Automated Vulnerability Management in CI/CD Pipeline: Proof of Concept

Radu CUCUTĂ and Alin PUNCIOIU

***Abstract*—This paper explores the evolution of vulnerability management, tracing its shift from manual processes to modern automated approaches driven by the increasing number of vulnerabilities in software systems. The emergence of DevSecOps (integrating security practices into the DevOps pipeline) is examined alongside continuous integration and continuous deployment (CI/CD) practices. The paper outlines the key components of a DevSecOps pipeline, with an emphasis on incorporating security checks at each stage. It also compares various vulnerability scanning and monitoring tools, highlighting their features and pricing models. Furthermore, the paper presents a proof of concept (PoC) integrating security into CI/CD pipelines, enabling automated detection, analysis, and remediation of vulnerabilities. The proposed scalable, cloud-native architecture enhances visibility, collaboration, and continuous security across software development lifecycles.**

***Index Terms*—automation, CI/CD, DAST, DevOps, DevSecOps, SAST, scanning tools, SDLC, security pipeline, vulnerability management.**

I. Introduction

In the late 1990s and early 2000s, vulnerability management was a manual process due to the limited number of vulnerabilities. However, the shift to remote operations, increased software usage, and changes in development practices led to a surge in vulnerabilities. The old approach of using objective scores like CVSS (Common Vulnerability Scoring System) for prioritization proved ineffective, as it didn't consider business risk or real-world threats [1].

Patching, the traditional "find it, fix it" approach, became impractical and risky with the growing number of vulnerabilities. The challenges included downtime, interference with other assets, and the risk of faulty patches. The cost of vulnerability management also increased significantly. Recent reports show that the global average cost of a data breach in 2025 hit the value of 4.4 million dollars, a slight 9% decrease over last year, driven by faster identification and containment [2-3]. Manual processes, along with the reliance on outdated tools, resulted in inefficient communication between security and IT/DevOps teams [3].

To address these issues, modern vulnerability management platforms focus on visibility, automation, and risk-based prioritization. Complete visibility of network and software assets is crucial for effective remediation. Automation helps save time and money while ensuring consistency and reducing the margin of error. Risk-based prioritization allows organizations to focus on the most urgent issues based on their unique system, encouraging collaboration between IT, security, and other stakeholders [4-5].

Numerous high-performing software development teams worldwide have embraced DevOps practices, streamlining processes such as software builds, testing, and deployment. Despite its effectiveness in these areas, DevOps falls short in addressing a crucial aspect: security and vulnerability management. Vulnerabilities can emerge at various stages within the DevOps pipeline. For instance, developers might unintentionally write insecure code, or an application could utilize a library with a security flaw. Even containerized applications are susceptible to vulnerabilities in the operating system running inside the container [2, 5-6].

This article presents a comprehensive analysis of DevSecOps principles and CI/CD practices as the foundation of modern software development pipelines. It explores the core tools, architectural components and automation mechanisms that ensure both agility and security across the development lifecycle.

The original contribution of this work is represented by the proposed architectural model and Proof of Concept (PoC), a newly designed DevSecOps based solution that integrates multiple tools and processes into a unified, secure, and automated ecosystem. The PoC demonstrates how these technologies can be orchestrated together within a cloud-native environment (Azure), ensuring that development, security, and operational practices coexist seamlessly.

This model highlights the practicality of a scalable, open-source DevSecOps platform, providing a foundation that can be adapted by startups and small organizations.

II. Code and Vulnerabilities

Vulnerabilities often originate before any code is written. During the planning and design phase, insecure architectural decisions or incomplete threat assessments can leave systems exposed [7]. Common issues include overlooking

R. CUCUTĂ is with the Military Technical Academy "Ferdinand I", Bucharest, Romania (e-mail: radu.cucuta@yahoo.ro).

A. PUNCIOIU is with the Military Technical Academy "Ferdinand I", Bucharest, Romania (e-mail: alin.puncioiu@mta.ro).

Digital Object Identifier 10.32754/JMT.2026.1.03

potential attack vectors, failing to enforce data protection principles, or neglecting to design for secure authentication and authorization. The absence of formal threat modeling or risk assessment allows weaknesses to persist undetected, setting the stage for future exploitation [8].

Developers often rely on other people's code. Most software projects start out with importing packages using a package manager. This code can be open source or proprietary, and it often relies on additional packages. Especially when writing JavaScript and using npm, it's packages all the way down [7-8].

However, these external dependencies are not without risk. Packages may contain incorrect or malicious code. Companies often release new versions of their software and packages that include new security updates, but those updated packages may introduce new vulnerabilities (or bugs) [7-8].

Once software reaches deployment, vulnerabilities continue to emerge through insecure configurations, exposed interfaces, and mismanaged credentials. Cloud-based systems are particularly prone to configuration errors, such as open storage buckets or unrestricted network ports. Outdated or unpatched software components increase the risk of exploitation [7]. Weak monitoring practices, poor log management, and lack of visibility into runtime environments further exacerbate the problem by allowing attackers to operate undetected [7].

III. DevSecOps and CI/CD Practices

A. *What Is DevSecOps?*

DevSecOps, short for Development, Security, and Operations, is an approach that emphasizes integrating security practices into the DevOps pipeline from the very beginning of the software development lifecycle (SDLC). The concept of DevSecOps started to gain traction in response to the need for improved security measures in fast-paced and continuous delivery environments [3, 9].

