

INTEGRATION OF ZERO TRUST IN DEVOPS
PRACTICES: ENHANCING SECURITY IN
CONTINUOUS INTEGRATION / CONTINUOUS
DEPLOYMENT (CI/CD) PIPELINES

NUR FATIN NAJHAH BINTI KAMALUDDIN

UNIVERSITI KEBANGSAAN MALAYSIA

INTEGRATION OF ZERO TRUST IN DEVOPS PRACTICES: ENHANCING
SECURITY IN CONTINUOUS INTEGRATION / CONTINUOUS DEPLOYMENT
(CI/CD) PIPELINES

NUR FATIN NAJIHAH BINTI KAMALUDDIN

PROJECT SUBMITTED IN PARTIAL FULFILMENT FOR THE DEGREE OF
MASTER OF CYBER SECURITY

FACULTY OF INFORMATION SCIENCE AND TECHNOLOGY
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI

2025

INTEGRATION OF ZERO TRUST IN DEVOPS PRACTICES: ENHANCING
SECURITY IN CONTINUOUS INTEGRATION / CONTINUOUS DEPLOYMENT
(CI/CD) PIPELINES

NUR FATIN NAJIHAH BINTI KAMALUDDIN

PROJEK YANG DIKEMUKAKAN UNTUK MEMENUHI SEBAHAGIAN
DARIPADA SYARAT UNTUK MEMPEROLEH IJAZAH
SARJANA KESELAMATAN SIBER

FAKULTI TEKNOLOGI DAN SAINS MAKLUMAT
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI
2025

DECLARATION

I hereby declare that the work in this thesis is my own except for quotations and summaries which have been duly acknowledged.

I acknowledge the use of ChatGPT to generate topics for my project.

Prompt: I entered the following prompts: "Relevant issues on DevOps."

Use: I modified the output to match with my project scope as a starting point for initial research before narrowing down the topic for my assessment.

09 January 2025

NUR FATIN NAJIHAH
BINTI KAMALUDDIN
P117258

ACKNOWLEDGEMENT

I am grateful to Allah, the Almighty, for providing me with innumerable blessings, opportunities and knowledge, which have allowed me to complete this thesis at last.

Having Dr. Rossilawati bt Sulaiman as my research supervisor is a great blessing. I am so grateful for all of her support and assistance. This thesis would never have happened without her support and direction. Next, I would like to sincerely thank my examiners for their insightful remarks and direction that went into this thesis, as well as for their kind donation of their valuable time.

I would want to express my gratitude to all of the postgraduate cyber security students as well as my classmates and friends for their support, camaraderie and facilitation of a positive learning environment throughout my time at UKM. My deepest appreciation to the visionary leaders of my corporate family for their unwavering support, invaluable encouragement, and steadfast belief in me throughout my thesis journey.

Thank you, my amazing Umami, Puan Azlina bt Abdullah, my beloved Walid, Encik Kamaluddin bin Talib, my siblings, for always trying to keep me on track no matter what obstacles I had to overcome during the academic year and in finishing this thesis. I express my gratitude for all of your unwavering support and encouragement.

Lastly, I want to express my gratitude to myself for persevering and giving it my all the way through. It is my goal that this study will, in some little way, advance the area and be useful as a guide for further research.

ABSTRAK

Peningkatan ancaman siber memerlukan penerapan langkah-langkah keselamatan yang kukuh dalam proses pembangunan perisian, terutamanya dalam saluran Integrasi Berterusan dan Penghantaran Berterusan (CICD). Disertasi ini menangani masalah keselamatan saluran CICD dengan mengintegrasikan prinsip *Zero Trust* ke dalam amalan DevOps. Soalan penyelidikan memfokuskan pada mengenal pasti kelemahan dalam proses CICD semasa dan menilai bagaimana *Zero Trust* dapat mengurangkan risiko ini. Tujuan kajian ini adalah untuk membangunkan strategi atau amalan terbaik yang mengintegrasikan prinsip *Zero Trust* ke dalam kitaran hayat DevOps, dengan itu meningkatkan keselamatan saluran CICD. Kajian ini tidak memfokuskan pada kawasan geografi tertentu tetapi berlaku kepada mana-mana organisasi yang melaksanakan amalan DevOps. Kaedah penyelidikan melibatkan pendekatan campuran, termasuk ulasan literatur dan temu bual bersama pakar. Penemuan kajian menunjukkan bahawa mengintegrasikan *Zero Trust* ke dalam amalan DevOps secara signifikan meningkatkan tahap keselamatan saluran CICD. Hasil utama termasuk pembangunan strategi baru untuk pengesahan identiti, akses paling rendah, pemantauan berterusan dan mikro-segmen dalam aliran kerja DevOps. Pelaksanaan strategi-strategi ini membawa kepada peningkatan pengesanan ancaman, pengurangan permukaan serangan dan peningkatan masa tindak balas insiden. Tesis ini menyumbang kepada bidang ini dengan menyediakan pendekatan berstruktur untuk mengintegrasikan prinsip *Zero Trust* ke dalam DevOps, menawarkan pandangan praktikal dan rangka kerja yang disahkan untuk organisasi yang ingin mengamankan proses penghantaran perisian mereka di tengah-tengah landskap ancaman yang sentiasa berubah. Kajian ini memperkenalkan istilah dan strategi baru untuk penggabungan lancar *Zero Trust* dalam saluran CICD, menetapkan preseden untuk penyelidikan dan pembangunan masa depan dalam amalan DevOps yang selamat.

