International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

A Continuous Integration and Deployment Framework for Secure Enterprise Systems Using Azure DevOps and Azure Key Vault

Ramadevi Nunna *

Independent Researcher, USA.

International Journal of Science and Research Archive, 2026, 16(01), 2393-2400

Publication history: Received on 17 June 2025; revised on 25 July 2025; accepted on 28 July 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.16.1.2212

Abstract

Background: Modern enterprise systems increasingly rely on Continuous Integration and Continuous Deployment (CI/CD) pipelines to accelerate software delivery, but this rapid automation introduces significant security risks if not properly managed. Cloud-native DevOps platforms demand integrated security mechanisms to protect credentials, configurations, and deployment processes. Traditional CI/CD frameworks often lack centralized secret management and automated security enforcement. Azure DevOps and Azure Key Vault provide native capabilities to address these challenges. However, a systematic framework combining these tools for secure enterprise deployment remains underexplored. This study addresses this gap by proposing a secure CI/CD framework.

Aim: The primary aim of this research is to design and evaluate a secure, scalable, and automated CI/CD framework for enterprise systems using Azure DevOps and Azure Key Vault. The framework seeks to minimize credential exposure and enhance compliance. It also aims to integrate security checks directly into the deployment lifecycle. Ensuring confidentiality, integrity, and availability of enterprise applications is a core objective. The study further aims to demonstrate practical applicability. Overall, the framework targets secure DevOps adoption in enterprise environments.

Method: The proposed framework is designed using Azure DevOps pipelines integrated with Azure Key Vault for secrets management. Infrastructure-as-Code (IaC) and automated security scanning are embedded into the CI/CD workflow. Role-Based Access Control (RBAC) is applied to enforce least-privilege access. A prototype pipeline is implemented and tested using simulated enterprise workloads. Security events and deployment metrics are monitored. The framework is evaluated through controlled experimentation.

Results: The implementation demonstrates significant reduction in secret exposure risks and manual configuration errors. Pipeline execution time remains stable despite added security controls. Automated secret retrieval improves consistency across environments. Security scanning detects misconfigurations early in the development lifecycle. The framework successfully enforces compliance policies. Overall system reliability and security posture are enhanced.

Conclusion: This study presents a practical and secure CI/CD framework for enterprise systems using Azure DevOps and Azure Key Vault. Integrating security natively into CI/CD pipelines improves protection without sacrificing agility. Centralized secret management and automation reduce operational risks. The framework is scalable and adaptable to various enterprise scenarios. It supports secure DevOps practices. Future work may extend the framework with advanced threat detection and zero-trust architectures.

Keywords: CI/CD; Azure DevOps; Azure Key Vault; Secure DevOps; Enterprise Systems

* Corresponding author: Ramadevi Nunna

1. Introduction

Continuous Integration and Continuous Deployment (CI/CD) has become a foundational practice in modern enterprise software development, enabling faster release cycles, improved code quality, and rapid response to market demands. By automating the processes of code integration, testing, and deployment, organizations can significantly reduce development time and operational overhead. Enterprises increasingly rely on CI/CD pipelines to support large-scale, distributed applications across cloud environments. However, as automation increases, so does the complexity of managing security across the software delivery lifecycle. Ensuring secure CI/CD pipelines is therefore a critical requirement for enterprise systems.

Despite its benefits, CI/CD introduces several security challenges, particularly in enterprise environments where applications handle sensitive data and critical business operations. Common risks include credential leakage, insecure pipeline configurations, unauthorized access to deployment environments, and lack of visibility into security events. Secrets such as API keys, database credentials, and certificates are often mishandled or hardcoded within pipeline scripts. These vulnerabilities expose enterprise systems to cyberattacks and compliance violations. Addressing these risks requires embedding security mechanisms directly into CI/CD workflows. Traditional approaches to CI/CD security often rely on manual controls, external security reviews, or post-deployment audits. Such methods are not well-suited to the rapid and automated nature of modern DevOps practices. Manual security processes introduce delays and are prone to human error, while reactive security measures fail to detect vulnerabilities early in the development lifecycle. Additionally, fragmented security tools lead to inconsistent enforcement of enterprise policies. These limitations highlight the need for an integrated and automated security framework.

