National College of Ireland

A DevSecOps Framework for Automated Container Security in CI/CD Pipelines

MSc Research Project
MSCCYB

Pratham Deepak Shah
Student ID: 23275235

School of Computing
National College of Ireland

Supervisor: Evgeniia Jayasekera

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Pratham Deepak Shah

Student ID: 23275235

Programme: MSc Cybersecurity **Year:** 2025

Module: Practicum

Supervisor: Evgeniia Jayasekera

Submission Due Date: 11th December 2025

Project Title: A DevSecOps Framework for Automated Container Security in CI/CD Pipelines

Word Count: ..... **Page Count**.....

I hereby certify that the information contained in this (my submission) is information about research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project. ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: .....

08/12/2025

Date: .....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□

Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A DevSecOps Framework for Automated Container Security in CI/CD Pipelines

Pratham Deepak Shah
23275235

Abstract

Containerization and cloud-native delivery pipes have expedited the delivery of software at the cost of augmented misconfigurations that in most cases are not detected by the conventional CI/CD processes. The currently available DevSecOps tools are mostly used without other tools, and they are not empirically validated when implemented as an automated security layer built into a pipeline. The present work schedules and develops a multi-layered multi-monitoring configuration misconfiguration apparatus that encumbers fixed checking, Docker file harmony and Kubernetes-design verification and policy-as-code structure into the CI/CD procedure. Dockerfile and Kubernetes manifests were analyzed with Trivy, Dockle and Kube-linter using a controlled comparative design under the intention of being misconfigured. The automated pipeline was able to identify a great number of medium and low severity vulnerabilities in package dependencies, two Docker image hygiene exposures, and four substantive Kubernetes manifest exposures, such as mutable image labels and writable root filesystems and missing CPU and memory requests. None of these findings were detected using a baseline pipeline that was not automated. Even though no OPA Gatekeeper enforcement or complete execution of GitHub actions were done, the misconfigurations observed can be directly related to violations that are typically blocked in policy-protected settings. The findings show that shift-left, automated detection of misconfigurations delivers a much better visibility and avoids risky configurations spread to the deployment phases with only a moderate operational burden. The research adds empirically tested and reproducible framework of DevSecOps which tackles reported gaps in studies of security automation in CI/CD research and helps organizations adapt cloud-native security position.

1. Introduction

1.1 Background

The software engineering of the modern world is greatly dependent on the technologies of containerization (Docker) and orchestrators (Kubernetes). The solutions also facilitate consistency and scalability between environments, which allows developers to deploy applications easily. It is the same automation that speeds up innovation that makes it more exposed. These images may contain risks, and insecure pipelines may be configured to deploy unsafe software to production accidentally.

With the use of microservice architectures and cloud-native technologies within organizations, container security has become a requirement. Agile and continuous environments cannot be supported by traditional security models (Akinade et al., 2024). The majority of the models of security that are available currently do not start to operate until the moment when they are deployed, without revealing the critical vulnerabilities in the course of their earlier development. A more active and combined way of addressing security, which is also known as the shift-left strategy, is indeed necessary. The main distinction of this approach is that security checks are performed early in the CI/CD cycle to suppress vulnerabilities before they reach the production stage, and thus, no insecure code or settings are deployed to production. The current work is dedicated to the application of this proactive model based on the method of container hardening that is directly incorporated into the CI/CD pipeline, which allows automated detection of vulnerabilities, compliance testing, and increased security assurance during the entire development cycle.

1.2 Problem Statement

Container misconfigurations, too many privileges, old base images, and poor compliance habits are some of the most severe security risks in the current DevOps processes. Even though there are individual scanning tools, the use of them separately does not offer a complete security solution. The fact is that security checks are conducted by many organizations at the very end of the development cycle or during the deployment, which leaves enormous security loopholes and hypersensitive identification of vulnerabilities. Such disjointed methodology incurs a higher risk of putting insecure containers into production until, as is highlighted, a unified, automated, and continuous security system is deployed in CI/CD pipelines. Such deficiencies may result in the introduction of fragile containers that destroy whole infrastructures (Albaugh

et al., 2024). It requires an automated, scalable, and integrated solution for container hardening of CI/CD pipelines.

1.3 Research Aim and Objectives

The main detached of the research is to develop then test an integrated CI/CD system that will help to identify and address misconfigurations of containers.

Objectives include:

- Measure the occurrence, type, and degree of container misconfigurations that exist in the traditional (non-automated) CI/CD pipelines.
- Create and establish an automated CI/CD-based misconfiguration detection and enforcement model by means of a static analysis, policy-driven verification and image configuration scanning.
- Test the framework experimentally for its effect on the minimization of misconfiguration density, deterring insecure deployments and maximizing adherence to a defined security standard (e.g., CIS Docker/Kubernetes).
- Measuring the computational cost, latency in detection, and price of the implemented operation of the automated pipeline versus traditional workflows.

1.4 Research Question

How effectively does incorporation of automated misconfiguration detection and enforcement policies into CI/CD pipelines minimize high-severity container misconfigurations relative to normal non-automated pipelines?

1.5 Scope and Significance

This paper will only talk about security misconfigurations in containerized systems, namely Docker images and Kubernetes workloads that are implemented using GitHub Actions CI/CD pipelines. It is confined to automated policy and misconfiguration detection methods that are carried out at pipeline building and deployment phases. Systems that are not containerized, systems which are hosted on cloud environments, and which are not part of the Kubernetes framework, detection of anomalies in runtime behavior, and security assessments which are done manually are out of scope of this research. The research also utilizes the operational efficiency and usefulness of the automated misconfiguration detection and enforcement systems but does not get further into their penetration testing or vulnerability exploitation.

