

Literature Review for: Serverless vs Containerized Microservices in Scalable Cloud Architectures: A Comparative Review

Name: Odai Abdelbaky
Date: July 10, 2025
Academic Code: 252000312

Table of Contents

Introduction 2

Review of the selected papers: 2

Paper 1 Serverless Computing vs. Containerization: Comparative Analysis for Cloud-Native Application Deployment..... 2

Paper 2 Engineering Scalable Microservices: A Comparative Study of Serverless Vs. Kubernetes-Based Architectures..... 3

Paper 3 Comparing Cost and Performance of Microservices and Serverless in AWS: EC2 vs Lambda 3

Paper 4 Serverless Architectures: A Comparative Study of Performance, Scalability, and Cost in Cloud-native Applications..... 4

Paper 5 Microservices and serverless functions—lifecycle, performance, and resource utilisation of edge based real-time IoT analytics 4

Paper 6 A comparison between traditional and Serverless technologies in a microservices setting 5

Connections 5

Conflicts Between Research Findings..... 6

Synthesis Matrix for Literature Review 7

Conclusion..... 12

References..... 12

Introduction

Literature Review for “Serverless vs Containerized Microservices in Scalable Cloud Architectures: A Comparative Review”

As cloud-native development continues to grow, two primary deployment models have become the biggest players: **serverless** and **containerized** microservices. Serverless platforms, like AWS Lambda, provide a simplified approach to cloud service deployment, as well as automatic scalability; whereas containerized microservices are far more flexible and provide developers with more control and options for implementing a wide range of microservices. In this report, I've reviewed 6 recent research papers that directly compare serverless to containerized microservices based criteria defined for cost, performance, scalability, and operational complexity. The goal of this report is to showcase the benefits and drawbacks of each model, and provide insight on the most appropriate use-cases for each for building scalable, cloud-native architectures.

Review of the selected papers:

Paper 1 Serverless Computing vs. Containerization: Comparative Analysis for Cloud-Native Application Deployment

- **Summary:** The objective of this research is to study serverless computing by using AWS Lambda in comparison to containerized microservices deployed via Kubernetes. This research includes testing cloud-native application deployments utilizing both AWS Lambda and Kubernetes. The authors created a sample application in both environments and tested performance along various measures such as cold start latency, scalability, operational complexity, and cost. The authors concluded that serverless computing allows for faster development, lower costs for low-traffic workloads, and almost zero operational overhead. On the other hand, containerized deployments perform better under sustained load and provide researchers with better control over the runtime environment. The authors, therefore, recommend using a hybrid model involving both approaches depending on the workload.
- **Critical Analysis:** This paper provides a straightforward comparison of serverless architectures and containerized architectures, using one application deployed on both AWS Lambda and in Kubernetes as a case study. One of the strengths of this paper is the applied aspect: the testing of both compute models using the exact same code base provides a fair performance & cost comparison but it also was clear about the operational trade-offs. However, this study is limited in scope. It used only one cloud platform (AWS) and it did not explore more complex workloads, nor the long-term performance trends. The lack of any more in-depth data on real-world deployments of cold-start mitigations also contributes to limiting its usefulness in terms of the bigger picture.

Paper 2 Engineering Scalable Microservices: A Comparative Study of Serverless Vs. Kubernetes-Based Architectures

- **Summary:** In this paper, authors compare and contrast two microservice deployment options and containers, leveraging AWS Lambda (serverless), and Kubernetes (containerized). The authors built the same microservices in the AWS Lambdas and Kubernetes, using Apache JMeter to run the microservices under loads, the important measurements were: cost, latency, and scalability. Serverless is advantageous if the environment is dynamic or has low traffic and can scale automatically with the ability to see costs and scale metrics when deployed. Kubernetes is best suited for higher throughput scenarios where resources need to be managed for cost, allowing persistent control. The authors recommend there must be a hybrid combination of serverless for event-based processes and containers for core service or infrastructure.
- **Critical Analysis:** One of the main strengths of the paper is its overarching benchmarking approach to benchmarking using multiple workloads modeled through Apache JMeter to compare Lambda and Kubernetes. They provide valuable observations and implications around how workloads are classified and offered hybrid recommendations based on workloads. Their quantitative results are based on metrics, response latency, and monthly cost, and are valid and justifiable. One of the salient drawbacks is that the study solely investigated AWS taking away broader implications in other areas like Azure or GCP. They also didn't consider any implications related to CI/CD or DevOps, failure or real-time observability of errors between the two contrasting environments.

