A Resilience Assessment Framework for Hybrid Monolithic-Microservices Systems: A Comparative Case Study of Odoo ERP Integration

Alex Kurniawan
Data Science and Cyber Security, Faculty of Information Technology, Swiss German University, The Prominence Tower Alam Sutera, Jl. Jalur Sutera Barat No. 15, Banten 15143, Indonesia
Email: alexkurniawan.ng@gmail.com

The migration of traditional monolithic Enterprise Resource Planning (ERP) systems to modern architectures often results in complex hybrid systems where a core monolith coexists with new microservices. These hybrid models present significant challenges in ensuring end-to-end reliability and observability. This paper proposes and validates a systematic framework for identifying, analyzing, and resolving performance bottlenecks in such environments, targeting the ISO/IEC 25010 reliability sub-characteristics: Maturity, Availability, Fault Tolerance, and Recoverability. Employing a quantitative pre-test/post-test methodology, a hybrid Odoo-NestJS system was instrumented with a comprehensive observability stack and subjected to targeted stress tests using k6. Baseline tests revealed critical vulnerabilities, including a 7.80% failure rate during resource-intensive requests and severe cascading failures. By implementing targeted architectural interventions, namely asynchronous worker pools, request timeouts, and retry patterns, the optimized system achieved a 100% reduction in processing failures, doubled its throughput, and reduced user-facing downtime errors by 88%. The findings validate that specific operational patterns provide substantial, quantifiable improvements to system stability in a hybrid monolithic-microservice context.
Keywords: ISO/IEC 25010, Hybrid Architecture, ERP, Monolith Microservices, Performance Engineering.

Corresponding Author:
Alex Kurniawan
Faculty of Information Technology, Swiss German University
The Prominence Tower Alam Sutera, Jl Jalur Sutera Barat No. 15, Banten 15143, Indonesia
alexkurniawan.ng@gmail.com

1. Introduction

In the era of digital transformation, Enterprise Resource Planning (ERP) systems serve as the operational backbone of modern enterprises, integrating critical functions ranging from finance and human resources to supply chain management. Historically, these systems, including widely used open-source platforms like Odoo, have been architected as large, tightly-coupled monoliths. While this traditional approach offers simplicity in initial development and maintains strong data consistency via a single shared database, it frequently encounters significant hurdles in scalability, maintainability, and long-term reliability as business requirements evolve and system complexity grows.

To address the limitations of the monolith, the software engineering industry is increasingly adopting microservice architectures [1], which structure applications as a collection of loosely-coupled, independently deployable services [2]. However, for many established enterprises and Small-to-Medium Enterprises (SMEs), a complete "big bang" rewrite to microservices is prohibitively expensive and carries unacceptable operational risks. Consequently, a pragmatic hybrid architecture has emerged as the standard transition path: the core monolithic ERP is retained for legacy stability, while new, agile capabilities are developed as standalone microservices. [3].

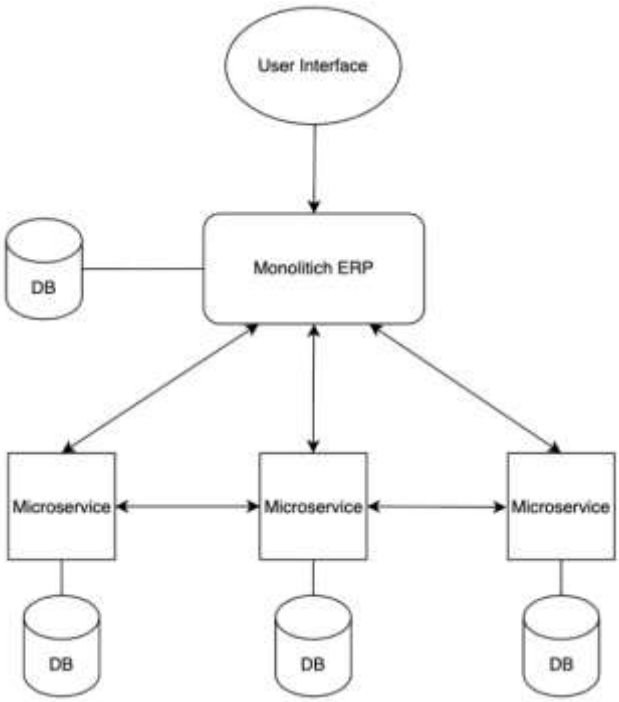

Fig. 1. Hybrid Monolithic with Microservices architecture

Despite its practical benefits, this hybrid model introduces new complexities in managing end-to-end system performance and resilience. The integration points between the centralized monolith and dynamic, distributed microservices often create "observability gaps," where identifying the root cause of a delayed response or system failure becomes exceptionally difficult. This lack of visibility forces engineering teams into reactive troubleshooting, which directly compromises the system's reliability, specifically its Maturity, Availability, Fault Tolerance, and Recoverability, as defined by the international ISO/IEC 25010 standard. [4].

