Sciforce

International Journal of Cloud Computing and Supply Chain Management

Journal homepage: www.sciforce.org

ISSN : 3067-0535

Open Access Research Article

Predictive Performance Modeling of Java-Based Microservices in Dynamic Cloud Environments Using Machine Learning Focus: ML prediction and Java optimization

Tirumala Rao Gundala*

Consulting Technical Manager & Performance Architect, Oracle, United States

Abstract

This research investigates performance inefficiencies in cloud-native microservice applications built with Java. Using machine learning algorithms, we analyze system metrics including request rate, response time, and CPU utilization to predict and optimize throughput in dynamic cloud environments.Introduction: Microservices architectures enable scalability and resilience but present performance engineering challenges. Java-based microservices face memory consumption, startup time, and garbage collection issues. Current literature lacks systematic performance-oriented analysis and optimization approaches for cloud-native microservice systems.Research Significance: Performance engineering for microservices remains underexplored despite its criticality. This research addresses data collection, monitoring, and diagnostic challenges across multiple system layers. Findings help optimize Java microservice deployments, reducing latency, improving resource utilization, and enhancing user experience in cloud environments.Methodology: We collected 100 performance observations from cloud-native Java microservices, measuring request rate (rps), average response time (ms), CPU utilization (%), and system throughput (rps). We applied Linear Regression and Random Forest Regression models to establish correlations between metrics and predict system throughput, evaluating model accuracy using standard error metrics.Results and Discussion: Descriptive statistics revealed significant performance variability: request rates ranged from 113 to 1999 rps (mean: 1072.58), response times from 24.31 to 789.77 ms (mean: 394.71), and CPU utilization from 10.43% to 93.78% (mean: 53.52%). Linear Regression achieved R^2 of 0.9577 on training data; Random Forest Regression significantly outperformed with R^2 of 0.9922, demonstrating superior predictive accuracy. Both models generalized reasonably to test data, with RFR exhibiting better robustness in handling extreme values. Future Scope: Investigate dynamic workload management and containerization optimization techniques for Java microservices.

Keywords: Cloud-native computing, Microservices architecture, Performance engineering, Java optimization, Machine learning prediction, System throughput.

Introduction

While performance is a fundamental requirement for achieving scalability and resilience in microservice-based systems, performance engineering for microservices has received relatively little attention from both the microservices and performance engineering research communities. Most existing studies primarily focus on architectural design, deployment strategies, and service orchestration, while performance-oriented analysis, modeling, and optimization are less explored. This lack of emphasis highlights the need for systematic performance engineering approaches that can effectively support the scalability and resilience demands of modern microservice architectures. [1]

Microservice-based architectures introduce several domain-specific challenges that need careful consideration. For example, due to frequent releases and continuous deployment cycles, comprehensive system-level testing becomes impractical. Consequently, modifications to traditional testing and performance evaluation approaches are required to effectively address the dynamic and rapidly evolving nature of microservice environments.The measurements collected in this manner are later used to identify anomalies such as unusually high response times or excessive resource consumption. In microservice architectures, performance-related data can be collected at multiple levels: from within individual microservices, from the containers running them, and from the interactions between interdependent microservices. However, collecting data across these layers and establishing correlations between them presents several challenges in terms of data integration, monitoring overhead, and accurate performance diagnostics. [2]

This research paper investigates performance inefficiencies in cloud-native applications operating within dynamic cloud environments. Such environments are often characterized by fluctuating workloads, elastic resource provisioning, and continuous deployment practices, all of which can contribute to unstable and suboptimal application behavior. These performance deficiencies negatively impact user experience by increasing latency and reducing service reliability, while also decreasing operational efficiency through inefficient resource utilization

Received date: December 09, 2025 **Accepted date:** December 19, 2025; **Published date:** December 27, 2025

***Corresponding Author:** Gundala, Tirumala Rao, Consulting Technical Manager & Performance Architect, Oracle, United States, E- mail: Tirumalagundala7@gmail.com