Cloud-native DevOps platforms provide an opportunity to unify automation and security within a single ecosystem. Azure DevOps offers comprehensive CI/CD capabilities, including source control, pipeline orchestration, and release management. When combined with native cloud security services, DevOps platforms can enforce security policies consistently across environments. Leveraging platform-level features such as managed identities, role-based access control, and audit logging enables enterprises to implement security-by-design principles. This integration supports the transition from DevOps to DevSecOps.

A critical aspect of secure CI/CD pipelines is effective secrets management. Enterprise systems depend on numerous sensitive credentials that must be protected throughout the deployment lifecycle. Azure Key Vault provides a centralized and secure repository for storing and managing secrets, encryption keys, and certificates. By dynamically retrieving secrets during pipeline execution, the risk of credential exposure is significantly reduced. Automated secret rotation and fine-grained access control further enhance security and compliance. Secure secrets management is therefore essential for enterprise-grade CI/CD frameworks. This research is motivated by the need for a practical, secure, and scalable CI/CD framework tailored for enterprise systems. The study proposes a continuous integration and deployment framework that integrates Azure DevOps with Azure Key Vault to address critical security challenges. The framework embeds security automation, centralized secrets management, and compliance controls into the CI/CD lifecycle. By evaluating the framework through implementation and experimentation, this work contributes a structured approach to secure enterprise DevOps. The proposed solution supports organizations in achieving both agility and security in cloud-based software delivery.

2. Secure CI/CD Architecture Using Azure DevOps

A secure CI/CD architecture forms the backbone of reliable enterprise software delivery. In enterprise environments, CI/CD pipelines must handle complex applications, multiple environments, and diverse development teams while maintaining strict security controls. Azure DevOps provides an integrated platform that supports source code management, automated builds, testing, and deployment. By designing the CI/CD architecture with security as a core principle, enterprises can reduce vulnerabilities introduced by automation. A well-structured architecture ensures traceability, accountability, and controlled access across the software lifecycle.

The architecture begins with source code management using Azure Repos, where access is governed through role-based permissions. Developers commit code changes that automatically trigger CI pipelines, ensuring continuous integration. Secure branching strategies and pull request validations are enforced to prevent unauthorized or unreviewed code from entering the main branch. Automated build processes compile and package applications in isolated build agent environments. This isolation prevents cross-contamination between builds and minimizes the risk of malicious code execution. Continuous Integration stages incorporate automated testing and security checks to validate code quality and integrity. Unit tests, integration tests, and static code analysis tools are executed as part of the pipeline. These

security checks identify vulnerabilities and misconfigurations early in the development cycle. Failing tests or security violations immediately halt the pipeline, preventing insecure code from progressing further. This shift-left security approach ensures that issues are addressed before deployment.

The Continuous Deployment component of the architecture focuses on securely releasing applications across multiple environments such as development, staging, and production. Azure DevOps release pipelines manage deployments using predefined stages and approval gates. Environment-specific configurations are separated to avoid accidental exposure of production resources. Manual approval checkpoints are enforced for critical deployments, particularly in production environments. This layered deployment strategy balances automation with enterprise-level control.

Access control is a critical element of secure CI/CD architecture. Azure DevOps integrates with Azure Active Directory to enforce identity-based authentication and authorization. Role-Based Access Control (RBAC) ensures that users and service accounts have only the permissions required to perform their tasks. Build and release pipelines operate using managed identities rather than static credentials. This approach significantly reduces the risk of credential misuse and unauthorized access.