The primary objective of this research is to design and implement a systematic, data-driven framework to proactively resolve these reliability bottlenecks. By deploying a comprehensive observability stack and subjecting a hybrid Odoo-microservice system to realistic stress scenarios, this study seeks to operationalize the ISO/IEC 25010 standard into an active engineering tool. Through this iterative approach, the research validates how specific architectural patterns can provide quantifiable improvements to system stability, bridging the gap between theoretical microservice concepts and real-world hybrid ERP execution.

2. Literature Review and Problem Statement

Current research extensively compares monolithic and microservice architectures, frequently highlighting a trade-off between raw performance and system resilience. For instance, Raharjo et al. utilized the ISO/IEC 25010 standard to evaluate these architectures, finding that while monolithic systems may process requests faster on average, microservices demonstrate significantly better fault tolerance and recoverability during failure scenarios. [5] Similarly, Al-Debagy and Martinek provided empirical evidence that monolithic architectures can outperform microservices under high-concurrency constraints, challenging the assumption that migrating to microservices automatically guarantees performance gains. [6] However, these studies primarily assess pure, homogeneous systems (either entirely monolithic or entirely microservices), leaving a gap regarding the transitional hybrid state. [7].

Furthermore, much of the existing literature addressing the migration from monolith to microservices relies on high-level strategic frameworks rather than operational execution. Studies by Levezinho et al. explore

the application of the Strangler Fig Pattern for monolith decomposition, while Correia and Rito Silva focus on database refactoring patterns to maintain data consistency. [8][9] While these theoretical and architectural roadmaps are essential for project planning, they lack a hands-on, technical methodology for diagnosing and optimizing the runtime reliability of a hybrid system while it is actively operating under user load.

The core problem addressed in this study is this lack of an operational framework to prevent systemic failures at the integration points of hybrid ERP ecosystems. In such architectures, an unoptimized interface can cause severe issues; a single slow downstream microservice can exhaust the monolith's worker threads, triggering a cascading failure that renders the entire upstream application unresponsive [10]. While message broker is a powerful pattern for improving system resilience and perceived performance, blindly implementing a message broker without proper analysis is premature. [11]. It is an architectural solution that shift the bottleneck, but it does not eliminate it. [12]. This research bridges the gap by operationalizing the ISO/IEC 25010 standard alongside modern observability tools: metrics, centralized logs, and distributed traces, to establish a quantitative pre-test/post-test methodology that pinpoints and resolves these specific runtime bottlenecks.

3. Method

This study utilizes a quantitative, experimental pre-test/post-test methodology executed within a controlled cloud staging environment designed to replicate production realities. The infrastructure consists of an Odoo Community Edition v17.0 monolith running on an 8-vCPU Ubuntu instance, integrated with NestJS microservices hosted on a separate server. To enable deep diagnostic analysis, the system was instrumented with an observability stack comprising Prometheus (metrics), Jaeger (distributed tracing), and the ELK Stack (centralized logging). [13].