Citation: Gundala, Tirumala Rao. (2025). Predictive Performance Modeling of Java-Based Microservices in Dynamic Cloud Environments Using Machine Learning Focus: ML prediction and Java optimization. International Journal of Cloud Computing and Supply Chain Management, 1(4), 1-7. doi: https://doi.org/10.55124/ijccscm.v1i4.250

and high management overhead. Addressing these challenges is therefore crucial for ensuring reliable and scalable cloud-native systems. [3]

This provides a comprehensive overview of how microservices can be effectively utilized within cloud-native applications to enhance scalability, resilience, and adaptability. By exploring architectural principles, deployment practices, and evolving technologies, this study underscores the crucial role of microservices in supporting robust and scalable cloud-native systems.This flexibility is commonly referred to as polyglot programming, which allows developers to use multiple programming languages and technologies within a single system. The shift from monolithic architectures to microservices represents a fundamental change in the design and management of software applications. Traditionally, monolithic architectures have been used for decades; in this approach, the entire application is built and deployed as a single, tightly integrated block of code. While this approach simplifies initial development, it often limits scalability, flexibility, and maintainability as applications grow in complexity. [4-5]

In the development of cloud-native applications, which are specifically designed to fully utilize the capabilities of cloud computing, the microservices architecture has gained considerable importance. Cloud-native applications prioritize resilience, scalability, and ease of maintenance, enabling them to perform efficiently in dynamic and distributed environments.In a microservices architecture, individual services can be updated or modified independently without affecting the entire application. This isolation reduces the risk associated with system changes and enables frequent, incremental updates, thereby supporting continuous integration and continuous deployment practices. [6]

While Java is a popular and reliable choice for building microservices, it presents several challenges, including high memory consumption, long startup times, and performance issues associated with garbage collection. These limitations can impact the performance and responsiveness of microservice-based applications, especially in resource-constrained or highly dynamic environments.Several studies comparing Java and Go show that Go generally performs better than Java in terms of sustained performance. This performance advantage is largely due to Go being a statically compiled language with minimal runtime overhead, resulting in faster execution and efficient resource utilization.[7]

Traditional Java threads generally consume more memory, and creating and terminating them requires significant explicit effort. This added overhead can limit scalability and performance, especially in systems requiring high concurrency, such as microservice-based applications.[8]

A comprehensive study on microservices testing approaches published on the arrive platform highlights that inconsistencies in the environment lead to behaviors that cannot be reliably reproduced in testing systems. Such discrepancies pose significant challenges in verifying microservices and ensuring consistent performance across development, testing, and production environments.[9]

that microservices need to sustain resilience under network-related challenges, effectively managing situations where dependent services experience slow responses or become unavailable. Ensuring such fault tolerance is critical for maintaining the reliability and stability of distributed microservice-based systems. [10]

For efficient testing of microservices, limitations in observability and diagnosability pose significant challenges. According to research published on the distributed nature of request processing across multiple services creates considerable complexities. This makes it difficult to fully understand and analyze the behavior of the system during testing.[11]

In recent years, a new architectural paradigm, the cloud-native microservices architecture, has emerged as a prominent approach in software system design. This architecture designs applications as a collection of loosely coupled services; these services can be developed, deployed, and scaled independently as needed. This approach enhances flexibility, resilience, and scalability, making it highly suitable for modern cloud-based environments.[12]

One of the key challenges in microservices-based systems is maintaining data consistency and state management across multiple services, while simultaneously ensuring smooth and reliable data transitions. Addressing this challenge is crucial for safeguarding the system's integrity and supporting seamless interactions between independently deployed services. [13]

The process of refactoring a monolithic application into a microservices-based architecture introduces numerous technical and organizational challenges. These challenges arise from decomposing tightly coupled components, managing data consistency, redefining service boundaries, and ensuring the reliability of the system during and after the transformation.[14]

Cloud-native computing, fostered by a vendor-neutral ecosystem, encompasses a set of technologies that break down applications into microservices and package them into lightweight containers. These containerized services are then deployed and orchestrated across various computing environments, thereby enabling portability, scalability, and efficient resource utilization. [15]

