National College of Ireland

Practical Implementation of Zero Trust Security in DevSecOps Pipelines

MSc Research Project
MSc Cloud Computing

Siva Suriya Kandasamy
Student ID: 23341963

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Siva Suriya Kandasamy		
Student ID:	23341963		
Programme:	Masters in Cloud Computing	**Year:**	2024 - 2025
Module:	MSc Research Project		
Supervisor:	Sean Heeney		
Submission Due Date:	11/08/2025		
Project Title:	Practical Implementation of Zero Trust Security in DevSecOps Pipelines		
Word Count:	6427	**Page Count:**	20

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: 

Date: 11/08/2025

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):	

Practical Implementation of Zero Trust Security in DevSecOps Pipelines

Siva Suriya Kandasamy
23341963

Abstract

With practices of DevSecOps becoming ever more utilized in current software development, the need to have sound security models has also seen the increased upward trend with rapid software delivery made possible by increased DevSecOps practices. Traditional perimeter-based security offerings are ineffective in controlling the dynamic nature of the threats, particularly such environments that have been built on the principle of microservices, adaptive orchestration and containerized application. Based on such challenges, this paper applies Zero Trust Security (ZTS) principles to cloud-native DevSecOps flow. This is done via making a comparative analysis of a standard DevSecOps pipeline (Pre-ZTS) to an enhanced pipeline, with ZTS controls (Post-ZTS). The key components will be the Open Policy Agent (OPA) able to perform dynamic admission control, the Istio Service Mesh that will enable service to service authentication and micro-segmentation, and the lightweight Kubernetes (K3s) orchestration. The use of complimentary security scanning tools which were OWASP ZAP, Trivy, and SonarCloud were employed to mimic a real-life threat detection process by attacking the vulnerability such as SQL injection and container image exploit. OWASP Juice Shop application was used as the testbed to subject the capabilities of the runtime policy enforcement and strengths of access control, as well as exposure to vulnerabilities. Results have shown that the ZTS metrics do not alter the output of a fixed vulnerability scanner but have the potential to significantly improve runtime security state as well as access control without materially impacting the deployment rate. The current piece provides one such implementation route that fills that gap of purely theoretical descriptions of Zero Trust structures and how they are used in practice in existing CI/CD environments.

Keywords: Zero Trust Security, DevSecOps, OPA, Istio, Trivy

1 Introduction

1.1 Background and Motivation

DevSecOps is the latest trend that added security processes to all phases of the CI/CD pipeline and includes it in the software development lifecycle (SDLC). The customary perimeter-based security mode has grown to be ineffective following the popularity of microservices, cloud-native architectures, and containerized apps. This traditional model assumes of implicit trust of an internal network that cannot cover lateral movement, insider threat and vulnerability of distributed systems. Because perimeter-based security assumes that all traffic within a network is trustworthy, it poses a huge blind spot especially in DevOps environments where workloads often change and scale.

A new solution, Zero Trust Security (ZTS) was introduced, based on the assumption of "never trust, always verify." All access requests under ZTS undergo access authentication,

authorization and constant validation regardless of source. This strategy will make sure that similar strict security checks are applied to internal as well as external traffic. The driving force behind pursuing this study is to get to understand how the ZTS concepts can be incorporated successfully into contemporary DevSecOps pipeline in order to harden security position without sacrificing the speed of delivery that CI/CD offers. With the help of real-world simulations and tooling, the research project will help provide practical contributions to the understanding of how to make software delivery secure in cloud-native contexts.

1.2 Problem Statement

Our modern, dynamic and decentralized environments do not fit traditional models of perimeter-based security. In contemporary DevSecOps pipelines these environments are set up in hybrid and public cloud infrastructure environments where microservices are deployed, which present a multitude of challenges and risks in the form of static firewall rules and implicit trust thinking. These pipelines do not regularly enforce the runtime policy or conduct ongoing identity checks, and these pipelines are prone to insider attacks and misconfigurations as well as advanced attack vectors like SQL injection, container image poisoning and privilege escalation.

In overcoming these challenges, there is an urgent need to develop a security architecture that pro-actively implements security policies, monitors run-time behavior as well as dynamically validates identities. Zero Trust Security and least-privilege access along with a focus on the context-aware verification fits these requirements.

1.3 Research Question

"How can Zero Trust Security be practically implemented in DevSecOps pipelines to enhance security without significantly impacting deployment speed?"

This research question also addresses below questions:

- What are the most appropriate suitability strategies to build Zero Trust concept into a CI / CD pipeline architecture?
- What are quantifiable effects of ZTS on vulnerability detection, and runtime security?
- Which frameworks and tools can be better used to enable dynamic policy enforcement and micro-segmentation as well as identity verification in CI/CD pipelines?

1.4 Proposed Solution

To realize the goal, Zero Trust Security implementation with attached DevSecOps pipelines is proposed to be achieved with the following tools and technologies.

- Open Policy Agent(OPA) - dynamic policy to govern access control.
- OWASP Juice Shop - test environment application that offers simulation on the vulnerabilities.
- Light weight Kubernetes (K3s) - set the microservices and manage the workloads.
- Istio Service Mesh - micro segmentation / internal communication (service to service).
- Trivy, OWASP ZAP and Sonar Cloud - Vulnerability scanners to know the security threats.

The given CI/CD may be tested in two ways. First - simple pipeline (without ZTS) and second pipeline with Zero Trust Security integrated enforcements.