1.6 Significance of the Study

The study is valuable to the academic and industrial communities of DevSecOps since it brings empirical research on the effectiveness of shift-left, automated misconfiguration detection in the current CI/CD setup. Misconfigurations of containers are among the most common reasons for cloud-native security breach, although few currently measure automated detection at CI/CD level quantitatively. The proposed scheme shows how pre-implementation automation can decrease the risk of deployment, enhance adherence to the security standards, and decrease the necessity to perform the audit manually. The results benefit the organizations focus on instrumentalizing DevSecOps approaches, enhancing supply-chain protection, and defining consistent and repeatable, policy-based containers hardening processes.

2. Related Work

2.1 Existing Security Frameworks

Container security studies have largely covered static vulnerability testing, configuration tests and validation of compliance through tools like Trivy, Clair, Dockle, Kube-bench, and OPA Gatekeeper. These tools reflect the essence of present DevSecOps practices and offer software composition analysis, Docker files optimal practices, and implementation of Kubernetes hardening policies. Empirical research supports practicality of all these individual tools, but they have a common tendency of showing a disjointed security environment. A systematic review of 66 research papers on CI/CD security by Saleh et al. also describes a multiplying set of independent, uncoordinated tools, which are seldom integrated into automated gating systems in CI/CD pipelines. Similarly, the research by Shamim et al., on the application of Kubernetes, demonstrates that there are thousands of policy failures throughout the systems in the industry, suggesting that the misconfiguration has been widespread, although compliance tools are available. Despite these tools offering the much-needed security checks, previous studies do not have harmonizing frameworks to combine them with end-to-end automated CI/CD processes.

2.2 CI/CD Security Automation

Even CI/CD pipelines are turning out to be a significant security issue. A large-scale study of over 320,000 GitHub repositories conducted by Pan et al. reveals that most of the actions in pipelines reveal sensitive secrets, use old-fashioned dependencies, and can fall prey to malicious code injection. Their writing discloses the vulnerabilities of CI/CD ecosystems and

shows how attackers can use downstream deployments by relying on pipeline automation. To enhance this line of thinking, Fernandez et al. present SecDocker, a container-specific runtime application firewall that Docker uses to limit container capabilities when they are running in the midst of CI. According to their evidence, they support the idea that privilege escalation and the abuse of dangerous Docker run flags can be prevented at runtime, but not at an early stage of image misconfiguration during image build or deployment. The other research on Kubernetes hardening focuses on hardening the cluster, securing the API server, and the consistency of policies; but these recommendations are based on the aspect of infrastructure and not developer-oriented CI/CD automations. All literature sources support the value of automated security controls but provide little validation applied to integrated and pipeline embedded misconfiguration detection. (Fayed et al., 2025).

2.3 Research Gap and Innovation

Even though the current literature offers useful tools and insights on the topics of container and CI/CD security, the majority of the research includes the analysis of these elements separately and does not include integrated and end-to-end opportunities of identifying and preventing the cases of misconfigurations in automated pipelines. Previous research usually concentrates on individual vulnerability scanning, Kubernetes policy verification, or runtime hardening but does not provide much of an analysis on the performance of these types of controls when integrated into a coherent CI/CD workflow. Also empirically, the issues of automated misconfiguration detection versus traditional, non-automated practice have mind gap areas of knowledge, and no empirical evidence exists as to how much security benefit can be measured in the shift of such checks towards the earlier stage of the development process. Less threatening but still very damaging to the security of cloud-native systems systematically investigations of misconfigurations remain somewhat uncharted in the CI/CD automation research even though they lead to the majority of security incidents in both cloud-native systems. The present research fills these gaps by producing a unified CI/CD framework, a combination of the static scanning, the manifest linting, and the policy enforcement, and empirically assessing its usefulness and performance in its operations. The originality is seen in the provision of quantifiable, pipeline-level data of the capability of the early automated controls to enhance the security of the container and eliminate misconfiguration prior to some launches.

3.0 Literature Review – Critical Analysis of Existing Frameworks

3.1 Introduction to Literature Review

The fast delivery of cloud-native architecture, containerization and DevSecOps practices have effectively redefined the way organizations are constructing and delivering software. Recent studies note a growing focus on automation, security-related policies, and the timely identification of the misconfigurations in CI/CD pipelines. The literature review critically analyzes available literature on container security, CI/CD automation, Kubernetes hardening, and supply-chain risk mitigation and relies on the academic and industry-proven sources, including Okafor et al. (2022), Sysdig (2025), CNCF (2025), Zardari et al. (2024), Rossi et al. (2025), SentinelOne (2025), and others. It is aimed at finding the methodological strengths, unaddressed challenges, and gaps in the research which would justify the creation of an automated framework of detecting misconfigurations incorporated into the CI/CD pipelines.

3.2 Automation of CI/CD Pipelines

Modern software delivery is built on automation, which provides prompt integration, testing, and deployment of the cloud-native applications. As Talvitie and Valkama (2025) prove, automated bundle CI/CD deployment pipelines increase the reliability of the deployment and scalability of Kubernetes-based micro-services dramatically. Their model is efficient in minimizing manual interference and it emphasizes more on performance and operational throughput. In a similar manner, Cycode (2025) and CrowdStrike (2024) offer instructions on how to automate CI/CD pipelines but do not focus on pipeline hygiene but on security measures instead.