Paper 3 Comparing Cost and Performance of Microservices and Serverless in AWS: EC2 vs Lambda

- **Summary:** In this paper, the authors analyzed the performance and cost of deploying a microservice-based API with AWS Lambda and Docker containers running on EC2. To simulate user traffic they used Locust and concluded that serverless (Lambda) deployment was more cost effective and more scalable for small-to-medium workloads but restricted under high-load workloads (especially with cold starts), whereas EC2 containers provided a better throughput and more consistent performance in heavy use cases but required more management of the infrastructure. The paper concludes that the choice of deployment should depend on the volume of use and how sensitive of the performance issues.
- **Critical Analysis:** This research article provides a beneficial side-by-side comparison of containerized microservices with EC2 (Elastic Compute Cloud) versus AWS Lambda. The analysis is founded upon actual AWS billings in two data centers and the use of real workloads. Its main strength is its real-world cost analysis and clear reporting on throughput performance under increasing load conditions. The study lacks any discussion of orchestration platforms, such as Kubernetes, that could influence the performance of containers. In addition, the study does not address other relevant

enterprise adoption concerns such as developer productivity, integration complexity, or ongoing operational maintenance.

Paper 4 Serverless Architectures: A Comparative Study of Performance, Scalability, and Cost in Cloud-native Applications

- **Summary:** In this paper, they contrast serverless and containerized architectures on AWS and Azure, using two different cloud-native applications: video encoding and data ingestion pipelines. The authors compared both platforms along the following three dimensions: latency; resource utilization; and developer effort. In terms of their applications, they found that serverless was the better choice for lightweight or event-driven workloads, which are less latency-sensitive, as it was more agile and cost-effective than containerized applications. Containerized workloads could provide more predictable latency and a better grasp of the amount of resources consumed, and so they would recommend containerized applications for high-performance or stateful applications.

The authors also proposed a decision rule based upon the high-level property of latency tolerance, which they believe provides a structured way of evaluating container or serverless workloads by applying a good enough heuristic, whereby, for latency-sensitive services, customers should consider a container, and for flexibility, customers should consider serverless.

- **Critical Analysis:** This study's strength is its multi-cloud evaluation of AWS and Azure, providing further context on how serverless behaves across providers. It is also novel in terms of considering developer effort and utility using a latency-based decision-making framework. The consideration of complex workloads (e.g. video processing, data pipelines, etc) also makes this paper more relevant to the real-world. Nevertheless, this study did not include any container alternatives such as AWS Fargate or edge-based deployments. It also did not include infrastructure-as-code, monitoring practices, or anything in the deployment lifecycle, which are all concepts that are important in production-grade systems.

Paper 5 Microservices and serverless functions—lifecycle, performance, and resource utilisation of edge based real-time IoT analytics

- **Summary:** In this paper, the focus is on the edge of computing and the exploration of serverless functions and Docker-based microservices for real-time IoT sensor data analytics. The authors assessed applications deployed on edge nodes for metrics like energy consumption, CPU usage, and cold starts. The findings indicated that containerized microservices had better latency and resource efficiency on constrained interfacing edges. While the serverless model could be initiated easily. The authors propose a layered architecture in which containers interact and process data at the edge, while serverless functions manage aggregation of data on the cloud side and alerts.

- **Critical Analysis:** The paper considers both a serverless and a containerized microservices architecture of edge computing, which are frequently missed in a lot of cloud architecture studies. The paper also articulates performance measurements CPU utilization, memory, and energy consumption. Nevertheless, the focus on edge devices provides a different angle than the other papers. The workloads were also correspondingly limited to sensor reading workloads that also wouldn't generalize to more demanding edge use-cases like AI inference or video analytics. Likewise, the authors also do not reference hybrid orchestration frameworks like KubeEdge or AWS Greengrass that could be appropriate hybrid solutions to combine serverless and container-based.

Paper 6 A comparison between traditional and Serverless technologies in a microservices setting

- **Summary:** In this paper, the authors implemented nine microservices on three platforms AWS Lambda (serverless), AWS Fargate (managed containers), and Kubernetes (self-managed containers) and compared AWS Lambda (serverless), AWS Fargate (managed containers), and Kubernetes (self-managed containers). The authors compared the three platforms based on cost, latency, cold starts, and aggregating operational complexity. AWS Lambda was shown to be the most economical for sparse, short-lived workloads. Fargate provided generally scalable performance without cold start issues. Kubernetes provided, generally, the greatest control; however, it had the highest level of management effort. This paper is used to establish the importance of the application and the patterns of workload to help decide which deployment models to use, and how much operational overhead is acceptable, advocating for hybrid or mixed strategies.
- **Critical Analysis:** This is the most comprehensive study of the group that compares Lambda, Fargate, and Kubernetes across real-world microservices instances (nine instances in total). It has very good coverage, as it provides a balanced consideration of latency, cold starts, cost, and operational complexity. Including the AWS Fargate option, as a middle, in the deployment comparison is a valuable addition as it also provides some nuance in thinking about serverless versus containers. However, it does focus exclusively on AWS services and so does limit any insights across platforms. It also does not explore multi-cloud strategies or transitioning from one deployment model to another, which is increasingly becoming common in enterprises today.