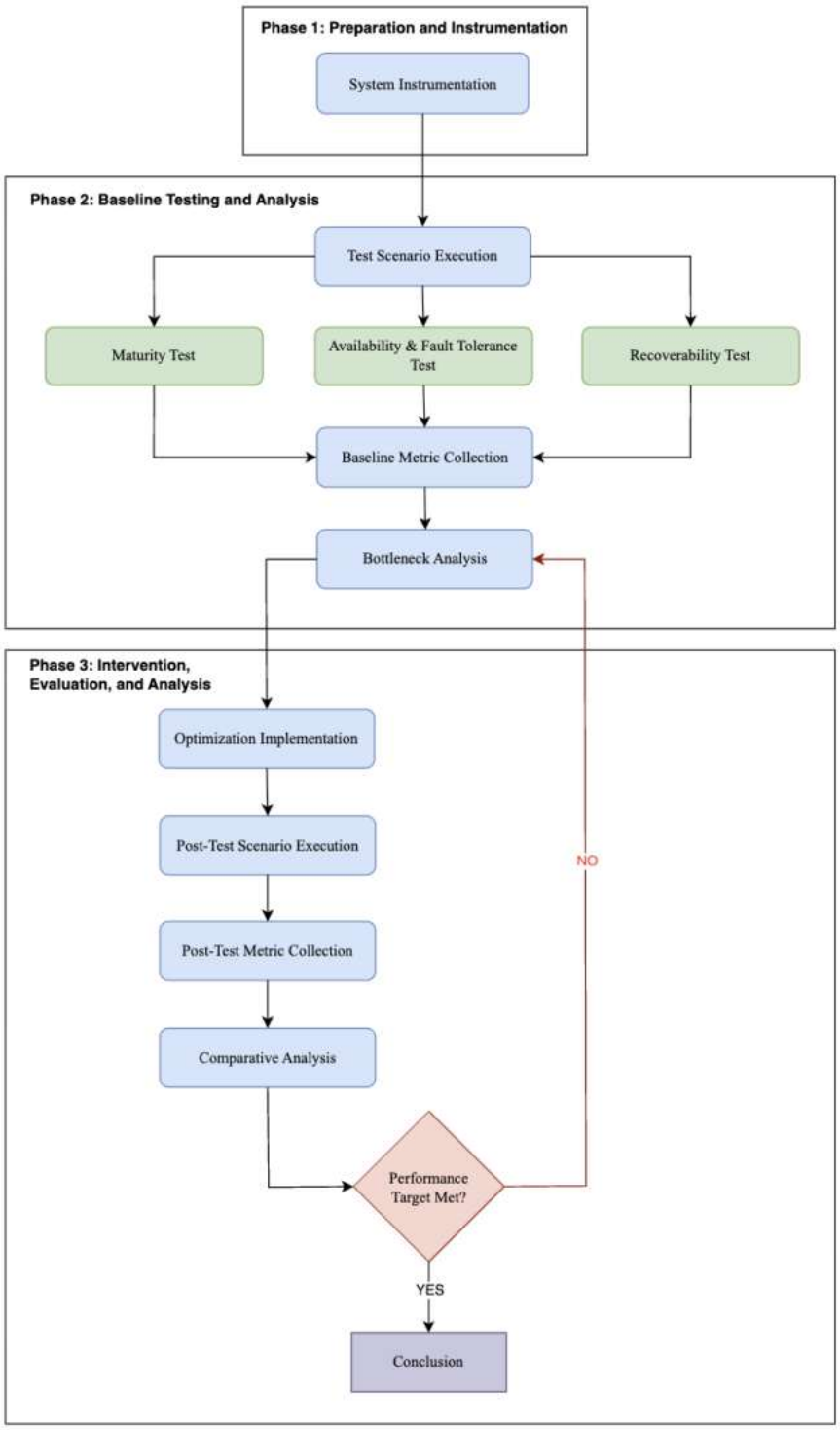

Fig. 2. Research Framework

The research was conducted through an iterative, three-phase framework: (1) Preparation and Instrumentation, (2) Baseline Testing and Analysis, and (3) Intervention and Evaluation. To generate objective data, the load-testing tool k6 was utilized to simulate specific user behaviours mapped directly to the ISO/IEC 25010 reliability sub-characteristics. Three distinct scenarios were defined:

a. Maturity Scenario ("Poison Pill"): Evaluates system stability by injecting a single, massive request containing over 20,000 JSON objects into a product synchronization endpoint while 20 Virtual Users (VUs) sustain normal operational load.

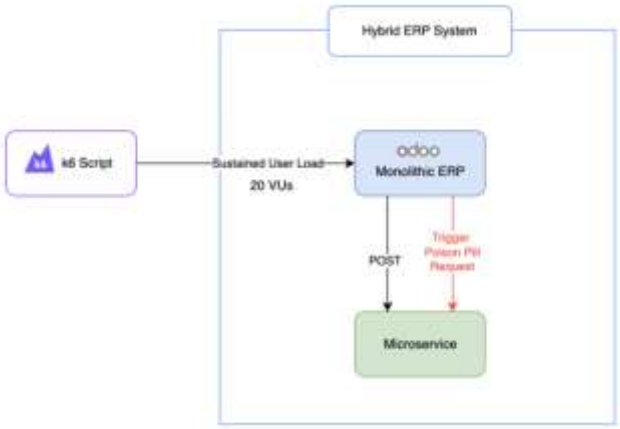

Fig. 3. Maturity Scenario

b. Availability & Fault Tolerance Scenario: Tests fault isolation by routing traffic to an intentionally "sic" microservice with a 15-second response delay, monitoring whether this local fault cascades to impact the latency of unrelated, healthy endpoints on the monolith.