The exact timeline for the emergence of DevSecOps can be a bit subjective, but it can be traced back to the early to mid-2010s when organizations began to realize the limitations of traditional security approaches in the face of agile and DevOps methodologies [1]. As development cycles accelerated and software deployment became more frequent, there was a growing recognition that security couldn't be a separate, isolated phase but needed to be integrated throughout the entire development process [7].

DevSecOps focuses on making security an integral part of an organization's DevOps processes. According to the 2019 "State of DevOps Reports" by Puppet, CircleCI, and Splunk, teams implementing DevOps are already more secure on average than teams without these practices. However, with DevSecOps, security becomes an explicit and integral part of the DevOps workflow. The goal is to make everyone responsible for security [3, 9].

B. *CI/CD Pipeline and Practical Vulnerabilities*

A continuous integration and continuous deployment (CI/CD) pipeline is a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation [3], [7].

DevOps practices aim to automate as much work as possible to save time and minimize human error. Central to this automation are CI/CD pipelines. Pipelines automatically trigger builds, run tests, and deploy software, either in the cloud or on your private server [3].

While CI/CD pipelines are key to successfully implementing DevOps, they also come with potential security threats. Unauthorized access to the pipeline can put a company at risk [3].

CI/CD pipelines may contain sensitive information, like usernames and passwords, for things such as an automated database release or an application that needs to connect to a database. Security management must be implemented correctly, or developers will have direct access to production databases that contain sensitive data.

C. *What Is the DevSecOps Pipeline?*

The typical DevOps pipeline included phases like Plan, Code, Build, Test, Release, and Deploy. In DevSecOps, specific security checks are applied in each phase of the DevOps pipeline, as illustrated in Fig. 1. Here we can understand the security checks used by adopting DevSecOps in the CI/CD pipeline.

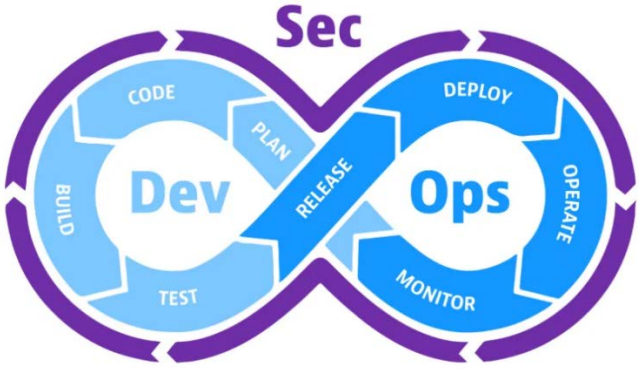

Figure 1. DevSecOps cycle [5]

• Plan: During the planning phase, comprehensive security analysis is undertaken to define the security objectives, scope, and requirements of the project [10]. This stage involves the identification of potential threat scenarios and the formulation of a strategic plan that determines how, where, and when security testing activities will be conducted throughout the SDLC [11-13].

• Code: In the coding stage, developers implement secure coding practices supported by automated tools such as linters and Git-based version control mechanisms. These tools assist in enforcing coding standards, detecting potential security misconfigurations, and preventing the

 Digital Object Identifier 10.32754/JMT.2026.1.03

inadvertent exposure of sensitive credentials such as passwords and API keys within source repositories [11-13].

• Build: Prior to and during the build phase, Software Composition Analysis (SCA) and Static Application Security Testing (SAST) are performed to ensure that both proprietary and third-party components meet security requirements. SCA tools analyze external dependencies and open-source libraries to detect known vulnerabilities or outdated packages, while SAST tools examine the source code itself for language-specific weaknesses such as injection flaws, insecure data handling, and improper input validation. Together, these analyses enable the early identification of security flaws before compilation and deployment, thereby minimizing the likelihood of introducing vulnerabilities into the production environment [11-13].

• Test: The testing phase encompasses multiple layers of validation to assess both the functional integrity and security resilience of the application prior to release. At the lowest level, unit testing verifies that individual components and functions behave as intended, thereby preventing logic-based vulnerabilities and supporting early fault detection. Unit tests also serve as regression checks for previously identified security flaws, ensuring they are not reintroduced in subsequent iterations. The testing process further involves functional and integration-level testing, encompassing unit, integration, and system tests that validate both the internal logic of individual components and the correctness of their interactions [11-13].

During the testing phase, Dynamic Application Security Testing (DAST) tools are employed to evaluate the application's behavior under runtime conditions. This testing detects vulnerabilities related to authentication, authorization, SQL injection, and API endpoint security, ensuring that potential attack vectors are identified prior to release.

In parallel, Infrastructure as Code (IaC) templates, used to define and provision application environments, are analyzed with specialized security tools to detect misconfigurations, exposed secrets, or noncompliant resource definitions [11-13].

• Release: Immediately preceding deployment, the release phase involves reviewing vulnerability scanning and penetration testing reports to validate the security posture of the software. These assessments verify that previously identified vulnerabilities have been mitigated and that the application complies with established security baselines before it is made publicly available [11-13].

• Deploy: After completing the above test in runtime, send a secure infra or build to production for final deployment [11-13].

• Operate: Once deployed, the application enters the operational phase, where it is actively managed and maintained within a live environment. This stage emphasizes logging, monitoring, and configuration management to ensure that implemented security controls remain functional and effective. Continuous oversight during operation supports early detection and remediation of performance or security anomalies [11-13].