Connections

1. Shared Recognition of Trade-offs

All six papers agree that serverless and containers have different strengths. Papers 1, 2, and 6 state explicitly, there is no one size can fit all; the deployment model must match the workload. This consensus is the basis across this body of literature: serverless is for simplistic and burst traffic; containers are for complex, stateful or high-throughput systems.

2. Recommendation of Hybrid Architecture

Papers 1, 2, and 6 recommend hybrid configurations where serverless is used for event-driven or less critical workloads and containers are used for performance-critical services. This connection shows an industry trend of adopting a polyglot architecture and blending deployment models with the aim of optimizing flexibility and minimizing costs.

3. Cold Start as a Universal Bottleneck

Papers 2, 3, 4 and 5 all highlight cold start latency as a common issue in serverless talks. Paper 4 advocated and indicated the extent of this with suggested latency thresholds (e.g., don't use serverless if the need is to consistent sub-200ms response time), which does lend some fidelity to the qualitative concerns raised in the others.

4. Scalability is Typical Strength of Serverless

Papers 1, 2, and 3 consistently suggest that auto-scaling is a common strength of serverless - particularly for workloads that are less regular or planned occurrences. They are the same in use case and their capacity, from the most basic APIs (for example, Paper 3) to a cloud-native pipeline (example, Paper 4).

5. Cost Sensitivity Findings

Papers 1, 3, and 6 offer specific cost comparisons and reach conclusions that serverless tends to be cheaper at low-to-medium workloads but tends to be cheaper than containers at scale. One point made in paper 6 was difficulty running Fargate at a cost between serverless and containers and reminded the audience that cost modelling is part of any architecture decision making - same with any modelling.

6. Developer Experience / Maintenance

Papers 1, 2 & 4 identified that it is easier to develop and maintain serverless-like infrastructures. There is a connection here too, since ultimately if there is less out of DevOps, there will be an improvement in managing DevOps, but also was recognized in paper 6 that containers provide the same flexibility for complex systems.

Conflicts Between Research Findings

1. Serverless Performance Reliability

Papers 1 and 2 find that serverless is performing well across a range of loads with acceptable latency and auto-scaling. Papers 3 and 4 argue that cold-start latency and concurrency limits in serverless models can add considerable latencies for API-driven or real-time systems.

Conflict: serverless appropriate for latency sensitive applications? Some papers say yes (if managed) while others say no.

2. Best Deployment for Cost Efficiency

Papers 1 and 3 are very clear that serverless is the most cost-effective solution for intermittent workloads. On the other hand, Paper 6 argues that AWS Fargate (a managed container service) has an equal or better cost-performance balance in many situations, especially while considering cold starts and idle latency.

Conflict: Some papers found serverless to clearly be cheaper while other studies show that managed containers may present the best total cost.

3. Scale Advantage

In both Paper 2 and Paper 3, we see that serverless has the advantage of scaling on demand, without the need to intervene manually. On the other hand, Paper 4 shows that Kubernetes containers also provide strong horizontal scaling, and with more predictability once workloads are sustained.

Conflict: There are competing views as to whether one can scale reliably at high volume or not.

4. Preferred Architecting Model

Paper 4, however, prefers strict use-case separation: containers for anything latency-sensitive and persistent, serverless for light, burstable work. In contrast, Papers 1 and 6 favor hybrid forms, stating that many applications can, and should mix both forms.

Conflict: Some papers prefer definitional separation of use cases and some are advocating for a blended approach.