ABSTRACT

The increasing sophistication of cyber threats necessitates the adoption of robust security measures within software development processes, particularly within Continuous Integration and Continuous Deployment (CICD) pipelines. This dissertation addresses the problem of securing CICD pipelines by integrating Zero Trust principles into DevOps practices. The research questions focus on identifying vulnerabilities in current CICD processes and evaluating how Zero Trust can mitigate these risks. The aim of the study is to propose strategies that embeds Zero Trust principles into the DevOps lifecycle, thereby enhancing the security of CICD pipelines. The study does not focus on a specific geographical area but applies to any organization implementing DevOps practices. The research method involves a mixed approach, including a literature review and interview with the experts. The findings reveal that integrating Zero Trust into DevOps practices significantly improves the security posture of CICD pipelines. Key results include the development of new strategies for identity verification, least privilege access, continuous monitoring and micro-segmentation within DevOps workflows. The implementation of these strategies leads to enhanced threat detection, reduced attack surfaces and improved incident response times. This project report contributes to the field by providing a structured approach in integrating Zero Trust principles into DevOps, offering practical insights and strategies for organizations seeking to secure their software delivery processes amidst an evolving threat landscape. The study introduces new strategies for the seamless incorporation of Zero Trust within CICD pipelines, setting a precedent for future research and development in secure DevOps practices.

TABLE OF CONTENTS

		Page
DECLARATION		iii
ACKNOWLEDGEMENT		iv
ABSTRAK		v
ABSTRACT		vi
TABLE OF CONTENTS		vii
LIST OF TABLES		x
LIST OF ILLUSTRATIONS		xi
LIST OF ABBREVIATIONS		xii
CHAPTER I	INTRODUCTION	
1.1	Research Background	1
1.2	Problem Statement	9
1.3	Research Aim And Objectives	12
1.4	Research Questions	12
1.5	Significance of Research	12
1.6	Research Scope	13
1.7	Project Organization	14
1.8	Summary	15
CHAPTER II	LITERATURE REVIEW	
2.1	Introduction	16
2.2	DevOps Methodology	16
2.3	Current State of Security in CI/CD	24
	2.3.1 Vulnerabilities	28
	2.3.2 Legacy systems	30
2.4	Threats and Risks in the Existing CI/CD Process	32
2.5	Access Control Solutions	36
	2.5.1 Defense in Depth (DiD)	36
	2.5.2 MITRE Att&ck Framework	40
	2.5.3 Cloud Security Alliance (CSA)	42
	2.5.4 Zero Trust Concept	44
	2.5.5 Discussion	48

2.6	Possible Technology Solutions and Best Practices to Secure DevOps	50
2.6.1	Existing Frameworks	52
2.6.2	WhiteSource	52
2.6.3	Zed Attack Proxy	53
2.6.4	AquaSecurity	53
2.6.5	Code AI	53
2.6.6	Prisma Cloud	53
2.6.7	SaltStack	54
2.6.8	IriusRisk	54
2.6.9	Discussion	55
2.7	Integration Strategies to Implement Zero Trust in DevOps Workflow	56
2.7.1	Integrating Security Into CI/CD Pipelines	56
2.8	Summary	57
CHAPTER III METHODOLOGY		
3.1	Introduction	58
3.2	Research Design	58
3.3	Data Collection Methods	59
3.3.1	Literature Review	59
3.3.2	Expert Interview	66
3.4	Development of Interviews	69
3.4.1	Preliminary Validation of Interviews	70
3.4.2	Interviews Design	71
3.5	Data Analysis	75
3.5.1	Descriptive Statistic	75
3.6	Summary	76
CHAPTER IV RESULTS AND DISCUSSION		
4.1	Introduction	77
4.2	Presenting Data of Interview	77
4.3	Analyzing Interview Result	77
4.4	Results of Interview	78
4.4.1	Demographic Information	78
4.4.2	Elements of Threats	80
4.4.3	Elements of Technology	82
4.4.4	Experts Recommendations and Suggestions	85
4.5	Development of Enhanced DevOps Workflows	91

	4.5.1 Discussion	93
4.6	Summary	96
CHAPTER V CONCLUSION AND FUTURE WORKS		
5.1	Introduction	97
5.2	Research Discussion	97
5.3	Contribution Of The Study	99
5.4	Future Works	100
REFERENCES		104
APPENDICES		
Appendix A	Interviews Form	116
Appendix B	Interviews Results	123

