J-INTECH (Journal of Information and Technology)
Accredited Sinta 4 Ministry of Higher Education, Science and Technology Republic of Indonesia SK No. 10/C/C3/DT.05.00/2025
E-ISSN: 2580-720X || P-ISSN: 2303-1425

Implementation and Analysis of Cloud-Native Approaches to Enhance Scalability and Performance in Dwi Jaya Understel Workshop Information System

Moch Bagus Tri Cahyo [1], Hamzah Setiawan [2] , Ika Ratna Indra Astutik [3]

[1 2 3] *Universitas Muhammadiyah Sidoarjo, Fakultas Sains dan Teknologi, Program Studi Informatika, Jl. Raya Gelam No. 250, Candi, Sidoarjo, Jawa Timur 61271, Indonesia*

Keywords

Amazon RDS; AWS Lambda; Cloud-Native Architecture; Information; Information System; Monolithic Architecture; Scalability; Serverless Computing; System Performance.

Corresponding Author:
baguschy13@gmail.com

Abstract

This study aims to analyze the differences in scalability and performance between a traditional monolithic system hosted on a Virtual Private Server (VPS) and a cloud-native serverless architecture using AWS services for an automotive workshop information system. An experimental method was employed using a post-test only control group design. Performance testing was conducted with K6 as the stress testing tool under a ramp-up load pattern of up to 60 Virtual Users (VU) to simulate peak traffic conditions, while Grafana was used for real-time monitoring and visualization of system metrics.The results indicate that under peak load scenarios, the cloud-native architecture reduced the average response time by 89.1% (from 6.05 seconds to 657.10 milliseconds) and eliminated the error rate completely (from 0.154% to 0%), compared to the monolithic system. Additionally, the throughput improved by 38.2%, demonstrating better responsiveness and stability. These findings confirm that serverless cloud-native systems offer superior scalability and reliability in handling dynamic and high-demand workloads, making them well-suited for public service platforms such as automotive workshop information systems.

1. Introduction

The advancement of information technology has driven modern information systems to not only operate stably but also possess adaptive capabilities to handle sudden spikes in workload and fluctuating user demands. This is particularly critical in public services such as automotive workshop information systems, where transactions, customer data, and service requests may increase abruptly and simultaneously. Failure to handle such load can lead to queues, transaction data loss, and decreased customer satisfaction . The integration of cloud services into mobile-based information systems has also been shown to improve service management efficiency and flexibility in handling dynamic user demands, supporting the adaptability required by workshop environments[1] .

Compared to traditional cloud computing practices that typically run monolithic applications on virtual infrastructure, the cloud-native approach offers greater resource efficiency and flexibility for dynamic business needs[2]. Common issues in workshops still relying on manual data processing—such as incomplete

transaction records or inaccurate financial reports—can be addressed through the adoption of cloud-based systems equipped with real-time features[3].

Cloud-native architecture provides a more flexible and efficient solution by leveraging services such as AWS Lambda, Amazon S3, and Amazon RDS, which support auto-scaling and real-time monitoring. Automation tools such as AWS CodeDeploy also enable more resilient and streamlined deployment processes. This approach is suitable for workshop information systems that serve users in dynamic volumes and require a high degree of reliability[4].

Several prior studies have explored the performance of cloud-native architecture; however, most are generic and do not specifically address the context of automotive workshop information systems. For instance, a benchmarking approach for evaluating the scalability of cloud-native applications was proposed in[5] , but it was not validated in high-transaction environments such as workshops. The comparison between monolithic and microservice applications in[6] focused solely on throughput and did not consider other critical factors such as cold start and error rate. Additionally, the evaluation in [7]. excluded serverless technologies like AWS Lambda, which are becoming increasingly relevant in modern scalability contexts. These limitations indicate that further research is needed to comprehensively assess how serverless architectures perform in systems with dynamic demands and daily transactional complexities like those found in workshop environments.

This study aims to address that gap by presenting an experimental investigation based on a real-world automotive workshop information system, employing a serverless and cloud-native architecture. The primary objective is to evaluate the system's autoscaling capabilities and operational stability when facing sudden user surges, with a particular focus on throughput and error rate—factors that have not been extensively discussed in prior literature. This research is expected to offer practical insights for developers and system architects when selecting implementation strategies for workloads with high scalability requirements.