• Monitoring: In addition to the operate stage, this phase also encompasses active incident detection and response activities, enabling security teams to promptly investigate, contain, and remediate identified threats [11-13].

DevSecOps integrates security mechanisms and controls directly into the Continuous Integration and Continuous Deployment (CI/CD) pipeline, facilitating the automation of security testing and verification at every phase of the software development lifecycle. This integration ensures that security assurance activities are performed continuously and systematically, without impeding development velocity or deployment frequency. By embedding security as a parallel process rather than a sequential checkpoint, DevSecOps promotes both rapid software delivery and a proactive security posture [10].

IV. Comparison of DevSecOps Tools

The following section undertakes a comparative examination of a diverse set of DevSecOps tools, focusing on their functional scope, application domains, and relevance across different stages of the software development lifecycle. Each tool is situated within its respective category such as static or dynamic analysis, dependency management, secret detection, or monitoring and considered in relation to its usability, accessibility, and licensing model (freemium, open-source, or subscription-based). This examination offers a structured perspective on how these tools support secure software development practices and how their economic and accessibility characteristics affect their adoption within organizational development environments.

Snyk (SAST, SCA / Container and IaC Security – Freemium)

Snyk is a Software Composition Analysis (SCA) tool that helps developers identify and remediate vulnerabilities in open-source dependencies, containers, and IaC files [3, 10].

Assertible (API and Functional Testing – Freemium)

Assertible[2] focuses on API and functional reliability testing, verifying service uptime, API correctness, and data consistency [10].

StackHawk (DAST – Subscription-based)

StackHawk[3] is a Dynamic Application Security Testing (DAST) tool designed to detect vulnerabilities in web applications and APIs during the testing and integration stages of the CI/CD pipeline. It automates runtime security scans to identify issues such as injection flaws, authentication and authorization weaknesses, and misconfigurations [10].

[2] https://assertible.com/
[3] https://www.stackhawk.com/

Clair (Container Image Security / SCA – Open-source)

Clair is an open-source container image security and vulnerability scanning tool that performs static analysis of container images to detect known vulnerabilities in system packages and dependencies. By maintaining a continuously updated vulnerability database, Clair enables developers and DevSecOps teams to identify and remediate image-level security risks before containers are promoted to production environments [3], [10].

SonarQube (SAST / Code Quality and Security Analysis – Freemium)

SonarQube is a Static Application Security Testing (SAST) and code quality analysis platform that examines source code to identify security vulnerabilities, bugs, and code smells before deployment [3], [10].

BrowserStack (Testing and Quality Assurance / Cross-Browser and Device Testing – Subscription-based)

BrowserStack[4] is a cloud-based testing and quality assurance platform that enables developers to test web and mobile applications across a wide range of browsers, operating systems, and real devices [10].

Trivy (Container and IaC Security / Vulnerability Scanning – Open-source)

Trivy is an open-source vulnerability and misconfiguration scanning tool designed for containers, source code, and Infrastructure as Code (IaC). It performs comprehensive security checks to identify known vulnerabilities in operating system packages, dependencies, and configuration files [3], [10].

Splunk (Monitoring and Security Analytics / SIEM – Subscription-based)

Splunk is a Security Information and Event Management (SIEM) and observability platform that collects, indexes, and analyzes machine-generated data from applications, systems, and network devices. It enables continuous monitoring, threat detection, and incident response through real-time log analysis and security analytics. Integrated within the operate and monitoring stages of the software development lifecycle, Splunk provides actionable insights that enhance situational awareness and support compliance auditing [3], [10].

AppSignal (Monitoring and Performance Management / Application Observability – Subscription-based)

AppSignal[5] is a performance monitoring and application observability platform designed to collect and analyze metrics, errors, and performance traces from web applications and services. It provides insights into system behavior, application performance, and potential anomalies, assisting in identifying issues that may impact reliability or security [10].

Honeybadger (Error Monitoring and Incident Management / Application Observability – Subscription-based)

Honeybadger[6] is an error monitoring and incident management platform designed to detect, track, and analyze exceptions and outages in web applications. It collects runtime error data, monitors uptime, and integrates with development workflows to provide real-time notifications and diagnostics for faster issue resolution. Positioned within the operate and monitoring phases of the software development lifecycle, Honeybadger enhances application reliability by correlating errors with code changes and performance metrics [10].

OWASP Dependency-Check (Software Composition Analysis / Vulnerability Detection – Open-source)

OWASP Dependency-Check is an open-source Software Composition Analysis (SCA) tool that identifies known vulnerabilities within a project's external dependencies. It analyzes libraries, frameworks, and third-party components to detect publicly disclosed security issues based on Common Vulnerabilities and Exposures (CVE) data. Integrated into the coding and build phases of the software development lifecycle, Dependency-Check enables early detection of dependency-related risks and supports continuous vulnerability management. Its open-source licensing model makes it accessible for both academic research and enterprise integration within DevSecOps pipelines [3], [10].

Syft (Software Composition Analysis / SBOM Generation – Open-source)

Syft is an open-source Software Composition Analysis (SCA) tool developed by Anchore[7], designed to generate detailed Software Bills of Materials (SBOMs) for container images and file systems. It identifies and catalogs software packages, libraries, and dependencies to improve visibility and traceability within complex software supply chains. Integrated primarily into the build and release phases of the software development lifecycle, Syft supports multiple output formats (e.g., SPDX, CycloneDX) and facilitates vulnerability assessment by serving as a foundational inventory for downstream security tools [10].

