National College of Ireland

A DevOps-Based Security Automation Framework Using IaC and CI/CD for Cloud Infrastructure

MSc Research Project
MSc in Cloud Computing

Jiyoung Kim
Student ID: 24142816

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Jiyoung Kim……

Student ID: ……24142816……

Programme: ……MSc in Cloud Computing…… **Year:** ……1……

Module: ……MSc Research Project……

Supervisor: ……Shreyas Setlur Arun……

Submission Due Date: ……11th December 2025……

Project Title: … A DevOps-Based Security Automation Framework Using IaC and CI/CD for Cloud Infrastructure ……

Word Count: ……6,680…… **Page Count**……22……

I hereby certify that the information contained in this (my submission) is information pertaining to 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: ……Jiyoung Kim……

Date: ……09th December 2025……

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 DevOps-Based Security Automation Framework Using IaC and CI/CD for Cloud Infrastructure

Jiyoung Kim
24142816

Abstract

The significance of cloud infrastructure security is on the rise since the use of cloud services constantly increases across both industry and academia. Traditional security practices are usually not able to match the complexity and speed of the modern software devlopment. This architecture achieves integration by bridging Infrastructure-as-Code (IaC) and CI/CD pipelines. It also adopts a "Shift-Left" security methodology that enhances fast and efficient detection and response of a variety of vulnerabilities. Furthermore, this suggested framework aims to reduce manual intervention and to identify cloud security threats by incorporating tools like static and dynamic analysis scanners. The model is validated through a practical implementation, which is based on open-source tools, such as Terraform, tfsec, SonarCloud, GitHub Actions, OWASP ZAP, and OWASP Juice Shop. Specially, the last framework offers a perfect, cost-efficient approach to small enterprises with restricted security budgets, effectively applying DevSecOps principles across the entire development process in a cloud-native ecosystem.

Keywords— DevSecOps, Shitf-Left Security, IaC, Automated Scanning, Cloud Security

1 Introduction

Nowadays, various companies and education corporations are constantly adopting cloud services to accomplish scalability and cost efficiency. Nevertheless, due to the increase in the adoption of clouds, protecting the security of these environments has become a major issue. The risks that can occur to cloud infrastructure are misconfiguration, data breaches, and unauthorized access. The traditional manual approach of the companies makes them susceptible to these risks, and they struggle to handle them properly. The situation is especially acute when a special security team is not available. Thus, the necessity of automated and effective mechanisms to proactively identify and address vulnerabilities in cloud infrastructure is increasing.

This work is aimed at connecting vulnerabilities measurement tools to Infrastructure-as-Code (IaC) administration to create a functional DevSecOps pipeline. The evolution of this framework is based on the exploitation of a number of open-source tools. **Terraform** is used for IaC management, and automated CI/CD workflows are implemented through **GitHub Actions**. In the case of security scanning, tfsec identifies security vulnerabilities. Additionally, **SonarCloud** supports static vulnerability analysis, and **OWASP ZAP** supports dynamic vulnerability analysis. The tools will provide a hierarchical analysis and resolution of security concerns in various levels and allow the detection of vulnerabilities in early and ongoing stages of the development lifecycle.

Research Question and Objectives

The research question posed in this study explores:

To what extent can a DevSecOps-oriented security automation architecture, based on Infrastructure-as-Code (IaC) and CI/CD pipelines, streamline vulnerability detection rates and reduce the manual work in the cloud infrastructure management?

To address the research question, the following specific sets of research objectives were derived:

1. Explore the latest word in DevSecOps, IaC practices, and automate security activities to establish a base of the proposed structure.
2. Create stronger DevSecOps pipeline that involves the use of IaC and automated vulnerability scanners.
3. Implement the suggested DevSecOps framework by configuring open-source tools (e.g., Terraform, tfsec, SonarCloud, GitHub Actions, and OWASP ZAP).
4. Assess the impact of the framework on weaknesses, fewer handwork actions, and further performance.

For evaluation, the OWASP Juice Shop (intentional vulnerable application) will be used as the target system to validate the effectiveness of the proposed framework.

Contribution

The crucial contribution of this study is the affordable DevSecOps framework that does not merely connect the elements but actually unites Infrastructure-as-Code (IaC) and CI/CD pipelines and a continuous, automated testing layer. The research provides quantifiable assessment measures, such as vulnerability detection rates, vulnerability detection times, and less manual intervention. This contribution offers not only scholarly sections but also a feasible roadmap to organizations with weak security standards.

Structure of Paper

Chapter	Title	Primary Focus and Contribution
2	Related Work	Offers a Critical reviews and analysis of prior studies with DevSecOps, Infrastructure as Code (Ia) methodology, and various security automation frameworks.
3	Research Methodology	Outlines the essential research design, which includes the comparative setup for the automated and non-automated workflows.
4	Design Specification	Presents the proposed architecture and specifies all required components. The overall system design is also presented here.
5	Implementation	Explains how to operate open-source tools and how to build the final DevSecOps pipeline.
6	Evaluation	Presents the experimental findings and explains such important metrics as detection rate, runtime performance, and manual effort.
7	Conclusion and Future Work	Serves to conclude the key contributions. It also reviews limitations of this project and suggests additional points for future study.

2 Related Work

This section aims to establish an academic foundation for the core topic of this study, IaC security integration in the DevSecOps pipeline, and ultimately, to validate this research. To this purpose, I have critically chosen 16 important papers published within the last five years and discussed the accomplishments and shortcomings of the existing research. The following discussion is categorized into five main themes, and this analysis will clearly identify the **key research gaps** in current research.

2.1 IaC Security and Static Analysis

Unlike the older versions, Infrastructure-as-Code (IaC) is much quicker to deploy. This IaC exposes code to a wide range of security risks. This is why early security verification, such as Shift-Left, is crucial. **Alonso et al. (2023)** and **Saavedra et al. (2023)** talked that early inspection of IaC code is one of the best options for system safety. They introduced a tool called GLITCH. This tool detects scripting issues, such as code smells. This research demonstrates that static testing (SAST) helps detect some basic issues early.
However, SAST has clear drawbacks. **Chiarl et al. (2022)** found that most SAST tools only verify compliance with rules. SAST tools often miss many types of security errors. **Konala et al. (2023)** also took note of it: static tools cannot identify those issues that emerge only when the system is running. This is why SAST, and dynamic testing (DAST) should be applied simultaneously to provide greater protection.

2.2 CI/CD Threats & Dynamic Testing (DAST)

The pace of CI/CD pipeline is much too fast, honestly. This makes new security blind spots. In fact, many kinds of risks usually occur right from the building and deployment steps**. Pan et al. (2024)** explained serious problems, like old and hardcoded secrets, are very common in open-source projects. It totally looks like the pipeline setup itself is easily a weak point.
We can very catch easily the same types of issue in real industry fields. **Paule et al. (2019)** considered actual industrial systems. The lack of serious security gaps confirmed by both **Pan et al. (2024) and Paule et al. (2019)** was due to poor access controls. Through those, these two papers verified that the security of pipeline is a very real problem for everyone.
We really need Dynamic Analysis (DAST) to handle this situation. In **Rangnau et al. (2020)**, he suggested putting DAST tools right into the CI/CD flows. The step was helpful for checking security. But honestly, this idea is not enough to detect. DAST only checks the final running app and ignores errors that have already been built into the IaC code (Section 2.1). Because of this huge gap, the current research is effectively only half-completed. We must combine both static and dynamic verification.

2.3 DevSecOps Frameworks and Strategy

The security automation is not just simply about linking tools. It needs a strong strategy with the whole organization. **Rajapakse et al. (2022)** stated that many companies face huge

challenges beyond the technical aspects. These challenges include both cultural resistance and **difficulties integrating tools**. Therefore, effective planning is needed, not just simple solutions. Small and medium-sized enterprises (SMEs) also face even mor serious challenges. **Cheenepalli et al. (2025)** observed that SMEs suffer because they have fewer budgets and specialists. **Nawas et al. (2022)** proposed a particular framework of decision-making to resolve these problems. This means that choosing the right strategy should be done long before we write a single code.

