International Journal of Electrical and Computer Engineering (IJECE)
Vol. 16, No. 2, April 2026, pp. 873~882
ISSN: 2088-8708, DOI: 10.11591/ijece.v16i2.pp873-882

Architectural trade-offs: comparative analysis across K3s, serverless, and traditional server deployments

Prajwal P., Naveen B. Teli, Nishal H. N., Nimisha Dey, Pratiba Deenadhayalan, Ramakanth Kumar Pattar, Pavithra Hadagali, Skanda P. R.
Department of Computer Science and Engineering, R.V College of Engineering, Bangalore, India

Article Info

Article history:

Received Apr 7, 2025
Revised Dec 7, 2025
Accepted Jan 16, 2026

Keywords:

Amazon web services
Kubernetes
Lambda
Microservices
Observability
Serverless

ABSTRACT

In modern software architecture, combining serverless computing, microservices, and containers improves scalability, performance, observability, and resilience. However, choosing the right deployment strategy is crucial. Current individual deployment methods often limit productivity because of poor integration options. This study looks at three deployment approaches: Kubernetes cluster, AWS Lambda (serverless), and Traditional Java Server. We tested performance under different workloads using virtual machines and simulations. The results show that the K3s cluster provides high throughput and low latency because it manages resources directly. AWS Lambda's pay-as-you-go model, along with its built-in cost optimization, works well for event-driven workloads. In contrast, Java Microservice is cost-effective but needs manual tuning to control latency and error rates. Bringing these scenarios together into a single service mesh architecture could help optimize costs, performance, and system resilience.

Corresponding Author:

Skanda P. R.
Department of Computer Science and Engineering, R.V College of Engineering
Bangalore, Karnataka, 560059, India
Email: skandapr.cs22@rvce.edu.in

1. INTRODUCTION

Microservices and containerization have revolutionized software engineering by enabling applications to be modular, independently deployable, and highly scalable. This transition from monolithic systems to microservices-based architectures allow organizations to innovate faster, respond to market changes, and improve system resilience. While previous studies have examined microservices deployment in isolation, whether it be on Kubernetes clusters [1], [2], serverless platforms such as AWS Lambda [3], [4], or traditional on-premise servers [5], [6], direct and controlled comparisons between these environments under identical workloads remain very limited. Such comparisons are very useful for organizations making architecture decisions based on performance, cost, and scalability requirements.

Despite the maturity of microservices technologies, decision-making around deployment strategies remains difficult due to limited quantitative, side-by-side evaluations. Existing research has examined containerized deployments [7]–[9], challenges such as cold starts and concurrency limits in serverless computing [3], [4], [10], [11], and the operational trade-offs of traditional hosting models [5], [6], but these studies typically focus on a single environment or compare different applications, limiting their generalizability. As a result, organizations often rely on anecdotal evidence or vendor claims when selecting deployment models, which can lead to suboptimal choices. Recent comparative studies have analyzed serverless, containerized, and monolithic architectures under cloud-native workloads, highlighting

differences in performance, scalability, and cost characteristics, while emphasizing that architectural suitability is highly workload dependent [12], [13]. This motivates the research question: Which deployment approach to use, let it be a K3s Kubernetes cluster, AWS Lambda serverless functions, or a traditional Java server? Which one offers the best trade-off between cost, performance, scalability, and resource efficiency for microservices applications? Furthermore, hybrid architectures that combine these models within a service mesh remain underexplored, representing an additional gap in the current state of the art.

This paper addresses these gaps by i) implementing the same Spring Boot e-commerce application across three deployment paradigms—K3s Kubernetes cluster, AWS Lambda, and traditional Java server—to ensure a fair comparison; ii) conducting controlled performance tests under varying workloads, measuring latency, throughput, error rates, resource utilization, and cost; iii) comparing our results to previous research and identifying strengths, weaknesses, and trade-offs for each deployment model; and iv) proposing the concept of a hybrid service mesh architecture that leverages the advantages of multiple deployment approaches.

Section 2 of this research reviews related work. Section 3 describes our methodology in detail, enabling replication. Section 4 presents results integrated with discussion and comparisons to prior studies. Section 5 concludes with key findings, implications, and future research directions.