LIBRARY FETSM

LIST OF TABLES

Table No		Page
Table 2.1	Threat Enumeration using STRIDE	33
Table 2.2	Summary Frameworks	49
Table 2.3	Summary Technologies Solutions	55
Table 3.1	Approaches to Literature Reviews	60
Table 3.2	Inclusion Criteria and Exclusion Criteria	62
Table 3.3	Cybersecurity Malaysia Information Security Professional Requirements	67
Table 3.4	Structure of the Interviews	72
Table 3.5	Sub-section A Questions	73
Table 3.6	Sub-section B Questions	74
Table 3.7	Sub-section C Questions	75
Table 4.1	Demographic Data of Experts	78
Table 4.2	Roles and Responsibilities	79
Table 4.3	Frequency and Cumulative Percentage of Agreement for Elements of Threats	81
Table 4.4	Frequency and Cumulative Percentage of Agreement for Elements of Technology	83
Table 4.5	Analysis of Interview Results About Threats and Risks That Involves Information Security Within the Existing CI/CD Process	86
Table 4.6	Current State of Security within the CI/CD Process	88
Table 4.7	Best Practices Within Current DevOps Process	91
Table 4.8	List of Standards	91
Table 4.9	Enhanced Activities in DevOps after Integrating Zero Trust	92
Table 4.10	State of DevOps Before and After Integrating Zero Trust	93
Table 4.11	Comparison Results and Vighe (2024)	94

LIST OF ILLUSTRATIONS

Figure No.		Page
Figure 1.1	DevOps LifeCycle	2
Figure 1.2	CI/CD Pipeline	4
Figure 2.1	Zero Trust Access	18
Figure 2.2	Security Factors in CI/CD Pipelines	24
Figure 3.1	Data Collection Process from Interview	69

LIBRARY FETSM

LIST OF ABBREVIATIONS

CD	Continuous Deployment
CI	Continuous Integration
CNII	Critical National Information Infrastructure
CSM	Cybersecurity Malaysia
DAST	Dynamic Application Security Testing
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act
IAM	Identity and Access Management
IaC	Infrastructure as Code
MFA	Multi-Factor Authentication
NIST	National Institute of Standards and Technology
OSS	Open-Source Software
OWASP	Open Web Application Security Project
PDP	Policy Decision Point
PEP	Policy Enforcement Point
ZAP	Zed Attack Proxy
ZT	Zero Trust
ZTA	Zero Trust Architecture

CHAPTER I

INTRODUCTION

1.1 RESEARCH BACKGROUND

The software development community has been compelled to reassess how new applications are created and deployed to the rise of the cloud computing paradigm and improvements in development procedures. Organizations can now achieve agility and velocity that were previously unattainable with traditional methods of development and delivery by implementing DevOps ideas and processes (J. Humble and D. Farley, 2010).

The phrase "DevOps" was introduced in 2009 by Patrick Debois, and it has since developed into an extensive collection of practices focusing on automation, teamwork and ongoing enhancement. The core tenets of DevOps are influenced by agile methods, lean approaches and the Theory of Constraints. According to A. Luz and F. Alih (2023), DevOps is a concept created by merging "development" (Dev) and "operations" (Ops), represents a cultural and technical strategy designed to connect software development and IT operations groups. At its essence, DevOps aims to enhance collaboration, communication and integration among these usually isolated functions, ultimately striving to provide high-quality software products and services more effectively and dependably. DevOps focuses on automation, continuous integration, continuous delivery (CI/CD) and applying agile principles to enhance the software development lifecycle (SDLC). By dismantling organizational silos and promoting a culture of collective accountability, DevOps allows teams to produce software more swiftly, adapt faster based on feedback and react promptly to evolving business demands.