Moreover, a literature review by **Ramos & Yoo (SLR, 2025)** supports the idea that automating security processes, especially in conjunction with IaC, is the key point to any system that needs to grow. Despite this excellent blueprint, the gap is huge.

Most frameworks only offer high-level roadmaps. They often fail to provide the details of how they will connect SAST and DAST in their IaC pipeline. It is that missing technical plan we need to provide.

2.4 Automation Challenges and Monitoring

Landuyt et al. (2024) evaluated modeling tools in a CI/CD environment. The paper revealed that automated analysis is unable to encompass all possible threat scenarios. This issue is closely connected with the drawbacks of test automation. According to **Patel et al. (2022)**, the coverage of test automation throughout the DevOps pipeline is still unsatisfactory. Automated security audits do face far more challenges than they should during implementation. Automated security audit completed results are not a realistic expectation on their own.

The more difficult one is after development monitoring. **Díaz et al. (2019)** talked about the need for self-service cybersecurity monitoring in DevSecOps. Elimination of threats is not possible. But existing tools have fundamental limitations in real-time detection, and post-development management. The goal of automation should be risk minimization, and this is why. Existing tools are not easily handled. This leaves development organizations open to the exploitation of critical security vulnerabilities.

2.5 Multi-Cloud and Execution Environment

DevSecOps research on the security has different boundaries on the nature of the environment. This issue highlighted about highly complex ecosystems in research. Indicatively, **Zhang et al. (2025)** identified the need of specialized security solutions to federated cloud-fog infrastructures. Such environments are not conventional cloud systems. On the same note, **Mahboob & Coffman (2021)** also investigated CI/CD pipeline security. They designed a system based on Trusted Execution Environment (TEE) that is specific to Kubernetes.

The issue is not the quality of the research, but its scope. Consequently, solutions are limited to certain cloud platforms or technologies. This lack of broad applicability limits the usefulness of most existing research. This is where the most important and critical research gap arises. Current papers address security issues in a fragmented manner, clearly exposing their limitations.

2.6 Critical Analysis Table

Table 1: Critical Analysis for Related Work

#	Source (Author, Year, Title)	Core Findings	Strength / Gaps	My Conclusion / Notes
1	Alonso et al. (2023) IaC Extension for DevSecOps	Suggest a DevSecOps framework integrating IaC practices.	**Strength**: IaC integration at the high-level. **Gap**: Lack of technical detail (Failure to explain connecting SAST/DAST).	This framework is too abstract and cannot be used for direct technical implementation.
2	Saavedra et al. (2023) IaC Smell Detection (GLITCH)	IaC code smells with GLITCH.	**Strength**: Solves IaC security vulnerabilities directly and at an early stage. **Gap**: Ignores critical runtime configuration errors.	It shows plainly the shortfalls of SAST in the ability to take capture issues of the execution context.
3	Chiari et al. (2022) Static Analysis of IaC	Review static analysis approaches for Infrastructure as Code.	**Strength**: Extensive literature reviewed on SAST tools. **Gap**: Conformity rules with no security vulnerabilities are mostly targeted.	It demonstrates the existing form of rule-based IaC analysis and states the high necessity of DAST integration.
4	Konala et al. (2023) Static Config Analysis in IaC	Classify static configuration analysis methods in IaC.	**Strength**: Scientific knowledge of SAST tools. **Gap**: Static analysis ignores serious runtime problems.	It demonstrates that the use of static methods is half-complete without checks of dynamic nature.
5	Pan et al. (2024) Security Threats in CI/CD Pipelines	Distinguish security threats in open-source CI/CD pipelines.	**Strength**: Verifies real pipeline detects. **Gap**: More concentrated on threats, not provide solution suggestions.	It points out a crucial weakness of CI/CD pipeline setup.
6	Paule et al. (2019) Industrial CD Vulnerabilities	Examine vulnerabilities in industrial Continuous Delivery systems.	**Strength**: Offers evidence to the industry based on vulnerabilities. **Gap**: Not investigate IaC or automated DAST integration.	It contains some applicable information on access control; however, it is too limited to be a contemporary DevSecOps solution.

7	Rangnau et al. (2020) Integrating DAST in CI/CD	Integrate DAST technologies within CI/CD workflows.	**Strength**: Provides step to shift DAST left. **Gap**: DAST ignores IaC build flaws (shortage of coverage).	It proposes only a partial solution and misses the need to check the entire IaC build process for complete security.
8	Rajapakse et al. (2022) DevSecOps Adoption Challenges	Adoption challenges research barriers and solutions to DevSecOps.	**Strength**: Good focus on non-technical barriers. **Gap**: Offers High-level strategies without technical blueprint.	It gives needs of culture (why) and nothing about technical implementation (how) in details.
9	Ramos & Yoo (2025) Cybersecurity in DevOps (SLR)	Audit DevOpsSec cybersecurity practices.	**Strength**: Systematically plots the current research environment. **Gap**: Display the harsh decentralization of solutions.	It maps current DevSecOps practices but reveals that solutions are fragmented and lack a unified model.
10	Cheenepalli et al. (2025) DevSecOps in SMEs	Recognize DevSecOps challenges and best practices in SMEs.	**Strength**: Concentrates on peculiarities of SMEs. **Gap**: Only general best practices and is specific to SMEs.	It recognizes the realistic financial limits that should be considered within a realistic framework.
11	Nawaz et al. (2022) DevSecOps Decision Framework	Propose a decision framework for DevSecOps implementations.	**Strength**: Helps organizations to choose the right strategy early. **Gap**: Only decision-based, the limitation of technical validation.	It is useful for managers but is useless for engineers who need actual technical connection.
12	Landuyt et al. (2024) Threat Modeling in CI/CD	Access the threat modelling tools of CI/CD environments.	**Strength**: Concentrates on moving the threat identification to the left. **Gap**: Automated tools cannot identify all possible threat scenarios.	It understands that automation is not the ideal solution and cannot fully substitute human supervision.
13	Patel, A., et al. (2022) Test Automation in DevOps	Test automation coverage on DevOps pipelines.	**Strength**: Directly handles automation limitations. **Gap**: Test coverage across the pipeline is stall lacking.	It shows clearly the fact that test coverage is a very serious drawback. Framework should address this issue.
14	Díaz, J., et al. (2019) Self-Service Cybersecurity in DevSecOps	Empower self-service DecSecOps cybersecurity monitoring.	**Strength**: Focuses on post-deployment management and self-service. **Gap**: Essential constraints in real-time detection and management.	It lacks post-deployment gap. The monitoring solutions are not yet developed.

15	Mahboob & Coffman (2021) Kubernetes Pipeline with TEE	Develop a secure Kubernets CI/CD pipeline using TEE.	**Strength**: An original application of TEEs to certain security assurances. **Gap**: Solutions do not work universally across all cloud platforms, rather, they only work with Kubernetes.	The study is magnificent, yet the area is too narrow. This solution lacks the much-needed multi-cloud applicability objectives.
16	Zhang et al. (2025) Secure Federated Cloud-Fog	Secure federated learning in cloud-fog infrastructures.	**Strength**: Solves a multifaceted and innovative space (Cloud-fog). **Gap**: Concentrates on only a few specialized areas and is not very useful in general.	This overall situation is too specific to a certain environment. This clearly shows that the current work is fragmented by environment type.

2.7 Conclusion and Research Gaps

The conclusion of all the previous paper mentioned above are clear. The current IaC security approaches are not only severely fragmented bur also inefficient as we examined. Three major unresolved gaps stand out in this process, the first of which is that the **technical verification itself is incomplete**. SAST tools only verify the code, but they can easily miss important runtime issues that occur during system execution. Conversely, DAST can identify mistakes, but it fails to be able to reverse and identify the line of IaC code that generated the mistake. Since the two tools are not adequately linked in this manner, there will be a critical security gap left on the pipeline.