Grype (Vulnerability Scanning / Container and Dependency Security – Open-source)

Grype is an open-source vulnerability scanner that analyzes container images, file systems, and SBOMs to identify known security issues based on vulnerability databases such as CVE and NVD. It is frequently used in conjunction with Syft to correlate identified packages with public vulnerability data, providing comprehensive risk visibility for both open-source and containerized components [10].

[4] https://www.browserstack.com/
[5] https://www.appsignal.com/
[6] https://www.honeybadger.io/
[7] https://anchore.com/

 Digital Object Identifier 10.32754/JMT.2026.1.03

Falco (Runtime Security and Intrusion Detection / Behavioral Monitoring – Open-source)

Falco is an open-source runtime security and intrusion detection tool developed by the Cloud Native Computing Foundation (CNCF). It continuously monitors system calls and container activity to detect anomalous behavior, policy violations, and potential intrusions in real time. Operating primarily within the operate and monitoring phases of the software development lifecycle, Falco enhances runtime observability by alerting teams to suspicious activity across containers, hosts, and Kubernetes environments [3, 10].

Datadog (Monitoring and Security Analytics / Cloud Observability Platform – Subscription-based)

Datadog is a cloud-based monitoring, observability, and security analytics platform that provides real-time visibility into applications, infrastructure, and network performance. It collects and correlates metrics, traces, and logs to detect anomalies, monitor service health, and identify potential security threats. Operating primarily within the operate and monitoring phases of the software development lifecycle, Datadog integrates with CI/CD pipelines and diverse cloud environments to support continuous performance assessment and incident response [3, 10].

Prometheus (Monitoring and Metrics Collection / Observability – Open-source)

Prometheus is an open-source monitoring and alerting system widely used for collecting and analyzing metrics from applications, containers, and infrastructure components. It stores time-series data and enables the definition of custom alerts based on real-time performance indicators [3, 10].

TruffleHog (Secret Scanning and Credential Detection– Open-source)

TruffleHog[8] is an open-source secret scanning tool designed to detect exposed credentials, API keys, and other sensitive information within source code repositories, configuration files, and commit histories. It employs pattern matching and entropy-based analysis to identify potential secret leaks across platforms such as GitHub, and GitLab [10].

Argo CD (Continuous Delivery and GitOps Open-source)

Argo CD is an open-source continuous delivery (CD) and GitOps tool designed for automating the deployment and lifecycle management of applications within Kubernetes environments. It continuously monitors Git repositories as the declarative source of truth and synchronizes the desired application state with the running infrastructure [3, 10].

To facilitate a structured comparison, Table I summarizes the examined DevSecOps tools according to their primary function and pricing model, highlighting distinctions between open-source and commercial solutions.

[8] https://trufflesecurity.com/trufflehog

TABLE I. SECURITY TOOLS: TYPES AND PRICING MODELS

Tool	Type	Pricing Model
Snyk	Security	Freemium and subscription-based
StackHawk	Security	Subscription-based
Clair	Security	Open-source
Assertible	Testing	Freemium and subscription-based
SonarQube	Code Quality	Open-source (Community) and Commercial
BrowserStack	Testing	Subscription-based
Aqua Security Trivy	Security	Open-source
Splunk	Monitoring	Subscription-based
AppSignal	Monitoring	Subscription-based
Honeybadger	Monitoring	Subscription-based
Dependency-Check (OWASP)	Security	Open-source
Syft	Security	Open-source
Grype	Security	Open-source
Falco	Runtime Security	Open-source
Datadog	Monitoring	Subscription-based
Prometheus	Monitoring	Open-source
TruffleHog	Secret detection	Open-source
Argo CD	Continuous delivery	Open-source

V. POC OVERVIEW

A. Why Implementing a PoC?

Automated Vulnerability Management represents a critical component in maintaining software security across the entire development lifecycle. Numerous platforms and tools have been developed to support this process, enabling the continuous identification, assessment, and remediation of vulnerabilities. However, most existing solutions remain at the conceptual or architectural level, with few being fully implemented or publicly available for empirical evaluation.

In this context, a Proof of Concept (PoC) was developed to demonstrate the practical implementation of an automated vulnerability management approach within a DevSecOps environment. The PoC aims to illustrate how security automation can be integrated across different stages of the software development lifecycle to improve vulnerability detection, analysis, and remediation efficiency.

The following section provides a detailed description of the PoC architecture, its components, integrated security tools, and the workflow designed to validate the feasibility of the proposed solution.

B. Context

The PoC was conceived within the context of establishing a startup oriented automated development platform, where the necessity for a DevSecOps integrated solution becomes essential. The proposed solution aims to consolidate tools, standards, and best practices under a unified DevSecOps framework, ensuring that each phase of the CI/CD pipeline from planning and coding to deployment and monitoring embeds security as a continuous and automated process.

A core design principle of the solution is the inclusion of automated vulnerability management mechanisms across all stages of software development and deployment. The PoC integrates multiple open-source and commercial tools to automate vulnerability detection, tracking, and remediation while maintaining flexibility and scalability within different project structures. Furthermore, the platform incorporates management and performance optimization components to enhance operational efficiency and observability.