1.5 Research Objectives

This research focuses on the main task, which is to create, deploy and test a CI/CD pipeline that will incorporate Zero Trust Security (ZTS) principles into a cloud-native setting. This is to show how to use the real-time policy enforcement by employing Open Policy Agent (OPA) in a Kubernetes-based deployment, to have dynamic and context-based admission controls. Moreover, the work dwells upon the deployment of service segmentation and secure transport of services via Istio Service Mesh, thus securing authentication and authorization of microservices. Another fundamental step of the assessment is a pre/post vulnerability assessment of integrating the ZTS, which implements widely used complex tools like container image scan applicability to Trivy, dynamic application security test support by OWASP ZAP and wind up the practice with static code analysis by SonarCloud. Moreover, the study evaluates the deployment performance to identify whether it is greatly affected by the introduction of ZTS components on the speed of the execution of pipelines. This will ensure the research will provide practical reference architecture, in addition to a practical implementation roadmap, that will help the organizations that need to incorporate the Zero Trust principles to integrate into modern cloud-native CI/CD pipelines without sacrificing agility.

2 Related Work

The challenges of integration of Zero Trust principles into DevSecOps and CI/CD pipelines have been addressed in a number of recently published articles. These works can be used as a theoretical and empirical foundation of this research, particularly with regards to the workload identity and policy enforcement aspects and secure-by-design pipeline structures.

2.1 Secure-by-Design CI/CD Pipelines

Bondalapati et al. (2025) have offered a Secure-by-Design CI / CD system where Zero Trust Security concepts are very centralized in the cloud-native delivery automations. This practice makes them not apply security as supplemented but incorporating it in the pipeline i.e., source code verification during deployment to production. The framework proposes quality control checks that are automated on the integrity of code, dependency vulnerability, misconfiguration of the infrastructure and security at run time. The primary advantage of the study is the in-depth methodology linked to the interpretation of security as the process, not the control gate. The authors concentrate on the significance of the things that are impossible to change the deployment artifacts that lock configuration baselines, but a good policy is applied in real-time. This is directly related in our activities that implements the same when automated vulnerability scanning (Trivy) and dynamic application testing (OWASP ZAP) are performed among other processes in the CI/CD work stream which also ensures that Zero Trust controls are implemented into our pipeline lifecycle.

2.2 Autonomous Identity-Based Threat Segmentation

Ahmadi (2025) suggested an ID-based autonomous threat segmentation approach of Zero Trust structures based on the concept of extending the identity-based segmentation mechanism by enabling both identity-based and behavior-based segmentation. This reduces the attack surface because the attack disruptor cannot use system lateral movement in case of hijacking of one of the components. This implementation is based on adaptive policies that are automatically conditioned to segmentation points (based on real-time situation), offering that balance between security and the business process flow. Such empirical findings would not be made without the aspect that this study is proactive in the sense that the task of segmentation does not require manual updates as this is done automatically. To endorse this concept, this work is echoed in Istio service-to-service segmentation policy since communication is intentionally facilitated only in the presence of granted microservices. It is a very straightforward example of how theoretical abstractions regarding identity-based segmentation can be applied in practice in a Kubernetes-driven DevSecOps pipeline.

2.3 Intent-Aware Authorization for Zero Trust CI/CD

Avirneni (2025a) proposed intent-based authorization model for the Zero Trust CI/CD environment, which would also use the identity and the purpose of the operation to determine the access control decisions. Unlike the conventional attribute-based or role-based access control, the model evaluates the request to possess the targeted action within the context of the particular stage of the workflow along with the risk and assessment of security position. This type of granular approach will eliminate excess grants and will be a dynamic least-privilege-access. The research outlines the circumstances whereby even the legitimate users are denied the ability to go beyond their intended scope by ensuring that the insider threat is minimized. Same principles have been applied to our work and use OPA Gatekeeper to apply Kubernetes policies that can apply to deployment via both fine-grained deployment enforcement, as well as context-based enforcement whereby only deployment with the necessary security labels and configuration may be permitted to perform enforcement. The above implies that one can convert the abstract principles of intent-awareness to implementable policies in automated chains.

2.4 Establishing Workload Identity for Zero Trust CI/CD

Avirneni (2025b) recently explored how workload identity based on SPIFFE could be applied to replace static credentials and secrets that are used in CI/CD pipelines to obtain cryptographically verifiable workload authentication. In this strategy, it does not require long-lived keys or passwords, which are possible to leak, but rather short-lived identities that are assigned to workloads. This provides a very secure, expandable and automatable authentication, as per Zero Trust. The advantage of this publication is that it aims to get rid of one of the primary causes of CI/CD vulnerabilities of hardcoded credentials. Although our work does not explicitly deploy SPIFFE, the workload-based trust principle cuts through to mutually configure the TLS (mTLS) in Istio, which verifies and encrypts inter service messages in post-ZTS setting. This indicates that as long as the security objectives remain the same, workload identity may be enforced by several technical frameworks.

2.5 Enhancing Secure Software Development with AI-Integrated Zero Trust

Coston et al. (2025) suggested the usage of the AZTRM-D model to eliminate vulnerabilities as the artificial intelligence solution based on DevSecOps practices, risk management frameworks, and Zero Trust model. Artificial intelligence is applied in the model in order to conduct continuous threat modeling in prioritizing the vulnerabilities in terms of the impact and the possibility of being exploited. The adaptive security that comes with the integration of AI allows a better response to new threats that occur, limiting exposure time. The strength of this study is that the detection and decision-making will be automated hence there will be minimum human error that will contribute to the speed at which the decisions will be made. Our research does not incorporate the risk-modeling driven by AI, but shares a philosophy of never-ending, automated security validation. Our empirical evidence of Pre and Post ZTS vulnerability scan provides the way that this kind of continuous monitoring builds system resilience over time.