Secondly, **the current founded frameworks are just abstract strategies that have minimal implementation capabilities**. They stated ‘Why’ integration is necessary but fail to specifically show ‘How’ to technical link SAST and DAST for practical implementation. Thirdly, **the applicability of research is excessively specialized**. Even quality research can be confined to narrow environments or technologies (e.g., Kubernetes), which is practically inapplicable in a variety of cloud environments. In conclusion, it clearly indicates that the existing studies are currently divided into limited fields.

The three significant gaps are exactly the reason why our study is vital. This research is going to focus on the solutions for handling these main problems: unconnected and imcompleted validation, abstract strategies, and environmental specialization. We plan to address several basic concerns that have not been adequately addressed in previous studies by proposing a practically usable technical framwork that allows SAST and DAST to be seamlessly integrated throughout the pipeline.

3 Research Methodology

This chapter outlines the methodological approach used in this study. It summarizes that the research design, development process, and related tools and environments. It also introduces the evaluation methodology used to access the effectiveness of the proposed system.

3.1 Research Design

The research follow an experimental design approach to examine the effectiveness of a DevSecOps-based security automation framework. The design was also aimed at supporting a feasible test of the way such as automanted *IaC validation, static analysis, and dynamic analysis* can be used in conjunction in a **CI/CD pipeline**. The study evaluates the behaviour of the framwork in a controlled and high-risk environment by building a working pipeline and testing it on two target systems, an original Django application and the deliberately insecure Juice Shop.

The project is congruent with the research objectives since it gives a quantifiable method of analyzing various *vulnerability detection*, *automation coverage*, and *time-to-detection* at some tools and phases. This method has ensured that the proposed framework is not only analyzed in the light of technical accuracy, but also the light of its application and relevance in the real-life activities of cloud-native development.

3.2 Research Procedure

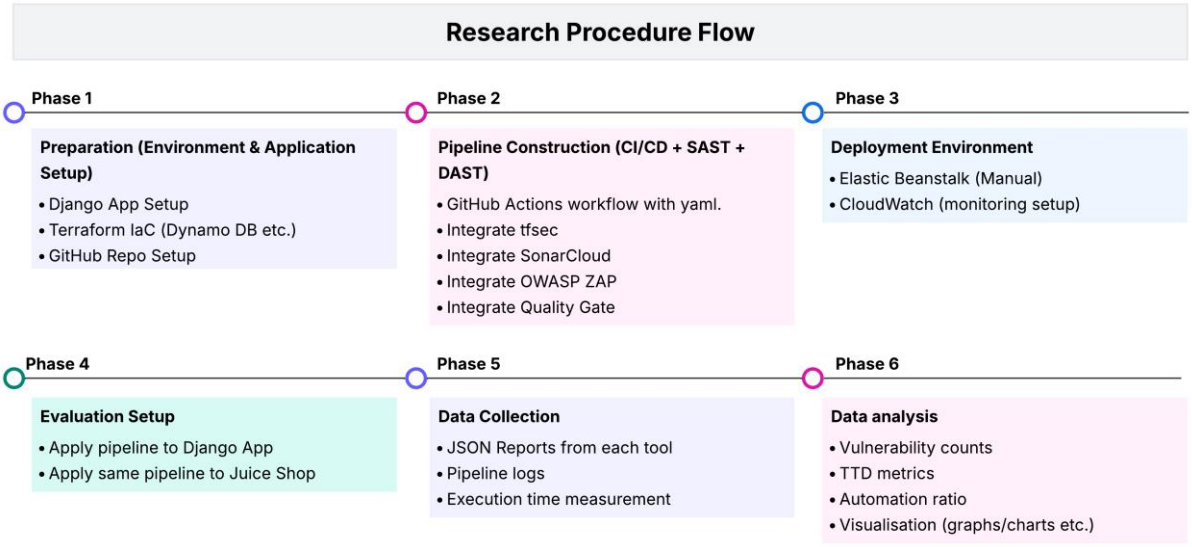


Figure 1. Overall research procedure followed in this study.

This research procedure is consists of six phases to provide a structured and measurable approach for evaluating the proposed DevSecOps framework. Figure 1 illustrates the sequential involved tasks, from first step (environment setup) to final step (data analysis with visualization).

3.3 Tools and Environment

The provided experimental environment is based on the integration of various cloud services, open-source security tools, and development resources. All tools and services are summarized in Table 2. Those are used in the implementation as well as evaluation of the proposed framework.

Table 2. Tools and Environment Summary

Category	Tool/Service	Purpose	Usage Description
Cloud Environment	AWS Learner Lab	Cloud hosting & sandbox environment	It services temporary AWS environment with IAM, S3, DynamoDB, CloudWatch, and Elastic Beanstalk.
IaC (Infrastructure as Code)	Terraform	IaC definition & resource provisioning	It uses to provision DynamoDB tables for storing information and configures necessary cloud resources.
Application Layer	Django	Base test application	It uses a basic registration functionality that is applied in pipeline validation.
CI/CD Platform	GitHub Actions	Automated pipeline execution	It executes IaC checks, analysis phases (SAST, DAST), and quality gate logic triggered on code commits.
SAST (IaC Security)	tfsec	Static analysis for Terraform	It identifies the security problems and misconfigurations in the IaC templates.
SAST (Application Code)	SonarCloud	Static analysis for Django code	It determines hard codes, bugs, vulnerabilites, code smells, and gives the visualization to the metrics of the code quality.
DAST	OWASP ZAP	Dynamic analysis during runtime	It automatically performs security scanning against the running application.
Deployment Platform	AWS Elastic Beanstalk	Application deployment & rollback	It has the Django application and it allows versioned rollback deployments.
Monitoring	AWS CloudWatch	Logs & metrics for monitoring	It gathers operational logs and performance metrics following deployment.
Evaluation Target	OWASP Juice Shop	Vulnerable benchmark application	It involves in testing the performance of the pipeline under high-risk conditions that were deliberate.

3.4 Evaluation Methodology

The proposed framework is evaluated for its vulnerability identification and automation of security processes. Its goal is to design a set of quantitative and qualitative metrics for measuring its effectiveness in detection. This study is consists of two applications used as test targets. One is a simple *Django-based platform*, which represents a controlled environment.

The other is *OWASP Juice Shop* for representing a high-risk scenario. The given dual configuration can examine the pipeline under varying levels of complexity.

The evaluation reports are used to carry out both **quantitative** and **qualitative** evaluations. The quantitative evaluation focuses on quantity of the vulnerabilities, the time of detection, and the proportion of tasks completed without manual intervention. On the other hand, it has qualitative observations. There are considered from various perspectives, such as pipeline stability and the clarity of generated reports to examine the practical usability of the framework. All experiments are carried out under similar conditions, the same GitHub actions and the AWS Learner Lab, to be consistent.