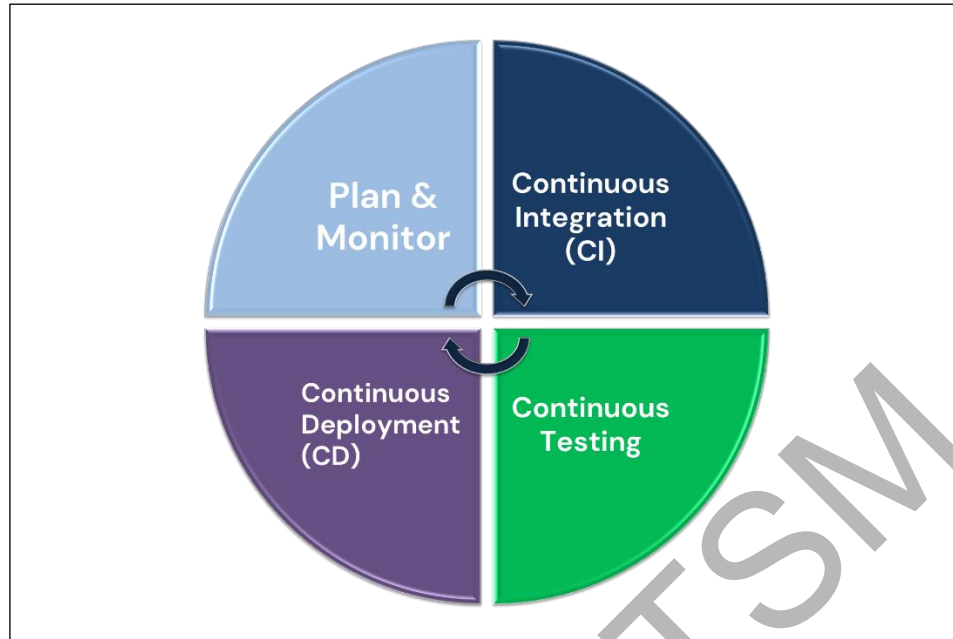


Figure 1. 1: DevOps Lifecycle

Source: Gokarna, 2021

DevOps is an extension of Agile principles, integrating multiple phases to ensure rapid, continuous, and secure software delivery. The DevOps lifecycle combines Agile methodologies with automation, forming a seamless workflow that includes (1) Plan and Monitor, (2) Continuous Integration (CI), (3) Continuous Testing (CT), and (4) Continuous Deployment (CD) (Gokarna, 2021).

The Plan and Monitor phase align with Agile's iterative planning, ensuring continuous feedback and improvement (Bass, 2018). CI supports Agile's emphasis on frequent code integration, promoting collaboration and reducing integration issues (Forsgren et al., 2018). CT enhances Agile testing by automating security and functionality tests, ensuring reliability and compliance (Sharma et al., 2022). CD enables Agile's goal of delivering working software frequently with minimal manual intervention, improving deployment efficiency and reducing downtime (Gruhn & Schäfer, 2020). By embedding security at every stage, DevOps extends Agile beyond development, ensuring a secure, automated, and scalable software lifecycle (Yadav et al., 2021).

Organizations aiming to establish a secure DevOps environment combine methods and procedures to aid in safeguarding applications and their information throughout all phases of operation. Integration begins at the planning phase and extends throughout the entire lifespan of an application, encompassing continuous integration and continuous deployment. Security is integrated with development and operations processes, developing a method illustrated in Figure 1.1. DevOps with integrated security primarily centres on transitioning it to the beginning of a software delivery pipeline rather than placing it at the conclusion. Security is incorporated into every stage - planning, integrate, testing, and deploy. The best practice is to automate security measures to improve detection and mitigation of possible threats or risks during initial stages of development while concurrently preserving the speed of the DevOps procedure.

Secure DevOps requires increased collaboration and enhanced transparency. Security should be viewed as a collective accountability from each team member involved in the application progress and provision. Incorporation of security procedures into the delivery process of created software which DevOps incorporates early and consistent risk management through the use of testing and monitoring tools that offer perspectives on weaknesses. It might also foster a culture of security, aiding the progression to consider security factors consistently. Developers and testers will consequently be more aware of security risks encountered during each stage. The expenses of software development may decrease due to shortened delivery duration. When the security vulnerabilities are addressed promptly, they might be improved initially with greater efficiency than afterwards.

In the Plan and Monitor phase, DevOps teams define project scope, requirements, and objectives while continuously tracking performance and security metrics. Security features in this phase include threat modelling and risk assessment (Zhang et al., 2021), security monitoring tools like SIEM (Security Information and Event Management), and compliance checks with standards such as OWASP and ISO 27001. The Continuous Integration (CI) phase involves frequent code merges, triggering automated builds and tests. To enhance security, DevOps teams implement Static Application Security Testing (SAST) (Kaur, 2020), secrets management solutions like HashiCorp

Vault, and dependency vulnerability scanning tools such as OWASP Dependency-Check. In the Continuous Testing (CT) phase, automated tests validate code functionality, performance, and security. Security measures here include Dynamic Application Security Testing (DAST) (Sharma et al., 2022), API security testing using tools like Postman and OWASP ZAP, and penetration testing automation with Metasploit or Burp Suite. Finally, the Continuous Deployment (CD) phase ensures that validated code is automatically deployed to production with minimal manual intervention. Security best practices in this phase involve Infrastructure as Code (IaC) security scanning using tools like Checkov and Terraform Sentinel, implementing a Zero-Trust security model (Yadav et al., 2021), and conducting container security scanning with solutions like Aqua Security and Trivy.

Central to this transformation are the practices of Continuous Integration (CI) and Continuous Delivery (CD), which have been extensively studied and implemented in various contexts. Organizations have been able to switch from code development to "sprints" which is a team focuses on completing a specific set of tasks or goals to deliver incremental progress on a project thanks to the automation of the tasks that previously slowed down this process. This entails quicker development cycles and the quick release of new features, with a quicker time to market as the ultimate goal. This is accomplished through the use of a set of procedures and equipment referred to as continuous delivery and integration.