Cloud-based capabilities offer significant potential for improving software engineering practices in many ways. However, realizing these benefits across the broader software engineering community presents considerable technical, organizational, and procedural challenges. [16]

Material and Methods

Request Rate (rps): Request rate, measured in requests per second (rps), represents the volume of incoming client requests handled by the system over a given time interval. It is a key indicator of workload intensity and system demand, reflecting how frequently services are invoked. Monitoring the request rate helps in understanding traffic patterns, identifying peak loads, and evaluating whether the system can scale effectively to accommodate fluctuating user demand.

Average Response Time (ms): Average response time, measured in milliseconds, indicates the mean duration taken by the system to process and respond to incoming requests. This metric is critical for assessing user-perceived performance, as higher response times often lead to poor user experience and reduced service satisfaction. Analyzing average response time enables the identification of performance bottlenecks and helps ensure that service-level objectives are consistently met.

CPU Utilization (%): CPU utilization, expressed as a percentage, measures the proportion of processing capacity consumed by the application during operation. It provides insights

Citation: Gundala, Tirumala Rao. (2025). Predictive Performance Modeling of Java-Based Microservices in Dynamic Cloud Environments Using Machine Learning Focus: ML prediction and Java optimization. International Journal of Cloud Computing and Supply Chain Management, 1(4), 1-7. doi: https://doi.org/10.55124/ijccscm.v1i4.250

into how efficiently computational resources are being used and whether the system is underutilized or experiencing resource saturation. High CPU utilization may signal performance constraints, while persistently low utilization may indicate inefficient resource allocation.

System Throughput (rps): System throughput, measured in requests per second, represents the number of requests successfully processed by the system within a specific time frame. It reflects the overall processing capability and efficiency of the system under varying workloads. Higher throughput generally indicates better performance, provided that response time and resource consumption remain within acceptable limits.

Machine Learning Algorithms

Linear Regression: Linear Regression is a fundamental statistical and machine learning technique used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear association and estimates model parameters by minimizing the error between predicted and observed values. Due to its simplicity, interpretability, and low computational cost, Linear Regression is often used as a baseline model for prediction and performance comparison in data-driven studies.

Random Forest Regression: Random Forest Regression is an ensemble learning method that combines multiple decision trees to improve predictive accuracy and robustness. By training each tree on a randomly selected subset of data and features, the model reduces overfitting and enhances generalization. Random Forest Regression is particularly effective in capturing complex, nonlinear relationships and handling high-dimensional data, making it well suited for performance prediction and optimization tasks in dynamic and heterogeneous environments.

Result and Discussion

Table 1. Descriptive Statistics

	Request_Rate_rps	Avg_Response_Time_ms	CPU_Utilization_percent	System_Throughput_rps
count	100.0000	100.0000	100.0000	100.0000
mean	1072.5800	394.7119	53.5221	691.7277
std	532.1617	233.5579	25.1205	401.4429
min	113.0000	24.3073	10.4302	56.9000
0.2500	649.0000	204.7593	30.5704	369.4425
0.5000	1098.0000	370.9048	56.9172	612.1250
0.7500	1519.0000	621.4135	72.2867	1044.2675
max	1999.0000	789.7718	93.7803	1568.5300

The descriptive statistics indicate considerable variability in system performance across the observed samples. The request rate ranges from 113 to 1999 requests per second, with a mean of 1072.58 rps, reflecting a highly dynamic workload. Correspondingly, the average response time varies widely between 24.31 ms and 789.77 ms, with a mean value of 394.71 ms, suggesting inconsistent system responsiveness under different load conditions. CPU utilization spans from 10.43% to 93.78%, with an average utilization of 53.52%, indicating alternating periods of underutilization and high processing demand. System throughput also shows significant fluctuation, ranging from 56.9 to 1568.53 rps, with a mean of 691.73 rps. Overall, these results highlight the strong interdependence between workload intensity, resource utilization, and system performance, emphasizing the need for effective performance optimization and dynamic resource management in cloud-native environments.

