American Journal of Applied Science and Technology

VOLUME Vol.05 Issue 11 2025
PAGE NO. 283-287

Architecting Secure and Sustainable Enterprise Java Delivery Pipelines: Managing Mixed Java Versions, Legacy Modularity, and DevSecOps Automation in Non-Containerized CI/CD Environments

Dr. Alexander M. Rothwell

Department of Computer Science, Westbridge University, United Kingdom

Received: 10 September 2025; **Accepted:** 02 October 2025; **Published:** 05 November 2025

Abstract: Enterprise Java ecosystems continue to operate under complex constraints shaped by long-lived legacy systems, mixed Java version dependencies, stringent regulatory requirements, and cautious adoption of containerization technologies. While cloud-native and container-based paradigms dominate contemporary discourse, a substantial proportion of mission-critical enterprise systems still rely on non-containerized continuous integration and continuous delivery (CI/CD) pipelines. These environments face unique challenges, particularly when managing multiple Java versions, modularization transitions, dependency risk propagation, and integrated security controls without disrupting operational stability. This research presents an in-depth, theory-driven examination of enterprise-grade CI/CD pipeline architectures for mixed Java version environments using Jenkins in non-containerized contexts. Drawing exclusively from established academic, industry, and standards-based references, the study synthesizes insights from Java platform evolution, modularity research, garbage collection optimization, dependency management, and DevSecOps governance frameworks. The methodology employs a qualitative analytical approach, integrating comparative literature analysis, architectural reasoning, and process-oriented interpretation to derive best-practice patterns. The findings reveal that sustainable pipeline design in non-containerized environments depends on deliberate version isolation strategies, disciplined dependency governance, modular refactoring aligned with Java Platform Module System principles, and deeply embedded security automation. The discussion critically evaluates trade-offs between modernization and operational risk, highlighting how policy-as-code, static and dynamic analysis tools, and compliance-driven observability can be harmonized within Jenkins-centric pipelines. The study concludes that non-containerized CI/CD architectures, when systematically engineered, remain viable and strategically relevant, offering a pragmatic modernization pathway for enterprises balancing innovation with legacy continuity.

Keywords: Enterprise Java, CI/CD pipelines, Jenkins, DevSecOps, Java modularity, dependency management

INTRODUCTION

The Java programming language and its surrounding ecosystem have remained foundational to enterprise software development for over two decades. Despite waves of technological transformation driven by cloud computing, microservices, and container orchestration platforms, Java continues to underpin a vast array of mission-critical systems in finance, telecommunications, healthcare, and government sectors. These systems are often characterized by long operational lifespans, complex dependency structures, and conservative change management practices shaped by regulatory and risk considerations (Gupta & Saxena, 2020). As a result, many enterprises operate heterogeneous Java environments where multiple Java versions coexist, ranging from legacy long-term support releases to more recent iterations incorporating modern language and runtime features (Venkat & Saito, 2022).

Continuous integration and continuous delivery practices have become central to maintaining software quality and delivery velocity in such environments. However, the dominant narratives around CI/CD frequently assume containerized deployments and cloud-native infrastructures. This assumption marginalizes a significant portion of enterprise reality, where non-containerized pipelines remain prevalent due to operational inertia, compliance constraints, performance predictability requirements, or the prohibitive cost of large-scale architectural rewrites (Tomlinson, 2021). Jenkins, as a widely adopted automation server, continues to serve as the backbone of CI/CD processes in these contexts, offering extensibility and flexibility that align with enterprise governance models (Jenkins, 2023; Jenkins Project, 2024).

Managing mixed Java versions within a single CI/CD pipeline introduces multifaceted challenges. Build tools, runtime behaviors, garbage collection mechanisms, and module systems vary across Java releases, creating subtle compatibility risks that can manifest as performance degradation, security vulnerabilities, or deployment failures (Chen & Thakkar, 2021; OpenJDK, 2021). The introduction of the Java Platform Module System (JPMS) in Java 9 further complicated the landscape by imposing explicit modular boundaries, exposing architectural weaknesses in legacy codebases that evolved under classpath-based assumptions (Deligiannis, Smaragdakis, & Chatrchyan, 2019).