The solution is particularly optimized for microservices-based architectures, offering support for monorepo configurations, where the entire project is maintained within a single repository and individual microservices are managed through dedicated branches. This approach promotes a unified workflow and improved oversight, aligning with practices adopted by large-scale technology organizations.

From an operational perspective, the PoC defines multiple user roles within the GitLab DevSecOps ecosystem (GitLab Dev / GitLab Ops), ensuring differentiated access, responsibilities, and workflow automation.

The underlying infrastructure and deployment environment are provisioned using Infrastructure as Code (IaC) tools, specifically Ansible and Terraform, which facilitate automated installation and configuration processes. The PoC is implemented within the Microsoft Azure Cloud environment with minimum costs, leveraging cloud-native capabilities while emphasizing the integration of open-source or cost-free tools suitable for small organizations or startups.

Additionally, Ansible playbooks have been developed to automate the installation and configuration of solution components where feasible, ensuring reproducibility and deployment efficiency. The design also adheres to the "Everything as Code" paradigm, promoting flexibility, maintainability, and transparency in both infrastructure and pipeline configurations.

An overview of the proposed architectural model is provided in Fig. 2.

C. Main Components

As shown in Fig. 3, the PoC is implemented within a virtualized environment that serves as the core of the integrated DevSecOps platform. This environment hosts all components required for the continuous integration and continuous delivery (CI/CD) processes, configured according to a predefined architectural template. A virtual machine functions as the central node of the solution, where multiple DevSecOps and security tools are installed and orchestrated to facilitate automated development, testing, deployment, and monitoring operations. The platform incorporates a comprehensive suite of tools, including Confluence, Jira, GitLab Dev Environment, GitLab Ops Environment, GitLab Runners, SonarQube, OWASP Dependency-Check, Trivy UI, Splunk, Nginx, JUnit, Syft, Grype, and a local registry, among others, all deployed and managed through Docker and Docker Compose technology.

The developed application is deployed across three operational environments, Development (Dev), Testing (Test), and Production (Prod), each equipped with specific monitoring and operational tools. These environments enable continuous data flow analysis, performance evaluation, and real-time detection of unauthorized or malicious access attempts. The integration of runtime monitoring tools also supports the implementation of a security incident response mechanism, ensuring timely mitigation of detected threats.

D. Testing

The testing and validation of the proposed solution were conducted using a custom-developed application specifically designed for this purpose. The application's primary objective was to generate JSON Web Tokens (JWTs) for authentication and authorization, enabling controlled access to protected resources based on token validation. The application was developed using the .NET framework and was architected as a system composed of two microservices, each serving a distinct functional purpose within the testing process.

The first component, the Authentication Service, was designed to manage user registration, login, and logout operations. The registration functionality required user input consisting of a username, email address, and password. The authentication process verified the provided credentials and, upon successful validation, issued a JWT used to authorize access to the second microservice. The logout mechanism invalidated existing refresh tokens, ensuring that session termination was handled securely.

The second component, the Data Consumption Service, was responsible for processing requests authenticated through valid JWTs issued by the Authentication Service. This service returned protected data only after verifying token authenticity, thereby simulating a real-world authorization workflow within a distributed microservices architecture.

To rigorously evaluate the proposed automated vulnerability management framework, the application was intentionally developed with predefined security vulnerabilities designed to test detection, analysis, and mitigation capabilities. This approach allowed for the systematic assessment of both static and dynamic security tools integrated within the DevSecOps pipeline.

The testing phase validated multiple tool functionalities, including secret detection using TruffleHog, static application security testing (SAST) configuration and rule customization via SonarQube, and dependency and container vulnerability scanning using Syft, Grype, and OWASP Dependency-Check. Furthermore, the behavior of the CI/CD pipeline was analyzed under failure conditions, verifying that specific jobs were correctly terminated when preconfigured vulnerability thresholds were exceeded.

The test suite also included JUnit-based unit tests,

 Digital Object Identifier 10.32754/JMT.2026.1.03

developed specifically for this application, to evaluate both functional correctness and integration stability. Within the operational and continuous monitoring stages, tools such as Trivy (for real-time Docker image analysis), Datadog (for metric collection, performance monitoring, and incident response), Falco Runtime Security (for real-time anomaly detection), Splunk (for log aggregation), Prometheus (for metrics collection), Grafana (for visualization of Prometheus data), were thoroughly evaluated to validate their interoperability and efficiency within the PoC architecture.

Performance measurements conducted during the testing phase indicated that the complete CI/CD pipeline from the initial code commit to the final deployment of the application in the production environment required an average of approximately **5 minutes** to execute. Following deployment, an additional **2-3 minutes** were required for the application's containerized components to be instantiated and become fully operational within the Kubernetes environment.

E. Repository

The complete testing workflow and installation scripts developed for the Proof of Concept are version-controlled within a dedicated Git repository at https://github.com/radu-cucuta/poc-automated-management-of-vulnerabilities. This repository provides full traceability of configuration files, automation playbooks, and deployment manifests, thereby validating the reproducibility and consistency of the proposed architectural model.