Table 2. Model Performance Comparison on Training Dataset

Data	Symbol	R2	EVS	MSE	RMSE	MAE	MaxError	MSLE	MedAE
Train	LR	0.957655	0.957655	7025.26	83.81683	64.73825	206.7129	32.57055	49.24015
Train	RFR	0.992237	0.992267	1287.97	35.8883	28.04706	116.3131	0.00512	22.8106

The table 2 shows that, the performance of two models—Linear Regression (LR) and Random Forest Regression (RFR)—on the training dataset shows notable differences. The LR model achieved an R^2 of 0.9577 and an explained variance score (EVS) of 0.9577, indicating it explains approximately 95.8% of the variability in the data. Its error metrics are relatively higher, with a mean squared error (MSE) of 7025.26, a root mean squared error (RMSE) of 83.82, a mean absolute error (MAE) of 64.74, a maximum error of 206.71, a mean squared logarithmic error (MSLE) of 32.57, and a median absolute error (MedAE) of 49.24.In comparison, the RFR model performs substantially better on the training set, achieving an R^2 of 0.9922 and an EVS of 0.9923, showing it explains over 99% of the variance. Its error metrics are significantly lower, with an MSE of 1287.97, RMSE of 35.89, MAE of 28.05, maximum error of 116.31, MSLE of 0.0051, and MedAE of 22.81, indicating much higher accuracy and more reliable predictions than the linear model.

Citation: Gundala, Tirumala Rao. (2025). Predictive Performance Modeling of Java-Based Microservices in Dynamic Cloud Environments Using Machine Learning Focus: ML prediction and Java optimization. International Journal of Cloud Computing and Supply Chain Management, 1(4), 1-7. doi: https://doi.org/10.55124/ijccscm.v1i4.250

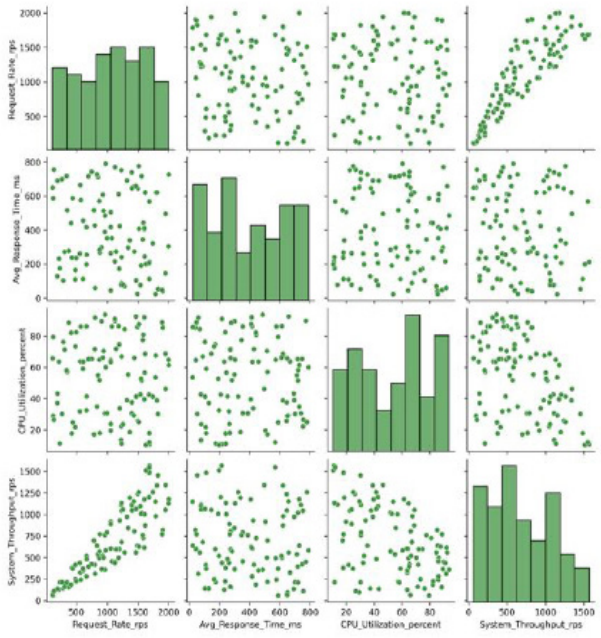

Figure 1: Pairwise relationships and distributions of system performance metrics

Figure 1 presents a pairwise visualization of key system performance metrics, including request rate, average response time, CPU utilization, and system throughput. The diagonal plots illustrate the distribution of each metric, revealing wide variability across observations, while the off-diagonal scatter plots highlight the relationships between different performance indicators. A strong positive correlation is evident between request rate and system throughput, indicating that throughput increases proportionally with incoming workload up to higher load levels. In contrast, average response time and CPU utilization exhibit more scattered relationships with other metrics, suggesting nonlinear and workload-dependent behavior.