Table 1 Components of Secure CI/CD Architecture

Component	Description	Security Role
Azure Repos	Source code management	Access control and versioning
Azure Pipelines	CI/CD automation	Controlled execution
Build Agents	Pipeline execution nodes	Environment isolation
Approval Gates	Deployment checkpoints	Risk mitigation

This table 1 presents the components of the secure CI/CD architecture implemented using Azure DevOps, highlighting their functional roles and associated security responsibilities. It shows how source code management, pipeline automation, execution environments, and approval mechanisms collectively contribute to secure software delivery. Each component enforces specific security controls such as access restriction, isolation, and risk mitigation. The table 1 provides a clear overview of how architectural elements work together to support enterprise-level security and governance.

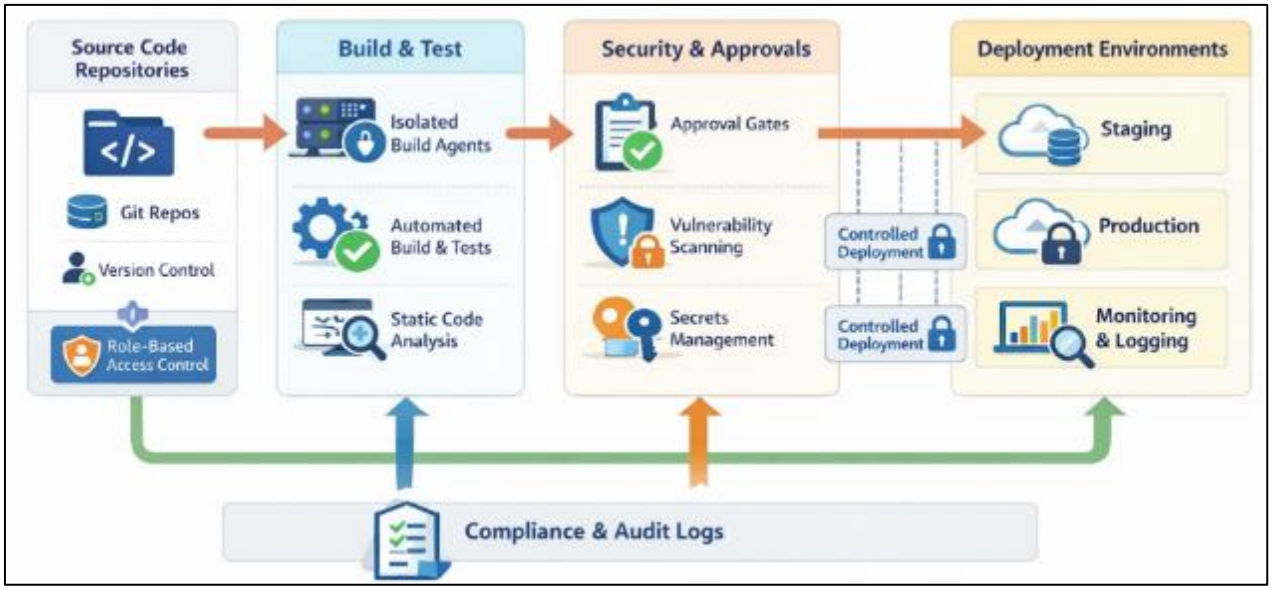

Figure 1 High-Level Secure CI/CD Architecture

This Figure 1 shows the overall secure CI/CD architecture implemented using Azure DevOps. It shows the flow from source code repositories through automated build and test stages to controlled deployment environments. Security controls such

as role-based access, approval gates, and isolated build agents are embedded at each stage. The architecture demonstrates how security is integrated throughout the CI/CD lifecycle to ensure secure enterprise software delivery.