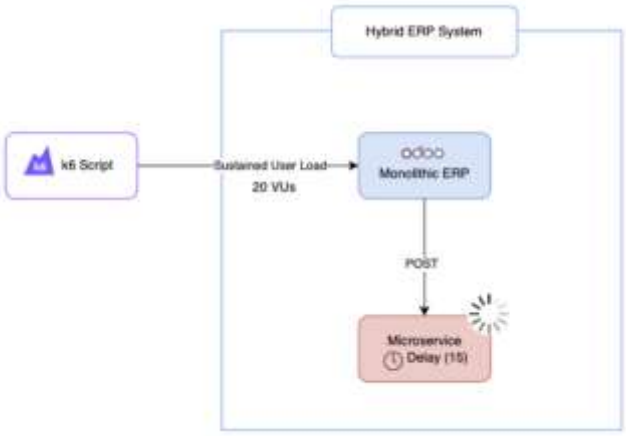

Fig. 4. Availability & Fault Tolerance Scenario

c. Recoverability Scenario: Assesses application resilience by subjecting the monolith to sustained load, manually terminating the target microservice container for two minutes, and measuring both the Time to Recover (TTR) and the user-facing error rate during the outage.

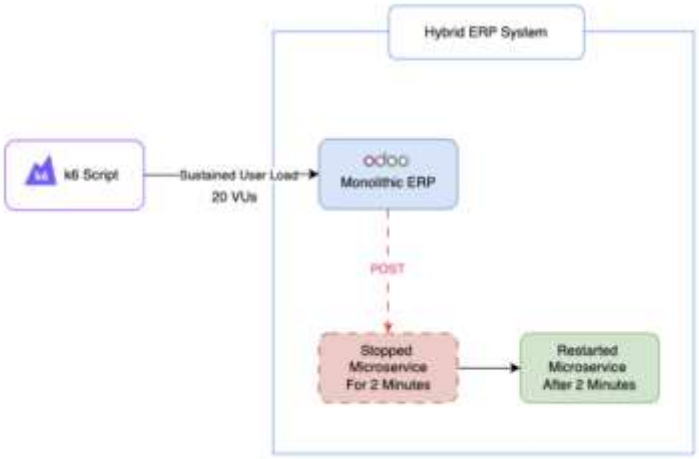

Fig. 5. Recoverability Scenario

4. Results and Discussion

The experimental data confirmed the hypothesis that targeted architectural interventions, informed by observability data, yield substantial improvements in hybrid system reliability. The comparative analysis of pre-test and post-test metrics validates the effectiveness of the proposed framework. [14].

Maturity: Managing Resource Contention. In the baseline test, the "Poison Pill" payload overwhelmed the system's synchronous processing model, leading to CPU spikes, database connection exhaustion, and a 7.80% failure rate with an average throughput of only 2.2 req/s. Result: The architecture was optimized by decoupling the request intake from the intensive business logic using an asynchronous worker pool and an in-memory queue. This allowed the system to absorb the large data payload at a controlled pace. Post-test results showed the failure rate plummeted to 0.00%, while average throughput increased by 104.5% to 4.5 req/s, and P_{95} latency improved by 42.9% (reduced to 6.66 seconds). [15].

Availability & Fault Tolerance: Eliminating Cascading Failures. The baseline architecture demonstrated a severe lack of fault tolerance. When the downstream microservice was intentionally delayed by 15 seconds, the monolith's worker threads waited indefinitely, causing the latency of unrelated, healthy endpoints to surge to 28.28 seconds, rendering the entire ERP functionally unavailable. Result: A request timeout pattern (configured to 5 seconds) was implemented on the monolith's API client. In the post-test, this "fail-fast" mechanism successfully released worker threads before they could be exhausted. Consequently, the healthy endpoint's latency remained under 1 second with a 0% failure rate, proving the fault was successfully isolated.

Recoverability: Enhancing Application-Level Resilience. During the simulated 2-minute service outage, the baseline system immediately passed infrastructure failures to the end-users, resulting in a severe 42.57% error rate. The infrastructure's Time to Recover (TTR) was measured at 24 seconds. Result: An automated retry pattern was built into the monolith's communication layer. While the infrastructure TTR naturally remained at 24 seconds, the retry logic effectively masked the transient downtime from the users. By patiently retrying failed requests, the system reduced the user-facing error rate by 88.2% (down to 5.02%), drastically improving perceived recoverability.