4 Design Specification

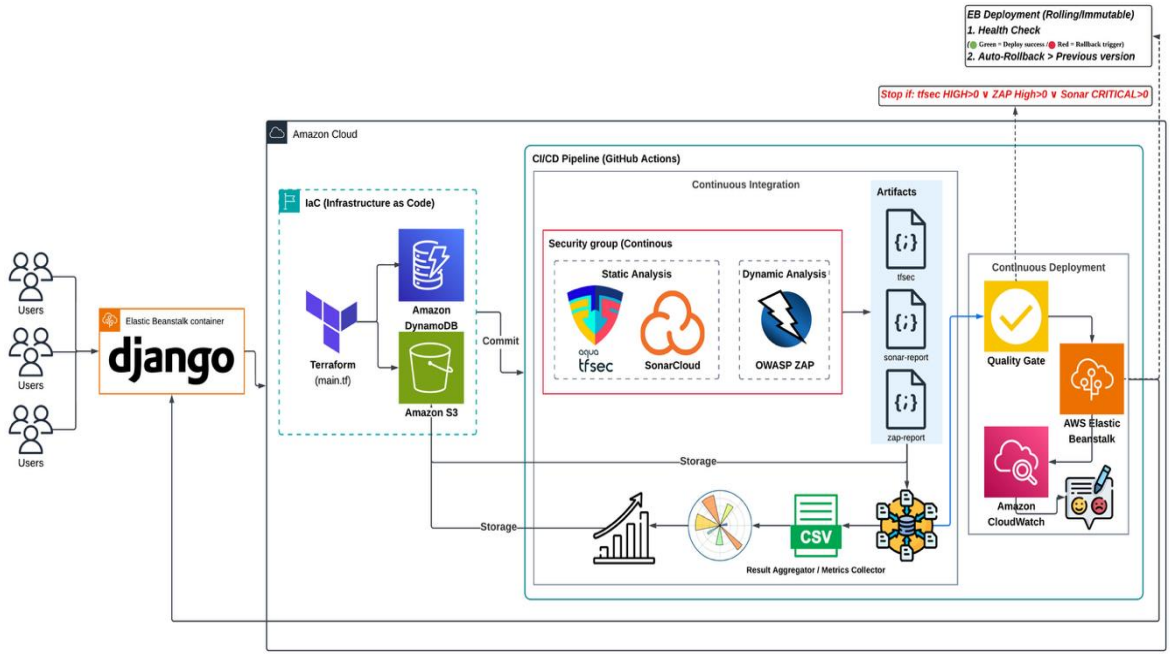


Figure 2. System Architecture

4.1 Overview

This research proposes a **DevOps-based security automation framework**. It combines *Infrastructure as Code (IaC)* with *Continuous Integration (CI)* and *Continuous Deployment (CD)* pipeline to aid the protection of the cloud. The design architecture focus on automating different vulnerability checks and reducing the number of manual security checks through both static and dynamic analysis with direct integration into the deployment process.
As demonstrated in Figure 1, the architecture is consists of four main layers:
(1) an **Application Layer**: It uses *Dango* and *AWS Elastic Beanstalk*.
(2) an **IaC Layer**: It manages by Terraform to provision resources such as *Amazon S3* and *DynamoDB* to store all results.
(3) a **CI/CD pipeline Layer**: It implements via GitHub Actions with a *yaml* file.

(4) a **Security Automation Layer**: It integrates some security tools (*tfsec*, *SonarCloud*, and *OWASP ZAP*).
These components support a fully automated **DevSecOps** pipeline to conduct security validation at all stage of deployment.
This model also consists of a Quality Gate mechanism which is used as a pre-deployment checkpoint. Practically, when a static analysis tool such as tfsec or SonarCloud finds a critical vulnerability, or when OWASP ZAP (dynamic analysis tool) finds out a high-risk vulnerability during the implementation of a pipeline gate, the suggested pipeline halts the practice instantly. In addition, *AWS Elastic Beanstalk* process automated rollback to the latest version. These control steps prevent unsafe version and code from being deployed to production. After the deployment, the system keeps monitoring through *AWS CloudWatch* which collects logs and performance metrics. This collected data helps developers ongoing security detection and safe operation. Additionally, this support completing the feedback loop between deployment and maintenance.
Overall, this architecture provides a practical security automation model to related figures.
It implements IaC control and automated CI/CD pipeline combining Git Actions which allows detection of vulnerabilities early and continuously compared to manual workload. Moreover, this framework shows how to successfully combine open-source tools in order to create a swift and low cost DevSecOps ecosystem in prac

4.2 Requirements and Constraints

This section briefly describes the key requirements for the design and implementation of the proposed framework. These are categorized into three groups namely functional, non-functional, and system constraints.

4.2.1 Functional Requirements

These functional requirements outline the key functions that a framework should offer to facilitate the automated analysis of cloud security.

1. *Automated Security Pipeline*
   The framework creates an automated CI/CD pipeline with *GitHub Actions*. Each time the developers commit and push some code changes or configuration changes into Terraform, a security analysis workflow is automatically executed by the build pipeline without having to be executed manually.
2. *IaC-Based Resource Management with Terraform*
   This architecture uses *Terrafrom* to implement operationally required cloud resources and follows the *Infrastructure as Code (IaC)* principle. The *DynamoDB* table is created using *Terraform* to store user information data for the application. Configuration versions are managed during the static analysis and can be analyzed using *tfsec*.
3. *Integrated Multi-Layered Security Scanning*
   The pipeline should include different security tools that deal with different layers of the system:

1) *tfsec* to check misconfigurations of *Terraform*, 2) *SonarCloud* to analyze the static vulnerabilities, and 3) *OWASP ZAP* to check the dynamic analysis for Django-based application page.
All three tools should execute automatically as a part in the same pipeline.

4.2.2 Non-Functional Requirements

Non-functional requirements stipulate the quality and other features that the structure must possess besides its main functionality.

1. *Reproducibility*
   Regarding the security analysis, reproducibility should be present across all commits. Furthermore, comparing the same version of code and *Terraform* configuration, a pipeline should always give identical results on repetitive execution.
2. *Scalability*
   The framework must be easily scalable to other applications or target systems, such as *OWASP Juice Shop*. Modifications to the current CI pipeline structure and tools should be feasible even with minor changes.
3. *Efficiency and Reduced Work*
   The checks of security by the manual methods should be reduced with help of automated pipeline. Time and effort spent on a static and dynamic analysis, such as the use of *tfsec*, *SonarCloud*, and *ZAP*, should be minimum when compared to the implementation of these tools outside the pipeline.

4.2.3 Constraints

This limitation is the technical and environmental constraints in terms of the implementation and review of the framework.

1. *AWS Learner Lab Limitations*
   This is implemented on the AWS Learner Lab and uses temporary credentials and limited IAM permissions. Consequently, the pipeline is not able to push output information to the *S3 bucket* by using long-term access keys. For this reason, some automation scenarios are unavailable.
2. *CI Runner Resource Limits*
   This framework utilizes *GitHub Actions Runner*, which limits the implementation of time and computation resources. Under these limitations, it may not be possible to have long-term scans.
3. *Tool Accuracy and Coverage*
   The open-source security tools with this framework could produce false positives or false negatives. Moreover, any tools focus on specific vulnerabilities only and thus the security assessment cannot be complete.

5 Implementation

5.1 CI Pipeline Architecture Overview

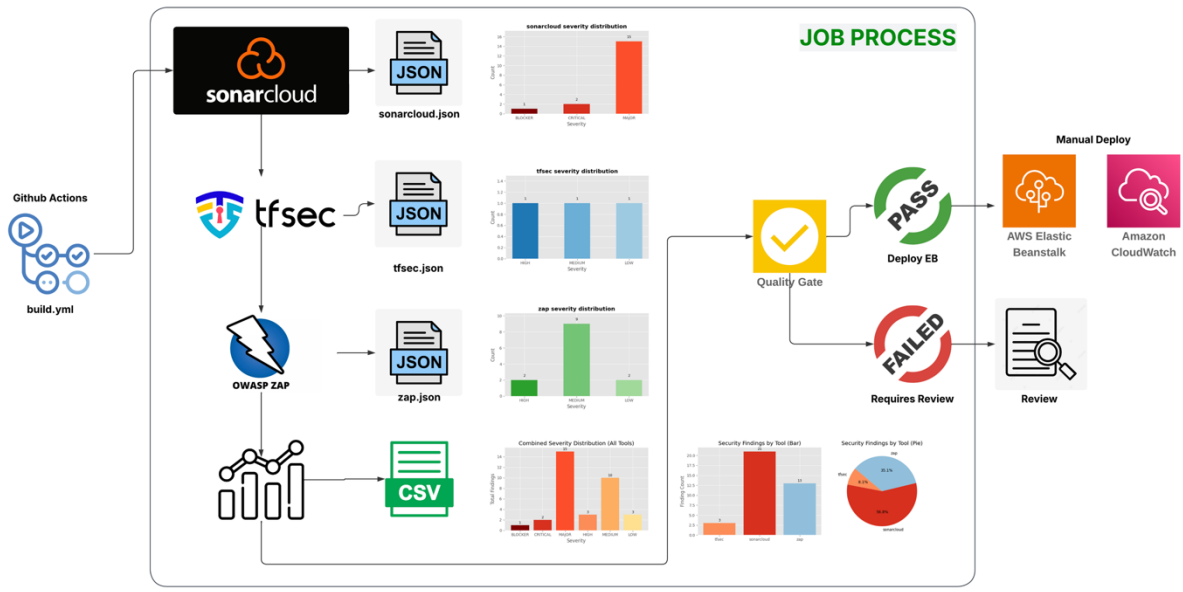


Figure 3. Implemented Pipeline Workflow and Generated Outputs

The Figure 3 is a diagram illustrating the overall implementation steps. This implementation is designed with **five steps**, and each step is embedded within a workflow which incorporates the following steps, the **static analysis**, the **IaC (Infrastructure as Code)**, the **dynamic analysis**, **result aggregation** and the final **quality gate** mechanism. In this case, this is a single GitHub Actions workflow, which is activated whenever commit is made to the main branch. It is a sequential workflow of the security analysis process, which begins with the analysis of the static analysis. All code submissions are reviewed before deployment and pass through the required automated security audit framework.