2. RELATED WORK

According to several studies [1], [2], some of the earlier initiatives that have contributed to the understanding and evolution of microservices and serverless computing architectures include. It is also work that has been carried out in various ways to advance serverless systems. Areas of interest include the resource of the serverless system, the performance of the serverless system, and the cost of the serverless system, especially on Amazon AWS Lambda. Notably, studies about employing serverless computing to facilitate the migration of developers' intricate applications to the nascent new paradigm also provided knowledge regarding the possibility and challenge of this administrative model of computation. Other related research has also focused on the integration of other serverless computing technologies, such as the use of containerization together with Kubernetes, in addition to the suitability of microservices as well as how to adopt, deploy, and manage microservices for large-scale applications on-premises and cloud solutions such as Amazon web services (AWS). Recent performance comparison studies further evaluate trade-offs between microservices deployed in containers versus serverless architectures. Fan *et al.* [14] benchmarked a cloud-native application across both approaches and revealed significant variations in throughput and application bootstrapping behavior. Allen *et al.* [15] examined the financial and performance implications of using EC2 instances versus AWS Lambda for microservices and found that serverless is not always cheaper at scale. In total, these insights and the research assure the steady development of the computing architecture strategy as well as provide guidance to practitioners about the future patterns of microservices and serverless computing.

They support monitoring and provide information to characterize the end-to-end life cycle of apps that use microservices deployed and run on serverless platforms. It also explains how the application systems behave when they encounter a cold start scenario, how the slacks of the application vary at different stages, and the mechanism of container scaling. The concern of the paper is centered on the problems concerning the current schedulers employed by the providers of serverless platforms for applications based on microservices [3], [4]. This then raises the issue of start-up times and the issue of time for resource formation on serverless platforms. The impact of cold-start latency continues to be a dominant concern in serverless computing, with a systematic taxonomy and mitigation analysis provided by Golec *et al.* [16], covering commercial FaaS platforms and container-level optimization techniques. To facilitate error handling and availability of fast local serverless function execution or re-execution, the authors recommend that non-volatile memory (NVM) [10], [17] be used as the persistence layer because of the constraints of current serverless systems in managing specialized hardware and long-running processes. In the context of serverless computing, this method aims at supporting specialized hardware and further execution. They discuss the challenge of random placement of serverless function instances, and this leads to suboptimized outcomes.

To explore some of the more nuanced security and privacy issues particular to the implementation of Kubernetes at the edge [18], [19]. It highlights the threats, recommends their countermeasures, and discusses the consequences of implementing security techniques for ensuring data providential at the edge computing context. To Spring Boot [5], [6], [20], a productive framework for building and running microservices has emerged. It explains the benefits of using Spring Boot to improve efficiency and minimize challenges with developing applications that utilize microservice architecture.

Microservices orchestration has been a significant focus in recent research due to its potential for improving the scalability and resilience of applications. Mathew *et al.* [21] present a practical approach using

AWS Step Functions, highlighting the advantages of this method in managing complex workflows in cloud environments. Their study demonstrates how AWS Step Functions can streamline microservices orchestration, providing a more efficient and scalable solution compared to traditional methods. This approach is particularly beneficial for applications requiring dynamic resource allocation and robust error handling. By leveraging the features of AWS Step Functions, such as visual workflows and built-in error handling, organizations can achieve higher efficiency and reliability in their microservices deployments. The authors provide a comprehensive analysis of the performance benefits, offering valuable insights for both practitioners and researchers in the field of cloud computing and microservices architecture.

Recent work has also explored hybrid orchestration strategies. Tusa *et al.* [22] demonstrated that combining microservices and serverless functions for edge-driven analytics can yield performance advantages depending on workload composition. Similarly, Golec *et al.* [16] compared container orchestration frameworks with serverless computing platforms and concluded that neither is universally superior, and optimal performance is workload dependent. These findings reinforce the relevance of examining architecture selections through direct experimentation. Additional comparative analyses between serverless and containerized architectures have been presented by Samson and Timilehin [12], who evaluated cloud-native deployments and reported that neither approach consistently outperforms the other, with performance and cost efficiency strongly influenced by workload characteristics.

It reviews the best practices for hosting applications on AWS [23] in different deployment patterns, such as monolithic and microservices, as well as how proper utilization of AWS services' scalability, reliability, and cost efficiency can be achieved. The proposal of architectural patterns and guidelines for effective work with serverless technologies on AWS. It includes facets like AWS Lambda, application programming interface (API) gateway [24], and other serverless services to understand that they can be used for hosting microservice or monolith applications effectively.

3. METHOD

An e-commerce application is built in Spring Boot, which is a microservices Java framework. In the application, there are services such as the product service, the order service, and the notification service. Each service has its own roles and tasks, for instance, managing products, receiving orders, and distributing notifications. For instance, the product service enables the creation of products, the modification of the attributes of the products, the deletion of products, and the querying for products with specific attributes. The operation of the e-commerce API is illustrated in Figure 1. This approach to structuring the application facilitates better modularity, maintainability, scalability, and flexibility.