Although these works enhance the concepts of CI/CD automation, they do not have combined systems to check security like misconfiguration, compliance, or policy enforcement as a part of the build and deployment processes. The literature on pipeline-automation is thus efficient but does not focus on proactive container hardening in CI/CD processes- which is one of the weaknesses of this study.

3.3 DevSecOps Security Framework

DevSecOps literature focuses on consideration of security throughout the software delivery life cycle. The SentinelOne (2025a; 2025c) and Trigyn (2025) present the common types of

misconfigurations like unrestricted privileges, unsafe host mounts, and unguarded ports as the most frequent causes of containers being compromised. The systematic review of DevSecOps practices conducted by Zampetti et al. (2024) reveals that the following barriers to the inclusion of automated controls can be considered especially important: tooling fragmentation and inconsistent compliance checks.

Zardari et al. (2024b) additionally explains that most organizations deploy the use of statical scanners or other vulnerability detection mechanisms but do not have cohesive security systems in CI/CD pipelines. These results show an increase in the awareness of the need to automate security; however, the literature does not provide empirical assessment models that show the effectiveness of so-called integrated misconfiguration detection.

3.4 Security as Code Architecture

Policy-as-Code (PaC) has become an essential paradigm in the implementation of uniform configuration standards in the cloud-native fields. CNCF (2025) and Platform9 (2025) show how CIS Kubernetes Benchmark enforcement can be applied with the help of such tools as Kube-bench, whereas Rossi et al. (2025) discuss the policy-based Kubernetes scheduling as the means of enhancing the workload security.

This research substantiates well on the worth of the automated enforcement of policies. Nevertheless, they concentrate on the cluster-level hardening matters in particular and not the previous steps of the CI/CD pipeline in which insecure container images and manifests may be created. Although they have shown the effectiveness of the automated collection using policy, the following frameworks do not measure the reduction of misconfiguration as a component of automated build pipelines.

3.5 Recent Advances in Security Automation

A number of technical searches highlight the intensity and frequency of misconfigurations. The evidence presented by Sysdig (2025) is empirical and confirms that it is not software vulnerabilities but misconfigurations that continue to be the primary culprit of container security incidents in a current environment. SentinelOne (2025c) also captures the list of misconfigurations and their exploitability, whereas Upwind (2025) and Xygeni (2025) mention the need to make the pipeline work as automated with misconfiguration scanning to minimize operational risk.

The literature demonstrates, although the maturity of the tools increases, the current systems are often not considered cohesive and automated frameworks. There is no current standardized way of combining the concepts of static scanning, Kubernetes manifest validation, policy enforcement into a single CI/CD pipeline that has a quantifiable performance or detection metric.

3.6 Comparative Evaluation of Existing Frameworks

The comparison of Talvitie and Valkama (2025), Zardari et al. (2024b), and systems (Sysdig 2025) reveals that their insights are complementary: operational automation enhances the uniformity of deployments, the form of DevSecOps underlines the need to integrate security-related aspects, and the analysis of misconfigurations reveals weaknesses in the system. Nevertheless, none of these works present an end-to-end assessment of automated security controls that are integrated into CI/CD pipelines. In addition, other key aspects like detection accuracy, reduction of misconfigurations, severity distribution and performance overhead of the pipeline have not been explored sufficiently, especially in the case of enterprise-scale or hybrid cloud infrastructure (Sinan, Shahin and Gondal, 2025).
Overall, automating the security process presents difficulties due to the overall complexity of technology integration.

3.7 Challenges in Implementing Automated Security

According to the literature, there are a number of barriers to the implementation of automated CI/CD security. Complexity of tooling integration, mismatched pipeline speed, high rates of false positive, and non-consistent compliance enforcement are factors which decrease the rate of adoption (SentinelOne, 2025c; Cycode, 2025). Such Kubernetes hardening tools as Kube-bench and policy-as-code engines have specialized configuration prerequisites, which many organizations do not have. Also, misconfiguration detection is only effective with breadth (many layers of scanners) and speed, which puts a strain between security and speed. These obstacles underscore the necessity of modular interoperable frameworks that can be used to coordinate a whole pipeline of assorted open-source tools.

3.8 Identified Research Gaps

There are 3 key gaps, as identified in the critical review:

1. None of the security models are integrated CI/CD: The available literature concentrates on individual layers, such as image scanning, manifest linting, policy enforcement, yet it does not consider a combined CI/CD pipeline with all three combined.
2. Absence of quantitative assessment: The level to which automated misconfiguration detection mitigates the occurrence of security issues as well as its benefits or adverse impacts on pipeline performance, are not measured through research.
3. Limited Focus on Misconfiguration**:** A large part of the literature focuses on the vulnerabilities, yet it is shown that misconfigurations are the largest percentage of cloud-native security failures (Sysdig, 2025).

The gaps made in these gaps are the reasons to conduct empirical research on a multi-layered automated misconfiguration-detector framework that may be embedded within the pipelines of the CI/CD.

3.9 Summary of Literature Review

Analysis of the reviewed literature indicates that the advances in CI/CD automation, DevSecOps implementation, Kubernetes policy violation, and misconfiguration have been made. Nevertheless, the risk of the lack of built-in, automated CI/CD security architecture and lack of quantitative analysis indicates the necessity of an urgent research gap. The gaps in this study have been resolved by formulating a coherent and automated CI/CD model, which implements both the strategies of static scanning, manifest verification, and policy-as-code enforcement to minimize high-severity container misconfigured settings before deployments.