1.2 Related Work

In the context of modern information systems, applications no longer serve merely as record-keeping tools but have become the operational backbone that demands high stability and adaptability to workload fluctuations[8] . Web-based workshop information systems have been widely implemented to address common issues such as manual transaction logging, inaccurate financial reports, and inefficient inventory management [9].

The *cloud-native* approach has emerged as a modern solution for designing large-scale and dynamic systems. This concept emphasizes automation, elasticity, and resource management efficiency, which are highly beneficial in environments with fluctuating workloads. One of the key technologies in this architecture is containerization, with Docker being a widely adopted platform that enables portable and isolated deployment processes [10].

According to the Cloud Native Computing Foundation (CNCF), *cloud-native* is a set of technologies that decomposes applications into microservices and packages them into lightweight containers, which are then deployed and orchestrated across various server environments [11].

A study conducted by Christian and Bisma demonstrated that microservice-based web applications performed better under heavy loads compared to monolithic architectures. However, this performance improvement comes with increased CPU consumption, while RAM usage remains relatively similar across both architectures [12]. These findings provide a valuable reference in evaluating the effectiveness of migrating from traditional VPS-based architecture to cloud-native architecture using AWS services.

2. Research Method

This study employs an applied experimental approach with the aim of comparing the performance and scalability between a monolithic architecture based on a Virtual Private Server (VPS) and a cloud-native architecture using AWS services. The research design used is the post-test only control group design, where

observations are conducted on two different architectures without a pre-test phase. This research was conducted at Dwi Jaya Understel Automotive Workshop, Pasuruan, East Java, over a period of three months.

The testing was carried out by gradually applying traffic load using a stress test with a ramp pattern, which means incrementally increasing the load over time. This method is similar to the approach used by [6], who also applied the ramp pattern to compare the performance of monolithic and microservice-based systems .

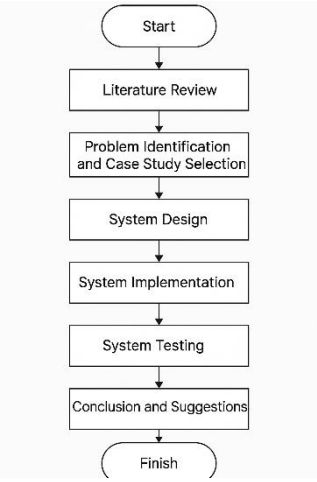

Figure 1research process

Variables in This Study

This study involved three main variables. The independent variable was the hosting environment used in the experiment, namely a Virtual Private Server (VPS) and a cloud-native AWS service. The dependent variable was system performance, measured using metrics such as response time, throughput, and scalability. To ensure consistency and reliability throughout the experiment, several control variables were maintained, including the test data used, the workload scenarios implemented, and the testing tools implemented throughout the performance evaluation.

Data Collection Techniques

Data collection in this study was conducted through several integrated methods. Direct observation was performed to capture real-time system responses and identify anomalies during testing. Workload simulation was carried out using K6 with a ramp-up pattern targeting up to 60 Virtual Users (VUs). This number was based on a reasonable estimation of peak concurrent users, following the commonly accepted DAU-to-PCU (Daily Active Users to Peak Concurrent Users) ratio of 10:1 to 20:1. This rule of thumb is widely used in performance planning to estimate realistic peak loads in the absence of precise traffic logs. The goal was to simulate high-concurrency scenarios that typically occur during busy hours and evaluate how well the system architecture handles such conditions. This method aligns with best practices in cloud-native performance testing, where stress testing is designed to validate scalability and resilience under realistic yet demanding usage conditions [13].

During the test, performance metrics such as response time, throughput, and error rate were sent to InfluxDB and visualized using Grafana dashboards for interactive monitoring. Additionally, AWS CloudWatch was utilized in parallel to validate and ensure data consistency by monitoring server-side metrics such as Lambda execution duration, error count, and API Gateway latency. The integration of K6, Grafana, and CloudWatch enabled comprehensive performance evaluation and accurate comparison between monolithic and cloud-native architectures.