The first three stages of the pipeline are automated security testing. The first step uses SonarCloud to perform static application security tests and generates a *sonarcloud. json* report based on severity. The second stage is the tfsec stage which evaluates Terraform-based IaC resources and makes result's output as a *tfsec.json* file. Then, the OWASP ZAP is used to perform dynamic security testing at runtime. It generates a *zap. json* vulnerability report. The output generated by these security analysis tools is collected through the Metrics Aggregator and structured CSV files such as *metrics.csv* and *metrics_detailed.csv* file. Next, it creates a summary graph as a PNG file to prove quantification and visualization of the risks.

The final stage of the CI pipeline is the **Quality Gate**. This step measures the aggregate outcomes and decides to go ahead with or cancel the pipeline application. When builds that are rated **High** or **Critical** in severity, it is automatically marked as failed and required developers to manually review and fix any security issues found. Lastly, built-in successful releases can be deployed to AWS Elastic Beanstalk manually with user approval. This deployment is currently implemented in DevSecOps practice. After deployment, the application environment is monitored via Amazon CloudWatch.

5.2 SAST Implementation (SonarCloud)

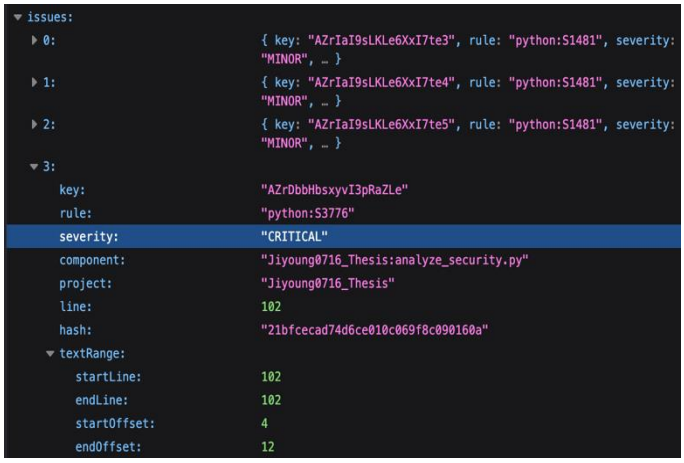


Figure 4. Example snippet of SonarCloud JSON report

The initial step of the planned security automation line conducts a Static Application Security Testing (SAST) with **SonarCloud**. Dynamically Python files and templates logic code analysis to determine vulnerabilities within code security including *rule violations* and *unsafe code patterns*. Whenever a developer makes a code change, GitHub Actions is triggered, and it generates a JSON file that has the precise location and components of the line.

Besides the JSON artefacts (machine-readable reports), SonarCloud is also able to present a *visual dashboard* to summarize detected issues across related categories, such as **Security**, **Reliability**, and **Maintainability**. This visual dashboard, shown in Figure 5, represents the overall quality of the related state of the pipeline execution. The two outputs are the artifacts of the SAST stage and they are inputted into the Metrics Aggregator.

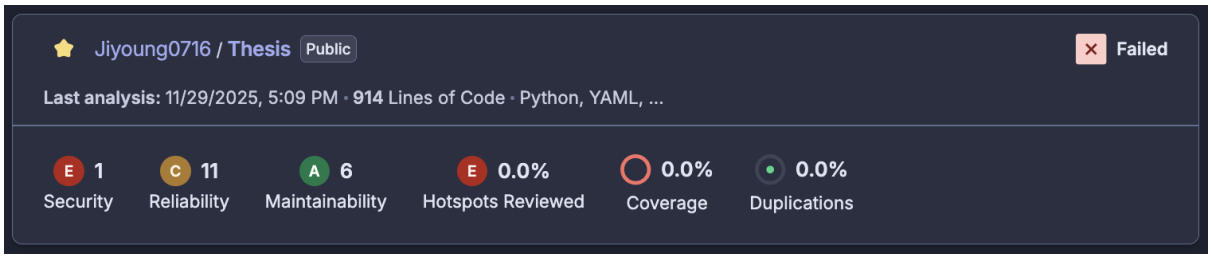


Figure 5. SonarCloud dashboard summary

5.3 IaC Security Implementation (tfsec)

The second phase of the pipeline performs **Infrastructure as Code (IaC)** by using **tfsec** scanning. The tool reviews the Terraform configuration which deploys AWS S3 and DynamoDB resources. It is an automatic tool that is executed in a workflow in GitHub Actions when the static analysis is completed. The tfsec allows the analysis of Terraform code to automatically identify any possible security vulnerabilities and misconfigurations.

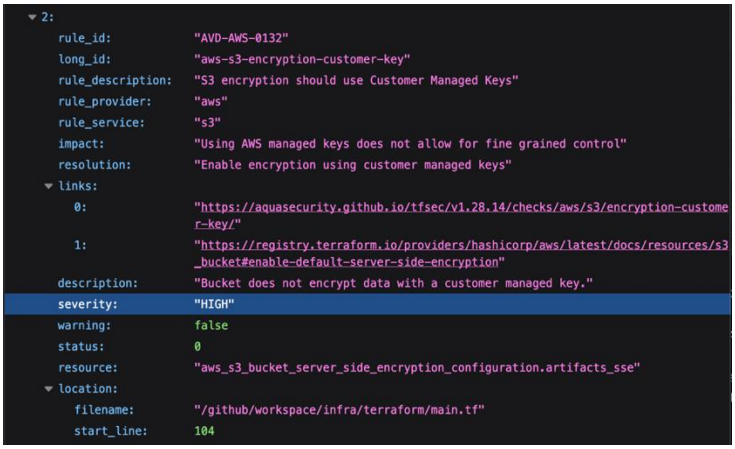
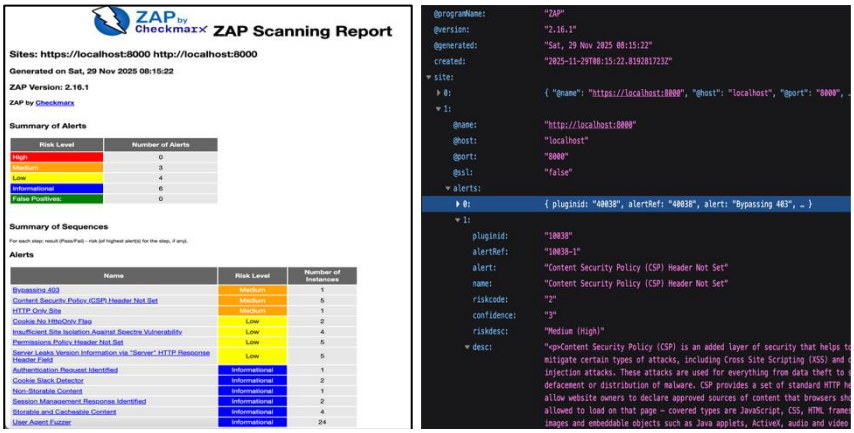


Figure 6. Example snippet of tfsec JSON report

Since scanning of tfsec is completed, it generates a tfsec.json file with each finding identified such as severity, rule identifier, related file, and line number. An example sample of this JSON output is shown in Figure 6.