These findings demonstrate that hybrid ERP bottlenecks are rarely solved by simply adding more hardware; they require architectural realignment. The integration of asynchronous processing, proactive fault isolation (timeouts), and application-level resilience (retries) directly addresses the inherent vulnerabilities of distributed integration. The ISO/IEC 25010 standard, when paired with comprehensive observability, proved to be a highly effective operational tool rather than just a theoretical benchmark.

5. Conclusion

This research successfully established and validated a systematic framework for assessing and improving the reliability of hybrid monolithic-microservice systems. By operationalizing the ISO/IEC 25010 standard into targeted stress tests (Maturity, Availability, Fault Tolerance, and Recoverability), the study exposed critical vulnerabilities within a baseline Odoo-NestJS architecture, including resource exhaustion and cascading thread failures.

The pre-test/post-test methodology quantitatively validated that specific architectural patterns directly remedy these vulnerabilities. Implementing an asynchronous worker pool eliminated data-processing failures and doubled throughput; introducing request timeouts successfully isolated slow dependencies to maintain system-wide availability; and applying retry patterns reduced user-facing errors by 88% during transient outages. Ultimately, this research provides the software engineering industry with a practical,

data-driven blueprint for migrating legacy ERP systems, proving that with modern observability and targeted design patterns, complex hybrid architectures can achieve high reliability.

6. References

[1] M. Fowler and J. Lewis, “Microservices.” Martin Fowler, 2017.

[2] G. Blinowski, A. Ojdowska, and A. Przybyłek, “Monolithic vs. microservice architecture: A performance and scalability evaluation,” *IEEE Access*, vol. 10, pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[3] M. Chebrolu, “Building Scalable Applications: Transitioning from Monolithic to Microservices with Modularized Design,” *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 7, no. 3, 2025, doi: 10.56726/IRJMETS70797.

[4] J. Estdale and E. Georgiadou, “Applying the ISO/IEC 25010 quality models to software product,” in *European Conference on Software Process Improvement*, Springer, 2018, pp. 492–503. doi: 10.1007/978-3-319-97925-0_42.

[5] A. B. Raharjo, P. K. Andyartha, W. H. Wijaya, Y. Purwananto, D. Purwitasari, and N. Juniarta, “Reliability evaluation of microservices and monolithic architectures,” in *2022 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, IEEE, 2022, pp. 1–7. doi: 10.1109/CENIM56801.2022.10037281.

[6] O. Al-Debagy and P. Martinek, “A comparative review of microservices and monolithic architectures,” in *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, IEEE, 2018, pp. 000149–000154. doi: 10.1109/CINTI.2018.8928192.

[7] D. S. H. Tam, Y. Liu, H. Xu, S. Xie, and W. C. Lau, “Pert-gnn: Latency prediction for microservice-based cloud-native applications via graph neural networks,” in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 2155–2165. doi: 10.1145/3580305.3599465.

[8] M. Levezinho, S. Kapferer, O. Zimmermann, and A. R. Silva, “Domain-Driven Design Representation of Monolith Candidate Decompositions Based on Entity Accesses,” *ArXiv Prepr. ArXiv240702512*, 2024, doi: 10.1007/978-3-031-78338-8_10.

[9] A. Correia Jr *et al.*, “GORDA: An open architecture for database replication,” in *Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007)*, IEEE, 2007, pp. 287–290. doi: 10.1109/NCA.2007.26.

[10] R. Su, X. Li, and D. Taibi, “From microservice to monolith: a multivocal literature review,” *Electronics*, vol. 13, no. 8, p. 1452, 2024, doi: 10.3390/electronics13081452.

[11] R. Maharjan, M. S. H. Chy, M. A. Arju, and T. Cerny, “Benchmarking message queues,” in *Telecom*, MDPI, 2023, pp. 298–312. doi: 10.3390/telecom4020018.

[12] R. Reagan, “Message Queues,” in *Web Applications on Azure: Developing for Global Scale*, Springer, 2017, pp. 343–380.

[13] R. Ewaschuk and B. Beyer, *Monitoring distributed systems*. O’Reilly Media, Incorporated, 2016.

[14] H. T. Tran, M. Balchanos, J. C. Domerçant, and D. N. Mavris, “A framework for the quantitative assessment of performance-based system resilience,” *Reliab. Eng. Syst. Saf.*, vol. 158, pp. 73–84, 2017, doi: 10.1016/j.ress.2016.10.014.

[15] S. Weerasinghe and I. Perera, “Optimized strategy for inter-service communication in microservices,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 2, 2023, doi: 10.1109/ICITR57877.2022.9992918.