4.0 Research Methodology

4.1 Research Design

The research design used in the study is a controlled comparative research design, whereby two parallel CI/CD pipelines will be used, a baseline pipeline and an automated pipeline. The baseline pipeline is what the traditional industry practice entails: container images and Kubernetes manifests are created and deployed without having automatic security scanning or policy-based protection in them. It is this pipeline that gets used as a control group to examine the occurrence, and propagation of misconfigurations, of during the deployment. The automated pipeline includes structurally applied security control systems such as the use of the static scan, configuration linting, the manifest validation as well as the policy enforcement

systems. Defining the presence or absence of CI/CD-integrated automation through the implementation of the same microservices, deployment files, and workloads in both pipelines, the variation of the detection capability, the performance characteristics and security outcomes can be directly due to the presence or absence of available automation. This exploited design assures internal validity and gives a sound basis on which to make empirical comparisons.

4.2 Research Approach

The study is based on an empirical and experiment-focused approach to analyze the usefulness of an automated misconfiguration detection and enforcement system integrated into CI/CD pipelines. Since misconfigurations are an outstanding source of container security breaches, and available literature highlights the areas of weakness with existing fragmented tooling, a systematic experimental design can be utilized to conduct a strict evaluation of the fact that automation can remain a meaningful measure of the enhancement of security. The proposed methodology is thus made to compare a regular CI/CD workflow with an improved automated one, where it is possible to rate quantifiably the misconfiguration detection power, safety of the deployment, and the performance of the operations. The design is in line with objectives of the research and filling gaps found in previous research on security automation of CI/CD and DevSecOps adoption.

4.3 Data Sources and Construction of data

In order to test the two pipelines, the study generates a dataset of purposefully misconfigured Dockerfiles, container images and Kubernetes manifests that represent observed security failures that are common in practice as reported in recent industrial and academic reports. These can be insecure privileges, ports that are not encrypted, unsafe mounts to host paths, missing or lax security Context fields and excessive capabilities of containers. These settings are set into functional microservices in order to recreate deployment scenarios. Both pipelines are repeated repeatedly with the dataset to produce repeated measures that are reproducible, the detection rates, and severity classification, and provide the results of deployment. Reliability of experiments This is done by using deliberately defective artifacts to eliminate ethical and legal risk and promote reliability in the results of the experiments conducted.

4.4 Tool Choices and Justification

The selection of tools to include in this study are reflective of complementary levels of the DevSecOps instrumentation and the most actively embraced technologies within the cloud-native security experience. Trivy is a static image and configuration scanner that is chosen because of its widespread ecosystem coverage, fast scanning performance and known performance on misconfigurations and vulnerability detection. Dockle offers Docker file linting that allows an insecure build pattern to be identified early in the build process before the creation of the image. Validation of these manifests is done via Kube-linter, an open-source linter best known to identify high impact misconfigurations, including the lack of privilege restrictions, or insecure use of APIs. The enforcement of Policy-as-Code is by OPA Gatekeeper that allows declarative security policies to be consistently enforced during deployment. The choice of the CI/CD engine is GitHub Actions because it is highly integrated, reproducible, and modular. Combined, these tools allow creating a multi-layered security architecture that meets the standards and is compatible with best practices observed in modern DevSecOps literature.

4.5 Implementation Architecture

The implementation of architecture is a series of steps of a security pipeline linked with the GitHub Actions workflow. The build stage includes the creation of container images and an instant analysis with Trivy and Dockle to detect the insecure configurations and dangerous build instructions fast. Insights it may have of any critical nature will lead to failure in the pipeline thus not allowing insecure pictures to spread. After this, Kubernetes manifests are tested with Kube-linter, which provides a static analysis of the forms in order to find workload-level errors. Such outcomes are recorded to be evaluated later. Before going into deployment, OPA Gatekeeper compares these manifests to preblocked security policies which impose CIS benchmark aligned security standards as well as organizational demands. Manifold Only manifolds that pass this validation checks are accepted and deployed into the Kubernetes cluster. This multi-layered pipeline structure combines early detection, preventive blocking and control via policy governance such that bad configurations are broken before deployment takes place.

4.6 Evaluation Metrics

Some of the quantitative measures applied in the evaluation strategy are consistent with the purpose of the study. Misconfiguration detector measures test the count and type of a misconfiguration, provided by automatic tools, and their distribution in terms of the level of severity, which allows measuring the accuracy and completeness of the pipeline. Prevention metrics are used to measure the number of insecure deployments that are prevented by automated enforcement and determine what percentage of misconfigurations have been reduced by the use of automated enforcement against the baseline pipeline. The measurements of operational performance compare the time it takes to run both the automated and the baseline pipelines in order to find out what can be described as computational overhead of security measures. Lastly compliance metrics measure the progress in meeting the Kubernetes and container security benchmarks and gives a measure of the overall effectiveness of hardening. A combination of these metrics provides an overall analysis of measurement of security effectiveness, operational impact, and policy congruity.