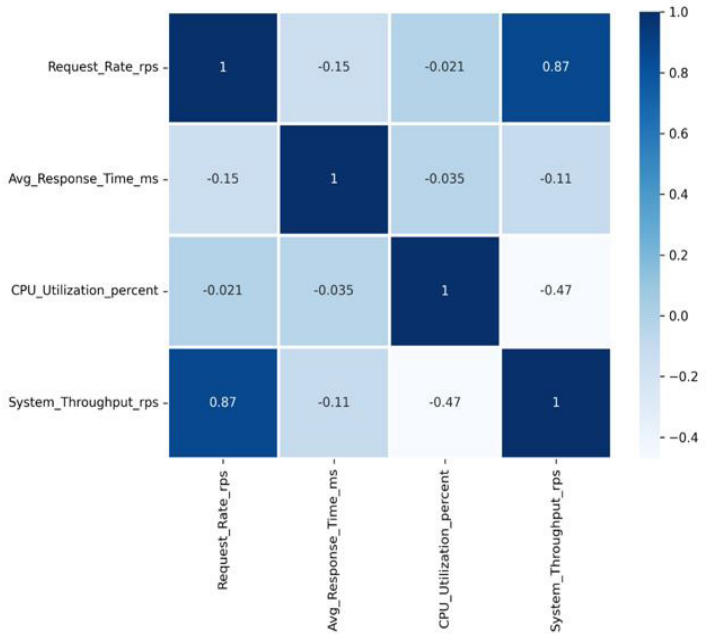

Figure 2: Correlation heatmap of system performance metrics

Citation: Gundala, Tirumala Rao. (2025). Predictive Performance Modeling of Java-Based Microservices in Dynamic Cloud Environments Using Machine Learning Focus: ML prediction and Java optimization. International Journal of Cloud Computing and Supply Chain Management, 1(4), 1-7. doi: https://doi.org/10.55124/ijccscm.v1i4.250

Figure 2 illustrates the correlation matrix among key system performance metrics, including request rate, average response time, CPU utilization, and system throughput. A strong positive correlation (0.87) is observed between request rate and system throughput, indicating that higher incoming workloads generally result in increased processing output. In contrast, CPU utilization shows a moderate negative correlation with system throughput (–0.47), suggesting potential efficiency degradation or resource contention at higher utilization levels. Average response time exhibits weak negative correlations with the other metrics, implying a complex and possibly nonlinear relationship with workload and resource usage.

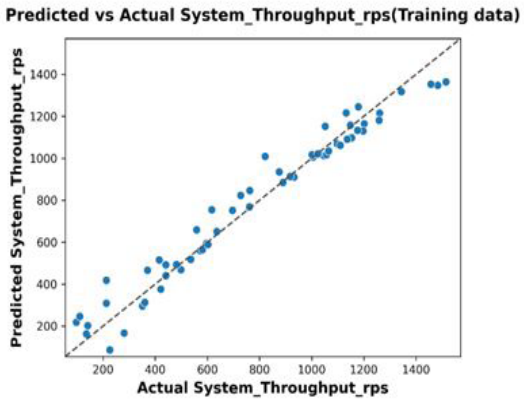

Figure 3: Predicted versus actual system throughput for the training dataset

Figure 3 compares the predicted system throughput values generated by the regression model with the corresponding actual throughput measurements from the training dataset. The majority of data points are closely aligned with the diagonal reference line, indicating strong agreement between predicted and observed values and demonstrating the model's ability to capture the underlying relationship between input features and system throughput. Minor deviations from the diagonal suggest the presence of localized prediction errors, particularly at lower and higher throughput ranges.

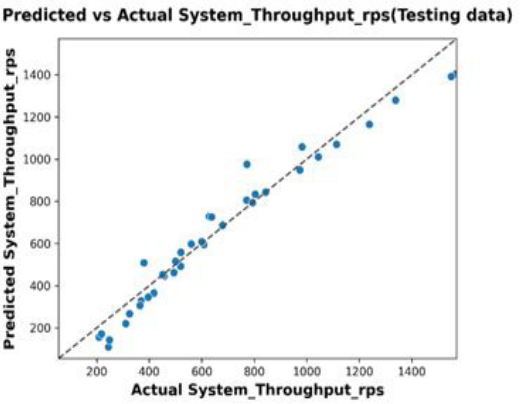

Figure 4: Predicted versus actual system throughput for the testing dataset

Figure 4 presents a comparison between the predicted and actual system throughput values for the testing dataset, providing an evaluation of the model's generalization capability. Most data points closely follow the diagonal reference line, indicating strong agreement between predicted and observed throughput under previously unseen conditions. Minor deviations at higher throughput levels suggest slight prediction errors when the system operates near peak capacity.