5.4 DAST Implementation (OWASP ZAP Full Scan)

The **Dynamic Application Security Testing (DAST)** is performed in the third stage of the security pipeline with the help of OWASP ZAP. This is unlike the analysis of the code, which is done in a static analysis, but then it examines the behavior of the actual application in the Django environment where it is running to identify vulnerabilities. This security tool sends some requests to the server and analyzes the responses to evaluate potential vulnerabilities in the service. ZAP offers automatic security vulnerability testing of typical web application problems including XSS (Cross-Site Scripting), SQL Injection, insecure security header settings, and CSRF (Cross-Site Request Forgery). The entire scanning process is much longer taking compared to the past stages because of the full scanning.



Upon completion, ZAP generates an artefact with *report_html.html* and *report_json.json* files. Both files feature all discovered warnings, including the risk level, rule identifier, affected URL, and the brief description of the problem.

Figure 7. Example of ZAP HTML full-scan & Snippet JSON report

5.5 Metrics Aggregation & Visualization

The Metrics Aggregator consolidates some outputs generated by previous stages (SonarCloud, tfsec, and OWASP ZAP) into a unified structure. Once all tasks are finished, this stage converts all generated JSON artefacts into structured CSV files. Then this stage generates *metrics.csv* and *metrics_detailed.csv*. The *metrics.csv* summarizes the number of findings and severity level of each tool. The other output is *metrics_detailed.csv* that gives more detailed data with **severity**, **rule identifier**, **message description**, **affected component**, and **line reference**. It is shown in Figure 8.

metrics_detailed

tool	severity	rule_id	target	location	message
tfsec	LOW	AVD-AWS-0025	/github/workspace/infra/terraform/main.tf	66-68	Table encryption explicitly uses the default KMS key.
tfsec	MEDIUM	AVD-AWS-0089	/github/workspace/infra/terraform/main.tf	84-93	Bucket does not have logging enabled
tfsec	HIGH	AVD-AWS-0132	/github/workspace/infra/terraform/main.tf	104-111	Bucket does not encrypt data with a customer managed key.
sonarcloud	MINOR	python:S1481	analyze_security.py		Replace the unused local variable "autotexts" with "_".
sonarcloud	MINOR	python:S1481	analyze_security.py		Replace the unused local variable "wedges" with "_".
sonarcloud	MINOR	python:S1481	analyze_security.py		Replace the unused local variable "texts" with "_".
sonarcloud	CRITICAL	python:S3776	analyze_security.py:102	102	Refactor this function to reduce its Cognitive Complexity from 24 to the 15 allowed.
sonarcloud	MAJOR	python:S125	experiment/forms.py:24	24	Remove this commented out code.
sonarcloud	MAJOR	Web:S5254	experiment/templates/index.html:2	2	Add "lang" and/or "xml:lang" attributes to this "<html>" element

Figure 8. Example snippet of the metrics_detailed.csv file

Besides CSV files, this step generates PNG charts to show the total number of findings by each security tool and the combined severity distribution by all tools. These charts help the developers to find out where they should do some remediation.

This combined visual summaries (Figure 9) give a general overview of the vulnerabilities that were identified during the CI pipeline and forms the basis of assure that will be conducted during the following phase of Quality Gate review.

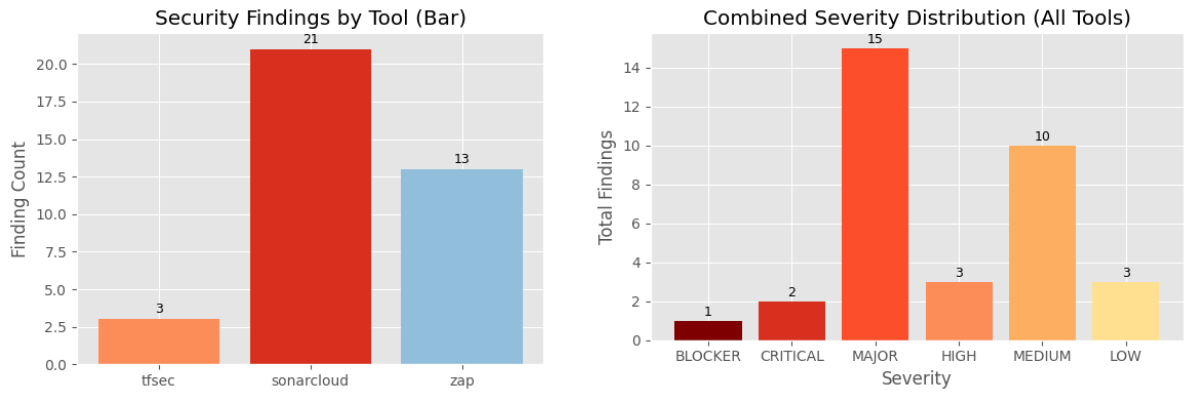


Figure 9. Comparative risk distribution and combined severity results across tools

5.6 Quality Gate Implementation

The last validation procedure in the CI pipeline workflow is the Quality Gate. It determines whether a build is deployable or not. It analyzes the *metrics.csv* file generated and aborts the pipeline whenever any severity results are classified as “CRITICAL” or “HIGH” warning.

During the implementation, several issues detected by tfsec (e.g., missing *server-side encryption*) and SonarCloud (e.g., *Cognitive Complexity* or *exposed secrets*) were addressed and improved. Conversely, OWASP ZAP determined a “HIGH” warning because of development environment artifacts such as *test server headers* and *CSPs*. Since these artifacts from ZAP do not represent actual security risks, the Quality Gate was extended to exclude unactionable ZAP messages during the process. With these improvements, the gate can pinpoint actual vulnerabilities, and non-harmful noise. Below Figure 10 demonstrates both the failure and successful execution cases.

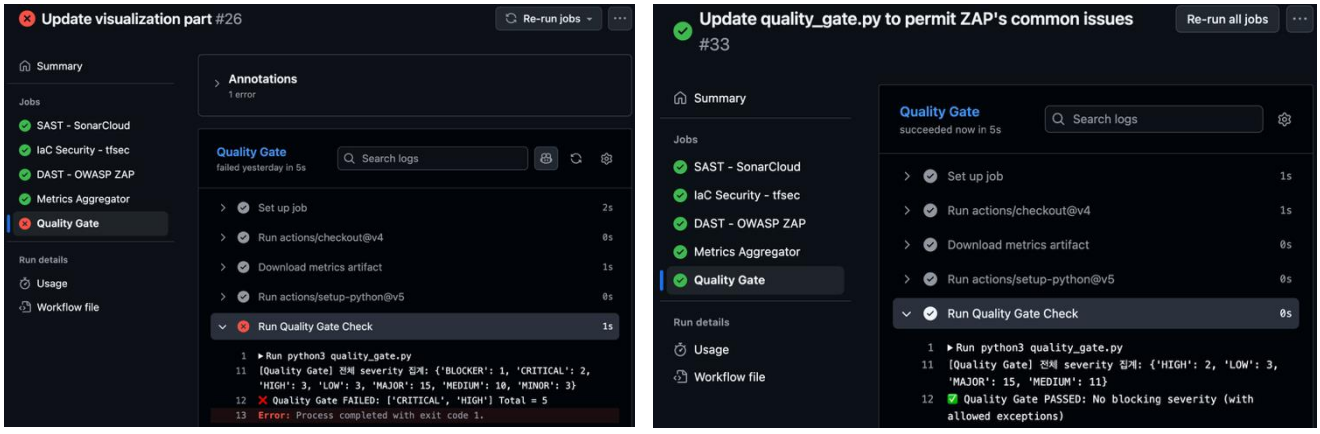


Figure 10. Comparison of Quality Gate outcomes: failed (left) and passed (right)

5.7 Manual Deployment to Elastic Beanstalk and CloudWatch

After the successful Quality Gate, the application was manually deployed to AWS Elastic Beanstalk. After that, the platform generated the service CNAME

http://x24142816-thesis-project-env.eba-pae62vxq.us-east-1.elasticbeanstalk.com/

It was automatically inspected for *CPU usage* and *network activity* via AWS CloudWatch through the Elastic Beanstalk console. These monitoring outputs indicate that the deployed environment was successfully operating. This move implies that a validated build can be launched on a cloud provider and monitored using AWS has built-in monitoring capabilities.