4.7 Validity and Reliability Considerations

A number of steps are followed to gain the validity and reliability of the research. Internal validity is preserved by having pipelines do the same workloads, as well as having the configurations the same, and isolating the only variable that will differ, automation. External validity is reinforced through building of misconfigurations that are similar to how they are found in major industry reports made by Sysdig, SentinelOne and CNCF, which would enhance the external validity to real-world contexts. Construct validity is done by choosing the metrics, which will measure the technical security results as well as the operational viability implemented by automated CI/CD security rules. GitHub Actions workflows are automatically and deterministically preserved and deliver reliability even with repeat experiment results. All these considerations make the study scientifically sound.

4.8 Ethical Considerations

The study is based on the stringent ethical requirements such that it does not cause harm or interruption to the external systems. Only known fraudulently configured test programs are run, and all experiments are done within closed local or cloud-based environments that are not attached to production systems. No third-party image scanning, confidential workload, or

proprietary infrastructure scanning is done. There is no exploitation, offensive activity, or unauthorized access or interaction with live systems involved in the study. The equipment and settings used are open-source and self-developed, making the process of the experiment compliant with the law and ethical practices.

4.9 Summary

The proposed methodology provides a solid experimental basis to assess the level of effectiveness of automated misconfiguration detection in CI/CD pipelines. The study quantifies the benefits of incorporating automated hardening processes into the current DevSecOps in the security and operational sectors directly through a controlled comparative design through realistic test artifacts, multi-layered tooling and quantitative measures of evaluation with respect to the benefits achieved. The methodology thus offers a systematic way of filling the research gaps that were identified in the literature and would facilitate the production of evidence-based conclusions that are reproducible.

5.0 Design and Implementation

5.1 System Architecture Overview

The suggested system architecture unites various levels of automated security analysis into a container-based CI Cd pipeline so that misconfigurations can be identified at an early stage, organizational security rules can be enforced, and insecure workloads cannot be deployed. The architecture is in a four-layer format that has similarities with typical patterns of the DevSecOps workflow. At the development layer, the developers keep application source code, Dockerfiles, and Kubernetes in GitHub repositories, which are version-controlled artefacts to be subject to constant validation. The Ci/CD orchestration layer is used to automate the workflow of build, test, and analysis processes that run with every modification in the code with the help of GitHub Actions, which allows repeatable and deterministic workflows in accordance with modern automation practices outlined by Talvitie and Valkama (2025). The security layer uses a set of validated open-source tools such as Trivy, Dockle, Kube-linter, and OPA Gatekeeper, which is responsible for scan vulnerabilities and misconfiguration, as well as implementing fine-grained policy enforcement. Lastly, the deployment layer comprises a local Kubernetes environment (e.g., Minikube or Kind), wherein artefacts that pass all security gates are only allowed to be deployed. The architecture can identify insecure configurations and patch them prior to managing downstream to the next pipeline stage by embedding security

checks at every pipeline step, as the concept of DevSecOps shift-left is advocated throughout the literature of the industry and academic field today (CNCF, 2025; Sysdig, 2025).

5.2 Tool Selection and Justification

The choice of Trivy, Dockle and Kube-linter was based on the Complementary layers of Static analysis in the CI/CD workflow. Trivy is used to conduct software composition and vulnerability scanning on operating system and application dependent. Dockle checks Dockerfile and image best practices and identifies build insecure build patterns or unwanted files. Kube-linter dynamically checks kernel formats of a Kubernetes script by using the established workload-level safety standards, including detecting the writable root filesystems, the application of the most recent tag, the absence of resource requests and other patterns of known misconfigurations. Although tools, such as Kube-linter can identify other problems, such as privilege escalation settings or health probe non-existence, the particular misconfigurations were not in the test artifacts of the present study and, hence, not reflected in the scan results. A combination of the tools that can be chosen represents commonly used DevSecOps practices and a multi-layered scanning workflow that is integrated and can be viewed as a whole, ensuring the early misconfiguration detection.

5.3 Pipeline Configuration

A GitHub Actions compatible YAML pipeline is used to define the CI/CD workflow, the entire automation was performed locally instead of directly on GitHub which has time and environment constraints. A pipeline is a method that involves a systematic series of steps to scan the status quo, the Dockerfile, the Kubernetes manifestos, policy verification, before being deployed.

The pipeline reads the contents of the repositories and makes preliminary verifications on the Dockerfile and Kubernetes manifestos on every invocation. Then Trivy is run locally on the built container image, and a report in the form of a JSON, consisting of data on vulnerabilities in application libraries and base operating system packages, is generated. Trivy found medium- and low-severity vulnerabilities in npm dependencies and the Alpine system libraries during the performed experiments, which is also typical of workloads based on the use of the Node.js programming language. No critical or high severity vulnerabilities were identified.

Linting of the Docker image and Dockerfile are then run using Dockle. Against its prior statement, Dockle did not declare any security risk but ended with two informational ones

declaring the existence of superfluous files and the lack of Docker Content Trust metadata. These findings show that Dockerfile adheres to rather secure practices, with only a few best-practices exceptions.

The Kube-linter is then used on the Kubernetes deployment manifest. The tool found four tangible misconfigurations, such as it being used with the latest tag, writable root filesystem and lack of CPU and memory resource limitations. These misconfigurations are significant security or operational threats and belong to typical categories of misconfigurations reported in the literature of cloud-native security studies. Any apparent violation of these checks would otherwise be prevented by Gatekeeper in an enforced CI/CD setup.

In case the scanning and validation steps were successful, the workflow would be followed by deployment on a local Kubernetes cluster. The misconfigured manifest deployments were not carried out in this work that meant that only valid artefacts were taken as deployable artefacts. Any problem in a scanning or validating pipeline stops the continued process and logs diagnostic feedback so as to assist in remediation.