Simultaneously, the growing emphasis on DevSecOps demands the seamless integration of security controls throughout the software delivery lifecycle. Enterprises are increasingly expected to embed vulnerability scanning, dependency analysis, policy enforcement, and compliance verification directly into CI/CD pipelines without compromising development flow (Mehta, 2022). Tools such as Trivy, OWASP Dependency-Check, SonarQube, and policy-as-code frameworks exemplify this shift, yet their effective orchestration within non-containerized, mixed-version Java pipelines remains underexplored in academic literature (Aqua Security, 2023; OWASP Foundation, 2023; SonarSource, 2024; Open Policy Agent, 2023).

This research addresses this gap by providing a comprehensive, theoretically grounded exploration of enterprise-grade CI/CD pipelines for mixed Java version environments operating without containers. The study synthesizes findings from empirical software engineering research, Java platform evolution studies, dependency risk analyses, and security governance frameworks to construct an integrated perspective on sustainable pipeline design. By focusing on Jenkins-centric architectures and emphasizing modularity, dependency management, and DevSecOps integration, the article contributes a nuanced understanding of how enterprises can modernize incrementally while preserving operational stability.

METHODOLOGY

The methodological approach adopted in this study is qualitative and analytical, grounded in an extensive synthesis of peer-reviewed academic literature, industry guides, standards documentation, and authoritative tool documentation. Given the complexity and socio-technical nature of enterprise CI/CD systems, a purely quantitative or experimental methodology would inadequately capture the architectural, organizational, and governance dimensions involved. Instead, the research employs an interpretive framework that integrates theoretical elaboration with comparative analysis across multiple sources.

The first methodological step involved categorizing the provided references into thematic domains, including Java platform evolution, modularity and legacy system transformation, CI/CD pipeline architecture, dependency and risk management, and DevSecOps security integration. This thematic classification enabled a structured analysis of how individual concepts intersect and influence pipeline design decisions. For example, studies on garbage collection optimization and modularization were analyzed not in isolation but in relation to build reproducibility and runtime consistency across Java versions (Chen & Thakkar, 2021; Deligiannis et al., 2021).

The second step focused on architectural reasoning, wherein concepts from Jenkins pipeline design were examined through the lens of non-containerized execution environments. Jenkins documentation and empirical studies on legacy CI/CD practices were used to infer patterns of toolchain orchestration, environment isolation, and version management (Kathi, 2025; Jenkins Project, 2024). This reasoning was further enriched by examining how policy-as-code and security scanning tools could be embedded into pipeline stages without introducing excessive friction or false positives (Open Policy Agent, 2023; SonarSource, 2023).

The third methodological component involved comparative synthesis. Contrasting perspectives from empirical studies on Java modularity adoption were analyzed to identify recurring challenges, such as split packages, reflective access violations, and dependency graph instability (Deligiannis, Spinellis, & Gousios, 2022). These challenges were then mapped to CI/CD implications, particularly in mixed-version pipelines where different modules may target different Java baselines.

Throughout the analysis, strict adherence to the provided references was maintained. All claims and interpretations were explicitly grounded in cited sources, and no external assumptions or undocumented practices were introduced. The methodology emphasizes depth over breadth, prioritizing exhaustive theoretical elaboration to illuminate the nuanced trade-offs inherent in enterprise pipeline design.

RESULTS

The analysis yields several interrelated findings that collectively illuminate the dynamics of enterprise-grade CI/CD pipelines in mixed Java version, non-containerized environments. One of the most prominent results is the centrality of deliberate version isolation strategies. Enterprises that successfully manage mixed Java versions do not rely on ad hoc configuration but instead implement structured mechanisms for selecting, validating, and enforcing Java runtimes at each pipeline stage (Kathi, 2025). Jenkins facilitates this through toolchains and environment directives, allowing builds to target specific Java versions while sharing a common automation framework (Jenkins, 2023).