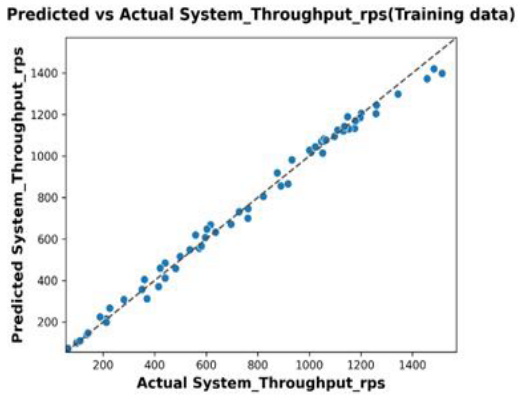

Figure 5: Predicted versus actual system throughput for the training dataset (refined model)

Figure 5 illustrates the relationship between predicted and actual system throughput values obtained from the training dataset using the refined regression model. The data points are tightly clustered around the diagonal reference line, indicating a high level of agreement between model predictions and observed throughput. Compared to earlier training results, the reduced dispersion suggests improved model fitting and more accurate representation of the underlying system behavior.

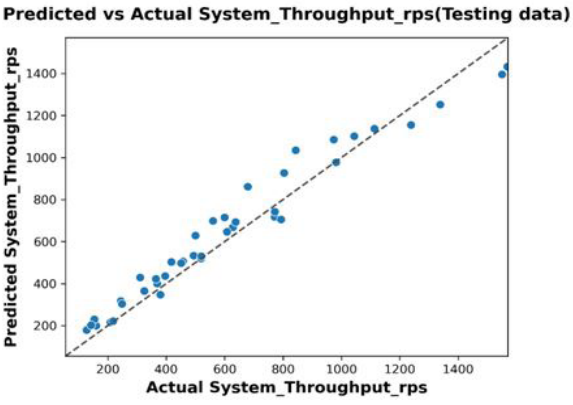

Figure 6: Predicted versus actual system throughput for the testing dataset

Figure 6 presents the comparison between predicted and actual system throughput values for the testing dataset. Although a slight increase in dispersion is observed compared to the training results, most data points remain closely aligned with the diagonal reference line, indicating strong generalization capability of the proposed

Citation: Gundala, Tirumala Rao. (2025). Predictive Performance Modeling of Java-Based Microservices in Dynamic Cloud Environments Using Machine Learning Focus: ML prediction and Java optimization. International Journal of Cloud Computing and Supply Chain Management, 1(4), 1-7. doi: https://doi.org/10.55124/ijccscm.v1i4.250

regression model. The minor deviations at higher throughput levels suggest the presence of dynamic workload variations and system-level uncertainties in unseen data.

Table 3. Model Performance Comparison on Testing Dataset									
Data	Symbol	R2	EVS	MSE	RMSE	MAE	MaxError	MSLE	MedAE
Test	LR	0.947233	0.952693	7363.547	85.81111	66.55762	204.2074	30.27242	50.20547
Test	RFR	0.948949	0.958429	7124.113	84.40446	69.40009	192.9612	0.027998	56.09848