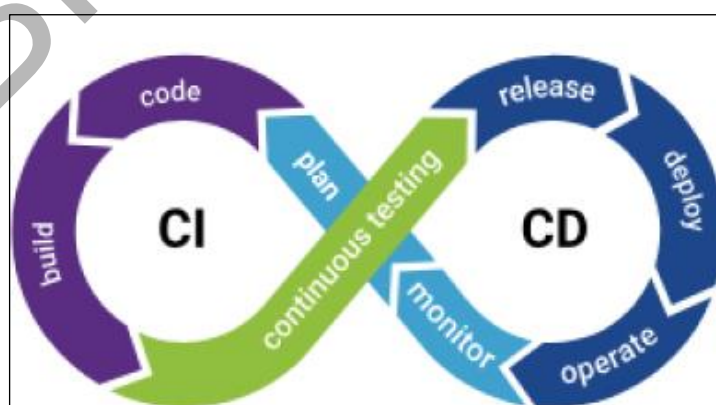


Figure 1. 2 CI/CD Pipeline

Source: Dieu Thu Vu, 2024

The approach to explore and executing essential practices of Continuous Integration (CI) and Continuous Delivery (CD) within the DevOps framework is organized into multiple phases, all of which are vital for the effective adoption of CI/CD practices. The planning stage includes grasping the existing condition of development and operational processes, pinpointing issues and establishing clear goals for CI/CD execution. This stage involves performing a comprehensive evaluation of current workflows, release cycles and operational practices. It involves stakeholders from different departments aids in collecting requirements and expectations, crucial for establishing precise goals and success criteria for CI/CD implementation.

Based on study by Subash Manala (2024), minimizing deployment duration, boosting code quality and increasing teamwork among groups could achieve the success criteria for CI/CD implementation. Choosing the appropriate tools is a vital stage in the methodology. The selection procedure considers compatibility with the current technology stack, simplicity of integration with version control systems and build tools, backing for automation and scalability and the presence of community support and documentation.

The CI/CD pipeline is a crucial component of modern DevOps practices, enabling organizations to automate and streamline the software development lifecycle. It integrates Continuous Integration (CI) and Continuous Deployment (CD) to facilitate frequent code changes, automated testing, and seamless deployment, ensuring higher efficiency, reliability, and security in software delivery. By implementing CI/CD, teams can reduce manual intervention, minimize errors, and accelerate time to market.

Figure 1.1 illustrates the DevOps lifecycle; meanwhile, Figure 1.2 shows the CI/CD Pipelines. Compared to Agile phases, Continuous Integration (CI) is the combination of the Code and Build phases, where developers frequently commit code to a shared repository, triggering automated builds and early-stage tests to detect and fix integration issues quickly (Forsgren et al., 2018). On the other hand, Continuous Deployment (CD) consists of the Release, Deploy, and Operate phases, ensuring that validated code is packaged, released, deployed to production, and actively monitored for performance, security, and reliability (Gruhn & Schäfer, 2020). By structuring DevOps in this way, organizations can achieve faster, more secure, and automated software delivery cycles.

Every phase guarantee that code modifications are incorporated, evaluated and launched effectively and dependably. Automated testing frameworks are utilized to conduct unit, integration and end-to-end tests, with the outcomes sent back to Jenkins to guarantee that only successful builds are incorporated into the main branch. Scripts for deployment automation and Kubernetes manifests enable smooth transitions to staging and production settings. The implementation stage consists of establishing and configuring the chosen tools and the created CI/CD pipeline.

Since code is committed at one end, tested throughout the steps of build, staging, production and commit and then made viable for release, CI/CD can be thought of as a pipeline. Iterative methods are also used in CI/CD practices to continuously analyse data and solicit input in order to improve the CI/CD procedure. A logical unit in the delivery process is a stage in the CI/CD pipeline (AWS Whitepaper, 2021). Every step serves as a doorway to verify the overall code. Since more of the code would have been examined in later stages, the quality of the code improves. When issues are found early in the pipeline, the code is prevented from moving forward.

If the software fails the testing phases, all further builds and releases are terminated. Therefore, a continuous integration/continuous development (CI/CD) pipeline builds the solution and runs a series of automated tests whenever a change is committed, giving developers quick feedback on their changes and increasing the value of the code (AWS Whitepaper, 2021). CI/CD pipelines, to put it briefly, are processes that guide source code through several stages, including building, functional testing, security scanning, packaging, and deployment, all of which are aided by automated tools and feedback mechanisms (Ramaswamy, 2022). Building, testing and deploying programs quickly and reliably is the main objective of contemporary software development teams (Daniel Pohl, 2020).