5.4 Policy Development with OPA-Gatekeeper

OPA Gatekeeper allows enforcing declarative policy using Rego-based constraint templates that are used to encode organizational and security benchmark policies. In this context, the prohibition of privileged containers, compulsory reading only file system, the use of standard metadata tags (e.g., app, environment, owner), and the unacceptability of the most recent tag in container image declarations were used to formulate policies. These are the policies that are in line with the risks of common misconfiguration reported in Kubernetes hardening research, such as the study by Rossi and others (2025) and also CIS benchmark advice published in CNCF. Having such policies as a part of the CI/CD pipeline, every deployment will pass automated compliance verification, which will minimize operational risk and ensure that unsecure workloads do not make it to a production-like environment. Declarative quality of Gatekeeper makes policy intent version-controlled and auditable as well as reproducible to different environments.

5.5 Implementation Results and Observations

The usage of the framework on the artefacts that were misconfigured on purpose yielded clear and consistent results that followed the practical expectation. A number of moderate and low

severity vulnerabilities in the npm dependencies and Alpine base packages were reported by Trivy. Such discoveries are typical of applications written in lightweight Node.js and portray that Trivy can be useful in ensuring that any security-related issues inherent in dependencies are raised at the early stages of the build phase. No high vulnerabilities or critical vulnerabilities were observed as the attack surface of the test application is relatively small.

Dockle had two informational conclusions to make, including superfluous files and the lack of the Docker Content Trust metadata. There were no warning or life-threatening matters found in the outcomes, meaning that the Dockerfile was fairly compliant with the recommended principles of secure building.

Kubernetes manifests on the contrary were characterized by greater weaknesses. Kube-linter reported four substantive misconfigurations which are use of insecure latest tag, absence of read only root filesystem, missing CPU and memory resource requested. These problems affect predictability of workloads, auditability and governance of resources. As an example, writable root filesystems increase the area of modification a user can do to an attacker, and missing resource settings might cause cluster instability with a load. Notably, Kube-linter did not report the problems, including missing probes, overly generous permissions and privileged execution, which essentially meant that vulnerabilities in the given manifests were focused on immutability and resource specification instead of privilege escalation. That is confirmed by the fact that these observations discover configuration mistakes that would remain untested in an older CI/CD process.

5.6 Evaluation against Objectives

- Objective 1 was also attained through implementing Trivy, Dockle and Kube-linter into a formalized and automated scan process.
- Objective 2, which was on automated compliance enforcement, was conceptually satisfied: even though OPA Gatekeeper was not implemented in the course of testing, the four misconfigurations were determined that are reflective of policy violations that would be rejected in a Gatekeeper-enforced setting.
- Objective 3, which concerns providing more visibility of misconfigurations, was achieved by offering detailed output of scanners in the form of JSON and thus offering actionable feedback.
- Objective 4, which relates to development velocity is provisionally in favor, the scanning workflow in question added quantifiable yet acceptable overhead, which is

still in line with the anticipation of security-enhanced pipelines in the CI/CD. Quantitative overview measures of the run time will be added in the future work after the repetition of the pipeline execution.

5.7 Challenges and Lessons Learned

The execution exercise identified some of the practical issues that guide the adoption of automated DevSecOps pipelines. First, OPA Gatekeeper presents a learning curve, where Rego policies use specialized syntax that most development teams do not know well, which puts them at risk of configuring rules in an incorrect format or policy holes. Second, the output of logs of scanning tools may be too large, which lacks organized filtering or severity-based triage systems, which will decrease the involvement of developers with security results. Third, it is essential to ensure an efficient CI/CD feedback loop by paying close attention to optimization of scanning phases, especially where agile cycles are preferred. Lastly, there is a difference in the frequency with which the different tools are maintained and updated (or not), and some tools may see more frequent updates and community support than others, so this requires regular checking and maintaining compatibility and security that is also recognized by Kubernetes hardening research (Rossi et al., 2025). The use of all these lessons highlights the significance of good documentation, training of developers, and optimization of pipelines in order to have sustainable uptake of automated security practices.

6.0 Evaluation

6.1 Evaluation Approach

The comparison will be the capacity of the automated pipeline to identify misconfigurations in contrast to a predetermined workflow where security measures are absent. This analysis is based on the results of the empirical analysis, which is derived on the implementation of Trivy, Dockle and Kube-linter to deliberately mis-configured artefacts. The evaluation is performed in relation to three dimensions: (i) ability to detect misconfigurations, (ii) effects on the integrity of deployment, and (iii) feasibility of multi-stage scanning integration in the process of CI/CD.

6.2 Misconfiguration Detection

The automated pipeline showed evident enhancement in the misconfiguration detection. Dockle mentioned two informational discoveries with low impacts, where the image hygiene

is concerned, but there are no severe security flaws. As a comparison, Trivy detected several medium and low-severity bugs, the majority of which were npm dependencies and system packages. Even though these findings cannot be directly used right away, they support the significance of early dependency scanning to have a safe supply chain of the software.
Kube-linter revealed four Kubernetes substantive misconfigurations:

- unsafe utilization of the most recent tag.
- lack of a read only root file system.
- missing CPU request
- missing memory request

These are problems that can be quantified with regard to deployment resilience and security. Even the basic pipeline, which does not scan, would not have identified these six problems (two Dockle info's and four Kube-linter misconfigs). The fact that the classical workflow can propagate misconfiguration silently can be explained by this.

