the local server, exposing the deployed services to the internet. This step enables remote testing and performance benchmarking under real-world conditions.

2.5 Performance Testing

The performance of both serverless and traditional deployments is benchmarked using custom Python scripts. These tests measure key metrics to evaluate the efficacy of each approach:

- **Latency**: The time taken to process a single inference request.
- **Throughput**: The number of requests handled per second under varying load conditions.

The benchmarking results provide a quantitative comparison of the two architectures, highlighting their strengths and trade-offs in terms of efficiency and responsiveness.

3 Results and Analysis

3.1 Performance Metrics

To evaluate the performance of both serverless and traditional server-based deployments, we conducted latency and throughput benchmarking.

Latency was measured by sending 150 requests to both the server-based and serverless setups, recording the time taken for each response. **Throughput** was tested by measuring the number of successful responses handled in 120 seconds. The setups, described in Section 2, were run on a local laptop, exposed to the internet via Ngrok, with requests originating from a separate computer.

The results are summarized in Table 1.

	Server-Based	Serverless
Latency (seconds)	0.5922	0.5713
Throughput (requests/second)	1.71	1.69

Table 1: Performance Metrics for Server-Based and Serverless Deployments

3.2 Observations

The performance results show that the latency and throughput of server-based and serverless deployments are nearly identical. This finding contrasts with our initial expectations. Based on knowledge from coursework and research, we anticipated two key differences:

1. **Higher Latency in Serverless:** Due to the cold-start problem in serverless architectures, we expected higher latency compared to server-based approaches.
2. **Higher Throughput in Serverless:** Serverless architectures are designed for auto-scaling, leading us to expect higher throughput than server-based approaches.

However, the observed results can be attributed to the limitations of our deployment environment using OpenFaaS CE, which lacks features to fully realize the potential of serverless architectures.

Cold-Start Limitations

OpenFaaS CE does not fully exhibit the cold-start behavior typical of serverless platforms. By default, the minimum number of replicas for a function is set to 1, ensuring that at least one instance is always active. This configuration prevents true scaling down to zero, which is necessary to measure the impact of cold-starts. Consequently, the latency test did not capture the delays associated with initializing a function from a cold-start state.

Auto-Scaling Limitations

In OpenFaaS CE, auto-scaling is governed by the `requests per second` (RPS) metric, monitored via Prometheus. If the RPS exceeds a defined threshold (5 RPS in our case), an alert triggers the creation of additional replicas of the function.

To observe this behavior, we conducted a separate test using a dummy function and a script to simulate high request rates. The results were monitored through Prometheus. Figure 2 illustrates the invocation rate over time, while Figure 3 shows the corresponding number of function replicas. When the RPS exceeded the threshold, scaling was triggered, increasing the number of replicas to a maximum of 5.

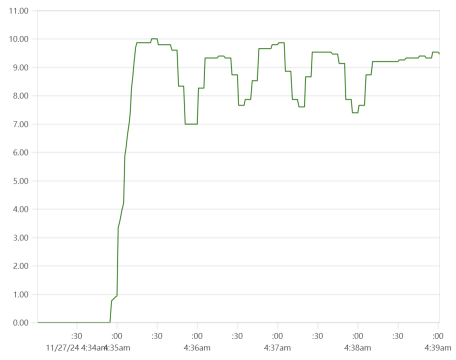

Figure 2: Invocation Rate of Function Over Time

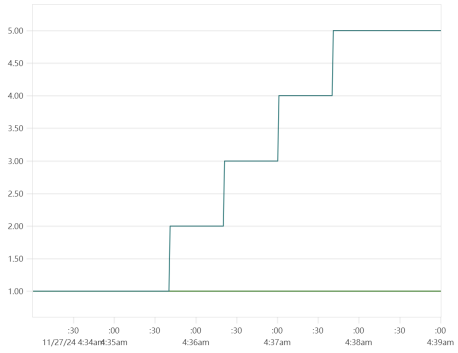

Figure 3: Number of Function Replicas Over Time

For our ML application, however, the inference time was too long to achieve sufficient RPS to trigger scaling. As a result, the throughput test did not measure the increased throughput typically associated with serverless auto-scaling.

4 Discussion

4.1 Advantages and Limitations

Based on our research and experimental results, both serverless and traditional server-based architectures have distinct advantages and limitations. The comparison is summarized in Table 2.

Aspect	**Serverless Architecture**	**Server-Based Architecture**
Scaling	Automatic scaling based on traffic demand, requiring no manual intervention.	Requires manual scaling and load balancing.
Latency	May experience cold-start latency when initializing functions from zero replicas.	Low latency due to always-on servers; no cold-start issue.
Cost	Pay-per-execution pricing model; cost-effective for fluctuating workloads.	Fixed costs based on server size, uptime, and capacity, even during low utilization.
Maintenance	Minimal maintenance; developers focus on application logic.	Requires ongoing maintenance of server infrastructure.
Resource-Intensive Tasks	Limited suitability for resource-heavy ML models or long-running tasks.	Well-suited for tasks requiring high computational power or prolonged execution times.
Use Case	Ideal for applications with sporadic or unpredictable workloads.	Best for applications with consistent workloads or stringent latency requirements.

Table 2: Comparison of Serverless and Server-Based Architectures

4.2 Challenges

This project introduced several challenges, primarily due to the exploration of new tools and technologies, including OpenFaaS, Kubernetes, Docker, and Ngrok. Significant time was spent researching and understanding the deployment process, from installation to implementation. Integrating these tools into a cohesive workflow required careful experimentation and troubleshooting, as well as adapting to the limitations of OpenFaaS CE.

5 Conclusion

This project demonstrated the deployment of a machine learning model using serverless and traditional server-based architectures. The results provided insights into the performance and applicability of these approaches, revealing trade-offs between flexibility, scalability, and cost-efficiency.

5.1 Summary

Serverless and traditional server-based deployments each offer unique advantages:

- **Serverless Architecture:** Highly dynamic and cost-effective for workloads with fluctuating demand, but limited by cold-start latency and scalability constraints in resource-intensive tasks.

- **Server-Based Architecture:** Reliable and responsive for consistent, high-demand workloads but requires more manual effort and incurs higher fixed costs.

Our experiments showed that OpenFaaS CE's limitations, such as minimum replica settings and scaling thresholds, influenced the performance, masking some of the expected benefits of serverless architectures. These constraints highlight the importance of platform choice in achieving the full potential of serverless solutions.

5.2 Recommendations

Based on the study, we propose the following guidelines for selecting deployment methods for ML models:

- **Serverless Architecture:** Serverless architectures are ideal for applications with sporadic workloads. For example:
  1. A mobile application offering on-demand image classification services globally.
  2. Applications that need cost-efficient scaling during peak traffic while minimizing idle resource costs.
- **Server-Based Architecture** Server-based deployments are better suited for applications with consistent workloads or stringent latency requirements. For example:
  1. Industrial systems performing automated quality control on a steady stream of image data.
  2. Applications requiring low-latency responses and high computational power without scalability concerns.

This project underscores the importance of understanding application requirements, platform capabilities, and workload characteristics when choosing between serverless and traditional deployment architectures. Future work could explore advanced serverless platforms and hybrid approaches to address the limitations identified in this study.

References

[1] A. Ellis, "OpenFaaS: Serverless Functions Made Simple," 2017. [Online]. Available: https://github.com/openfaas/faas

[2] "Kubernetes," 2014. [Online]. Available: https://kubernetes.io

[3] Inconshreveable, "ngrok," 2015. [Online]. Available: https://ngrok.com

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2019. [Online]. Available: https://arxiv.org/abs/1801.04381

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.