Before submitting work to the central repository, developers must run unit tests on all newly written code as soon as possible. The simplest and least expensive fixes are still those for issues discovered early in the software development process. It is recommended that developers frequently review their code in a central repository to identify mistakes early and speed up the entire development cycle (Akhil Jain, 2021).

To obtain quick feedback, a properly scaled build process should handle all tasks, that may occur during the commit step (AWS Whitepaper 2021). Therefore, developers communicate changes through Continuous Integration (CI), which involves routinely committing code to source control. Automated test scripts are created and incorporated into the CI pipeline to guarantee strong code quality. Docker files and Kubernetes manifests are created to package and manage applications, whereas automated deployment scripts implement updates to staging and production environments.

Monitoring and logging tools are set up to collect metrics and logs, offering immediate insights into application performance. Evaluating the CI/CD pipeline is essential to confirm it operates as expected and achieves set objectives. The pipeline executes with modified sample code to confirm each step of the process. Stress and load evaluations measure the pipeline's efficiency and scalability, with any problems or bottlenecks, like lengthy build times or unreliable tests, being recognized and resolved. Development and operations teams offer insights on the pipeline's usability and efficiency, crucial for ongoing enhancement.

Continuous Delivery (CD) involves more functional testing as well as the deployment of the application code in a staging environment. Infrastructure as code (IaC) is also used in the construction of the production environment, which comes next in the deployment/delivery pipeline sequence following the staging environment. This will make every successful build in continuous delivery is immediately pushed to all pre-production environments, with the understanding that quality is improving at every level.

The assessment outcomes are utilized to pinpoint areas needing enhancement, and input from stakeholders is integrated to ensure the pipeline aligns with the organization's changing requirements and goals. In conclusion, this method offers a thorough strategy for executing Continuous Integration and Continuous Delivery practices within a DevOps framework. By emphasizing planning, choosing tools, designing pipelines, executing, testing and assessing, organizations can attain quicker, more dependable and superior software releases, fostering enhanced business success (Subash Manala, 2024).

In conclusion, the rapid adoption of DevOps methodologies has compelled a re-evaluation of long-standing security paradigms. Concurrently, Zero Trust (Delbene, 2019) is becoming more popular as a proactive security strategy built on least privilege and continuous authentication. According to Rose (2020), Zero trust (ZT) encompasses a set of principles and notions aimed at reducing uncertainty when implementing precise, least privilege per-request access choices in information systems and services, especially in a network perceived as compromised. Zero trust architecture (ZTA) is a cybersecurity strategy for an organization that employs zero trust principles and includes component interactions, workflow design and access regulations. Thus, a zero-trust enterprise refers to the network infrastructure (both physical and virtual) and operational policies established for a business as a result of implementing a zero-trust architecture strategy.

This definition emphasizes the core of the matter, which is the aim to stop unauthorized access to data and services while also ensuring that access control enforcement is as detailed as possible. In other words, permitted and sanctioned subjects (a mix of user, application and device) can access the data, excluding all other subjects. To advance this idea, the term “resource” can replace “data,” making ZT and ZTA focused on resource access rather than solely data access. To reduce uncertainties (since they cannot be completely removed), the emphasis is on authentication, authorization, and narrowing implicit trust zones while ensuring availability and decreasing time delays in authentication processes. Access rules are designed to be as detailed as possible to ensure the minimum privileges necessary to carry out the requested action.

According to Rangnau (2020), agile development methodologies lower the risks involved in complex software development projects. The lack of sufficient security expertise in DevOps teams is typically cited by the involved parties when flaws in software produced via agile development are found (Rangnau, 2020). Poor security testing causes insecure systems, which in turn leads to inadequate security competency in DevOps (Conor Deegan, 2020).

There is an increase in attacks and threat actors are not taking a break but instead they are becoming more adept at willingly hacking targets. As a result, companies using CI/CD and learn how to quickly and effectively implement security measures in order

to counteract this threat. Zero Trust aids with precisely this goal by offering the essential insights on more robust and effective security technique implementation and execution (Rangnau, 2020). Development teams are using the agile development process to finish projects on schedule and within budget.

1.2 PROBLEM STATEMENT

Enterprises seeking to establish a secure DevOps framework incorporate procedures and instruments to facilitate the safeguarding of apps and their information across the whole lifecycle of the system. Planning and design are the early stages of integration, which continues throughout an application's lifetime and includes continuous integration and continuous delivery. The major goal of DevOps with integrated security is to move security from the end of a software delivery pipeline to the beginning in every stage (planning, development, testing, deployment and operation). When it is feasible, security should be automated with the use of testing and monitoring tools (Adam Skurla, 2018).

In this study, real-life problems in the workplace faced by the organization are securing the CI/CD pipeline, particularly in protecting code integrity, managing access control, and preventing supply chain attacks. Traditional security models often rely on perimeter-based defences, which are inadequate in modern DevOps environments where deployments occur frequently across cloud and hybrid infrastructures. As a result, unauthorized access, credential leaks, and insider threats pose critical risks to the software development lifecycle.