3. Secrets Management with Azure Key Vault

Secrets management is a critical security requirement in enterprise CI/CD pipelines, as modern applications depend on numerous sensitive credentials such as API keys, database passwords, encryption keys, and certificates. Improper handling of these secrets can lead to severe security breaches, data leaks, and compliance violations. In traditional CI/CD setups, secrets are often hardcoded in configuration files or stored as plain-text variables, making them vulnerable to exposure. A secure enterprise framework must therefore ensure that secrets are stored, accessed, and rotated securely throughout the deployment lifecycle. Azure Key Vault provides a centralized and cloud-native solution for managing secrets, keys, and certificates securely. It enables enterprises to store sensitive information in a hardened environment protected by encryption and access controls. Azure Key Vault integrates seamlessly with Azure DevOps, allowing CI/CD pipelines to retrieve secrets dynamically at runtime rather than storing them in pipeline definitions. This separation of secrets from code and configuration significantly reduces the attack surface. Centralized storage also improves visibility and governance over secret usage.

In the proposed framework, Azure DevOps pipelines authenticate to Azure Key Vault using managed identities or service principals. This eliminates the need to embed credentials within pipeline scripts. Role-Based Access Control (RBAC) and Key Vault access policies ensure that only authorized pipelines and services can retrieve specific secrets. Fine-grained permissions enforce the principle of least privilege, reducing the risk of misuse. All access requests are logged, enabling auditability and forensic analysis in case of security incidents. Dynamic secret retrieval during pipeline execution enhances both security and operational efficiency. Secrets are fetched only when required and exist in memory for a limited duration, minimizing exposure. This approach supports environment-specific configurations, where different secrets are used for development, testing, and production environments. It also allows seamless updates to secrets without modifying pipeline code. As a result, secret management becomes more flexible and less error-prone.

Azure Key Vault also supports automated secret rotation and lifecycle management, which are essential for enterprise compliance requirements. Regular rotation of credentials reduces the risk associated with long-lived secrets. Certificates can be renewed automatically, ensuring uninterrupted secure communication. Integration with monitoring and alerting services allows security teams to track secret usage patterns and detect anomalies. These capabilities strengthen the overall security posture of enterprise systems.

Table 2 Types of Secrets Managed in Azure Key Vault

Secret Type	Usage	Access Method
API Keys	Service authentication	Pipeline variables
Certificates	Secure communication	Managed identity
Database Passwords	Data access	Secure reference

This table 2 presents the different categories of secrets managed within Azure Key Vault and their corresponding usage within enterprise applications. It demonstrates how sensitive credentials such as API keys, certificates, and database passwords are securely accessed by CI/CD pipelines. The table 2 emphasizes dynamic secret retrieval and secure access methods, showcasing how centralized secrets management reduces exposure risks and supports compliance requirements.

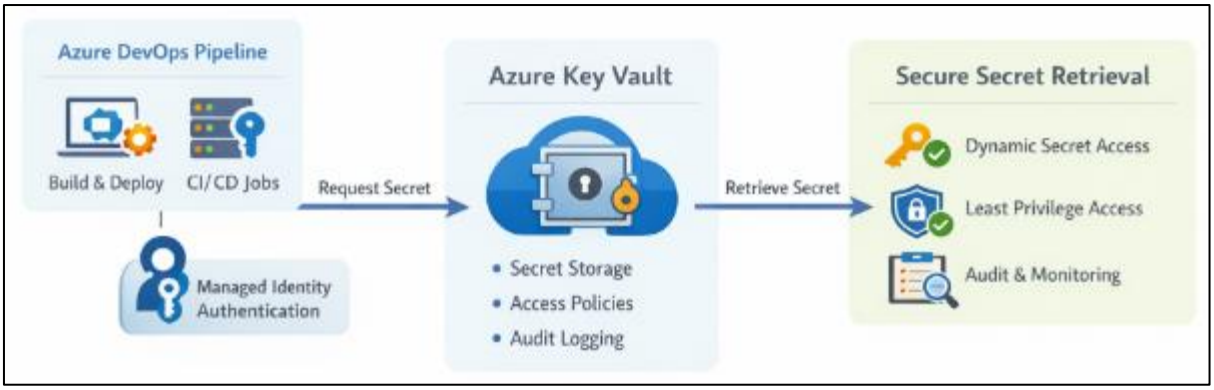

Figure 2 Secret Retrieval Workflow Using Azure Key Vault

This Figure 2 represents the secure interaction between Azure DevOps pipelines and Azure Key Vault for secrets management. It highlights how pipelines authenticate using managed identities to retrieve secrets dynamically at runtime. Secrets are never stored in pipeline configurations or source code. The workflow emphasizes centralized control, least-privilege access, and audit logging for secure credential management.