VI. PoC Architecture Diagrams

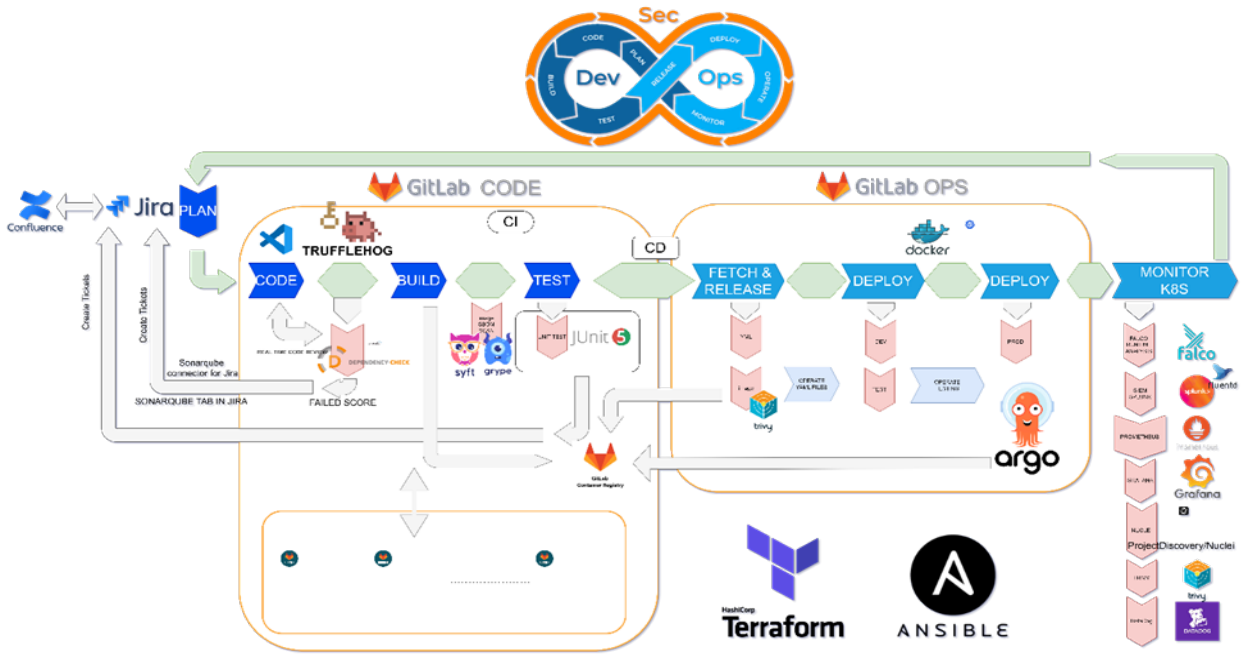

Figure 2. Architectural DevSecOps model (inspired of [11-12], [14-15])

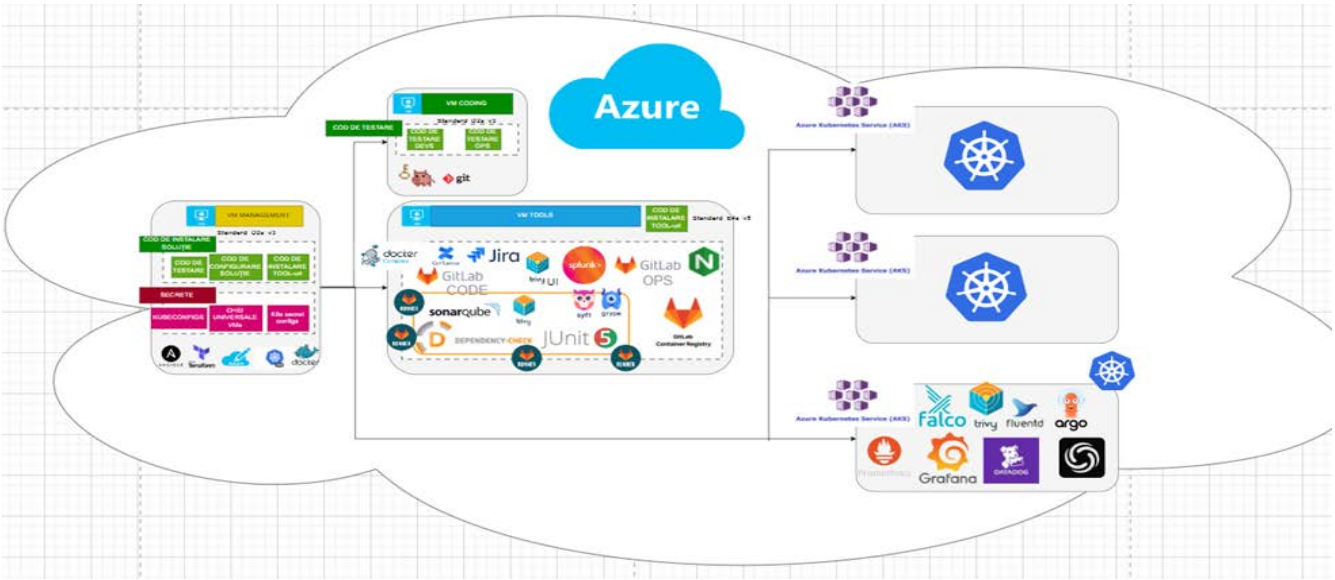

Figure 3. Architectural DevSecOps solution

Fig. 2 illustrates the overall architecture of the proposed DevSecOps PoC, integrating development, security, and operations into a unified automated workflow. The model follows a continuous integration and continuous delivery (CI/CD) structure embedded with security validation mechanisms at every stage of the software development lifecycle. While the previous Fig. 2 depicted the logical DevSecOps workflow, highlighting the integration of tools and processes across the CI/CD pipeline, Fig. 3 emphasizes the physical and infrastructural distribution of those components across virtual machines and Kubernetes clusters.