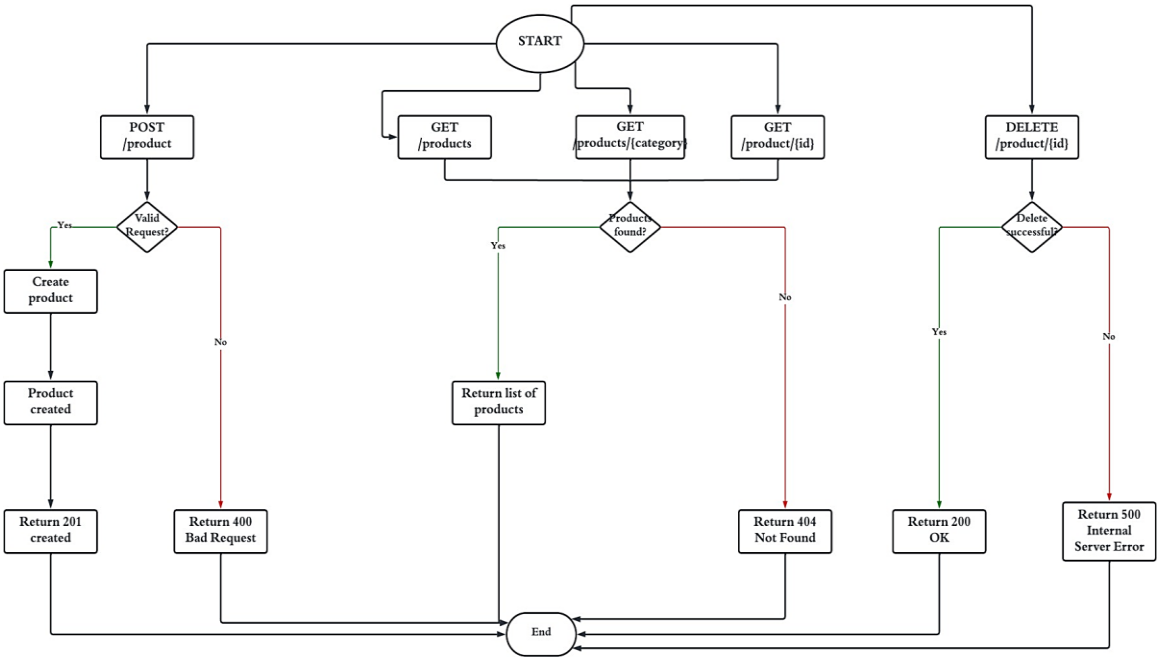

Figure 1. Working of an e-commerce application

Following development, each service can be containerized using Docker. In containerization [14], each service and its requirements are packed within their own small containers. This practice conforms to the microservices architecture [7], where services can be deployed, scaled, and managed independently. In the case of Docker [17], each service runs in its own container; this way, no other services will interfere with each other or their environment. This makes application development and deployment processes seamless, making the application's lifecycle more efficient.

The API endpoints' functionality, correctness, and healthiness are properly tested and debugged to input them into the system. This involves the development of robust test suites that provide test coverage for various uses and cases. As applied to software testing, automated testing frameworks may be used to run tests in an effective and accurate manner. Through an exhaustive API, there is increased detection of prospective problems, quirks, and discrepancies in the application, hence making the API more reliable and robust.

By deploying the microservices in a single application on a server, the Monolithic architecture (Traditional Java server deployment) method combines the microservices into a single unit. This approach simplifies deployment setup during installation and configuration since the application comprises deployed components consolidated in the SCA deployment unit. However, as the application grows to have more substantial facilities and functions, this architecture may prove disadvantageous as regards scalability and management ability. Lack of scalability and flexibility. On-premises software implementations are rigid due to their packaged-based model, meaning that modifications or updates to individual services necessitate redeployment of the entire application, hence causing expected downtime or inefficiency.

However, as per the Kubernetes cluster approach [1], a Kubernetes cluster is created by the utilization of four virtual machines, where the master-slave structure prevails. Kubernetes includes several capabilities that help in the automation of deployment scheduling, scaling, and management of containerized applications in the cluster [8]. Load balancing, service discovery, and self-healing are the added features that are used to make the microservices more scalable and resilient, along with the enhancement of the performance of the deployed microservices. Having distributed workloads on several nodes and containers, Kubernetes brings efficiency to work while guaranteeing that the application is highly available [9].