Data Analysis Techniques

The data in this study were analyzed quantitatively by focusing on several key performance indicators. These include the calculation of average response time to determine the overall system responsiveness, maximum response time to identify peak delays under load, and error rate to assess the system's reliability during testing scenarios. These metrics provide a comprehensive evaluation of system performance across both architectural implementations.

System Architecture Design

Existing System Architecture (Simple Monolithic)

The existing system operates on a VPS server using a monolithic approach with a LAMP stack (Linux, Nginx, MySQL, Flask). All system components run on a single machine, which simplifies the deployment and debugging processes. However, this approach presents significant limitations in terms of scalability, load distribution, and monitoring capabilities. This is in line with the findings of Blinowski *et al.* [7] who stated that monolithic systems, while easier to develop in the early stages, become inefficient and difficult to scale as system complexity increases. This statement is further supported by a study by Srinivasan *et al.* [14], which concluded that monolithic architectures tend to be inefficient in environments requiring high scalability and often lead to bottlenecks in complex systems due to their tight coupling within a single deployment unit and limited service isolation.

Figure 2 Monolithic Architecture

Cloud-Native System Architecture

The automotive workshop information system was transformed into a cloud-native architecture by leveraging various services from Amazon Web Services (AWS) to enhance scalability and infrastructure management efficiency. On the frontend, the website is hosted using Amazon S3 with static website hosting, enabling fast access and lower operational costs. For the backend, the system is built using the Flask framework and deployed on AWS Lambda, connected via API Gateway, thereby supporting the serverless concept and allowing requests to be handled elastically based on demand. The database used is Amazon RDS with MySQL, or alternatively, DynamoDB for more flexible requirements at a larger scale. Furthermore, monitoring and logging are conducted using Amazon CloudWatch, allowing real-time performance and activity tracking and simplifying the troubleshooting process. This cloud architecture approach aligns with best practices in cloud-native migration to achieve service elasticity, business process automation, and cost efficiency, as described by Bagir *et al.* [15]. Additionally, Kumar *et al.* [16] emphasized that serverless computing is becoming an evolutionary paradigm in modern application development due to its ability to provide automatic scaling, reduced infrastructure management overhead, and cost optimization within cloud-native environments.

Figure 3 Cloud-Native Architecture

System Architecture Comparison

Table 1 Comparative Analysis of System Architectures: Monolithic vs. Cloud-Native (AWS Lambda)

Aspect	Monolithic (Before)	Cloud-Native (AWS Lambda)	Aspect	Monolithic (Before)
Hosting	VPS (Single Server)	AWS Lambda (Serverless)	Hosting	VPS (Single Server)
Frontend Hosting	Stored on VPS	Amazon S3 (Static Hosting)	Frontend Hosting	Stored on VPS
Backend	Flask on VPS	Flask via AWS Lambda + API Gateway	Backend	Flask on VPS
Database	MySQL on the server	Amazon RDS (MySQL) or DynamoDB	Database	MySQL on the server

Table 1 presents a comparison between the system architectures before and after migration, specifically between the monolithic VPS-based approach and the cloud-native architecture built on AWS Lambda. The comparison includes several key aspects such as hosting, scalability, deployment, monitoring capabilities, and cost model.

The table clearly highlights the limitations of the monolithic architecture, which relies on a single server and manual configurations, leading to challenges in scaling and system maintenance. In contrast, the cloud-native model enables automatic scaling, serverless deployment, and a pay-per-use cost structure, making it more suitable for dynamic and high-demand environments such as automotive workshop information systems. These distinctions justify the architectural transition and support the rationale behind the performance evaluation conducted in this study.

3. Result and Discussions

This section presents the results, testing processes, and discussion of the research that has been conducted.

Result

Functional Testing Results

Table 2.Functional Testing Results

Test Case ID	Feature	Test Steps	Tes data	Status
TC F01	Access Dashboard	Enter email & password, then login	Username:q Password: q	✓
TC F02	Add Transaction	Input license plate, select spare part, save	B1234XYZ, Engine Oil, Rp80,000	✓
TC F03	Search Spare Part	Type "Oli" in the search field	Oli	✓
TC F04	Logout	Click the logout button	-	✓

Functional testing was conducted to ensure that all core features of the automotive workshop information system operate in accordance with user requirements and system specifications. The testing covered the login process, service transaction entry, spare part search, and logout functionality. All features passed the tests with a *Pass* status, indicating that the system functioned as intended without any errors. According to Jia *et al.* [17], functional testing is an essential part of software quality management, as it aims to verify whether the system meets user requirements and predefined specifications. This approach falls under the category of *black-box testing*, where testing is performed without considering the internal structure of the system and focuses on the system's inputs and outputs.

