Publisher
Singh Publication
www.singhpublication.com

E-ISSN:2583-2468

Applied Science and Engineering Journal for Advanced Research

Research Article
DevSecOps

2026 Volume 5 Number 2 March

DevSecOps Scan Engine: A Containerized Security Orchestration Framework

Kumar SS[1*], Vivekanandan P[2], Dineshkumar R[3], Sasitharan S[4], Abinaya N[5]

DOI:10.54741/ASEJAR/5.2.2026.183

[1*] S.Saravana Kumar, Assistant Professor(SS), Department of CSE(Cyber Security), Dr. Mahalingam College of Engineering and Technology, Coimbatore, Tamil Nadu, India.
[2] P. Vivekanandan, Head of the Department, Department of CSE(Cyber Security), Dr. Mahalingam College of Engineering and Technology, Coimbatore, Tamil Nadu, India.
[3] R. Dineshkumar, Student, Department of CSE(Cyber Security), Dr. Mahalingam College of Engineering and Technology, Coimbatore, Tamil Nadu, India.
[4] S. Sasitharan, Student, Department of CSE(Cyber Security), Dr. Mahalingam College of Engineering and Technology, Coimbatore, Tamil Nadu, India.
[5] N. Abinaya, Student, Department of CSE(Cyber Security), Dr. Mahalingam College of Engineering and Technology, Coimbatore, Tamil Nadu, India.

The rapid evolution of modern software delivery practices has significantly shortened development cycles, enabling organizations to release updates at an unprecedented pace. However, this acceleration has also increased the exposure of applications to security vulnerabilities, particularly when traditional security validation is performed only at later stages of development.

This paper presents the DevSecOps Scan Engine, a container-oriented orchestration framework designed to embed automated security analysis directly into continuous integration and continuous deployment workflows. The proposed system supports multiple categories of security testing, including static code analysis, dynamic application testing, and dependency vulnerability assessment, all executed within isolated and short-lived container environments. By leveraging containerization, the framework ensures consistency across executions while eliminating environmental dependencies and cross-process interference.

A standardized abstraction layer is introduced to unify the interaction between diverse security tools and pipeline components, transforming heterogeneous scanner outputs into a consistent structure for seamless integration with dashboards and automated decision-making systems.

Experimental evaluation demonstrates that the system supports concurrent scan execution with reliable isolation and efficient resource utilization, while effectively preventing deployments when critical vulnerabilities are detected. Overall, the proposed approach enables scalable and automated security validation while preserving the speed and flexibility required in modern development environments.

Keywords: DevSecOps, container isolation, CI/CD security, SAST, DAST, SCA, security automation, vulnerability orchestration

Corresponding Author

S.Saravana Kumar, Assistant Professor(SS), Department of CSE(Cyber Security), Dr. Mahalingam College of Engineering and Technology, Coimbatore, Tamil Nadu, India.
Email: saravanacs84@gmail.com

How to Cite this Article

Kumar SS, Vivekanandan P, Dineshkumar R, Sasitharan S, Abinaya N, DevSecOps Scan Engine: A Containerized Security Orchestration Framework. Appl Sci Eng J Adv Res. 2026;5(2):1-5.
Available From
https://asejar.singhpublication.com/index.php/ojs/article/view/183

To Browse

Manuscript Received	Review Round 1	Review Round 2	Review Round 3	Accepted
2026-02-02	2026-02-19			2026-03-07

Conflict of Interest	Funding	Ethical Approval	Plagiarism X-checker	Note
None	Nil	Yes	5.83	

1. Introduction

1.1 Background and Motivation

The adoption of continuous integration and continuous deployment practices has transformed the way software systems are developed and delivered. Organizations are now capable of releasing updates within hours or even minutes, allowing rapid adaptation to user requirements and market demands. While this shift improves productivity and responsiveness, it also introduces significant security concerns. In many cases, vulnerabilities are introduced during development and remain undetected due to insufficient or delayed security testing.

Traditional security assessment approaches, which often rely on manual testing or periodic evaluations, are not well suited for high-frequency deployment environments. These methods typically occur after development is complete, making it difficult to identify and resolve issues early in the lifecycle. As a result, vulnerabilities may propagate into production systems, increasing the likelihood of exploitation and potential data breaches.

1.2 Challenges in Integrating Security

Despite the growing importance of DevSecOps, integrating security into development pipelines remains a complex task. Security tools often require different runtime configurations and dependencies, making it difficult to maintain consistent execution environments. Furthermore, these tools generate outputs in varying formats, which complicates the aggregation and interpretation of results.

Another major challenge is the lack of isolation in shared environments. When multiple scans are executed simultaneously, resource contention and process interference can lead to inconsistent results. Additionally, limited visibility into ongoing scan activities can hinder effective monitoring and decision-making. These issues collectively create barriers to the seamless adoption of DevSecOps practices.

1.3 Proposed Solution

To overcome these challenges, this work proposes the DevSecOps Scan Engine, a unified orchestration framework that leverages containerization to standardize and automate security scanning processes.

Each scan is executed within a dedicated container, ensuring a clean and isolated runtime environment. This approach eliminates configuration drift and guarantees reproducibility across different executions.