3 Research Methodology

3.1 Research Approach

The study rests on the assumption of the experimental case study design to identify the principles of Zero Trust Security (ZTS) application in DevSecOps pipeline. This was aimed at creating as close a representation to a real-world cloud-native scenario in which all factors are held constantly apart from Zero Trust controls being introduced. Such methodology will allow ZTS to quantify its security and performance footprint in a consistent way, which will be replicable.

The research was conducted in two phases:

- Pre-ZTS Phase - A DevSecOps pipeline of zero trust-free.
- Post-ZTS Phase - The base pipeline updated with Istio service mesh and Open Policy agent (OPA) Gatekeeper to apply Zero Trust.

The two phases involved the same deployment environments, applications settings, and scan operations and allowed determining that any noticed difference could be explained exclusively by the presence of Zero Trust operations.

3.2 Test Environment Setup

In order to ease the experimental setup, an environment of K3s Light weight Kubernetes Cluster has been deployed in AWS cloud9 to represent the orchestration platform of container. As a target application, all of the scans, along with policy checks, have been conducted against the OWASP Juice Shop application that is also known to have several vulnerabilities built into it.

Trivy was utilized in scanning vulnerability with the aim of evaluating the weaknesses in one of the container images and OWASP ZAP was employed in conducting dynamic application security testing. In Post-ZTS, OPA Gatekeeper was used to implement policy on pod

deployments and Istio used to provide a service-to-service segmentation/network-level Zero Trust capabilities. The automation platform that was chosen as CI/CD was GitHub Actions, and the analysis of the static code was followed by using SonarCloud.

Kubernetes subcomponents were loaded in a pre-loaded state until they had a reproducible and stable test environment like CoreDNS, metrics-server and local-path provisioner, which were pre-loaded with the help of EBS volumes. This eliminated dependencies on third party image registries and rapid initialization of environment without waiting on container pulls.

3.3 Experimental procedure

The Pre-ZTS Phase started by deploying OWASP Juice Shop into Kubernetes cluster devoid of Istio and OPA. This allowed free communication amongst services, and there was no application of a piece-of-the-runtime policy. Trivy was executed to scan the application image itself to find vulnerabilities, OWASP ZAP was executed to find the flaws in the web applications, and the pipeline execution time was captured to be used as a performance comparison benchmark.

In the Post-ZTS Phase, an OPA Gatekeeper was used to enforce label-based pod compliance enforcement policies that unlabeled workloads could not be deployed. Istio was then to be deployed using strict Authorization Policies in order to have only allowed services to communicate and, thus, enabled micro-segmentation. After these measures had been in effect, it was repeated to conduct the same scans of the vulnerability and measures of the performance.

Both Pre-ZTS and Post-ZTS were launched directly off Git tags on GitHub Actions:

- pre-zts-* : Runs pre-ZTS scans and tests.
- post-zts-* : Runs post-ZTS scans and tests.

Moreover, each code push into the main branch generated a build workflow that ran test cases written in python and sent the results to SonarCloud.

3.4 Data Collection and Analysis

Both phases provided results of Trivy and OWASP ZAP scans in structured forms (table, JSON and HTML reports). The results were then compared, to identify changes in security posture. The proper metric that should be applied to quantify the performance of deployments is the time required to execute CI/CD jobs prior to the mechanism of ZTS integration and vice versa.

Although the difference in the number of vulnerabilities was not found in the scans performed, this is a reasonably expected result. Istio and OPA do not modify container images or application code to implement their runtime and network security which is why the number of vulnerabilities remain the same. Rather, they were manifested in blocked access conditions, limited communication channels and mandated compliance in pod installations.

Tool / Components	Purposes
AWS Cloud 9	Development and deployment environment
K3 Kubernetes	Lightweight Kubernetes for application deployment
OWASP Juice Shop	Vulnerable web application for security and scanning
Trivy	Container image vulnerability scanning
OWASP ZAP	Dynamic Application Security Testing (DAST)
OPA Gatekeeper	Policy enforcements during runtime
Istio Service Mesh	Micro segmentation and controls of ZTS
Sonar Cloud	Static Code Analysis for code quality (SAST)
GitHub Actions	CI/CD automation