The AWS Lambda serverless computing platform [20] is used when implementing the microservices, and they are created as separate functions. Instead of running and managing dedicated on-purpose servers, each microservice function is invoked as a response to specific events or queries. Since AWS Lambda provides concurrency control of these functions, workload distribution fulfills the optimal allocation of resources in a cost-efficient manner. In accordance with this serverless structure, programmers are more focused on writing code and thus contributing to the growth of the enterprise instead of spending time on setup. AWS Lambda's inherent scalability and automated error handling also enhance the reliability of the microservices that are adopted. All three approaches are clearly formulated as a block diagram, as shown in Figure 2.

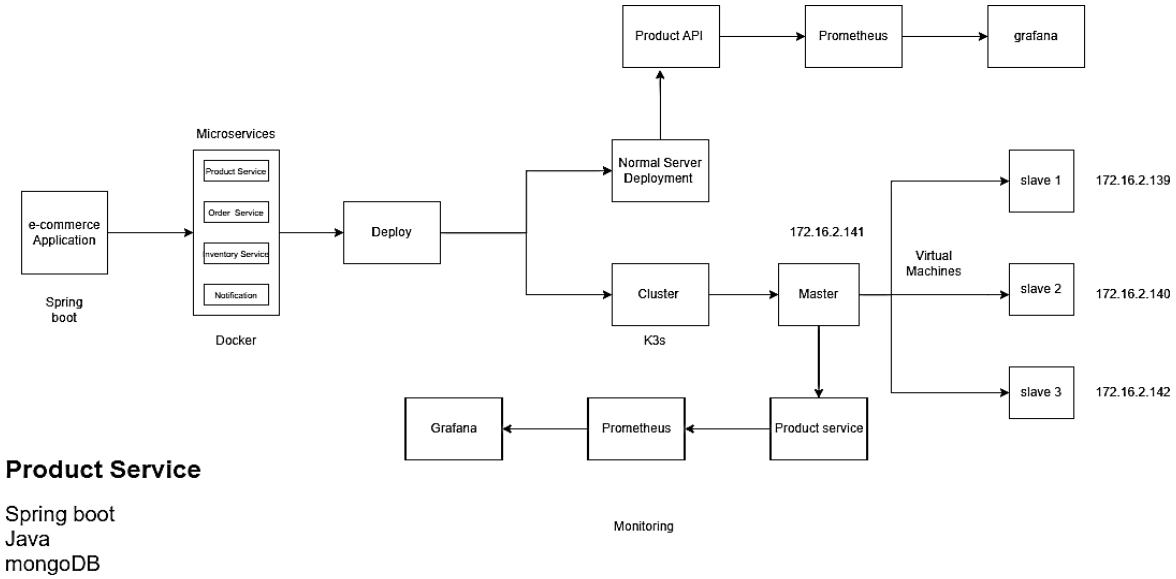

Figure 2. Block diagram of the proposed architecture

Some other services which are part of the architecture include Prometheus and Grafana, which are used in monitoring the resources and the clusters promoted. At the present time, the Prometheus system is applied as the monitoring and alerting toolkit, which is open source, and Grafana is used as the visualization and data analysis tool. This also encompasses the monitoring, alerting, or diagnosing process that occurs in real-time as an assurance that consistency and reliability of the application are maintained.

Load testing is simply carried out to measure its performance under the structure of the three architectural patterns and under various loads. Scalability, efficiency, and reliability are measured using parameters including response times, throughput, and errors, among other factors. Life-like usage patterns are modeled to provide an understanding of how the system behaves and where the weak points could be when used optimally, so that necessary fine-tuning can be done as well as capacity planning.

Autoscaling mechanisms across virtualization and container layers for network traffic are within CPU and RAM usage. This guarantees that resources are allocated or reallocated in a flexible way to create high utilization and low costs. Moreover, the HTTPS certificate validation is enforced via CA server configuration to raise security by specifying secure communication between the clients and the application servers. This helps secure key data and prevent unauthorized access, thereby enhancing overall system security.

4. RESULTS AND DISCUSSION

The evaluation metrics provide comprehensive insights into the performance of different architectures. Latency and response time measure responsiveness, while throughput assesses the system's capacity under varying loads. Error rate and failure recovery gauge reliability, and scalability examines resource allocation efficiency. Resource utilization and observability metrics offer deeper analysis into resource efficiency and monitoring capabilities. Comparative analysis between hybrid, serverless, and container-based approaches helps quantify the advantages of the hybrid solution, highlighting improvements in latency, throughput, error rates, and resource utilization.