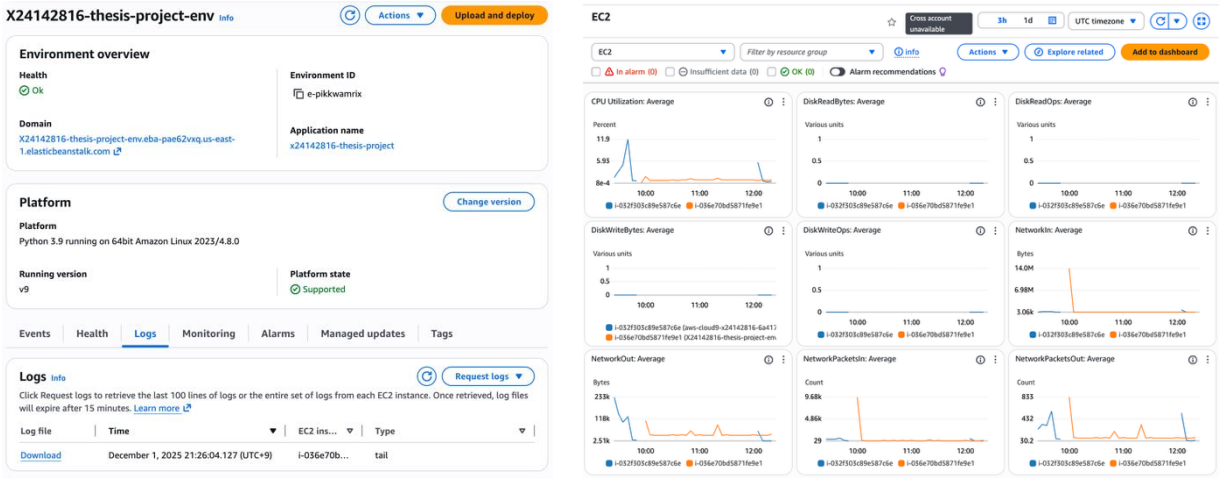


Figure 11. Elastic Beanstalk deployment and CloudWatch monitoring

6 Evaluation

This section evaluates the effectiveness of the proposed security automation pipeline for a DevSecOps environment. The evaluation consists of three key factors that meet the research objectives: ***vulnerability detection accuracy***, ***automation coverage***, and ***detection time efficiency***. The analysis is based on a test Django application designed for implementation and the OWASP Juice Shop for evaluation.

6.1 Experiment / Case Study

6.1.1 Experiment Design

This experiment was designed to evaluate the **functionality** and **relevance** of the suggested security automation pipeline in a cloud environment. It was tested on two different systems: a custom test application (*Django*) that was generated *as part of the project*, which represents a low-complexity system and *OWASP Juice Shop*, *an intentionally vulnerable web application* used as a high-risk benchmark. This allowed us to examine the proposed objectives under different conditions based on a consistent evaluation methodology.

These two systems were executed by the same GitHub Actions workflow using the same conditions. The pipeline combines three main automated security measures: *SonarCloud* performed code scanning as a static analysis, Infrastructure-as-Code was scanned by *tfsec*, and the dynamic part was analyzed by *OWASP ZAP* Full Scan. All the execution generated structured JSON reports, which were converted to CSV files and graphical visualizations using

metrics aggregation script. All executions were carried out under the same environment, including Ubuntu-based GitHub runners, to avoid any environment-related variations.

6.1.2 Vulnerability Detection Results

According to the results, they show that each tool differs significantly in detection focus. The highest number of issues was reported by SonarCloud, where more than one hundred findings were found mostly in the high-severity categories, which means that the coverage of static analysis is strong. Conversely, tfsec produced no results since the application does not have Infrastructure-as-Code resources written in Terraform. Therefore, scans are not applicable to this target.

OWASP ZAP reported fewer issues in general, but it was able to record runtime vulnerabilities, which cannot be observed by the static analysis. Those issues are notably related to headers and cookies. Even though its numerical output was lower, the results indicate dangers that become apparent in the performance. The tools complement each other and give a bigger perspective of security posture for the system.

Table 3. Vulnerability detection results for OWASP Juice Shop

Tool	Blocker	Critical	High	Info	Low	Major	Medium	Minor
tfsec	-	0	0	-	0	-	0	-
SonarCloud	6	32	-	2	-	31	-	29
ZAP	-	-	2	-	3	-	10	-

Note: A dash (-) denotes undefined levels, while "0" indicates the level is supported but no results were detected.

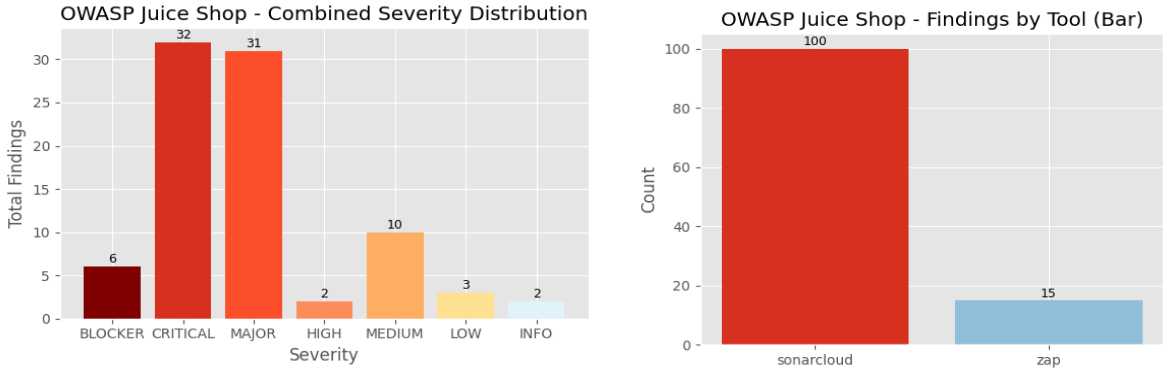


Figure 12. Vulnerability severity distributions for OWASP Juice Shop

6.1.3 Automation Coverage

The coverage of automation was different based on the structure of a given system. The Django pipeline executed all stages of the pipeline, consisting of static analysis, IaC scanning, dynamic analysis, metrics aggregation, and quality gate. In contrast, IaC resources were not included in the OWASP Juice Shop, and thus, the tfsec phase was not possible. The other stages worked as expected, and every target went through all applicable steps, without any human intervention, ensuring **100 percent automation** in all supported components of the pipeline.

The automation characteristics of the tools were also different. SonarCloud and tfsec produced results directly from code, while OWASP ZAP was used to detect runtime vulnerabilities. All

the outputs were automatically collected and unified with the use of the metrics script, and both targets could be processed without manual steps. In general, the findings reveal that the pipeline ensures flexible automation with various types of systems and naturally reveals weaknesses upon the lack of certain artefacts.

6.1.4 Time-to-Detection (TTD) Analysis

Tool / Stage	Django (sec)	OWASP Juice Shop (sec)
SonarCloud (SAST)	70	145
tfsec (IaC)	13	-
ZAP Full Scan (DAST)	354	777
Metrics Aggregation	24	21
Quality Gate	8	5
Total	469	948

Note: A dash (-) means that the step is not applicable to the corresponding system.

The time required to execute each of stages was different in the two systems because they had different structures and operations. In the case of Django application, the pipeline took 469 seconds, with the most of time was spent on the dynamic analysis (354 seconds). In contrast, static analysis and metrics aggregation required comparatively less time. For the OWASP Juice Shop, the total response time was 948 seconds, and this was mainly due to the ZAP Full Scan taking much longer time to crawl and evaluate the bigger and complex application. The IaC scanning was not executed because this target does not contain any Terraform-based resources.