Table 1: Tools and their purposes

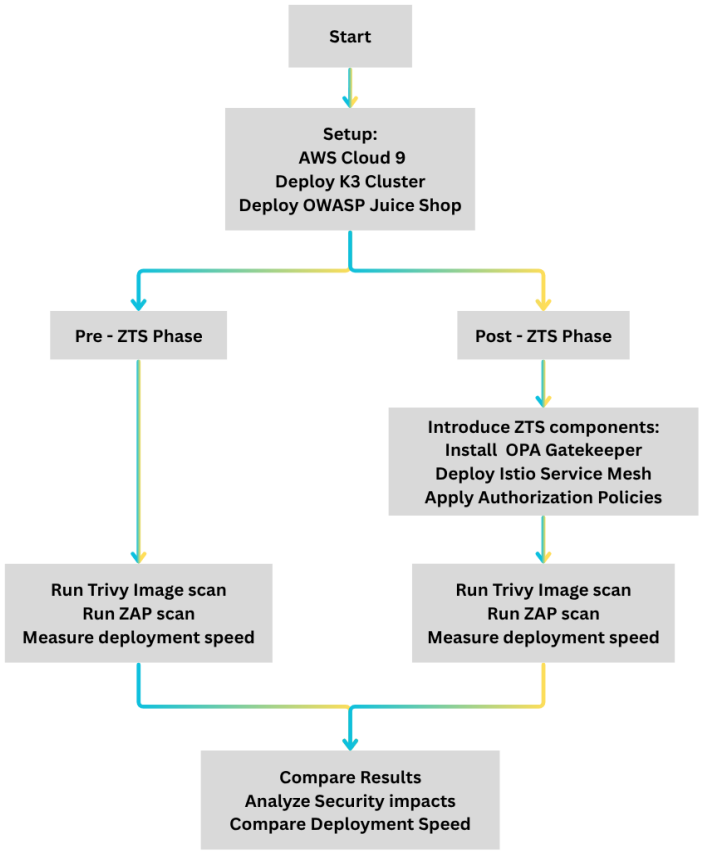

Figure 1: Flow Diagram Plan

4 Design Specification

The proposed architecture was aimed at incorporating the principles of Zero Trust Security (ZTS) into modern DevSecOps pipeline, and goal was to augment the application security without contributing great amounts of delay to the deployment speed. The methodology incorporated two controlled environments, one which was a base case (Pre-ZTS), wherein Zero Trust enforcement was not in place, and another one, a (Post-ZTS) complete Zero Trust controls were enforced. Running the same number of workloads and the same security scans in both of the environments made it possible to quantify the effectivity and performance implications of ZTS. Its architecture consists of three conceptual layers namely, the source code and CI/CD pipeline layer, the Zero Trust enforcement layer and the security scanning and analysis layer.

4.1 Source code and CI/CD Pipeline Layer

At the core of the system sits the source code repository which exists on GitHub and has an application, Kubernetes deployment configurations and workflow definitions. The tool chosen in the study is OWASP Juice Shop, a vulnerable web application which is developed on purpose and can be considered as the source of various security tests because of the excessive weaknesses that it possesses. It has also stored Kubernetes manifests used in the deployment of the application together with other test workloads of the application which include the httpbin and sleep pods. There is a GitHub Actions that acts as the CI/CD engine, and separate workflow files that have a run of Pre-ZTS and Post-ZTS. Manual enablement is based on the code tags with semantics i.e. pre-zts-*(base environment), and post-zts-*(Zero Trust enabled environment). Once activated, each workflow works through automated security testing steps with Trivy to check against vulnerability and OWASP ZAP to perform dynamic testing on applications. Each report generated is uploaded as an artifact hence tracing and reproduction of results can be done. Moreover, the repository also has a parallel build workflow which is automatically executed after each push to the main branch. This build pipeline executes previously specified Python testing scripts, data are then reported to SonarCloud to analyze the code statically, and track code quality and possible security flaws alongside the Zero trust-oriented searches.

4.2 Zero Trust Security Enforcement Layer

Zero Trust enforcement layer is proposed only in the Post-ZTS environment and serves as the means to govern both the workload admission to the cluster, as well as the interaction between the implemented services. This layer is composed of two different key technologies. The former is Open Policy Agent (OPA) Gatekeeper which works as Kubernetes admission controller. All API requests coming to OPA to create or modify cluster resources are evaluated and only requests that satisfy the predefined rules on security are passed. In our research, a constraint, a must-have-app-label was applied in such a way that any pod that lacks the app label could not be deployed. This produces a level of compliance and consistency prior even to the enlistment of workloads.

The second one would be Istio Service Mesh (control plane in istio-system), which is just installed in zts-test namespace with sidecar injection enabled. Istio sidecar injection introduces Envoy sidecar proxies into every pod in zts-test and relatively restricted AuthorizationPolicy rules that manage east west traffic (inter-node service-to-service communication). By default, this is untrusted, a so-called deny-all configuration is employed,

where all communications between services are denied unless explicitly allowed. A specific set of allow rules then get set up to allow only necessary communication e.g. the sleep service to the httpbin service. This will mean that even when an attacker cracks one service, he/she is not at liberty to move as desired within or across the cluster. A combination of OPA Gatekeeper and Istio also allows Zero Trust enforcement on both deployment and runtime, substantively minimizing the attack surface.

4.3 Security Scanning and Analysis Layer

The security-scanning-and-analysis layer is essential in carrying out a continuous security validation which proves important in making a comparison between the Pre-ZTS and the Post-ZTS environment. This layer combines various tools to carry out both test static and dynamic. Container image scanning to check operating system and application library vulnerabilities and configuration scanning to check insecure Kubernetes manifests is done with Trivy. During the Post-ZTS, such scans validate that Zero Trust policies do not have any impact on the vulnerability profile of the actual base application image. OWASP ZAP will be utilized to conduct application security tests dynamically by replicating actual attacks to the Juice Shop application that is deployed. In Post-ZTS scenario, ZAP would also expose walled or unreachable endpoints because of Istio network policies. SonarCloud is optionally incorporated to conduct static code analysis, which concentrates on the code quality and probable vulnerabilities in the code. Scanning both environments with identical security scans allows finding the vulnerabilities that persisted across the environments and also discover at what points Zero Trust implementation succeeds in defining what will be the attack vectors blocked.