The K3s cluster architecture offers moderate pricing, reflecting the costs associated with managing and maintaining the cluster infrastructure. However, this architecture demands high efforts due to the complexities involved in setting up and managing the cluster. On the performance front, the K3s cluster demonstrates low error rates, high throughput, and low latency, making it suitable for high-performance applications. However, cost optimization in this architecture is limited, requiring manual intervention to ensure efficient resource utilization. Performance analysis of the K3s cluster is shown in Figure 3 and Table 1.

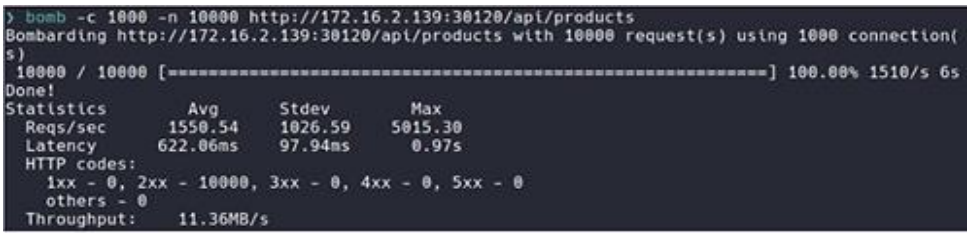

Figure 3. Analyzing the performance of K3s cluster

Table 1. Performance analysis metrics of K3s cluster

Architecture	No. of requests	Performance metrics
K3s cluster	1000	Minimal impact on latency and error rate. Throughput remains high. This is due to direct control over resources. Cost-effective for higher workloads due to shared infrastructure.
	10000	Latency remains low. Error rate stays low due to horizontal pod auto-scaling. Resource utilization increases, but auto-scaling helps maintain performance.
	100000	Latency stays under control due to autoscaling. Error rate is managed, ensuring stability. Autoscaling helps handle higher loads, but resource optimization may be necessary.
	1000000	Latency increases, but autoscaling attempts to maintain stability. The error rate may rise, indicating potential capacity issues. Resource saturation becomes evident, demanding scaling and fine-tuning, even with auto-scaling

The Java microservice architecture is associated with relatively low costs, attributed to the lightweight nature of individual microservices. Efforts required for development, deployment, and maintenance in this architecture fall between the other two architectures, requiring moderate efforts.

However, the performance attributes vary, with moderate to high error rates and latency, coupled with moderate throughput. Cost optimization in Java microservice architecture requires manual efforts to control costs effectively, as resource allocation and scaling are typically managed by the developer or operations team. Performance analysis of the local server is shown in Figure 4 and Table 2.

With its pay-as-you-go pricing structure, AWS Lambda provides cost-effectiveness by only billing for the resources used while executing a function. Because AWS Lambda abstracts away infrastructure administration, developers can concentrate entirely on designing and deploying functions, resulting in less effort required. Performance-wise, AWS Lambda offers low error rates and latency, albeit with moderate throughput, making it suitable for lightweight workloads. Furthermore, cost optimization is built in, as AWS Lambda automatically scales resources based on demand, optimizing costs without manual intervention. Performance analysis of the K3s AWS Lambda implementation is shown in Figure 5 and Table 3. Each architecture offers distinct trade-offs in terms of pricing, effort, performance, and cost optimization strategies, catering to different use cases and requirements. Table 4 summarizes the comparative attributes of the evaluated architectures, highlighting key trade-offs in cost, effort, latency, throughput, and optimization strategies.