To address these challenges, integrating the Zero Trust security model into CI/CD pipelines has become essential. Zero Trust enforces the principle of “never trust, always verify,” ensuring strict identity verification, least privilege access, and continuous monitoring across all DevOps stages. However, many organizations struggle with implementing Zero Trust due to complexities in policy enforcement, performance trade-offs, and integration with existing DevOps workflows. By incorporating Zero Trust into CI/CD pipelines, organizations can enhance security without compromising agility, ensuring that only authenticated and authorized entities can interact with critical

DevOps components. This research explores best practices, and strategies for achieving a secure CI/CD pipeline with Zero Trust integration.

Based on the study made by Vighe (2024), examining CI/CD pipelines requires developing models of the pipeline's framework, workflows and relationships, followed by evaluating different security elements. Start by recognizing the essential elements of a CI/CD pipeline where security testing occurs, such as source code repositories, build servers, testing frameworks and deployment processes. Once the key elements of the CI/CD pipeline responsible for security testing are identified, it is essential to determine vital security tools designed to uncover and address security vulnerabilities in the initial phases of development.

A development team may use a potentially dangerous application technology without fully considering the risks that could arise. According to Chittala (2024), the use of untested containers or a public secure API could all serve as examples. Inadequate development techniques or human mistake could also introduce risks into the code. The absence of a proper security review may cause these dangers to remain hidden for extended periods of time.

This maintains the DevOps process's velocity while ensuring improved identification and mitigation of possible hazards or dangers throughout earlier stages of development. By adding security procedures to the DevOps software delivery pipeline, early and systematic risk management is achieved through the use of testing and monitoring tools that offer insights into vulnerabilities. It might also encourage a security-conscious culture, encouraging the development team to consider security issues constantly.

Imparting Zero Trust values into DevOps procedures is a critical undertaking that aims to improve security in pipelines for Continuous Integration/Continuous Deployment (CI/CD). According to Adam Skurla (2018), the underlying issues with traditional security methods, which frequently rely on perimeter-based protections and implicit trust within the network, are the root of the current issue. These traditional methods are not up to the complexity of today's cyber threats in the setting of dynamic,

cloud-native environments and the lightning-fast software delivery speed enabled by DevOps practices.

The issue is further made worse if there are no thorough guidelines and best practices for integrating Zero Trust into DevOps workflows. Organizations frequently find themselves struggling to define access restrictions, set trust boundaries, and integrate security controls into CI/CD processes, among other practical implementation issues. Moreover, the predominance of post-development security considerations and manual security procedures in DevOps pipelines creates bottlenecks that impair the speed and agility of software delivery. Not only does this go against the principles of DevOps, but it also exposes enterprises to risks associated with compliance and security breaches (Adam Skurla, 2018).

The DevOps team uses increments and iterations to produce software more quickly. According to Akujobi (2021), the security team strives to make sure that software is resilient to threats while considering the ROI (return on security investment). They also make sure that in the event that security fails, there are backup plans in place. The two teams appear to be competing with one another since one wants rapid deployment while the other wants robustness, which takes time. Many businesses and organizations are unaware of the effectiveness of their security measures.

According to research conducted by Help Net Security (2021), 53% of businesses are unsure of the functionality of their security products. It becomes crucial to comprehend how much security may be enhanced by the addition of security solutions and how this may affect agility. Another issue is how security teams can operate more in line with DevOps, where security enhancements are integrated.

The question of whether these technologies enhance the security of the DevOps process becomes crucial. This objective enables a company to determine whether or not these enhancements are useful to them. Security is typically considered an afterthought in DevOps. To help enhance the industrial complex as a whole, security inspections must be moved further to the left. Nevertheless, a lack of awareness of their value makes

adding some of these checks unpopular. It is crucial to ascertain whether these security tool additions enhance DevOps security for this reason.

1.3 RESEARCH AIM AND OBJECTIVES

This research focuses on the development of new techniques for that encompasses the scope of security within the CI/CD process. In order to fulfil the purpose of this research, several objectives need to be achieved as follows:

1. To identify threats and risks that involves information security within the existing CI/CD process;
2. To identify current state of security within the CI/CD process; and
3. To propose integration of zero trust strategies within current CI/CD process.

1.4 RESEARCH QUESTIONS

In order to conduct this study, several questions need to be answered as follows:

1. What are the threats and risks that involves information security within the existing CI/CD process?
2. What are the procedures obtained from current state of security in within the CI/CD processes?
3. How the propose integration of zero trust strategies within current CI/CD process be developed?

1.5 SIGNIFICANCE OF RESEARCH

Organizations need information to make decisions. While many firms may have different information strategies, DevOps always strives to construct a system that meets the criteria and can be developed without issues within the allocated time. The main objective of this study is to look at suitable techniques for improving the CI/CD process.