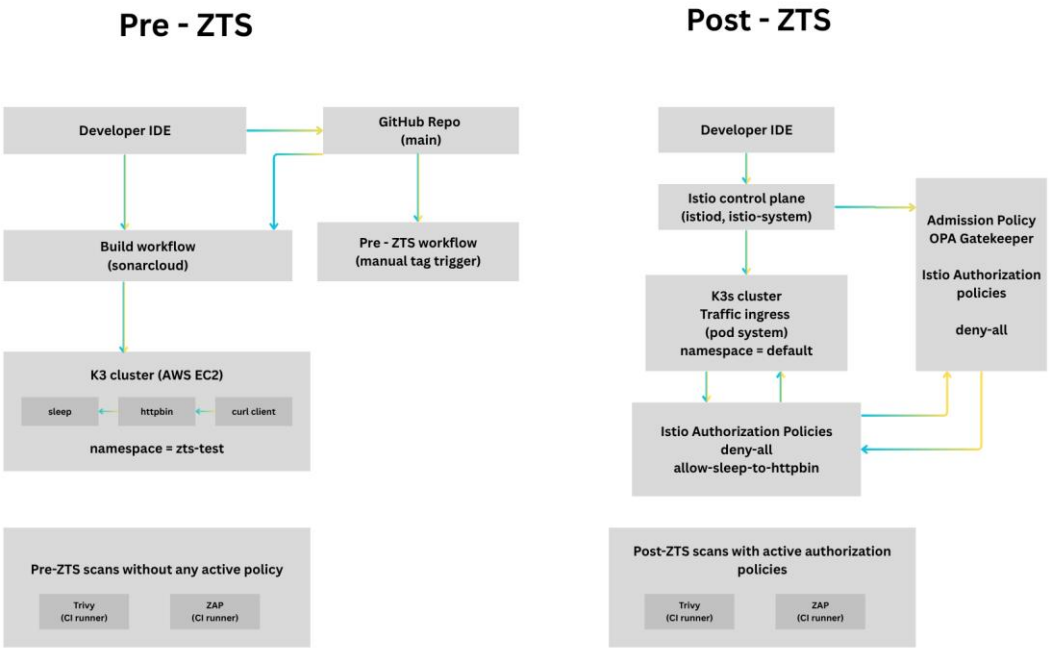

Figure 2: Architectural diagram

5 Implementation

The implementation of the given research project was characterized by the practical deployment of all necessary tools, configurations, and workflows in order to establish Pre-ZTS and Post-ZTS environments and perform security scans and assessments. It was done in well-defined phases so that the activities of each component can be authenticated and quantified separately.

5.1 Environment Setup

The first one was created on the Amazon Web Service EC2 instance and maintained through the Cloud9 IDE to provide ease of development and administration. K3s is a lightweight version of Kubernetes installed in order to run as a container orchestration system. K3s set up system services crucial to the working of the system such as CoreDNS to address service names; metrics-server to analyze the resources, Traefik to direct traffic and the local-path-provisioner to handle storage on the K3s cluster.

In order to maximize the time taken to set up cluster and prevent possible failures when accessing external registries, some of the key system images used in Kubernetes, such as CoreDNS, metrics-server and local-path-provisioner were pre-fetched and stored into an EBS (Elastic Block Store) volume. These were pulled at run time fresh images contained in the EBS volume, not new images pulled after the docker hub or other community repositories. This plan meant it would be easier and more dependable to initialize a cluster, particularly in the event where the internet connection or upstream registry framework is unpredictable. Furthermore, this was the only method that ensured compatibility of version of system components between Pre-ZTS and Post-ZTS phases so as to avoid the environmental drift which may render results of the evaluations biased.

Namespaces were used in order to guarantee logical segregation of workloads, and the ability to use Zero Trust controls without affecting unrelated workloads. As an example, the OWASP Juice Shop was deployed in the default namespace (outside the mesh) and the zts-test namespace was dedicated to Istio service-to-service communication experiments with sleep and httpbin pods (mesh enabled).

An OWASP Juice Shop, a specially vulnerable web application was provided through a Kubernetes Deployment manifest. It was publicly available by using NodePort service in order to support internal traffic to experiment with mesh communication and external connection to process vulnerability testing. Such architecture provided flexibility that security testing could be performed in several layers.

5.2 Pre-ZTS Baseline Configuration and Scans

Before the mechanisms of Zero Trust were brought out, a baseline environment was established (Pre- ZTS). This base was used as the control measure in the assessment of the Zero Trust security and performance impact.

Pre-ZTS baseline was intended in a completely open setting where Zero Trust elements and controls did not exist. At this point, the Kubernetes cluster had no policy of constrained communication amongst services and workloads deployed without any admission control

policy and service mesh sidecar proxies. This was a typical DevSecOps pipeline setup before the spread of Zero Trust, so that it was a neutral starting point that could be compared to the future advantages of security improvement.

Having created the background a Pre-ZTS Security Scans workflow is created in the GitHub Actions. This was done as a manual workflow with semantic tags (pre-zts-*) to avoid running accidentally on unrelated changes to the repository. There were two large scans undertaken as part of the workflow:

- Trivy Image Vulnerability Scan - This scan examined the bkimminich/juice-shop Docker image regarding any known vulnerabilities both at the operating system and library level. The scan results were limited to those reported as CRITICAL and HIGH vulnerability and was stored in the form of a structured report.

- OWASP ZAP Baseline Scan Test — This scan has executed dynamic application security test (DAST) against a publicly available Juice Shop endpoint and found a series of potential vulnerabilities, including security header-related problems, information exposure, and cross-site scripting-related risks. The reports were being generated as HTML, Markdown and JSON.

Further, the repository was integrated with SonarCloud to carry out static application security testing (SAST) on Python test files whenever a code was pushed to the main branch. This gave ongoing-code quality and vulnerabilities information independent of the Zero Trust scans.

5.3 Post-ZTS Configuration and Scans

The Zero Trust Security model was applied after conducting the baseline assessment by implementing two major components OPA Gatekeeper and Istio Service Mesh to ensure strict security measures were imposed.