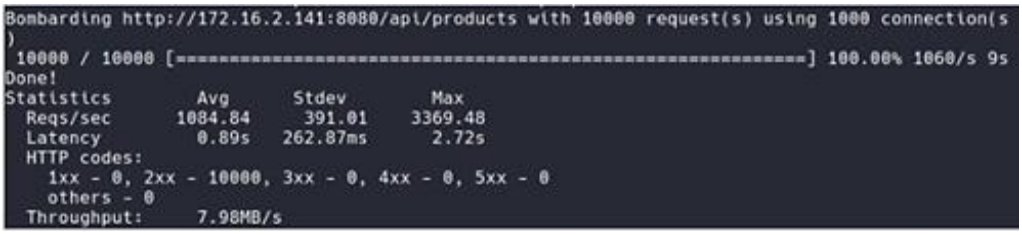

Figure 4. Analyzing the performance of local cluster

Table 2. Performance analysis metrics of Java server

Architecture	No. of requests	Performance metrics
Local Server	1000	Low latency and error rate due to direct control over resources. Throughput remains high, suitable for moderate workloads. Cost effective for small-scale usage.
	10000	Latency and error rates are still low but may start to rise slightly. Throughput holds up well, making it suitable for medium-scale loads. Costs remain relatively stable due to manual resource management.
	100000	Latency and error rate increase, impacting user experience. Throughput may plateau or decline, requiring optimization. Costs have risen due to increased resource demand and manual management.
	1000000	Latency spikes significantly, potentially leading to timeouts. Error rate could become unacceptable, necessitating immediate attention. Resource exhaustion and potential application crashes impact costs.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Thread Group - 500:Lambda Request	500	40046	0	64248	13960.98	0.80%	14.0/min	1.72	0.04	7574.7
Thread Group - 1000:Lambda Request	1000	40851	0	64022	14434.35	0.20%	14.3/min	1.78	0.04	7656.3
Thread Group - 10,000:Lambda Request	10000	42988	0	64944	10939.05	0.70%	13.9/min	1.72	0.04	7637.3

No of Threads (users) used for all the tests is 10.

Mostly the errors are from Dynamo DB – the default configuration of DynamoDB is not able to handle more concurrent requests. If you have any suggestion to improve this, let us know.

Figure 5. Performance analysis of AWS Lambda

Table 3. Performance analysis of AWS Lambda

Architecture	No. of Requests	Performance metrics
AWS Lambda	1000	Low latency and error rate due to automatic scaling. Throughput remains stable, making it suitable for light to moderate workloads. Cost effective for sporadic usage.
	10000	Latency and error rate are still low, as AWS manages scaling. Throughput may start to plateau, affecting performance. Costs remain reasonable due to pay-as-you-go pricing.
	100000	Latency remains relatively low, but throughput may degrade. Error rate could start to rise due to potential concurrency limits. Costs have increased but are still manageable compared to other options.
	1000000	Latency and error rate may start to increase significantly. Throughput may drop, impacting responsiveness. Costs escalate due to resource consumption and potential throttling.

Table 4. Comparison of attributes

Attribute	Comparison		
	K3s cluster	AWS Lambda	Server
Price	Moderate	Pay-as-you-go	Low
Efforts	High	Low	Moderate
Error rate	Low	Low	Moderate-high
Throughput	High	Moderate	Moderate
Latency	Low	Low	Moderate
Cost optimization	Limited	Built-in	Manual

5. CONCLUSION

The assessment emphasizes the distinct advantages and drawbacks of every deployment scenario. The K3s cluster exhibits outstanding throughput and low latency with few errors, rendering it ideal for demanding computational workloads. AWS Lambda offers a budget-friendly, event-driven framework with integrated optimization, yet it displays moderate performance and delay. The Java Microservice, though cost-effective, encounters increased error rates and moderate latency, requiring manual adjustments. Every method entails compromises in terms of cost, effort to implement, and overall performance. Combining these deployment techniques into a cohesive service mesh framework offers a valuable chance to leverage their collective advantages to improve cost effectiveness, system efficiency, and robustness. Comparative results provide crucial insights into emerging trends and future directions in microservices architecture.

The analysis reveals distinct characteristics of deployment scenarios: K3s cluster performs very well in terms of speed and low latency, accompanied by minimal errors, perfect for computation tasks. AWS Lambda offers great efficiency with its pay-as-you-go and event-driven services, although its performance in throughput and latency is moderate. This document indicates that the Java Microservice is less expensive than the monolithic architecture but carries a greater likelihood of errors and average latency. Each of the outlined situations entails expenses and efforts concerning established performance metrics. Therefore, the concept of incorporating these scenarios into a unified service mesh structure appears to align with upcoming developments due to their significant benefits regarding cost, performance, and availability. Therefore, the findings generated by the research are valuable for further exploring microservices architecture and for establishing new research pathways in this dynamic area of study.

Future work should explore hybrid orchestration strategies in production environments, investigate the impact of emerging Kubernetes distributions and serverless platforms, and evaluate additional performance factors such as energy efficiency and security posture. In summary, this paper advances the understanding of architectural trade-offs in microservices deployment, providing actionable guidance for practitioners and a foundation for further academic exploration in cloud-native systems engineering.

FUNDING INFORMATION

This work was supported by institutional funding from R.V. College of Engineering, Bangalore, Karnataka, India.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Prajwal P.	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓			
Naveen B. Teli		✓	✓	✓		✓		✓	✓	✓	✓			
Nishal H. N.			✓	✓		✓		✓	✓	✓	✓			
Nimisha Dey		✓		✓	✓	✓		✓	✓	✓				
Pratiba Deenadhayalan	✓	✓		✓	✓		✓			✓		✓	✓	
Ramakanth Kumar Pattar	✓	✓			✓		✓			✓		✓	✓	✓
Pavithra Hadagali		✓		✓	✓		✓			✓		✓	✓	
Skanda P. R.	✓	✓	✓	✓		✓		✓	✓	✓	✓			

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The authors confirm that the data supporting the findings of this study are available within the article.