The table 3 shows that, the performance of the models on the test dataset shows some interesting patterns. The Linear Regression (LR) model achieved an R^2 of 0.9472 and an explained variance score (EVS) of 0.9527, indicating it captures roughly 94.7% of the variance in the data. Its error metrics include a mean squared error (MSE) of 7363.55, a root mean squared error (RMSE) of 85.81, a mean absolute error (MAE) of 66.56, a maximum error of 204.21, a mean squared logarithmic error (MSLE) of 30.27, and a median absolute error (MedAE) of 50.21, showing reasonably accurate predictions but with some larger deviations.The Random Forest Regression (RFR) model achieved slightly better overall performance on the test set, with an R^2 of 0.9489 and an EVS of 0.9584, explaining nearly 94.9% of the variance. Its error metrics show mixed results: the MSE is 7124.11, RMSE is 84.40, and maximum error is 192.96, which are slightly better than LR, but the MAE (69.40) and MedAE (56.10) are somewhat higher, suggesting slightly larger typical deviations for most predictions. The MSLE for RFR is extremely low (0.028), indicating that on a logarithmic scale, the predictions are very close to actual values.

Conclusion

The analysis of both Linear Regression (LR) and Random Forest Regression (RFR) models shows that while both approaches perform well in predicting the target variable, the RFR model consistently demonstrates superior performance, particularly in the training phase. On the training data, RFR achieves a higher R^2 (0.9922) and explained variance (0.9923), along with significantly lower error metrics (MSE, RMSE, MAE, and MedAE) compared to LR, indicating a better fit and more accurate predictions.On the test data, both models generalize reasonably well, with R^2 values of 0.9472 for LR and 0.9489 for RFR. Although the Random Forest model shows slightly higher mean and median absolute errors than LR, it maintains a lower maximum error and extremely low MSLE, suggesting it handles extreme values more robustly. Overall, the RFR model proves to be more reliable and precise, making it the preferred choice for predictive tasks in this dataset, while LR can still serve as a simpler baseline model.

References

1. Heinrich, Robert, André Van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, and Johannes Wettinger. “Performance engineering for microservices: research challenges and directions.” In Proceedings of the 8th ACM/SPEC on international conference on performance engineering companion, pp. 223-226. 2017.

2. Varun Venkatesh Dandasi, Suresh Deepak Gurubasannavar, Raghavendra Sunku. (2023). Enhancing Smart Grid Security: A Multi-Criteria Evaluation Through GRA Method. International Journal of Information Technology and Management Information Systems (IJITMIS), 14(2), 153-167.

3. Heinrich, Robert, André Van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, and Johannes Wettinger. “Performance engineering for microservices: research challenges and directions.” In Proceedings of the 8th ACM/SPEC on international conference on performance engineering companion, pp. 223-226. 2017.

4. Gurubasannavar. S D, Sunku. R, Dandasi. V V. (2025). Selecting an Extract, Transform, and Load (ETL) Software Solution: A Comprehensive Evaluation and Comparison . International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-7. doi: https://doi.org/10.55124/ijccscm.v1i3.249

5. Divya Soundarapandian. (2024) Algorithmic Framework for Retail Media Optimization and Consumer Engagement Enhancement. Journal of Business Intelligence and Data Analytics, 1(3), 1-7. https://doi.org/10.55124/jbid.v1i3.259

6. Mamillapalli, Siva Kumar, and Ramya Devi Jeganathan. “Mastering Cloud-Native Performance: Strategies for Optimization.”

7. Perikala. K (2024). Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems. International Journal of Computer Science and Data Engineering, 1(3), 1–7 doi: https://dx.doi.org/10.55124/csdb.v1i3.268

8. Oyeniran, Oyekunle Claudius, Adebunmi Okechukwu Adewusi, Adams Gbolahan Adeleke, Lucy Anthony Akwawa, and Chidimma Francisca Azubuko. “Microservices architecture in cloud-native applications: Design patterns and scalability.” International Journal of Advanced Research and Interdisciplinary Scientific Endeavours 1, no. 2 (2024): 92-106.

9. Karri, SairamakrishnaBuchiReddy, Chandra Mouli Penugonda, Srujana Karanam, Mohd Tajammul, Srinivasarao Rayankula, and Prasad Vankadara. “Enhancing Cloud-Native Applications: A Comparative Study of Java-To-Go Micro Services Migration.” International Transactions on Electrical Engineering and Computer Science 4, no. 1 (2025): 1-12.