In the gatekeeper-system namespace OPA Gatekeeper was mounted. K8sRequiredLabels constraint template and a must-have-app-label constraint were implemented so that it could be used to declare mandatory labelling on the entire pods. The functional test was conducted with deployment of a compliant pod (good-pod.yaml) and an attempt at deployment of a non-compliant pod (bad-pod.yaml) that has been blocked due to the admission webhook of Gatekeeper.

Istio had been deployed to the istio-system namespace and sidecar injection was auto-enabled in zts-test namespace. This namespace has deployed the httpbin and the sleep pods to have an environment to simulate the communication of microservices. It has been configured with a deny-all AuthorizationPolicy to block by default the traffic between the east and west. This was then accompanied by allow-sleep-to-httpbin policy to let only requests to httpbin by the sleep pod and all other requests to be blocked. The work with these polices was successful since it was confirmed by connectivity testing using curl.

It involved a Post-ZTS Security Scans workflow implemented in GitHub Actions that was executed on a manual basis via the use of semantic tags (post-zts-*). This workflow depicted the Pre-ZTS scan procedure in which Trivy and OWASP-ZAP were executed against identical application and configuration to allow a quick comparison between the outputs.

Pod Name	Namespace	Labels Present	Pre-ZTS Result	Post-ZTS Result	Remarks
good-pod	default	app = good	Allowed	Allowed	Policy met
bad-pod	default	none	Allowed	Denied	Policy enforced correctly

Table 2: Pod deployment compliance check via OPA Gatekeeper

Source Service	Target Service	Pre-ZTS access result	Post-ZTS access result	Remarks
Sleep	Httpbin	Allowed	Allowed	Policy permits only this path
Httpbin	Sleep	Allowed	Denied	Verified enforcement
Curl Client	Httpbin	Allowed	Denied	Verified enforcement

Table 3: Service to service communication restriction test

5.4 Workflow Automation Strategy

As a way of ensuring a strict distinction between the states of testing, the workflows were set as follows:

- Pre-ZTS, Post-ZTS, workflows were invoked manually using tags (pre-zts-*, post-zts-*) and they only ran in their respective target environment set-ups.

- Build was configured to automatically launch on pushes to the main branch only and SonarCloud analysis was used without conducting security scans.

This separation eliminated the danger of carrying out a scan, in an inappropriate environment state thus leading to data integrity during the evaluation and results phases.

6 Evaluation

6.1 Evaluation Approach

The quality measurement approach of the study was based on ensuring the effectiveness of security, compliance correctness in the implementation of the policy, and performance on integrating Zero Trust Security (ZTS) onto a DevSecOps pipeline. This was to make sure that

the results could be quantitative, reproduceable and applicable in practical software delivery situations.

This comparison was carried out in two well-determined forms Pre-ZTS and Post-ZTS. The Pre-ZTS (Baseline) environment was wired such that DevSecOps pipeline worked without OPA Gatekeeper or Istio rules, so it resembled a classic security approach. The very same pipeline in the Post-ZTS environment was then enhanced with OPA Gatekeeper that set the runtime policy restrictions and Istio Authorization Policies that defined interaction between services.

The analysing process of security contained three validations. First, analysis of the vulnerability in scenes, Static and Container Image was conducted using Trivy to detect the high vulnerability and risk in the application images. Second, to determine the exploitable vulnerabilities during runtime of OWASP Juice Shop application, Dynamic Application Security Testing (DAST) using OWASP ZAP was performed. Third, Runtime Policy Enforcement Verification was guaranteed by the fact that OPA Gatekeeper appropriately prevented the deployment of non-compliant applications and Istio was able to limit network access according to the principles of Zero Trust.

To measure any performance impact, previous total pipeline execution time was recorded Pre-ZTS as well as Post-ZTS environments about build, deployment, and scan phases. It was aimed at learning whether the introduced security measures brought additional delays that could negatively impact the speed of DevSecOps.

To provide cohesion and to eliminate the unpredictable nature of building out and placing components into networks, all the fundamental Kubernetes control plane images, such as metrics-server, local-path-provisioner and CoreDNS, were pre-pulled and stored on Amazon EBS volumes. This enabled them to be deployed into the cluster without having to drill down on external repositories and variability of deployment could therefore be minimized and enable fair comparison of baseline and enhanced environments.

By means of this disciplined approach, there was a definite guideline on how efficiency was to be evaluated on whether the Zero Trust Security is practically implementable in a DevSecOps pipeline and introducing significant compromises in performance.

6.2 Experiment 1: Trivy Image Vulnerability Scans

The designed experiment will measure the level of increased security through the implementation of Zero Trust components on container image vulnerabilities. With GitHub Actions, both same OWASP Juice Shop Docker image (Bkimminich/juice-shop) were scanned at the Pre-ZTS and Post-ZTS points. The same configurations were used to run the scans on Trivy both in the first and second phases, where only CRITICAL and HIGH severity vulnerabilities were filtered with reports saved in the form of tables. Pre-ZTS consisted of the scanning of the image in a cluster with no Istio and no OPA, and Post-ZTS contained both enabled.

The findings indicated that there was no variation in numbers and types of vulnerabilities or in the two phases. This showed that Zero Trust controls or to be more specific, being role-based runtime policies and service segmentation measures, do not introduce a change to the static vulnerabilities in container images. The bottom line is that vulnerability scanning still is

an important but distinct issue from Zero Trust security. This conclusion helps explain that Zero Trust can defend against threats in run time, such as lateral motion, and security of images requires uncompromised focus in the building stage.