REFERENCES

[1] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "Serverless computing for container-based architectures," *Future Generation Computer Systems*, vol. 83, pp. 50–59, Jun. 2018, doi: 10.1016/j.future.2018.01.022.

[2] J. R. Gunasekaran, P. Thinakaran, N. C. Nachiappan, R. Srivatsa Kannan, M. T. Kandemir, and C. R. Das, "Characterizing bottlenecks in scheduling microservices on serverless platforms," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, Nov. 2020, pp. 1197–1198, doi: 10.1109/ICDCS47774.2020.00195.

[3] W. Lloyd, M. Vu, B. Zhang, O. David, and G. Leavesley, "Improving application migration to serverless computing platforms: latency mitigation with keep-alive workloads," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, Dec. 2018, pp. 195–200, doi: 10.1109/UCC-Companion.2018.00056.

[4] A. Tzenetopoulos, D. Masouros, D. Soudris, and S. Xydis, "Towards elastic memory allocation of serverless functions in disaggregated memory systems," in *Proceedings of the 4th Workshop on Heterogeneous Composable and Disaggregated Systems*, Mar. 2025, pp. 15–21, doi: 10.1145/3723851.3723855.

[5] S. Chinthalapally, "Maximizing scalability and flexibility: leveraging AWS Lambda with API gateway," *Saint Louis University*, 2024.

[6] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: a survey of opportunities, challenges, and applications," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–32, Jan. 2022, doi: 10.1145/3510611.

[7] B. Tanneru, "Developing scalable microservices with spring boot and docker," *International Journal For Multidisciplinary Research*, vol. 5, no. 1, Jan. 2023, doi: 10.36948/ijfmr.2023.v05i01.38178.

[8] N. Li, Y. Tan, X. Wang, B. Li, and J. Luo, "Dynamic resource allocation for containerized applications in edge computing," in *Proceedings of the International Conference on Internet of Things, Communication and Intelligent Technology. IoTCIT 2022, vol. 1015. Springer, Singapore*, 2023, pp. 121–130, doi: 10.1007/978-981-99-0416-7_12.

[9] M. Clement, "Security considerations in edge Kubernetes clusters," *ResearchGate*, 2024.

[10] V. Koneni, "Microservices architectures using spring boot: embracing containerization and observability," *International Research Journal of Engineering and Technology (IRJET)*, vol. 11, no. 7, pp. 384–396, 2024.

[11] E. Jonas *et al.*, "Cloud programming simplified: a Berkeley view on serverless computing," *arXiv:1902.03383*, 2019, doi: 10.48550/arXiv.1902.03383.

[12] F. Samson and O. Timilehin, "A comparative analysis of serverless and containerized architectures in cloud-native environments," 2025.

[13] M. D. Marieska, A. Yunanta, H. Aulia, A. S. Utami, and M. Q. Rizqie, "Performance comparison of monolithic and microservices architectures in handling high-volume transactions," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 9, no. 3, pp. 594–600, Jun. 2025, doi: 10.29207/resti.v9i3.6183.

[14] C.-F. Fan, A. Jindal, and M. Gerndt, "Microservices vs serverless: a performance comparison on a cloud-native web application," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science*, 2020, pp. 204–215, doi: 10.5220/0009792702040215.

[15] C. Allen, X. Li, A. S. Abdelfattah, T. Cerny, and D. Taibi, "Comparing cost and performance of microservices and serverless in AWS: EC2 vs Lambda," in *Next Generation Data Science*, 2024, pp. 1–14, doi: 10.1007/978-3-031-61816-1_5.

[16] M. Golec, G. K. Walia, M. Kumar, F. Cuadrado, S. S. Gill, and S. Uhlig, "Cold start latency in serverless computing: a systematic review, taxonomy, and future directions," *ACM Computing Surveys*, vol. 57, no. 3, pp. 1–36, Mar. 2025, doi: 10.1145/3700875.

[17] A. Sargunakumar, "Container orchestration: key challenges and opportunities with Kubernetes," *International Journal of Innovative Research and Creative Technology*, vol. 10, no. 5, pp. 1–9, 2024, doi: 10.5281/zenodo.15086842.

[18] A. K. Panchalingala, "AWS cloud architecture: a comprehensive analysis of best practices and design principles," *European Journal of Computer Science and Information Technology*, vol. 13, no. 37, pp. 109–116, May 2025, doi: 10.37745/ejcsit.2013/vol13n37109116.