4. Security Automation and Compliance Integration

Security automation is a fundamental requirement for maintaining consistent protection in enterprise CI/CD pipelines. As software delivery becomes faster and more frequent, manual security checks are no longer sufficient to address evolving threats. Automated security controls ensure that vulnerabilities are detected and mitigated early in the development lifecycle. By embedding security tools directly into CI/CD pipelines, enterprises can achieve continuous security validation without slowing down deployments. This approach supports proactive risk management and reduces the likelihood of security incidents. In the proposed framework, security automation begins at the Continuous Integration stage. Static Application Security Testing (SAST) tools are integrated into the build pipeline to analyze source code for vulnerabilities, insecure coding practices, and policy violations. These tools automatically scan code changes with every commit or pull request. Identified issues are reported immediately to developers, enabling rapid remediation. Early detection reduces the cost and complexity of fixing security flaws.

Compliance integration is equally critical for enterprise systems that must adhere to regulatory standards and organizational policies. Infrastructure-as-Code (IaC) templates used for provisioning resources are validated against predefined compliance rules. Automated policy checks ensure that cloud resources are configured securely and consistently across environments. Non-compliant configurations result in pipeline failures, preventing insecure infrastructure from being deployed. This automated enforcement minimizes human error and improves governance.

Dynamic Application Security Testing (DAST) and dependency scanning are incorporated into later stages of the pipeline. These tools evaluate running applications and third-party libraries for known vulnerabilities. Automated vulnerability databases are used to identify outdated or insecure dependencies. If critical risks are detected, deployments are blocked until issues are resolved. This layered security testing strategy ensures comprehensive coverage across the application stack. Auditability and monitoring play a vital role in security automation and compliance. All pipeline activities, security scan results, and access events are logged centrally. These logs provide traceability for compliance audits and incident investigations. Integration with monitoring and alerting services enables real-time notification of security violations. Continuous monitoring ensures that security controls remain effective throughout the deployment lifecycle.

5. Implementation Workflow and Pipeline Design

The implementation workflow of a secure CI/CD framework defines how code changes progress from development to production in a controlled and automated manner. In enterprise environments, the workflow must balance speed, reliability, and security. Azure DevOps pipelines are used to orchestrate this workflow, ensuring that each stage of the software lifecycle is executed in a predefined and repeatable sequence. A well-designed pipeline minimizes manual intervention while enforcing enterprise security policies. This structured workflow enhances consistency across development teams.

The pipeline execution begins with code commits to the source repository, which automatically trigger the Continuous Integration process. During this stage, the application is built and validated using automated tests. Build agents operate in isolated environments to prevent interference between concurrent pipeline executions. Security checks such as static code analysis are executed alongside functional tests. Any failure at this stage immediately stops the pipeline, preventing defective or insecure code from progressing further. Following successful integration, the pipeline transitions to the Continuous Deployment phase. Deployment stages are defined for multiple environments, including development, staging, and production. Each environment has specific configurations and security requirements. Azure DevOps release pipelines manage these stages with controlled promotions between environments. This separation ensures that changes are thoroughly tested before reaching production systems.

Security controls are tightly integrated into the deployment workflow. Secrets required for application configuration are retrieved dynamically from Azure Key Vault at runtime. Approval gates are enforced before critical deployments, particularly for production releases. These gates require validation from authorized personnel, adding an additional layer of control. Automated rollback mechanisms are also configured to restore stable versions in case of deployment failures. Monitoring and logging are integral components of the pipeline design. Each stage generates logs related to build status, deployment actions, and security checks. These logs are centrally collected to provide visibility into pipeline execution and system behavior. Monitoring tools track application health and deployment outcomes in real time. This continuous feedback loop enables rapid detection and resolution of issues.