The framework integrates directly with CI/CD pipelines through a set of well-defined APIs, enabling automated initiation and monitoring of security scans. By incorporating severity-based enforcement mechanisms, the system ensures that deployments are blocked when critical vulnerabilities are detected. This proactive approach allows organizations to address security issues before they reach production environments.

2. Review of Literature

Security integration within software development lifecycles has been extensively studied in recent years. Existing research highlights the importance of embedding security mechanisms early in the development process to reduce vulnerabilities and improve system resilience [1].

Rahman and Williams discuss the adoption of security practices within DevOps environments, emphasizing the need for automation and continuous monitoring [2]. Similarly, Fitzgerald and Stol explore continuous software engineering models that enable rapid development cycles while maintaining system quality [3].

Beller et al. analyze the effectiveness of static analysis tools, identifying limitations such as developer resistance and false positives [5]. Johnson et al. further investigate why developers often avoid such tools, citing usability and workflow integration challenges [6].

Dynamic testing approaches have also been explored, with Fonsca et al. evaluating web vulnerability scanners for detecting injection and cross-site scripting vulnerabilities [7]. The OWASP Top 10 framework provides a standardized classification of critical web application risks [8].

Containerization technologies, particularly Docker, have been widely studied for their role in improving deployment consistency and isolation [9]. Pahl highlights the importance of container-based architectures in cloud environments, enabling scalable and reproducible execution [10].

Overall, existing literature emphasizes the need for integrated, automated, and scalable security frameworks capable of operating within modern CI/CD ecosystems.

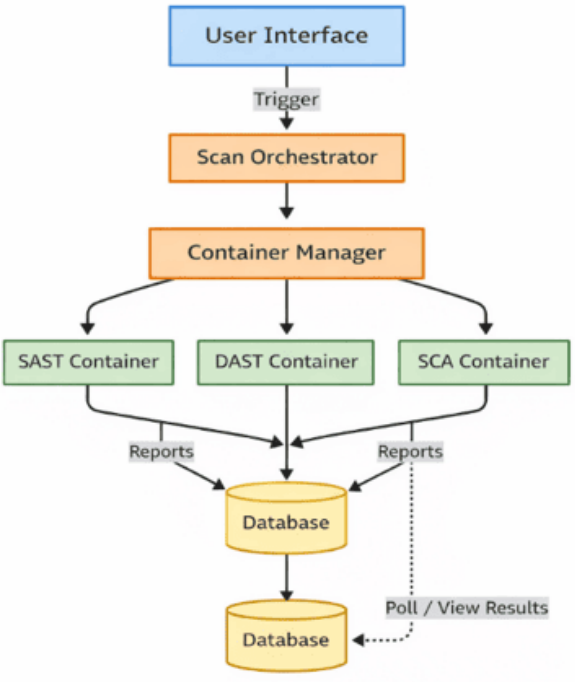
3. Devsecops Scan Engine Architecture

The DevSecOps Scan Engine is designed as a multi-layered architecture that coordinates user interaction, orchestration logic, and execution environments. The system is structured to ensure scalability, maintainability, and efficient resource utilization while supporting diverse security scanning tools.

3.1 Architectural Overview

The architecture is divided into three primary layers: the presentation layer, the orchestration layer, and the execution layer. The presentation layer provides a user interface that enables interaction with the system through API calls. Instead of maintaining persistent connections, the system adopts a polling-based approach to retrieve scan status updates at regular intervals.

The orchestration layer acts as the central control unit, responsible for validating requests, assigning unique identifiers, and managing the overall scan lifecycle. The execution layer interfaces with the container runtime to provision and monitor scanner instances. This separation of responsibilities ensures modularity and simplifies system maintenance.



3.2 Container Lifecycle Management

Each scan request progresses through a series of well-defined stages, beginning with initialization and ending with finalization. During initialization, the system validates input parameters and prepares metadata records. The provisioning stage involves allocating container resources and configuring the execution environment.

Once the environment is ready, the execution stage begins, where the selected security tool runs within a restricted container. After execution, the system processes the generated reports, extracting relevant information and converting it into a standardized format. Finally, the results are stored securely, and the container is removed to prevent residual data from persisting.

This lifecycle-driven approach ensures that each scan operates independently, eliminating interference and maintaining consistency across executions.

3.3 Tool Abstraction Layer

One of the key components of the system is the abstraction layer, which standardizes the interaction between different security tools. Since each tool produces output in its own format, the abstraction layer transforms these outputs into a unified schema. This schema includes essential information such as severity levels, vulnerability descriptions, and remediation suggestions.

By normalizing the data, the system enables consistent reporting and simplifies integration with dashboards and CI/CD pipelines. Design also allows new tools to be incorporated without requiring significant modifications to existing components.

3.4 CI/CD Integration and Enforcement

The framework is designed to integrate seamlessly with CI/CD pipelines through REST-based APIs. When a pipeline reaches a predefined stage, it triggers a security scan and periodically checks its status. Based on the results, the system enforces deployment decisions by evaluating the severity of detected vulnerabilities.