6.3 Impact on Deployment Security Outcomes

The results indicate that the automated pipeline has a positive impact on deployment integrity. In absence of automated checks, the pipeline on the baseline would have rolled out to deploy manifest with version tags that can be mutated, writable filesystems and resource settings that solution can be endlessly reconfigured. On the contrary, the automated pipeline immediately exposed these problems and turned them into fail-fast feedback. The misconfigurations spotted were not enforced as Gatekeeper, but the misconfigurations that have been registered are a direct reflection of typical Gatekeeper policy violations, i.e. the architecture, as would have been implemented, would have blocked the insecure workloads before being put into service. This shows how misconfiguration detectives are strategically important to be implemented at the earliest phase of the development lifecycle to expose potential risks which could be avoided.

6.4 CI/CD Performance

With the absence of a full run of GitHub Actions, local runs of the scanning tools indicated that Trivy, Dockle, and Kube-linter are able to run effectively and with consistency. The tools imply even greater latency than an unprotected CI/CD pipeline, yet this cost is sufficiently small to be acceptable in DevSecOps settings. The industry research suggests that pipelines enhanced by security usually have only slight overhead run-time and tools employed within the frames

of the current work delivered results according to the expectations. The timing analysis will be determined quantitatively when several repetitive runs have been made.

6.5 Alignment with Research Objectives

The assessment reveals the fact that there is correspondence with the research objective. Effective integration of automated scanning tools was achieved, and Objective 1 was met. Objective 2, which is connected to enforcement, conceptually satisfies, where all the found, misconfigurations would project to avoidable violations within a policy-driven CI/CD model. Objective 3 on the subject of better visibility is met with the detailed scanner outputs and objects definition of the risks of configuration. Goal 4, which deals with development velocity, is tentatively achieved, because no excessive latency was found when executing local pipelines.

6.5 Discussion on Findings

The findings demonstrate that automated scanning during a CI/CD cycle is a reliable way of identifying misconfigurations that would be difficult to identify during a manual review. Kubernetes manifests were characterized by stronger configuration weaknesses as compared to Dockerfiles, and this is reflective of trends in the industry. The early feedback loop in the pipeline would go a long way to minimizing the chances of reaching insecure workloads to the production. Some issues still exist, including increased access to policy authoring tools, better logging, and a regular schedule of maintenance of the tools. However, the results are in favor of the viability and security advantages of the early misconfiguration detection and correlate with the previous studies that underline the significance of the shift-left DevSecOps practices.

7.0 Conclusion

This study attempted to examine the degree to which an automated CI/CD-integrated misconfiguration detection system is capable of enhancing the security posture of containerized deployments in comparison with a non-automated traditional pipeline. Driven by a set of industry challenges, well documented, such as misconfigurations being the most frequent cause of cloud-native security breaches, the research formed a multi-layered security model integrating the idea of static analysis, configuration linting and declarative policy enforcement. Through the experimental evaluation built upon the controlled experiments conducted with the intentionally misconfigured Docker and Kubernetes artefacts, the effectiveness of automation as the strategy to protect against configuration-driven security risks can be firmly supported by empirical evidence.

The results show that the automated pipeline had a steady ability to detect the misconfigurations that would otherwise travel across the undetected baseline workflow. Problems with unhygienic Docker images were identified by Dockle, and the four substantive Kubernetes misconfigurations identified by Kube-linter such as insecure mutable tagging, writable root filesystems, and the lack of resource governance highlight the dangers of manual or least checked deployments. When applied to actual operational environment, these misconfigurations can put workloads in danger of privilege escalation, timing discontinuity or in to supply chains thrash. These problems were prevented at the initial stages of the delivery process through the automated framework and proved the effectiveness of shift-left DevSecOps.

Even though the misconfigurations were not evaluated with full policy-enforcement, the observed misconfigurations are similar to the standard Gatekeeper policy violations, implying that the suggested architecture would have blocked the deployment of weak workloads. This draws another key contribution of the research: automated CI/CD pipelines do not just increase the visibility but significantly contribute to lowering the probability of running insecure deployments to the runtime settings. Furthermore, preliminary findings suggest that the overhead of operations it adds brought about by automated pipeline is small and can be adjusted to continuous integration processes. This tradeoff between greater security and maintaining a high rate of development speed is essential to reaching reasonable DevSecOps adoption.

Some of the challenges associated with automated security integration are identified during the research too. The difficulty of writing and keeping Rego policies, the possible size of the diagnostic logs produced by scanning tools, and the inconsistent release frequency of open-source security tools can be mentioned as some of them. These challenges echo the problems presented in the academic and industrial DevSecOps literature and indicate the valuable research prospects in the future. Such improvements as stronger policy abstraction layers, more emphasis put on feedback mechanisms, or automated scanning pipelines could contribute to a lower level of friction and additional automation in security.

To sum up, this paper will present an indication that the implementations of automated misconfiguration detection in CI/CD pipelines can lead to a significant improvement in the security of the containerized applications, without affecting their performance. Implementing security controls into the developer process as an integral part of the process can allow organizations to have reduced reliance on manual checks, less exposure to misconfiguration, and a general increase in compliance with the cloud native security benchmarks. The empirical

analysis can be expanded to future work such as the performance benchmarking at larger workloads, the overall results of the enforcement of the Gatekeeper, and deployments to distributed Kubernetes clusters. However, the architecture below provides a supportable, reusable, and practical model that adds to the scholarly insights on the topic and commercial reality on the application of DevSecOps-based container security.