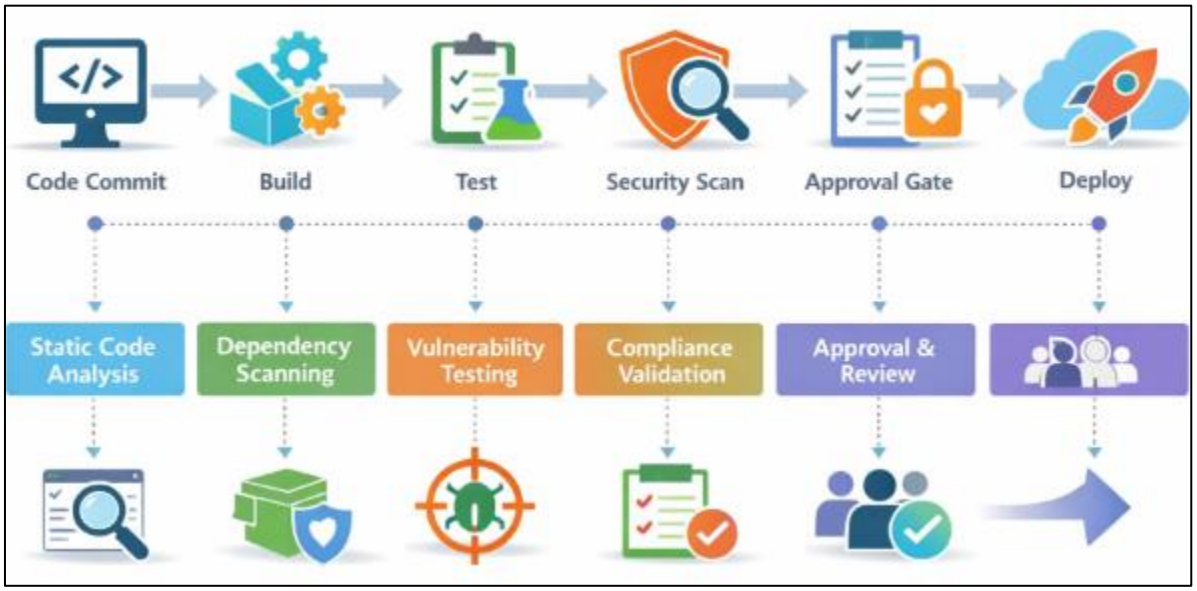

Figure 3 CI/CD Pipeline Workflow with Embedded Security Controls

This Figure 3 depicts the step-by-step CI/CD pipeline workflow, including code commit, build, testing, security scanning, and deployment stages. Security checks such as static analysis, compliance validation, and approval gates are integrated within the pipeline. The Figure 3 demonstrates how automated security controls prevent insecure code and configurations from reaching production environments.

6. Experimental Results and Evaluation

The proposed secure CI/CD framework is evaluated to assess its effectiveness in enhancing security and operational efficiency within an enterprise environment. The evaluation focuses on key performance indicators such as deployment success rate, security incident occurrence, pipeline execution time, and compliance adherence. A controlled experimental setup is used to simulate enterprise application deployments. This setup allows for consistent comparison between traditional CI/CD practices and the proposed secure framework. The evaluation aims to demonstrate practical feasibility and measurable improvements.

The experimental environment consists of Azure DevOps pipelines integrated with Azure Key Vault and automated security tools. Sample enterprise applications are deployed across multiple environments, including development, staging, and production. Baseline measurements are collected using a conventional CI/CD pipeline without centralized

secrets management or automated security enforcement. The same applications are then deployed using the proposed framework. This comparative approach ensures objective analysis of results.