Even though the time of implementation varied, the security framework did not require a human intervention in any of the steps supported. Static analysis, dynamic scanning, report conversion, and quality verification were executed fully automatically for both systems. As the results show, stable automation performance is achieved across a variety of application types. Overall, TTD is significantly affected by the size and complexity of the analysis target. The dynamic scanning phase, which has many analysis points, is particularly susceptible to this impact.

6.1.5 Quality Gate Evaluation

The Quality Gate of the Django application passes with no high or critical findings detected in the three main stages. Through this result, the automated security framework correctly validated a low-risk system (Django) and successfully built it without any manual checks.

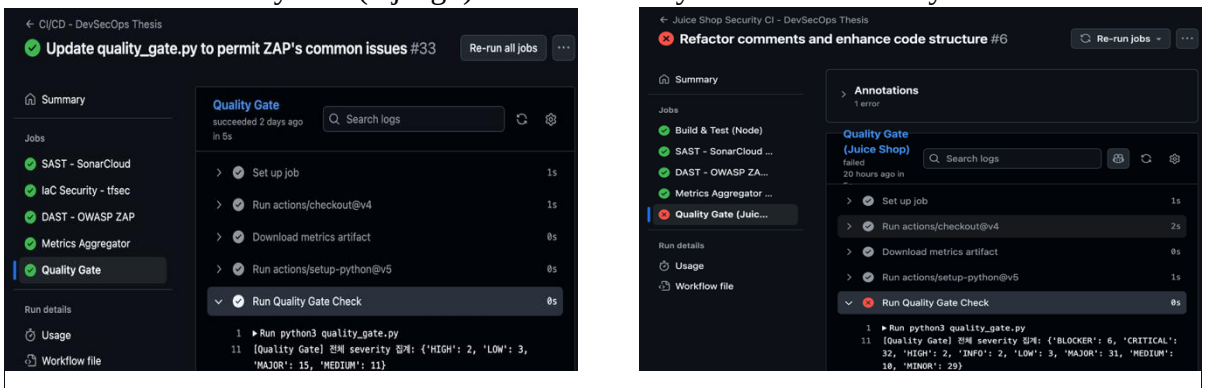


Figure 13. Comparison of Quality Gate for Django (passed) and OWASP Juice Shop (failed)

In the case of OWASP Juice Shop, this gate identified numerous *Critical* and *High* findings. Consequently, the pipeline failed and reflected intentional vulnerabilities. The failure demonstrates that the gate perfectly restricts unsafe builds with various potential issues. Based on this, it provides stability by accurately distinguishing between safe configurations and high-risk systems.

6.2 Discussion

The experimental results of the study are consistent with previous research demonstrating that static analysis, IaC auditing, and dynamic analysis each cover different security domains. Static analysis tools analyze the structural problems of the code, while dynamic tools find risks that occur at runtime. This trend was observed in this study as well, suggesting that the two methods are complementary. Based on this, the results support previous research that multi-layered automation provides a stronger detection confidence than relying on a single technique.

Several limitations were present in the experimental design. The Django application was a small-scale system with limited functionality, and the OWASP Juice Shop had no IaC resources, preventing tfsec from being. GitHub Actions environments also fixed time and resource limits that interfere with long scans. In the future research, the evaluation could be expanded the scope and increase representativeness by integrating more complex applications and a broader set of tools.

7 Conclusion and Future Work

This paper aimed to test the hypothesis of whether a proposed automation framework of DevSecOps which combines Infrastructure-as-Code, static analysis, and dynamic analysis could enhance the efficiency and feasibility of cloud security validation. It had been deployed with open-source tools such as Terraform, tfsec, SonarCloud, GitHub Actions, and OWASP ZAP. It was experimented with the small Django application and the OWASP Juice Shop. The pipeline was able to automate all the supported stages, minimize the manual intervention, and offer complementary detection on various security layers. The structural weaknesses were detected through the static analysis, and runtime problems were detected through dynamic analysis, which proved the usefulness of the combination. The Quality Gate did not fail either as the low-risk Django application was approved and the Juice Shop was blocked.

There were several limitations. The Django system was intentionally simple with low-risk, and Juice Shop lacked Terraform-based IaC, preventing evaluation of the tfsec stage. GitHub Actions lacks time and resource constraints, whereas AWS Learner Lab could not auto-export outputs because of the credential constraint. In the future, it could overcome these constraints using an unmanaged AWS environment (e.g., Free Tier) to enable automated deployment steps. Based on this, it can evaluate the pipeline on more complex IaC-driven systems. Overall, this research eventually proves a cost-effective and technically feasible DevSecOps model to organizations with limited security resources.

References

Alonso, J., Piliszek, R. and Cankar, M. (2023). 'Embracing IaC Through the DevSecOps Philosophy: Concepts, Challenges, and a Reference Framework', *IEEE Software*, 40(1), pp. 56–62. doi: 10.1109/MS.2022.3212194.

Pan, Z. et al. (2024) 'Ambush From All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines', *IEEE Transactions on Dependable and Secure Computing*, 21(1), pp. 403–418. doi: 10.1109/TDSC.2023.3253572.

Zhang, Z., Wu, L., Jin, J., Wang, E., Liu, B. and Han, Q.-L. (2025) 'Secure Federated Learning for Cloud-Fog Automation: Vulnerabilities, Challenges, Solutions, and Future Directions', *IEEE Transactions on Industrial Informatics*, 21(5), pp. 3528–3540. doi: 10.1109/TII.2025.3528569.

Paule, C., Düllmann, T.F. and Van Hoorn, A. (2019) 'Vulnerabilities in Continuous Delivery Pipelines? A Case Study', *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Hamburg, Germany, pp. 102–108. doi: 10.1109/ICSA-C.2019.00026.

Rangnau, T., van Buijtenen, R., Fransen, F. and Turkmen, F. (2020) 'Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines', *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, Eindhoven, Netherlands, pp. 145–154. doi: 10.1109/EDOC49727.2020.00026.

Chiari, M., De Pascalis, M. and Pradella, M. (2022) 'Static Analysis of Infrastructure as Code: a Survey', *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Honolulu, HI, USA, pp. 218–225. doi: 10.1109/ICSA-C54293.2022.00049.

Saavedra, N., Gonçalves, J., Henriques, M., Ferreira, J.F. and Mendes, A. (2023) 'Polyglot Code Smell Detection for Infrastructure as Code with GLITCH', *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Luxembourg, Luxembourg, pp. 2042–2045. doi: 10.1109/ASE56229.2023.00162.

Landuyt, D.V., Sion, L., Philips, W. and Joosen, W. (2024) 'From automation to CI/CD: a comparative evaluation of threat modeling tools', *2024 IEEE Secure Development Conference (SecDev)*, Pittsburgh, PA, USA, pp. 35–45. doi: 10.1109/SecDev61143.2024.00010.

Reddy Konala, P., Kumar, V. and Bainbridge, D. (2023) 'SoK: Static Configuration Analysis in Infrastructure as Code Scripts', *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, Venice, Italy, pp. 281–288. doi: 10.1109/CSR57506.2023.10224925.

Cheenepalli, J., Hastings, J.D., Ahmed, K.M. and Fenner, C. (2025) 'Advancing DevSecOps in SMEs: Challenges and Best Practices for Secure CI/CD Pipelines', *2025 13th International Symposium on Digital Forensics and Security (ISDFS)*, Boston, MA, USA, pp. 1–6. doi: 10.1109/ISDFS65363.2025.11011960.

Akbar, M.A., Smolander, K., Mahmood, S. and Alsanad, A. (2022) 'Toward successful DevSecOps in software development organizations: A decision-making framework', *Information and Software Technology*, 147, 106894. doi: 10.1016/j.infsof.2022.106894.

Díaz, J., Pérez, J.E., Lopez-Peña, M.A., Mena, G.A. and Yagüe, A. (2019) 'Self-Service Cybersecurity Monitoring as Enabler for DevSecOps', *IEEE Access*, 7, pp. 100283–100295. doi: 10.1109/ACCESS.2019.2930000.