Stress Test Results

Table 3 Stress Test Results

Test Case	Scenario	Virtual Users (VU)	SLA Target
TC-K01	Normal Load (open page, search spare parts) duration 1 minute	10 VU	Response < 1500ms, Error < 1%
TC-K02	Medium Load (input service transaction) duration 1 minute	20 VU	Response < 1000ms, Error < 1%
TC-K03	Peak Load (register simulation 1–60 VU) duration 1 minute	1-60 VU	Response < 1000ms, Error < 2%

Stress testing was conducted to evaluate the performance of the automotive workshop information system under various load scenarios, namely normal load, medium load, and peak load, by simulating different numbers of Virtual Users (VUs). This test measured metrics such as response time, error rate, and throughput based on the target Service Level Agreement (SLA). According to Castro and Harman [18], performance testing in serverless architectures such as AWS Lambda faces challenges in result stability, especially when the number of requests increases drastically. This highlights the importance of designing representative load scenarios and using testing tools capable of consistently simulating real-world conditions.

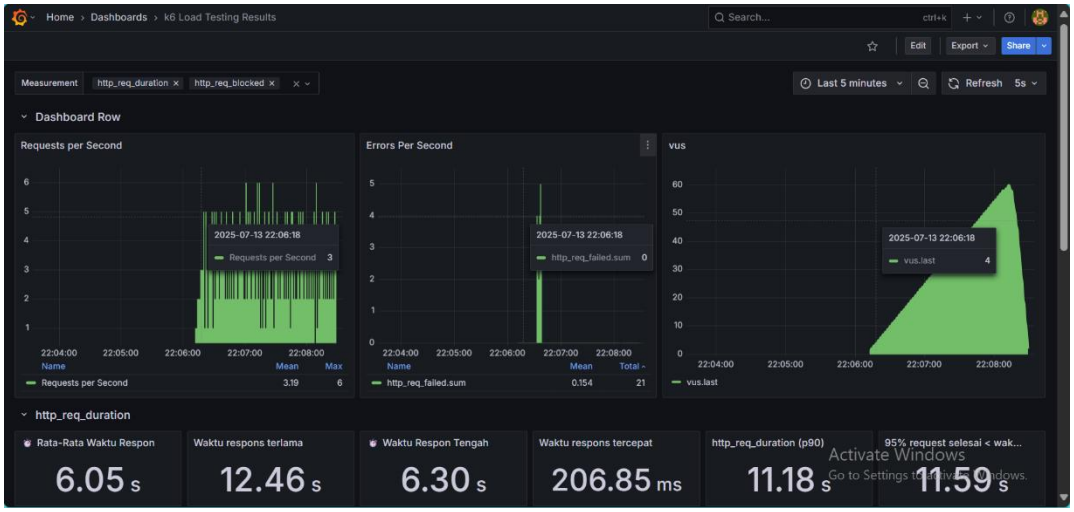

Figure 4 Monolithic System Performance Results on VPS

The figure presents the performance testing results of the monolithic system hosted in a traditional VPS environment. In the “Errors per Second” panel, there is a visible spike of 5 errors at the beginning of the test. Although no further errors occurred, the system recorded an average response time of 6.05 seconds and a peak response time of 12.46 seconds, both significantly exceeding the <2-second SLA target. Moreover, the "Throughput" panel illustrates that the system struggled to maintain a consistent request processing rate. These results indicate the limitations of the monolithic architecture in handling even moderate loads and its lack of elasticity in responding to traffic fluctuations. This aligns with the findings of Blinowski et al. [12], who highlighted the disadvantages of monolithic systems compared to distributed and scalable cloud-native architectures.