Scan Stage	Severity	No. of Vulnerabilities	Vulnerabilities found
Pre – ZTS & Post - ZTS	Critical	5	OpenSSL high-risk CVEs, outdated Node.js modules
Pre – ZTS & Post - ZTS	High	19	Known library vulnerabilities, outdated dependencies

Table 4: Evaluation of Trivy Image Vulnerability Scans

The scan results are the same between Pre-ZTS & Post-ZTS because Trivy is doing image-based scanning. It runs against the container image layers checking for known vulnerabilities in packages, dependencies. As the OWASP Juice Shop image (bkimminich/juice-shop) was the same in both environments, it says the vulnerability counts remain the same.

6.3 Experiment 2: OWASP ZAP Dynamic Security Testing

The purpose of this experiment was to judge the hypothesis of whether the addition of parts of Zero Trust can affect the finding of vulnerabilities in web applications at runtime that can be identified over external scanning. First, and then in the role of Post-ZTS test, OWASP ZAP Baseline Scan was executed against the publicly exposed target of the OWASP Juice Shop with GitHub Actions. The same configurations were applied in every stage, scanning using HTTP protocol to identify standard weaknesses, misconfigurations and vulnerabilities to security issues available in the HTTP response and client-side content of the application.

The findings indicated that the difference between the percentage of vulnerabilities reported did not change in a measurable amount between the two stages. This uniformity helps point out the fact that the Zero Trust measures in place, namely Istio service-to-service communication access restrictions and OPA Gatekeeper enforced policies at runtime, are doing so with the goal of securing the internal traffic paths and applying governance within the cluster, as opposed to affecting the behavior of the external-facing application. Since ingression of Juice Shop was always on at the public ports due to testing grounds, both stages could be scanned using OWASP ZAP to achieve the same targets, yielding the same results. This is a positive indicator that Zero Trust may be injected into a production DevSecOps pipeline without adversely affecting availability of existing external services, even as it offers valuable internal defence against lateral attack and unauthorised deployment.

Scan Stage	Risk Level	Number of Alerts	Issues Detected
Pre-ZTS & Post-ZTS	High	0	None
Pre-ZTS & Post-ZTS	Medium	2	CSP header not set, Cross-Domain Misconfiguration
Pre-ZTS & Post-ZTS	Low	5	Dangerous JS functions, Deprecated Feature Policy, Timestamp Disclosure
Pre-ZTS & Post-ZTS	Informational	4	Suspicious comments, Modern web application, Cacheable content

Table 5: Evaluation of OWASP ZAP Dynamic Security Testing

The results on the ZAP baseline scans were also more or less the same since the scans were done on the OWASP Juice shop application made accessible to the outside world through the NodePort/ Ingress nodes. In addition to side-routing rules, Zero Trust policies (OPA Gatekeeper and Istio Authorization Policies) in our deployment were used mainly to control internal service-to-service (east-west) connections and restrict workload admission policies. The external north-south traffic path taken by ZAP was neither blocked nor filtered, so the web vulnerability surface identified by ZAP was identical.

6.4 Experiment 3: Deployment Speed Comparison (Pre-ZTS vs. Post-ZTS)

In the third experiment, it was necessary to determine whether major changes had been observed in the deployments speed in a CI/CD pipeline because of Zero Trust controls introduction. Using deployment, the speed of GitHub actions is measured to complete the same deployment process in the Pre-ZTS and Post-ZTS situation. For Pre-ZTS pipeline, the concept is to build/deploy the OWASP Juice Shop application, without Istio and OPA. The Post-ZTS pipeline was identical except that Istio sidecars were injected into Pods and OPA Gatekeeper policies were enforced.

The measurements showed that a slight rise in deployment time occurred in the Post-ZTS environment. This was contributed by Istio sidecar injection where additional containers are added to every pod and OPA policy evaluation on admission. Additional time did not have any significant impact on the development processes, however. These findings are relevant to the central research question as the results indicate that the implementation of Zero Trust may not be associated with delays in delivery speed unacceptable to customers, if the deployment process is properly optimized.

Deployment Stage	Avg. Build scan (SonarCloud) (sec)	Avg. Trivy scan (sec)	Avg. ZAP scan (sec)	Total Pipeline Time (sec)
Pre-ZTS	55	54	90	102
Post-ZTS	55	62	90	114

Table 6: Evaluation of deployment speed comparison

6.5 Discussion

The results of this study once again prove that Zero Trust Security principles are applicable in a DevSecOps pipeline and do not affect the continuous delivery pipeline or the availability of external services. Although image-based vulnerability scanners (Trivy) and dynamic application testing tools (OWASP ZAP), did not reveal any differences between the Pre-ZTS and Post-ZTS stages, the outcome confirms that Zero Trust is meant to focus on runtime protection, internal network segmentation and enforcing access based on policies and access requirements, as opposed to changing the essential vulnerabilities of an application. As proven in the experiments with Istio, the rule of service-to-service communication enforcement worked and, therefore, lateral movement in the cluster without a relevant authorization request substantially declined. Likewise, OPA Gatekeeper policies allowed only compliant workloads to be deployed with governance on the Kubernetes level. These data demonstrate that even when the vulnerabilities cannot be eliminated, Zero Trust can create an essential added security layer by obstructing the paths used by an adversary to strike at runtime, which is highly consistent with the security objectives of current DevSecOps environments based on containers.