[19] M. L. El Bechir, C. S. Bouh, and A. Shuwail, "Comprehensive review of performance optimization strategies for serverless applications on AWS Lambda," *arXiv:2407.10397*, 2024, doi: 10.48550/arXiv.2407.10397.

[20] B. Tanneru, "Serverless computing with AWS Lambda: evaluating suitability and scalability," *Journal of Software Engineering and Simulation*, vol. 10, no. 10, pp. 71–73, Oct. 2024, doi: 10.35629/3795-10107173.

[21] A. Mathew, V. Andrikopoulos, and F. J. Blaauw, "Exploring the cost and performance benefits of AWS step functions using a data processing pipeline," in *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing*, Dec. 2021, pp. 1–10, doi: 10.1145/3468737.3494084.

[22] F. Tusa, S. Clayman, A. Buzachis, and M. Fazio, "Microservices and serverless functions—lifecycle, performance, and resource utilisation of edge based real-time IoT analytics," *Future Generation Computer Systems*, vol. 155, pp. 204–218, Jun. 2024, doi: 10.1016/j.future.2024.02.006.

[23] D. Yakubov and D. Hästbacka, "Comparative analysis of lightweight Kubernetes distributions for edge computing: performance and resource efficiency," in *Service-Oriented and Cloud Computing: Proceedings 11th IFIP WG 6.12 European Conference, ESOCC 2025, Bolzano, Italy,* 2025, pp. 81–95.

[24] D. Yakubov and D. Hästbacka, "Comparative analysis of lightweight Kubernetes distributions for edge computing: security, resilience and maintainability," in *Service-Oriented and Cloud Computing. ESOCC 2025. Lecture Notes in Computer Science, vol 15547. Springer, Cham.*, 2025, pp. 96–104.

BIOGRAPHIES OF AUTHORS

Prajwal P. is a final-year undergraduate student in the Department of Computer Science Engineering at R V College of Engineering, Bangalore, India. His research interests encompass Operating Systems and Computer Networks. He has developed sustainable solutions utilizing the latest technologies, with extensive work in web and backend engineering. Prajwal is committed to advancing the field through his research and practical applications in developing efficient systems. He can be contacted at email: prajwalp.cs20@rvce.edu.in.

Naveen B. Teli is a final-year student in the Computer Science Engineering Program at R V College of Engineering, Bangalore, India. His research interest includes web development, artificial intelligence, machine learning, and real-world applications. He can be contacted at naveenbteli.cs20@rvce.edu.in.

Nishal H. N. is a final-year student in the Computer Science Engineering Program at R V College of Engineering, Bangalore, India. His primary focus is on front-end development, where he applies his skills to create intuitive and effective user interfaces. Nishal excels in utilizing cutting-edge technologies to address and solve practical, real-world challenges. With a strong background in web development, he has spearheaded projects that emphasize user experience and performance. Nishal is dedicated to innovation and efficiency, contributing to the field through his creative and technical expertise. He can be contacted at nishalhn.cs20@rvce.edu.in.

Nimisha Dey is a final-year student in the Computer Science Engineering Program at R V College of Engineering, Bangalore, India. Her research interest includes artificial intelligence, machine learning, and real-world applications. She can be contacted at nimishadey.cs20@rvce.edu.in.

Pratiba Deenadhayalan is working as an associate professor in the Computer Science and Engineering Department at RVCE. She received her Ph.D. degree from VTU. She has published over 40 research papers. She has worked on various research and consultancy projects sponsored by Cisco, Citrix SCII, and Samsung. She can be contacted at email: pratibad@rvce.edu.in.

Ramakanth Kumar Pattar is a Professor and Dean of CSE Cluster in the Computer Science and Engineering department at RVCE. His research interests are digital image processing, pattern recognition, and natural language processing. He has published over 100 research papers. He has executed several funded research and consultancy projects sponsored by DRDO, ISRO, AICTE, GE India Pvt. Ltd, CABS, HP, and Nihon Communication Solutions Pvt. Ltd. He can be contacted at ramakanthkp@rvce.edu.in.

Pavithra Hadagali is an associate professor in the Computer Science and Engineering Department at RVCE. Her research interests are software defined networks, machine learning, deep learning, and software engineering. She has executed projects sponsored by Samsung, Toyota, etc. She can be contacted at pavithrah@rvce.edu.in.

Skanda P. R. is a third-year student in the Computer Science Engineering Program at R V College of Engineering, Bangalore, India. His research interest includes web development, artificial intelligence, and machine learning. He can be contacted at skandapr.cs22@rvce.edu.in.