Table 4 Monolithic Architecture Stress Test Results

Test Case	AVG Response	MAX Response	Error Rate	Status
TC-K01	1.25s	2.95s	0%	Failed
TC-K02	271.62ms	1.16s	0%	passed
TC-K03	6.05s	12.46s	0.154%	Failed

The test results show the performance of the monolithic system running on a VPS. In the normal load scenario (TC-K01), the system recorded an average response time of 1.25 seconds (1250 ms), which is still within the SLA threshold of <1500 ms. However, the maximum response time reached 2.95 seconds (2950 ms), indicating performance inconsistency as it exceeded the significant tolerance limit in certain conditions. In the medium load scenario (TC-K02), the system successfully met the SLA with an average response time of 271.62 ms and an error rate of 0%, indicating stable performance in this condition. Meanwhile, under peak load (TC-K03), the system experienced a significant performance drop with an average response time of 6.05 seconds and a maximum response time of 12.46 seconds, thus failing to meet the SLA for this scenario. These results indicate that the monolithic architecture still has limitations in handling sudden traffic spikes elastically, although it remains stable and responsive under medium load conditions.

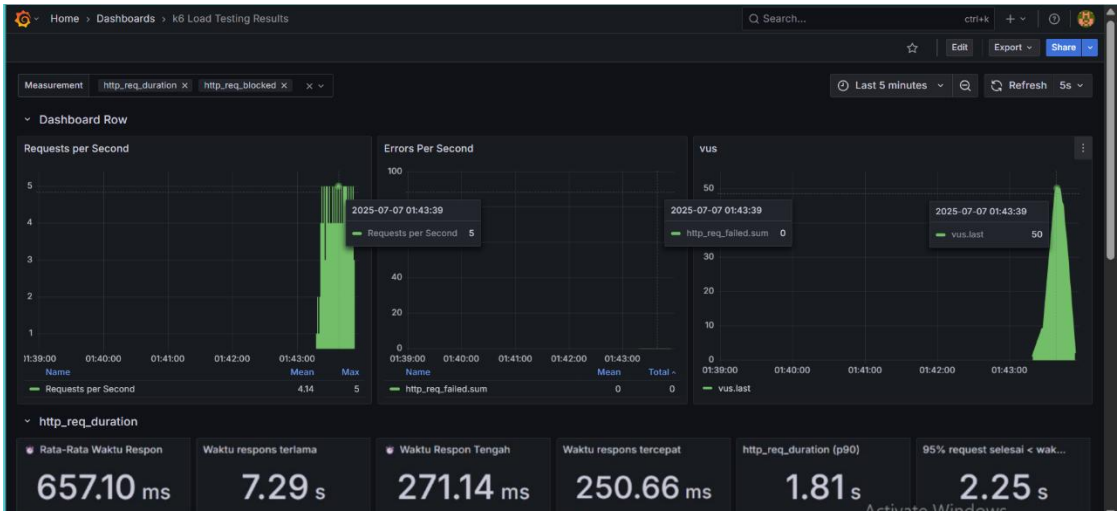

Figure 5Performance Test Results of AWS Lambda-Based Cloud-Native System Under Peak Load

Figure 5 illustrates the performance test results of the cloud-native system based on AWS Lambda under the TC-K03 scenario, where the system was tested under a peak load of 60 simultaneous users. The test was conducted to evaluate the system's ability to handle sudden traffic spikes.

The results indicate that the system successfully maintained an average response time of 657.10 milliseconds, which is well below the 2-second SLA threshold. Moreover, no errors were recorded during the test (0% error rate), demonstrating the system's reliability under high-traffic conditions.

The graph also shows that latency and throughput remained stable, with no significant spikes during the period when 60 users were actively accessing the system. This reflects the automatic scaling capability of the serverless architecture, where AWS Lambda dynamically adjusts its capacity to meet user demand.

These findings reinforce the advantages of the cloud-native approach, which enables systems to remain responsive and stable even during traffic surges—aligning with the findings of Blinowski et al. [7] regarding the elasticity and scalability of microservices and serverless architectures.