Another significant finding concerns the role of modularity in stabilizing pipeline behavior. Empirical studies consistently show that legacy Java systems often exhibit weak modular boundaries, leading to hidden dependencies and brittle builds when transitioning to newer Java versions (Deligiannis et al., 2021). The adoption of JPMS, while initially disruptive, ultimately enhances pipeline predictability by making dependencies explicit and enabling more precise impact analysis during upgrades (Deligiannis et al., 2019). Pipelines that incorporate module-aware compilation and testing stages are better equipped to detect incompatibilities early, reducing downstream failures.

Dependency management emerges as a critical determinant of pipeline security and reliability. Research on transitive dependency risks demonstrates that vulnerabilities and breaking changes often propagate indirectly through dependency trees, complicating upgrade decisions (Shah et al., 2020; Shah, Reddy, & Ma, 2022). The results indicate that pipelines integrating automated dependency analysis tools, such as OWASP Dependency-Check and Snyk-style vulnerability assessments, significantly improve risk visibility (OWASP Foundation, 2023; Snyk Ltd., 2023). When combined with policy-as-code enforcement, these tools enable consistent governance across projects and Java versions (Open Policy Agent, 2023).

From a performance perspective, garbage collection behavior remains a nontrivial concern in mixed-version environments. Different Java releases introduce distinct garbage collectors and tuning options, which can influence application latency and throughput (Chen & Thakkar, 2021). Pipelines that include microbenchmarking and performance regression checks, informed by tools like the Java Microbenchmark Harness, are better positioned to detect runtime regressions introduced by version changes (Oracle Corporation, 2022).

Security integration results highlight the feasibility of comprehensive DevSecOps practices even in non-containerized pipelines. Static analysis tools, dynamic testing frameworks, and runtime observability platforms can be orchestrated within Jenkins stages to provide layered security assurance (SonarSource, 2024; OWASP Foundation, 2023). Importantly, the findings suggest that non-containerized environments do not inherently preclude advanced security automation; rather, success depends on disciplined pipeline design and clear governance models (Mehta, 2022).

DISCUSSION

The results underscore the enduring relevance of non-containerized CI/CD pipelines in enterprise Java ecosystems. While containerization offers undeniable benefits in terms of portability and isolation, the assumption that it is a prerequisite for modern DevSecOps is not supported by empirical evidence. Instead, the analysis reveals that architectural intentionality and governance discipline are more decisive factors than deployment substrate.

One of the key interpretive insights concerns the tension between stability and modernization. Enterprises operating mixed Java versions often do so not out of neglect but as a rational response to risk and cost considerations (Gupta & Saxena, 2020). The gradual adoption of newer Java features, guided by empirical understanding of language and runtime evolution, allows organizations to balance innovation with reliability (Venkat & Saito, 2022). CI/CD pipelines serve as the mediating infrastructure through which this balance is negotiated.

Modularity emerges as both a technical and organizational construct. While JPMS provides formal mechanisms for encapsulation, its successful adoption requires cultural shifts in how teams reason about dependencies and ownership (Deligiannis et al., 2022). Pipelines that enforce modular boundaries through automated checks effectively institutionalize these shifts, embedding architectural discipline into everyday workflows.

The discussion also highlights the strategic importance of dependency governance. Transitive risks challenge traditional notions of control, as vulnerabilities may originate far from the code directly maintained by a team (Shah et al., 2020). Policy-as-code frameworks offer a compelling response by translating organizational risk tolerance into executable rules that operate consistently across pipelines (Open Policy Agent, 2023). This alignment between policy and automation is particularly valuable in regulated environments subject to standards such as PCI DSS (PCI Security Standards Council, 2022).

Limitations of the study include its reliance on secondary sources and theoretical synthesis rather than primary empirical data collection. While this approach enables broad integration of existing knowledge, it may overlook context-specific nuances present in individual organizations. Future research could complement this work with longitudinal case studies examining real-world pipeline transformations.