7 Conclusion and Future Work

The study showed how Zero Trust Security (ZTS) could be applied practically in a DevSecOps pipeline, where the development, testing, and production stages also include a runtime enforcement solution to provide service-level segmentation (e.g. Istio) and policy compliance (e.g. OPA Gatekeeper). The work managed to demonstrate that the structure of these Zero Trust elements could be implemented without drastic effects on the speed of deployment, as well as operational stability, in addition to the pre-existing CI/CD systems. Although no statistically significant changes between stages were demonstrated in static security analyses with Trivy and dynamic scans with OWASP ZAP, this difference reflects the same results we can obtain with the Zero Trust paradigm, which focuses on runtime controls and access and segmentation, instead of inherently correcting the vulnerability. The experiments established that ZTS enhances security position in the rejection of unauthorized internal communications, realm of workload adherence and minimization of lateral movement within cluster. This multilayered strategy of defence means that if the vulnerabilities exist the attack routes are greatly reduced.

In the described framework, the study has been carried out in a closed environment based on one vulnerable app (OWASP Juice Shop) and a small pool of Zero Trust components. The following studies can take this work further to include other Zero Trust guards as a

continuous identity-based authentication of workloads, multi-factor authentication of developers accessing pipelines and automated incident response playbooks based on policy violation. In addition, the adoption of the full-scale network observability based on distributed tracing and telemetry analysis might help to gain better insight into the service behaviour and possible anomalies. The scalability and flexibilities would be tested by scaling up the experiments to support a broad range of microservices, programming stack, and the representative production loads. Lastly, integrating Zero Trust runtime controls with other components of advanced supply chain security, including signed container images, Software Bill of Materials (SBOM) verifications, and dependency risk scoring automated, might provide a secured DevSecOps pipeline that inverts the traditional thought process, and hence considered holistically pre- and post-deployment.

References

Bondalapati, R. C., Kumpatla, L. A., & Karumanchi, S. R. (2025). Secure-by-Design CI/CD Pipelines: A Zero Trust Framework for Cloud-Native Deployment Automation. *Journal of Computer Science and Technology Studies*, *7*(5), 211-219.

Ahmadi, S. Autonomous identity-based threat segmentation in zero trust architectures. arXiv 2025. *arXiv preprint arXiv:2501.06281*.

Avirneni, S. T. (2025). Establishing Workload Identity for Zero Trust CI/CD: From Secrets to SPIFFE-Based Authentication. *arXiv preprint arXiv:2504.14760*.

Avirneni, S. T. (2025). Decoupling Identity from Access: Credential Broker Patterns for Secure CI/CD. *arXiv preprint arXiv:2504.14761*.

Avirneni, S. T. (2025). Intent-Aware Authorization for Zero Trust CI/CD. arXiv preprint arXiv:2504.14777.

Dhandapani, S. (2025). Enhancing Software Supply Chain Security Through STRIDE-Based Threat Modelling of CI/CD Pipelines. *arXiv preprint arXiv:2506.06478*.

Coston, I., Hezel, K. D., Plotnizky, E., & Nojoumian, M. (2025). Enhancing Secure Software Development with AZTRM-D: An AI-Integrated Approach Combining DevSecOps, Risk Management, and Zero Trust. *Applied Sciences*, *15*(15), 8163.

dos Santos, R. F. (2025). Applying Zero Trust to Kubernetes Clusters. *ARIS2-Advanced Research on Information Systems Security*, *5*(1), 57-71.

Rehan, H. Zero-Trust Architecture for Securing Multi-Cloud Environments.

Arora, S., & Hastings, J. (2024, December). Microsegmented Cloud Network Architecture Using Open-Source Tools for a Zero Trust Foundation. In *2024 17th International Conference on Security of Information and Networks (SIN)* (pp. 1-8). IEEE.

Cheenepalli, J., Hastings, J. D., Ahmed, K. M., & Fenner, C. (2025, April). Advancing DevSecOps in SMEs: Challenges and Best Practices for Secure CI/CD Pipelines. In *2025 13th International Symposium on Digital Forensics and Security (ISDFS)* (pp. 1-6). IEEE.

Pan, Z., Shen, W., Wang, X., Yang, Y., Chang, R., Liu, Y., ... & Ren, K. (2023). 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.

Stafford, V. (2020). Zero trust architecture. *NIST special publication*, *800*(207), 800-207.

Amazon Web Services. (2024). *Achieving Zero Trust Security on Amazon EKS with Istio | Amazon Web Services*. [online]
Available at: https://aws.amazon.com/blogs/opensource/achieving-zero-trust-security-on-amazon-eks-with-istio/.

Castro, H. (2024). *Building a Zero Trust Security Framework with DevSecOps in AWS*. [online] Available at: https://www.researchgate.net/publication/387270468_Building_a_Zero_Trust_Security_Framework_with_DevSecOps_in_AWS.

Sarah, W. (2024). *Policy-as-Code and Compliance Automation in DevSecOps Pipeline*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/391274412_Policy-as-Code_and_Compliance_Automation_in_DevSecOps_Pipeline.

Adanigbo, O. S., Adekunle, B. I., Ogbuefi, E., Odofin, O. T., Agboola, O. A., & Kisina, D. Implementing Zero Trust Security in Multi-Cloud Microservices Platforms: A Review and Architectural Framework. *ecosystems*, *13*, 14.

Kang, H., Liu, G., Wang, Q., Meng, L., & Liu, J. (2023). Theory and application of zero trust security: A brief survey. *Entropy*, *25*(12), 1595.

Sanders, G., Morrow, T., Richmond, N. and Woody, C. (2021). *Integrating Zero Trust and DevSecOps*. [online] Dtic.mil. Available at: https://apps.dtic.mil/sti/html/trecms/AD1145432/ [Accessed 4 Aug. 2025].

Vemula, V. R. (2022). Integrating zero trust architecture in DevOps pipeline: Enhancing security in continuous delivery environments. *Trans. Latest Trends IoT*, *5*(5), 1-18.