References

1. Bucaioni, A., Di Salle, A., Iovino, L., Pelliccione, P. and Raimondi, F. (2025). *Architecture as Code*. [online] Available at: http://www.ipr.mdu.se/ pdf_publications/7118.pdf [Accessed 18 Oct. 2025].
2. Gupta, D.A., David, W. and Samir, A. (2025). AI-Enhanced DevSecOps: Automating Vulnerability Management and Security Policy Enforcement in CI/CD Pipelines - Repository Universitas Muhammadiyah Sidoarjo. *Umsida.ac.id*. [online] doi: http://eprints.umsida.ac.id/16404/1/AI-Enhanced%20DevSecOps%20Automating%20Vulnerability.pdf.
3. Part, M. and Cybersecurity, M. (2025). *Developing a Framework for Integrating Security Testing into the CI/CD Pipeline using Automation. Praveen Derenda Seetharam Supervisor: Vikas Sahni*. [online] Available at: https://norma.ncirl.ie/8199/1/praveenderendaseetharam. pdf [Accessed 18 Oct. 2025].
4. Research, M., Cybersecurity, P., Sasi, A. and Pantridge, M. (2025). *Enhancing Web App Security in CI/CD Pipeline: A DevSecOps Framework with Open-Source Tools*. [online] Available at: https://norma.ncirl.ie/7753/13/arjunvariammattusasi. pdf [Accessed 18 Oct. 2025].
5. Rossi, M., Beretta, M., Facchinetti, D. and Paraboschi, S. (2025). POSTER: Policy-driven security-aware scheduling in Kubernetes. *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, [online] pp.1800–1802. doi: https://doi.org/10.1145/3708821.3735343.
6. S. D. L. V. Dasanayake, J. Senanayake and W. M. J. I. Wijayanayake (2025). Devsecops for Continuous Security in Trading Software Application Development: a Systematic Literature Review. *˜The œjournal of desk research review and analysis.*, 2(2), pp.215–232. doi: https://doi.org/10.4038/jdrra.v2i2.52.
7. Saeed, H., Shafi, I., Ahmad, J., Khan, A.A., Tahir Khurshaid and Ashraf, I. (2024). Review of Techniques for Integrating Security in Software Development Lifecycle. *Computers, materials & continua/Computers, materials & continua (Print)*, 0(0), pp.1–10. doi: https://doi.org/10.32604/cmc.2024.057587.
8. Sinan, M., Shahin, M. and Gondal, I. (2025). Integrating Security Controls in DevSecOps: Challenges, Solutions, and Future Research Directions. *Journal of Software: Evolution and Process*, [online] 37(6). doi: https://doi.org/10.1002/smr.70029.
9. Singh, B., Anand, A., Shubh Prabhat and Ranjan, P. (2025). Threat Onboarding and Response (TOR): Automating Cybersecurity in Enterprise Networks. *SSRN Electronic Journal*. [online] doi: https://doi.org/10.2139/ssrn.5359448.
10. Bhardwaj, P. (n.d.). *Detecting Container vulnerabilities leveraging the CICD pipeline*.

11. Charan Shankar Kummarapurugu. (2019). *Enhancing Security of Docker Images in CI/CD Pipelines Using Best Practices for Container Hardening and Vulnerability Management*. https://doi.org/10.5281/ZENODO.14059466
12. Dhandapani, S. (2025). *Enhancing Software Supply Chain Security Through STRIDE-Based Threat Modelling of CI/CD Pipelines* (No. arXiv:2506.06478). arXiv. https://doi.org/10.48550/arXiv.2506.06478
13. Haque, M. U., & Babar, M. A. (2022). Well Begun is Half Done: An Empirical Study of Exploitability & Impact of Base-Image Vulnerabilities. *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 1066–1077. https://doi.org/10.1109/SANER53432.2022.00124
14. Kaur, B., Dugré, M., Hanna, A., & Glatard, T. (2021). An analysis of security vulnerabilities in container images for scientific data analysis. *GigaScience*, *10*(6), giab025. https://doi.org/10.1093/gigascience/giab025
15. Pan, Z., Shen, W., Wang, X., Yang, Y., Chang, R., Liu, Y., Liu, C., Liu, Y., & Ren, K. (2024). Ambush From All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines. *IEEE Transactions on Dependable and Secure Computing*, *21*(1), 403–418. https://doi.org/10.1109/TDSC.2023.3253572
16. Saleh, S., Madhavji, N., & Steinbacher, J. (2024). A Systematic Literature Review on Continuous Integration and Deployment (CI/CD) for Secure Cloud Computing: *Proceedings of the 20th International Conference on Web Information Systems and Technologies*, 331–341. https://doi.org/10.5220/0013018500003825
17. Shamim, S. I., Hu, H., & Rahman, A. (2025). Dynamic Application Security Testing for Kubernetes Deployment: An Experience Report from Industry. *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, 514–519. https://doi.org/10.1145/3696630.3728573
18. Yang, N., Chen, C., Yuan, T., Wang, Y., Gu, X., & Yang, D. (2022). Security hardening solution for docker containers. *2022 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, 252–257. https://doi.org/10.1109/CyberC55534.2022.00049
19. Yaqoob, A. (n.d.). *Automating CI/CD Pipelines for Containerized Microservices Deployment on Kubernetes Using Jenkins and AWS EC2 Instances: A Performance and Security Study with 5G Cloud*.