10. Dhandapani, Aswinkumar. “Automation Testing in Microservices and Cloud-Native Applications: Strategies and Innovations.” Journal of Computer Science and Technology Studies 7, no. 3 (2025): 826-836.

11. Ingole, Anushka Anil, and Nikhil E. Karale. “Java in Microservices Architecture: A Study on Spring Boot and Cloud-Native Development.” International Journal of Ingenious Research, Invention and Development 4, no. 2 (2025).

12. Rahman, F. “Cloud-Native Microservices for Next-Gen Computing Applications and Scalable Architectures.” (2025).

13. Varun Venkatesh Dandasi, Suresh Deepak Gurubasannavar, Raghavendra Sunku. (2024). Data Mart Design and Predictive Modeling at Palandir Foundry: from Ontology to Operational Intelligence Using Machine Learning. International Journal of Computer Engineering and Technology (IJCET), 15(2), 274-294.

14. Suresh Deepak Gurubasannavar, Varun Venkatesh Dandasi, Raghavendra Sunku. (2025). Enhancing Smart Home Automation with AI And Topsis-Based Decision Making. International Journal of Information Technology and Management Information Systems

(IJITMIS), 16(6), 1-22.

15. Mishra, Mayank, Shruti Kunde, and Manoj Nambiar. “Cracking the monolith: Challenges in data transitioning to cloud native architectures.” In Proceedings of the 12th European conference on software architecture: companion proceedings, pp. 1-4. 2018.

16. Deng, Shuiguang, Hailiang Zhao, Binbin Huang, Cheng Zhang, Feiyi Chen, Yinuo Deng, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. “Cloud-native computing: A survey from the perspective of services.” Proceedings of the IEEE 112, no. 1 (2024): 12-46.

17. Perikala. K. (2025). Cloud-Native NoSQL Foundations for Large-Scale Generative AI Platforms. International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-6. doi: https://doi.org/10.55124/ijccscm.v1i3.248

18. Mitchell, Brian S. “Cloud Native Software Engineering.” arXiv preprint arXiv:2307.01045 (2023).

19. Marie-Magdelaine, Nicolas. “Observability and resources managements in cloud-native environnements.” PhD diss., Université de Bordeaux, 2021.

20. Walker, Sarah. “Integrating Cloud-Native Microservices for Agile Banking Operations and Continuous Innovation.”

21. Divya Soundarapandian. (2024) Reliability Analysis of Data Science Workflow Components Using SPSS A Correlation-Based Study. International Journal of Computer Science and Data Engineering, 1(2), 1–7. doi: https://dx.doi.org/10.55124/csdb.v1i2.266

22. Gurubasannavar, S. D. (2025). Performance Optimization for Micro-Frontend-Based Applications: A Predictive Analysis Using XG Boost Regression. Journal of Business Intelligence and Data Analytics, 2(3), 1-7. https://doi.org/10.55124/jbid.v2i3.256

23. Nadipalli, Rajesh. “Cloud-Native Java Application Security through DevSecOps Practices.” Journal of Mathematical & Computer Applications 1, no. 4 (2022): 1-5.

24. Theodoropoulos, Theodoros, Luis Rosa, Chafika Benzaid, Peter Gray, Eduard Marin, Antonios Makris, Luis Cordeiro et al. “Security in cloud-native services: A survey.” Journal of Cybersecurity and Privacy 3, no. 4 (2023): 758-793.

25. Raghavendra Sunku. (2024). AI-Powered Forecasting and Insights in Big Data Environments. Journal of Business Intelligence and Data Analytics, 1(2), 254. https://doi.org/10.55124/jbid.v1i2.254

26. Evans, Benjamin J., and James Gough. Optimizing Cloud Native Java: Practical Techniques for Improving JVM Application Performance. “ O’Reilly Media, Inc.”, 2024.

27. Perikala. K (2024). Architecting Retail-Scale Product Knowledge Graph Systems. International Journal of Artificial intelligence and Machine Learning, 2(3), 1-6. doi: https://doi.org/10.55124/jaim.v2i3.292