Table 5 cloud-native arcitekture Stress Test Results

Test Case	AVG Response	MAX Response	Error Rate	Status
TC-K01	2.14s	2.56s	0%	Failed
TC-K02	871.62ms	1.16s	0%	Passed
TC-K03	657.10ms	72.9s	0%	Passed

This section presents the performance testing results of the cloud-native system deployed using AWS services. In the normal load scenario (TC-K01), the system failed to meet the Service Level Agreement (SLA) as the average response time reached 2.14 seconds, exceeding the defined target. However, under the medium load (TC-K02) and peak load (TC-K03) scenarios, the system successfully met the SLA with average response times of 871.62 ms and 657.10 ms, respectively, and an error rate of 0%. These results indicate that cloud-native architecture offers superior scalability in handling high loads. This finding is consistent with the study by Blinowski *et al.* [7], which demonstrated that microservices-based and cloud-native systems offer advantages in elasticity and performance under heavy traffic conditions. Meanwhile, the failure observed in the light-load scenario can be attributed to cold start latency, a common issue in serverless environments such as AWS Lambda [18].

Discussion of Testing Results

Based on the conducted testing, there is a significant performance difference between monolithic and cloud-native architectures. From a functional perspective, all core features—including login, transaction creation, spare part search, and logout – were executed successfully on both architectures, indicating that the system meets its functional requirements.

In terms of performance, the monolithic architecture performed well under light and medium load scenarios. In TC-K01 (normal load), it recorded an average response time of 1.23 seconds, and 271.62 ms in TC-K02 (medium load), maintaining system responsiveness. However, under a traffic spike condition in TC-K03 (peak load), the system showed a significant performance drop, with an average response time of 6.05 seconds and an error rate of 0.154%, reflecting its limited scalability. This is consistent with the study by Fan et al.[19], who emphasized that monolithic systems can become performance bottlenecks under heavy traffic due to their tightly coupled architecture and inability to scale individual components.

In contrast, the cloud-native architecture showed greater resilience in handling heavier loads. Although it experienced a slightly longer average response time of 2.14 seconds in TC-K01 due to AWS Lambda's cold start latency, the system remained stable without any errors. In TC-K02, the cloud-native system maintained SLA compliance with 871.62 ms, and in TC-K03 (peak load), it successfully handled 60 concurrent virtual users with an average response time of 657.10 ms and 0% error rate.

These findings support the claims of Menéndez *et al.* [20], who compared microservices architectures using AWS Lambda and RDS, concluding that such cloud-native setups exhibit strong performance stability and reliability under high traffic, albeit with challenges related to initialization and cold starts. Additionally, Christian and Bisma[6] highlighted that microservices-based systems are more stable and scalable than monolithic systems, even though they may consume more CPU resources. This reinforces the conclusion that cloud-native architecture is more robust under high-demand conditions.

The contrasting performance patterns observed stem from the fundamental nature of each architecture. Monolithic systems are best suited for small-scale, low-variability environments due to their simplicity. However, their rigidity becomes a limitation when dealing with unpredictable or high-volume traffic. On the other hand, cloud-native systems are inherently elastic, with architectures such as AWS Lambda allowing individual components to scale on demand. Despite the advantages, one major drawback remains: cold start latency. Brasoveanu et al. [21] noted that cold starts in serverless environments introduce delay depending on the runtime environment, deployment size, and configuration, typically ranging from a few hundred milliseconds to several seconds.

Although cold start was not the primary focus of this study, it remains a common challenge in serverless applications. AWS offers a feature called provisioned concurrency, which allows Lambda functions to remain pre-initialized to reduce startup latency, although this comes with additional operational costs. For systems requiring consistently low response times, such mitigation strategies are worth considering, as highlighted in recent studies evaluating the effectiveness of such approaches in significantly reducing cold start impact [21].

4. Conclusions and Future Works

Based on the results of this study, it can be concluded that the cloud-native architecture outperforms the monolithic architecture in terms of scalability and system resilience under high load conditions, thanks to its support for autoscaling and real-time monitoring. On the other hand, the monolithic architecture delivers faster response times under light to moderate loads but suffers from significant performance degradation when subjected to peak load scenarios.

For an automotive workshop information system that operates in a dynamic environment and may experience sudden traffic spikes, the cloud-native approach is more recommended. However, optimization is needed to reduce Lambda cold start latency, in order to improve response times under light load conditions.