If critical issues are identified, the pipeline is halted, preventing insecure code from being deployed. Otherwise, the deployment proceeds automatically. This mechanism ensures that security validation becomes an integral part of the development process rather than an afterthought.

4. Experimental Evaluation

The system was evaluated under conditions that simulate real-world development environments with multiple concurrent scan requests. Performance measurements indicate that the system achieves low latency during scan initiation and efficient container provisioning. The polling mechanism provides consistent updates without introducing significant overhead.

Functional testing confirms that the system accurately processes and normalizes outputs from different security tools. Vulnerability classifications remain consistent across multiple runs, demonstrating reliability in severity aggregation. Additionally, the CI/CD enforcement mechanism effectively blocks deployments when high-risk vulnerabilities are detected.

Resource monitoring further verifies that each container operates within defined limits, ensuring isolation and preventing resource contention. The automatic cleanup process successfully removes all execution artifacts, maintaining a clean runtime environment.

5. Discussion

The use of container-based execution provides significant advantages in terms of consistency and isolation. By ensuring that each scan runs in a controlled environment, the system eliminates issues related to configuration drift and dependency conflicts. The polling-based monitoring strategy offers a practical balance between simplicity and responsiveness, avoiding the complexity associated with persistent communication channels.

The abstraction layer enhances the flexibility of the system by decoupling tool implementations from the core architecture. This allows for easy updates and integration of new scanning technologies. However, the system's reliance on container runtime infrastructure may introduce limitations in environments with restricted resources. Future enhancements could address these limitations by incorporating distributed orchestration mechanisms.

6. Conclusion

This paper introduced the DevSecOps Scan Engine, a container-based framework designed to integrate automated security scanning into CI/CD pipelines. By combining container isolation, standardized abstraction, and automated enforcement mechanisms, the system enables continuous and scalable security validation.

The experimental results demonstrate that the framework achieves reliable performance, effective vulnerability aggregation, and consistent enforcement of security policies. The proposed approach highlights the feasibility of integrating security into rapid development workflows without compromising efficiency.

Future work will focus on improving scalability through distributed execution models and enhancing risk prioritization techniques to further optimize security decision-making.

References

1. A. Sharma, & M. Barenkamp. (2022). DevOps and DevSecOps: A systematic literature review. *Journal of Software Engineering and Applications, 15*(3), 123–145.

2. Ramprakash, P., Sakthivadivel, M., Krishnaraj, N., & Ramprasath, J. (2014). *Host-based intrusion detection system using sequence of system calls*. *International Journal of Engineering and Management Research, 4*(2), 241–247.

3. B. Fitzgerald, & K. Stol. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software, 123*, 176–189.

4. Saranya, N., Sakthivadivel, M., Karthikeyan, G., & Rajkumar, R. (2023). *Securing the cloud: An empirical study on best practices for ensuring data privacy and protection*. *International Journal of Engineering and Management Research, 13*(2), 46–49. https://doi.org/10.31033/ijemr.13.2.6

5. M. Beller, R. Bholanath, S. McIntosh, & A. Zaidman. (2016). Analyzing the state of static analysis: A large scale evaluation in open source software. *Proc. IEEE SANER*, pp. 1–10.

6. B. Johnson, Y. Song, E. Murphy Hill, & R. Bowdidge. (2013). Why developers avoid static analysis tools. *Proc. Int. Conf. Software Engineering*, pp. 672–681.

7. J. Fonsca, M. Vieira, & H. Madeira. (2007). Testing and comparing web vulnerability scanners for injection and cross site scripting attacks. *Proc. Pacific Rim Int. Symp. Dependable Computing*, pp. 365–372.

8. OWASP Foundation. (2021). *OWASP Top 10: The ten most critical web application security risks*.

9. D. Merkel. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal, 2014*(239).

10. C. Pahl. (2015). Containerization and the platform as a service cloud. *IEEE Cloud Computing, 2*(3), 24–31.

11. L. Bass, I. Weber, & L. Zhu. (2015). *DevOps: A software architect's perspective*. Boston, MA, USA: Addison-Wesley.

12. J. Wettinger, U. Breitenbücher, M. Falkenthal, F. Leymann, & M. Zimmermann. (2014). Integrating configuration management with model driven cloud management based on TOSCA. *Proc. IEEE Int. Conf. Cloud Computing*, pp. 437–444.

13. S. McConnell. (2004). *Code complete*. (2nd ed.). Redmond, WA, USA: Microsoft Press.

14. T. M. Chen, & S. Abu-Nimeh. (2001). Lessons from Stuxnet. *Computer, 44*(4), 91–93.

15. N. Antunes, & M. Vieira. (2009). Comparing the effectiveness of penetration testing and static code analysis on the detection of SQL injection vulnerabilities. *Proc. IEEE ISSRE*, pp. 301–306.

16. A. Sabahi. (2016). Secure containers in cloud computing: A survey. *Proc. IEEE 8th Int. Conf. Cloud Computing Technology and Science*, pp. 395–402.

17. R. Buyya, C. Vecchiola, & S. T. Selvi. (2013). *Mastering cloud computing*. New York, USA: McGraw Hill.