CONCLUSION

This research demonstrates that enterprise-grade CI/CD pipelines for mixed Java version environments can achieve high levels of security, reliability, and adaptability without relying on containerization. By synthesizing insights from Java platform evolution, modularity research, dependency risk analysis, and DevSecOps frameworks, the study provides a comprehensive theoretical foundation for pipeline design in legacy-conscious enterprises. Jenkins-based automation, when combined with disciplined version management, modular enforcement, and integrated security controls, remains a viable and strategically sound approach. As enterprises continue to navigate the complexities of modernization, the principles articulated here offer a pragmatic pathway for aligning technical evolution with organizational realities.

REFERENCES

1. Aqua Security. (2023). Trivy open source vulnerability scanner.
2. Chen, L., & Thakkar, M. (2021). Garbage collection optimization in large-scale Java applications. Proceedings of the IEEE International Conference on Software Maintenance and Evolution.
3. Deligiannis, I., et al. (2021). Challenges in modularizing legacy Java systems: An empirical study. Empirical Software Engineering, 26(2), 25.
4. Deligiannis, N., Smaragdakis, Y., & Chatrchyan, S. (2019). Migrating to Java 9 modules: Lessons from the trenches. Proceedings of the ACM on Programming Languages, 3(OOPSLA), 1–25.
5. Deligiannis, N., Spinellis, D., & Gousios, G. (2022). Analyzing modularity in Java projects after JPMS adoption. Empirical Software Engineering Journal, 27(1), 1–29.
6. Gupta, M., & Saxena, A. (2020). An empirical study of Java LTS versions in enterprise software systems. Journal of Software Engineering and Applications, 13(8), 325–337.
7. Jenkins. (2023). Pipeline syntax and tools. Jenkins documentation.
8. Jenkins Project. (2024). Jenkins documentation: Pipeline and plugin ecosystem.
9. Kathi, S. R. (2025). Enterprise-grade CI/CD pipelines for mixed Java version environments using Jenkins in non-containerized environments. Journal of Engineering Research and Sciences, 4(9), 12–21. https://doi.org/10.55708/js0409002
10. Malhotra, S. (2021). Dependency management for Java frameworks: The case of Spring and Jersey. International Journal of Software Engineering & Applications, 12(4), 45–57.
11. Mehta, N. (2022). DevSecOps: A leader’s guide to producing secure software without compromising flow, feedback, and continuous improvement. IT Revolution.

12. OpenJDK. (2021). JEP index.

13. Open Policy Agent. (2023). Policy as code for secure CI/CD.

14. Oracle. (2021). Java SE support roadmap.

15. Oracle. (2023). Java SE support roadmap.

16. Oracle Corporation. (2021). CLDR in JDK 9 and later (JEP 252).

17. Oracle Corporation. (2022). Java microbenchmark harness.

18. OWASP Foundation. (2023). OWASP dependency-check.

19. OWASP Foundation. (2023). OWASP ZAP project.

20. PCI Security Standards Council. (2022). Payment card industry data security standard v4.0.

21. Shah, A., et al. (2020). Risks in transitive dependency upgrades in Java projects. Proceedings of the IEEE International Conference on Software Maintenance and Evolution, 27–36.

22. Shah, P., Reddy, A., & Ma, J. (2022). Risk propagation in Java dependency trees: A transitive analysis approach. Software: Practice and Experience, 52(9), 1754–1772.

23. SonarSource. (2023). Static analysis for Java applications.

24. SonarSource. (2024). SonarQube documentation.

25. Snyk Ltd. (2023). State of Java security report.

26. Tomlinson, B. (2021). CI/CD without containers: Lessons from legacy environments. Proceedings of the DevOps Enterprise Summit.

27. Venkat, G., & Saito, T. (2022). Modern Java language features: From Java 9 to Java 17. Java Magazine, Oracle.

28. Splunk Inc. (2023). Security information and event management best practices.