The architectural model presented in Fig. 2 is further elaborated in the following section to provide a detailed understanding of its components and interactions.

1. Planning and Project Management Layer

At the top-left section of the architecture, the workflow begins with Confluence and Jira, which serve as collaborative tools for project documentation, sprint management, and issue tracking. These components establish the planning phase of the lifecycle, defining user stories, requirements, and tasks that feed directly into the GitLab Code repository. This integration ensures full traceability between planning artefacts and version-controlled code.

2. Source Code and Version Control (GitLab CODE)

The "GitLab CODE" environment (as shown in Fig. 2) acts as the central version control system and CI orchestration platform. It hosts the application source code and associated configuration files, structured under a monorepo model that supports multiple microservices. This environment triggers the CI pipeline whenever a commit or merge request is performed, thereby initiating automated security and quality checks. Before code is merged, TruffleHog is employed to detect any hardcoded secrets or credentials within the source code. Once validated, the code moves through a sequence of CI pipeline stages, including build, test, and deployment preparation.

3. Continuous Integration (CI) Pipeline Stages

The CI pipeline encompasses several automated stages, each integrating specialized tools:

Code Stage:

The code undergoes secret scanning (TruffleHog) and dependency vulnerability analysis (OWASP Dependency-Check). This ensures that security is introduced early in the development process.

Build Stage:

The project is compiled and subjected to static analysis using SonarQube, identifying potential vulnerabilities, code smells, and maintainability issues. The Syft tool generates Software Bill of Materials (SBOMs) for each Docker image, while Grype evaluates these SBOMs to identify package-level vulnerabilities and assign risk scores.

Any critical vulnerability exceeding a defined threshold triggers an automated pipeline failure, preventing insecure builds from advancing.

Test Stage:

The JUnit5 framework performs unit and integration tests to validate functional correctness. Simultaneously, SAST results are aggregated for deeper analysis. This stage produces both test artefacts and vulnerability reports stored in GitLab. All resulting Docker images are pushed into the GitLab Container Registry, ensuring controlled versioning and provenance tracking of build artefacts.

4. Continuous Delivery and Deployment

After successful CI execution, the workflow transitions into the GitLab OPS environment, which manages CD (Continuous Delivery) operations. This phase includes multiple steps:

Fetch & Release:

The pipeline retrieves validated build artefacts and prepares them for deployment.

Deploy Stages (DEV → TEST → PROD):

Each environment follows a progressive deployment model, where Trivy performs container image scanning to identify runtime vulnerabilities before release.

Deployment automation is managed by Argo CD, ensuring GitOps-based synchronization between the manifest repository (GitLab Ops) and Kubernetes clusters. Any change to the manifest or Docker image automatically updates the corresponding environment in real time.

5. Runtime Monitoring and Security

The right section of the architecture represents the operational monitoring and runtime security layer deployed in the Kubernetes (K8s) environment. This layer integrates multiple observability and security tools. Falco Runtime Security detects anomalous or malicious behavior in containers and nodes in real time. Datadog provides centralized observability, performance analytics, and incident response. Prometheus collects metrics, while Grafana visualizes them through dashboards for continuous monitoring. Splunk aggregates and indexes logs across all components. Nuclei performs Dynamic Application Security Testing (DAST) to identify runtime web and API vulnerabilities. Trivy operates in continuous scan mode to detect newly introduced image vulnerabilities or misconfigurations. This combination ensures continuous runtime protection, event correlation, and efficient threat detection within the production environment.

VII. Research Perspectives

The development of the presented PoC has demonstrated the feasibility of integrating security, monitoring, and automation mechanisms into a unified DevSecOps framework. However, given the continuous evolution of both security paradigms and cloud-native technologies,

 Digital Object Identifier 10.32754/JMT.2026.1.03

several directions emerge for further research and enhancement of the proposed solution. These perspectives focus on extending the architecture's functionality, improving automation, and addressing advanced aspects of software and infrastructure security [3].

A crucial future enhancement involves the integration of a dedicated Identity and Access Management (IAM) framework within the platform. Such an integration would ensure centralized authentication, fine-grained authorization, and consistent policy enforcement across all components of the solution [3], [10].

The solution can be expanded into the domain of Platform Engineering, forming an additional abstraction layer above the existing DevSecOps architecture. This layer would provide self-service capabilities, governance automation, and developer enablement features, allowing teams to deploy secure, policy-compliant environments autonomously. The transition toward Platform Engineering would transform the current framework into a scalable internal developer platform (IDP), consolidating infrastructure, security, and compliance workflows under a unified operational model [16].

To improve the protection of sensitive data, the implementation of secret management tools at both the platform and cluster levels is essential. Tools such as HashiCorp Vault or Azure Key Vault can securely store, rotate, and manage credentials, encryption keys, and tokens [10].

Another key research direction concerns the deployment of centralized vulnerability management and analysis platforms, such as DefectDojo or ArcherySec [15].

The integrity and authenticity of software artefacts can be further ensured through the use of signing and verification utilities such as Cosign [10].

A promising research direction lies in exploring the integration of Artificial Intelligence (AI) and Machine Learning (ML) models into the DevSecOps ecosystem. These technologies could support predictive threat analysis, anomaly detection, and automated vulnerability classification. Furthermore, Generative AI could assist in automated code review, policy generation, and incident response playbook creation. Investigating AI assisted security practices would contribute to a new paradigm of autonomous and adaptive security management [17].