Synthesis Matrix for Literature Review

Study	Authors	Year	Key Findings / Results	Methodology	Limitations	Strengths
Serverless Computing vs. Containerization: Comparative Analysis for Cloud-Native Application Deployment	-Alessandro Matteo Giovanni -Luna James Charlotte	2025	Serverless was faster to make, cost less when the use was low or changing, and was easier to run. Containers worked better when there was a lot of use, and they were good where the goal is to keep	Made a simple microservice app, two ways: One as an AWS Lambda function (serverless). The other with containers run by Kubernetes. Measured how long it took to	It was just one app, on one cloud (AWS). No look at the long run (weeks or months) to see how autoscaling works over time. No look at how	Good way of doing the benchmark of the same app on two types of systems. Good description of the metrics used to measure: latency, cost per call, and how

			things going for a long time or keep data.	start up after a cold start, how well they scaled, how hard they were to run, and how much they cost at different levels of use.	to fix the cold start issue found in serverless (provisioned concurrency in Lambda).	much work has to be done to run the app. Good talk about when to use either (containers or serverless) depending on what kind of app it is.
Engineering Scalable Microservices: A Comparative Study of Serverless Vs. Kubernetes-Based Architectures	- Sreenivasulu Navulipuri	2025	Cold starts in serverless functions were notable under sporadic access but less of an issue under sustained traffic. Kubernetes had better predictable performance and was preferred for heavy/consistent workloads. Serverless was significantly cheaper for low-invocation workloads. Proposed a hybrid architecture where serverless handles burst traffic and containers manage core services.	Built identical microservice applications (e.g., file conversion, user management) in: AWS Lambda (FaaS) Docker containers deployed on Kubernetes (EKS) Simulated varied workloads using Apache JMeter. Benchmarked metrics: startup time, CPU/memory usage, response latency, scaling behavior, and monthly costs.	Only on AWS services (Lambda and EKS); might not work for GCP or Azure. No look at the impact on CI/CD pipelines or the overhead of DevOps on both setups. No user experience metrics or perf metrics for the integrations (e.g., API Gateway latency).	Lays out a multi-metric benchmarking setup with repeatable workloads. Gives a cost vs performance picture based on real cloud pricing. Brings up a workload classification idea to decide between deployment models.
Comparing Cost and Performance of Microservices and Serverless in AWS: EC2 vs Lambda	-Christopher Allen -Xiaozhou Li -Tom Černý	2024	Lambda was 30-60% cheaper at low/med traffic was easier to run and grow	Deployed a CRUD based REST API as: -Containerized service on EC2 (Docker) and	No coverage of Kubernetes or orchestration layers, only EC2 + Docker.	Use of real billing data from AWS Variety of load scenarios; show where problems

	-Amr S. Abdelfattah		EC2 based containers had better throughput at high weight allowing more control but cost more in care. Lambda had cold start issues and burst loads. Recommended using Lambda for event microservices and use EC2 for core work	-Stateless functions on AWS Lambda. Run different workloads on Locust: low, med, high. Monitor CPU and RAM use, response times and the monthly bill from AWS billing.	Didn't evaluate deployment pipelines, autoscaling logic, or DevOps complexity. Cold start mitigation options in Lambda not explored	occur. Can be utilized to analyze how the cloud system will respond to real time work load. Can be used to obtain the impact of the difference in the work load on the cloud.
Serverless Architectures: A Comparative Study of Performance, Scalability, and Cost in Cloud-native Applications	-Naveen Kodakandla	2021	Serverless offered versus what Containers offered: Serverless: -Less developer effort -Cost-efficiency with intermittent traffic -Auto-scaling with very little config Containers: -Better performance for stateful/compute-heavy workloads -More control over resources -More predictable latency He also established a latency threshold rule: serverless is	Compared implementations of video encoding service and event-driven data pipeline in: AWS Lambda & Azure Functions (serverless architecture) Kubernetes deployed containers Benchmarks: Cold starts, latency, resource usage, developer complexity, and scaling limits	Consider both AWS and Azure provides multi-cloud view. Realistic test cases encoding and data processing are common workloads in the cloud.	Not a direct comparison with containers on EC2 or managed container services. Did not consider hybrid use cases that other papers considered.

			an option for workloads with a latency tolerance of less than ~200ms. If the workload is at greater than 200ms latency, the selection will be containers.			
Microservices and serverless functions—lifecycle, performance, and resource utilisation of edge based real-time IoT analytics	-Francesco Tusa -Stuart Clayman -Alina Buzachis -Maria Fazio		Containers experienced: - Lower latency (approx. a 25% improvement) and better resource utilization. - Increased offline resilience when edge connectivity was minimal. Serverless experienced: - Ease of deployment/scalability in an IoT, cloud-based context. - Less suitable for real-time edge analytics due to longer start times. Containers are ideal for edge analytics, serverless is ideal for cloud-based aggregation and alerting.	Developed a sensor analytics pipeline of air quality and motion sensors using: -Containerized microservices on edge nodes. -Serverless functions pushed by cloud messaging services Monitored cold starts, delays in data transmission, compute cycles, energy usage, and resource constraints.	Restricted to sensor analytics generalization to other workloads (e.g., video, AI inference) not promised. Didn't investigate hybrid orchestration models (e.g., KubeEdge, AWS). Some data was simulated and not from long-term production environments.	Relevant to real-world IoT workflows, applicable to Smart Waste or Smart City rollouts. Metrics measured were over performance such as energy and memory usage. Provides guidance on designing hybrid cloud/edge workloads.
A comparison between traditional and Serverless	-Juan Mera Menéndez	2023	They developed nine microservices prototypes	- Lambda: Great for infrequent and burst	Does not consider hybrid models or	Contains AWS Fargate, usually excluded in