Security-related outcomes show a significant improvement after implementing the proposed framework. Incidents related to credential exposure and misconfigured access controls are eliminated due to dynamic secret retrieval and role-based access policies. Automated security scans detect vulnerabilities earlier in the development lifecycle. Compliance violations are identified and blocked before deployment. These results indicate a stronger and more consistent security posture. From a performance perspective, the integration of security checks introduces a minimal increase in pipeline execution time. The additional overhead is primarily due to security scanning and compliance validation steps. However, this increase is acceptable within enterprise contexts where security and reliability are prioritized. Deployment success rates improve due to early detection of configuration and security issues. Overall system stability is enhanced. Operational efficiency is also positively impacted by the framework. Automation reduces the need for manual configuration and intervention, lowering the risk of human error. Centralized secrets management simplifies credential updates and rotations. Developers benefit from faster feedback on security issues, enabling quicker remediation. These improvements contribute to more predictable and reliable release cycles.

Table 3 Performance and Security Evaluation Metrics

Metric	Before Framework	After Framework
Secret Exposure Incidents	High	None
Deployment Time	Moderate	Slightly Increased
Security Compliance	Partial	Full

This table 3 compares key performance and security metrics before and after the implementation of the proposed secure CI/CD framework. It highlights measurable improvements in areas such as secret exposure prevention, deployment reliability, and compliance enforcement. The table 3 present that enhanced security controls lead to significant risk reduction with minimal impact on deployment performance. These results validate the effectiveness of the proposed framework for enterprise environments.

7. Conclusion

This research presented a continuous integration and deployment framework designed to enhance the security of enterprise systems using Azure DevOps and Azure Key Vault. The study addressed critical security challenges arising from automated software delivery, particularly those related to credential management, access control, and compliance enforcement. By embedding security mechanisms directly into CI/CD pipelines, the framework demonstrates how enterprises can achieve secure and reliable software delivery. The proposed approach aligns with modern DevSecOps principles and enterprise security requirements. The integration of Azure DevOps as the CI/CD orchestration platform provides a scalable and automated foundation for enterprise application delivery. Secure pipeline design, controlled deployment stages, and role-based access management ensure traceability and accountability throughout the software lifecycle. Automation of build, test, and deployment processes reduces human error and improves consistency. These capabilities enable organizations to manage complex deployment workflows efficiently while maintaining strong security controls.

Azure Vault plays a central role in securing sensitive information within the CI/CD framework. Centralized secrets management eliminates hardcoded credentials and minimizes the risk of secret exposure. Dynamic secret retrieval, automated rotation, and fine-grained access control significantly enhance confidentiality and compliance. By decoupling secrets from pipeline configurations, the framework improves both security and operational flexibility. This approach is particularly beneficial for large-scale enterprise environments with strict governance requirements. The integration of security automation and compliance checks further strengthens the proposed framework. Automated security scanning, policy enforcement, and continuous monitoring ensure early detection of vulnerabilities and misconfigurations. Compliance requirements are enforced consistently across all environments, reducing audit complexity. Security becomes a continuous and shared responsibility rather than a post-deployment activity. This shift enables enterprises to maintain a strong security posture while accelerating software delivery. Experimental evaluation confirms that the proposed framework effectively improves security and reliability without imposing significant performance overhead. The results demonstrate reduced security incidents, improved deployment success rates, and enhanced compliance adherence. Although security checks introduce slight increases in pipeline execution time, the

benefits outweigh the costs in enterprise contexts. The evaluation validates the practical applicability and scalability of the framework.

In conclusion, this research contributes a structured and practical solution for secure enterprise CI/CD using Azure DevOps and Azure Key Vault. The framework supports secure automation, centralized secrets management, and compliance integration within modern cloud environments. It provides a foundation for enterprises seeking to adopt secure DevOps practices. Future work may extend this framework by incorporating advanced threat detection, zero-trust architectures, and artificial intelligence-driven security analytics. The proposed approach represents a significant step toward secure and resilient enterprise software delivery.

References

[1] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010. ISBN:978-0-321-60191-9. Pages: 512

[2] T. Rangnau, R. V. Buijtenen, F. Fransen, and F. Turkmen, “Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines,” in EDOC, 2020.