VIII. Conclusion

This research confirms that integrating security within all stages of the software development lifecycle is essential for modern, DevSecOps environments. Traditional manual vulnerability management is no longer viable due to the scale and complexity of current software ecosystems. The adoption of DevSecOps provides an effective framework for embedding continuous, automated security controls within CI/CD pipelines.

The Proof of Concept demonstrates that automation and tool integration are central to efficient vulnerability management. By orchestrating open-source and commercial tools across build, test, deployment, and monitoring phases, the proposed model achieves both security and operational agility. The results show that automation reduces human error, increases consistency, and enables real-time visibility of security posture without slowing down delivery.

Furthermore, the study highlights that open-source tools, when properly integrated, can deliver enterprise-grade security capabilities, making DevSecOps accessible for startups and smaller organizations. From a theoretical perspective, this research bridges the conceptual gap between DevOps automation and security governance, offering a reproducible model that operationalizes DevSecOps principles. Practically, the solution provides a reference implementation for organizations seeking to adopt automated vulnerability management using open technologies.

In conclusion, this work serves as a foundational starting point for further documentation and research in the field of DevSecOps and automated vulnerability management. The proposed PoC provide a practical reference architecture that can be expanded and refined to address emerging challenges in security automation, scalability, and intelligent threat detection.

References

[1] "Application Development Software – worldwide," *Statista*. [Online]. Available: https://www.statista.com/outlook/tmo/software/application-development-software/worldwide.

[2] "Cost of a data breach report 2025," *Ibm.com*. [Online]. Available: https://www.ibm.com/reports/data-breach.

[3] L. Prates and R. Pereira, "DevSecOps practices and tools," *Int. J. Inf. Secur.*, vol. 24, no. 1, Feb. 2025. doi:10.1007/s10207-024-00914-z

[4] S. D. Quinn, N. Ivy, M. Barrett, and G. A. Witte, "Prioritizing cybersecurity risk for enterprise risk management," National Institute of Standards and Technology, Gaithersburg, MD, Feb. 2025. doi:10.6028/NIST.IR.8286B-upd1

[5] H. R. Kadaskar, "Unleashing the power of DevOps in software development," *International Journal of Scientific Research in Modern Science and Technology*, vol. 3, no. 3, pp. 01–07, Mar. 2024. doi:10.59828/ijsrmst.v3i3.185

[6] J. Peterson, "CI/CD pipeline security: best practices beyond build and deploy," *Cycode.com*, Aug. 2025. [Online]. Available: https://cycode.com/blog/ci-cd-pipeline-security-best-practices/.

[7] M. C. Sanchez, J. M. C. de Gea, J. L. Fernandez-Aleman, J. Garceran, and A. Toval, "Software vulnerabilities overview: A descriptive study," *Tsinghua Sci. Technol.*, vol. 25, no. 2, pp. 270–280, Apr. 2020. doi:10.26599/TST.2019.9010003

[8] E. Iannone, R. Guadagni, F. Ferrucci, A. De Lucia, and F. Palomba, "The secret life of software vulnerabilities: A large-scale empirical study," *IEEE Trans. Softw. Eng.*, vol. 49, no. 1, pp. 44–63, Jan. 2023. doi:10.1109/TSE.2022.3140868

[9] D. Sandilands and M. Lee, "The State of DevOps Report 2024: The evolution of platform engineering is live – get your copy now," *Puppet*, Mar. 2024. [Online]. Available: https://www.puppet.com/blog/state-devops-report-2024.

[10] M. Souppaya, K. Scarfone, and D. Dodson, "Secure Software Development Framework (SSDF) version 1.1 : Recommendations for mitigating the risk of software vulnerabilities," National Institute of Standards and Technology, Gaithersburg, MD, Feb. 2022. doi:10.6028/NIST.SP.800-218

[11] "OWASP DevSecOps Guideline," *Owasp.org*. [Online]. Available: https://owasp.org/www-project-devsecops-guideline/.

[12] "OWASP DevSecOps Guideline - v-0.2," *Owasp.org*. [Online]. Available: https://owasp.org/www-project-devsecops-guideline/latest/.

[13] B. Leshchenko, B. Snisar, A. Stupak, and V. Osadchyi, "Integrating DevSecOps into the software development lifecycle: A comprehensive model for securing containerized and cloud-native environments," in *Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems II*, 2024, pp. 153–161.

[14] T. Lam and N. Chaillan, "DoD enterprise DevSecOps reference design," *Defense.gov*, Aug. 2019. [Online]. Available: https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf.

[15] "OWASP Devsecops Maturity Model," *Owasp.org*. [Online]. Available: https://owasp.org/www-project-devsecops-maturity-model/.

[16] J.-H. Soeldner, L. Ajouaoui, A. Fritzsche, and G. Soeldner, "Platform Engineering for cloud-native organizations," in *Libro de Actas*, 2023. doi:10.4995/BMT2023.2023.16741

[17] M. Bedoya, S. Palacios, D. Díaz-López, E. Laverde, and P. Nespoli, "Enhancing DevSecOps practice with large language models and Security Chaos Engineering," *Int. J. Inf. Secur.*, vol. 23, no. 6, pp. 3765–3788, Dec. 2024. doi:10.1007/s10207-024-00909-w