technologies in a microservices setting	-Jose Emilio Labra Gayo -Enrique Riesgo Canal -Aitor Echevarría Fernández		deployed in all three environments. They benchmarked: - Cold start time - Memory and CPU usage report - Execution latency - Cost across 30-day cycle They also considered infrastructure management complexity.	workloads, minimal management overhead, but had high cold starts for API service. - Fargate: Cost-effective, less management than traditional Kubernetes(container runtime), consistent performance, no cold start. - Kubernetes: Best fit for consistent workloads and high throughput, however the highest amount of management overhead. Recommendations are: - Use Lambda for event driven jobs - Use Fargate when there is a need to balance auto-scaling and cost - Use Kubernetes when complex systems are exist and there	transitions (e.g., Lambda to Fargate). Only AWS services, without GCP/Azure equivalent comparisons. Benchmarks did not include metrics on failover, high availability, or observability.	serverless/container blendings. All real-world prototype apps provide realistic benchmarks database applications. Contains metrics related to developer experience and operational complexity.
---	--	--	---	---	---	---

				is a management control		
--	--	--	--	-------------------------	--	--

Conclusion

This literature review has assessed the advantages, disadvantages, and compromises of serverless computing and containerized microservices when constructing scalable cloud architectures. The six reviewed papers shared a general consensus that these two models serve different use cases: serverless computing works best for lightweight, event-driven, and dynamic tasks, while containers provide better performance, control, and reliability when delivering complex or latency-critical services. The general conclusion is simple: the model chosen for deployment should be a function of the inherent workload characteristics, predicted scalability needs, operational objectives, and available infrastructure. An increasing number of scholars and practitioners promote the idea of hybrid cloud-native architectures that utilize a blend of serverless and containerized components that deliver the benefits of performance, cost efficiency, and operational maintainability. As cloud platforms continue to develop, they should continue providing new services such as AWS Fargate, Azure Container Apps and edge-native runtimes. Also, more research is needed on the transition paths between containerized and serverless models, security issues, and automation of mixed mode environments. However, this represents a great opportunity for the development of flexible, resilient, and efficient cloud-native systems.

References

Giovanni, Alessandro & Charlotte, Luna. (2025). Serverless Computing vs. Containerization: Comparative Analysis for Cloud-Native Application Deployment. https://www.researchgate.net/publication/390193315_Serverless_Computing_vs_Containerization_Comparative_Analysis_for_Cloud-Native_Application_Deployment

Tripathi, K. (2025). Engineering Scalable Microservices: A Comparative Study of Serverless vs. Kubernetes-Based Architectures <https://ijsret.com/2025/04/14/engineering-scalable-microservices-a-comparative-study-of-serverless-vs-kubernetes-based-architectures>

Allen, Christopher & Li, Xiaozhou & Abdelfattah, Amr & Černý, Tom & Taibi, Davide. (2024). Comparing Cost and Performance of Microservices and Serverless in AWS: EC2 vs Lambda. 10.1007/978-3-031-61816-1_5. https://www.researchgate.net/publication/381761885_Comparing_Cost_and_Performance_of_Microservices_and_Serverless_in_AWS_EC2_vs_Lambda

Kodakandla, Naveen. (2021). Serverless Architectures: A Comparative Study of Performance, Scalability, and Cost in Cloud-native Applications. 5. 136-150.

https://www.researchgate.net/publication/386876894_Serverless_Architectures_A_Comparative_Study_of_Performance_Scalability_and_Cost_in_Cloud-native_Applications

Tusa , F. et al. (2024) Microservices and serverless functions-lifecycle, performance, and resource utilisation of edge based real-time IOT analytics, Future Generation Computer Systems.

<https://www.sciencedirect.com/science/article/pii/S0167739X24000529>

Menéndez, J. M., Gayo, J. E. L., Canal, E. R., & Fernández, A. E. (2023) A comparison between traditional and Serverless technologies in a microservices setting.

<https://arxiv.org/abs/2305.13933>