[3] T. Chen and H. Suo, "Design and Practice of DevOps Platform via Cloud Native Technology," 2022 IEEE 13th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2022, pp. 297-300, doi: 10.1109/ICSESS54813.2022.9930226.

[4] Omoike, O. (2025). Designing a Cloud-Native DevOps Framework for Risk Management in Digital Financial Service. International Journal of Scientific Research and Modern Technology, 4(11), 16–21. https://doi.org/10.38124/ijsrmt.v4i11.954

[5] Theodoropoulos, T.; Rosa, L.; Benzaid, C.; Gray, P.; Marin, E.; Makris, A.; Cordeiro, L.; Diego, F.; Sorokin, P.; Girolamo, M.D.; et al. Security in Cloud-Native Services: A Survey. J. Cybersecur. Priv. 2023, 3, 758-793. https://doi.org/10.3390/jcp3040034

[6] M. Meyer. Continuous integration and its tools. IEEE Softw., 31 (3) (2014), pp. 14-16, 10.1109/MS.2014.58

[7] M. Fowler, Continuous Integration, 2006. Google Scholar

[8] Akond Rahman, Chris Parnin, and Laurie Williams. 2019. The Seven Sins: Security Smells in Infrastructure As Code Scripts. In Proceedings of the 41st International Conference on Software Engineering (Montreal, Quebec, Canada) (ICSE '19). IEEE Press, Piscataway, NJ, USA, 164-175.

[9] Khatri, A.; Khatri, V. Mastering Service Mesh: Enhance, Secure, and Observe Cloud-Native Applications with Istio, Linkerd, and Consul; Packt Publishing Ltd.: Birmingham, UK, 2020. [Google Scholar]

[10] Chernyshev, M.; Baig, Z.; Zeadally, S. Cloud-Native Application Security: Risks, Opportunities, and Challenges in Securing the Evolving Attack Surface. Computer 2021, 54, 47–57. [Google Scholar] [CrossRef]

[11] Rahaman, M.S.; Islam, A.; Cerny, T.; Hutton, S. Static-Analysis-Based Solutions to Security Challenges in Cloud-Native Systems: Systematic Mapping Study. Sensors 2023, 23, 1755. [Google Scholar] [CrossRef]

[12] Sun, Lei et al. “Research on Key Management Infrastructure in Cloud Computing Environment.” 2010 Ninth International Conference on Grid and Cloud Computing (2010): 404-407.

[13] Myrbakken H., Colomo-Palacios R. DevSecOps: A multivocal literature review. Software Process Improvement and Capability Determination, Vol. 770, Springer, Cham (2017), pp. 17-29, 10.1007/978-3-319-67383-7_2

[14] Rahman, A., Mahdavi-Hezaveh, R., Williams, L.: A systematic mapping study of infrastructure as code research. Inf. Softw. Technol. 108, 65–77 (2019). https://doi.org/10.1016/j.infsof.2018.12.004

[15] Morris, K.: Infrastructure as Code: Managing Servers in the Cloud. O’Reilly Media Inc, Sebastopol (2016)

[16] Peter W. Singer. et al. 2014. Cybersecurity and Cyberwar: What Everyone Needs to Know. ISBN: 9780199918096. Publisher: Oxford University Press. DOI: 10.1093/wentk/9780199918096.001.0001

[17] Matija Cankar, Nenad Petrovic, Joao Pita Costa, Ales Cernivec, Jan Antic, Tomaz Martincic, and Dejan Stepec. 2023. Security in DevSecOps: Applying Tools and Machine Learning to Verification and Monitoring Steps. In Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (Coimbra, Portugal) (ICPE ’23 Companion). Association for Computing Machinery, New York, NY, USA, 201–205.

[18] Damian A. Tamburri. et al. 2020. Cloud applications monitoring: An industrial study. Information and Software Technology, Volume 127, November 2020, 106376. DOI: https://doi.org/10.1016/j.infsof.2020.106376