Furthermore, the improvements in system responsiveness, scalability, and stability are expected to contribute positively to customer satisfaction and overall service efficiency. For an automotive workshop that depends on timely and accurate transaction processing, the reduction in error rates and consistent performance under heavy load can minimize service delays, reduce queuing time, and enhance customer trust. These factors may also improve key business metrics such as service throughput, system availability, and customer retention rate.

For future research, it is suggested to include evaluation parameters related to operational costs and resource efficiency, in order to provide a more comprehensive understanding of the benefits of migrating to cloud-native architecture, from both technical and economic perspectives.

5. References

[1] K. M. H. and M. A. Romli, “Application for Mental Health Consultation with Scheduling Function at the Counseling Guidance of Universitas Teknologi Yogyakarta”.

[2] L. Y. J. Lin, D. Xie, J. Huang, and Z. Liao, “A multi-dimensional extensible cloud-native service stack for enterprises”, doi: 10.1186/s13677-022-00366-7.

[3] B. T. H. B. H. Lavenia, W. Hayuhardhika, and N. Putra, “Pengembangan Sistem Informasi Point of Sales untuk Bengkel berbasis Cloud Computing (Studi Kasus: Bengkel Mas Pur Baturaja)”.

[4] D. D. Putra and M. R. P. Putra, “Load Balance Design of Google Cloud Compute Engine VPS with Round Robin Method in PT. Lintas Data Indonesia”, doi: 10.33395/sinkron.v3i2.10064.

[5] G. I. S. A’fa Nafasha, I. Putu, and E. Indrawan, “Analisis Perbandingan Biaya dan Serverless Computing pada Google Cloud Platform”.

[6] Y. C. and R. Bisma, “Studi Perbandingan Performa Aplikasi Web Monolitik dan Microservice Berbasis Apache Kafka”.

[7] G. Blinowski, A. Ojdowska, and A. Przybylek, “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation,” *IEEE Access*, vol. 10, pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[8] H. S. and W. Hasselbring, “A configurable method for benchmarking scalability of cloud-native applications”, doi: 10.1007/s10664-022-10162-1.

[9] D. N. and H. Mulyono, “Sistem Informasi Manajemen Bengkel Berbasis Web pada Bengkel Ikhsan Jaya Motor”.

[10] V. R. S. R. Rakhman, M. Shadiq, and D. Syaddad, “Implementasi Cloud Native dan Multi Cloud Pada Sistem Operasi Windows Dengan Menggunakan Docker dan Cara Penggunaannya”, doi: 10.5281/zenodo.13119907.

[11] S. D. and others, “Cloud-Native Computing: A Survey From the Perspective of Services”, doi: 10.1109/JPROC.2024.3353855.

[12] M. C. A. M. Ştefan, N. R. Rusu, and E. Ovreiu, “Empowering Healthcare: A Comprehensive Guide to Implementing a Robust Medical Information System”, doi: 10.3390/asi7030051.

[13] J. S. Patel, “Cloud-Native Performance Testing: Strategies for Scalability and Reliability in Modern Applications”, doi: 10.37745/ejcsit.2013/vol13n83249.

[14] D. Narsina, “Microservices vs. Monoliths: Comparative Analysis for Scalable Software Architecture Design”, doi: 10.18034/ei.v11i2.734.

[15] M. B. and others, “Migrasi Cloud Native Architecture API Development untuk Memaksimalkan 5G Monetization pada Jaringan AXIATA”.

[16] T. R. T. Bodner, T. Radig, D. Justen, and D. Ritter, “An Empirical Evaluation of Serverless Cloud Infrastructure for Large-Scale Data Processing”.

[17] X. Jia, “The Role and Importance of Software Testing in Software Quality Management”.

[18] S. Eismann *et al.*, “A Review of Serverless Use Cases and their Characteristics,” Jan. 2021, [Online]. Available: http://arxiv.org/abs/2008.11110

[19] M. G. C. F. Fan and A. Jindal, “Microservices vs serverless: A performance comparison on a cloud-native web application”, doi: 10.5220/0009792702040215.

[20] J. M. Menéndez, J. E. L. Gayo, E. R. Canal, and A. E. Fernández, “A comparison between traditional and Serverless technologies in a microservices setting”.

[21] R. A. A. Brasoveanu and M. Moodie, “Textual evidence for the perfunctoriness of independent medical reviews”.