Factual information, unlike technical knowledge, is solely understandable by experts or information holders; information technology or training are not necessary.

A thorough comprehension can prevent companies from being fixated on technical issues at the price of realizing how important knowledge is. A clear understanding can aid in preventing miscommunications that could lead to information loss or exposure pertaining to the information held by the business.

The information security framework and techniques to improve security will be suggested based on the experience gained from the study, which is essential research material. This security framework is necessary for small and midsize enterprises to maximize management, defense, day-to-day operations and future growth. The market will be satisfied by this study, which will also drive the trend toward improved and safer environments. The critical environment's sectoral dependencies will be lessened as a result of this study. The findings of this study will also help businesses determine how DevOps could inspire a company or organization to improve quality through the effective and successful implementation of DevOps.

1.6 RESEARCH SCOPE

In keeping with previous research that has identified challenges, concerns and dangers during the deployment phase, this study focuses on core DevOps components. The study's goal for analysis is small and midsize enterprises (SMEs) as the organization is facing real-life problems in the industry. The new information security framework for CI/CD will be created in response to the findings of the analyses. Interviews with DevOps engineers and security experts to gather information for this study are conducted.

Obtaining consent for interviews presents a hurdle because organizations have stringent policies about revealing company information. The research area is constrained due to Malaysia's poor DevOps practices, despite the fact that information is a strategic advantage for all enterprises.

1.7 PROJECT ORGANIZATION

This dissertation is divided into five chapters including research part. Chapter one discusses some components and introduces research background and research problem statement. Research questions, aim, objectives and scope of the study is defined based on the analysis of the current trend of methodology used by most of companies. The significance of the study discusses the importance and challenges of this research.

The second chapter discusses a review of the literature based on research questions. After a thorough screening process, a subset of the many publications was chosen to highlight important aspects of the study issues. From those articles, numerous dangers and threats were discovered. It will include further details and justifications for suggestions on how to improve security in CI/CD pipelines. Although DevOps principles are part of the current CI/CD pipeline, there are certain shortcomings, vulnerabilities and difficulties with improving security in DevOps processes. Analyses of case studies will be used to construct a conceptual framework.

Chapter three explains the methodology and methods used to collect data. The detailed steps on designing the framework and data acquisition method are explained in this chapter. Tools and resources selected are explained and this chapter will elaborate on the protocols in paper search process, paper selection criteria and interview protocol.

Chapter four presents the analysis of data collected. Percentage and frequency analysis will be used to analyze the input from expert opinion to get a clearer understanding of several elements of the survey distributed. The analysis results help this research to fulfill its objectives which is to propose strategies in enhancing security in CI/CD pipelines.

Chapter five discusses the result of the information gathering and the proposed strategies. Some extend points of the new strategies will be discussed in the context of security policies and the aim of the strategies to solve current issues, threats and vulnerabilities. Some recommendations for future researchers and development will be provided as well.

1.8 SUMMARY

The results of this study will be used for development, assisting SMEs in raising their level of cybersecurity protection and environment. This will assist in introducing a few elements and pertinent subjects. It will be beneficial to offer fundamental knowledge and expertise about the evolution of DevOps. DevOps ought to be viewed as a roadmap for achieving Malaysia's intended objectives for digital governance. Thus, it is imperative that this research be carried out in order to help firms enhance the CI/CD procedures that they already use.

LIBRARY FETSM

CHAPTER II

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter outlines the literature review conducted on the topic examined. The literature review included a semi-systematic assessment of previous research, that comprises textbooks, research papers and various pertinent materials. This offers an analytical overview, recap and evaluation of prior research relevant to the study issues related to this project. This chapter also analysed various threats and risks present in the current solutions. This chapter is composed of four (4) primary stages that encompass a comprehensive overview of the subject: (1) current software development methodologies, (2) the current state of security in CI/CD, (3) potential threats and risks present in the CI/CD process, and (4) access control strategies. This chapter seeks to establish a foundation by pinpointing deficiencies and obstacles within the existing DevOps practices, the foundation for suggesting enhancements that incorporates Zero Trust for efficient and safer DevOps methods.

2.2 DEVOPS METHODOLOGY

As mentioned in Chapter 1, DevOps is an approach that combines the traditional software jobs and focuses on improving communication to increase the frequency of software deployments and maintain their quality. In essence, DevOps is a methodology that brings together tools and procedures to support the advancement of core principles, such as collaboration, automation, continuous integration (CI), continuous delivery (CD), infrastructure as code and monitoring (Pratibha Jha, 2018). The term "DevOps" frequently refers to the increasingly proficient development that supports a collaborative working relationship between IT and improvement activities, resulting in

a rapid flow of planned work (high convey rates) while also enhancing the stability, security, dependability and flexibility of the creative environment. It may be acknowledged that "DevOps" combines operational responsibilities with progress. However, DevOps is not associated with grouping standard tasks into a single unit. Alternatively, DevOps may be a set of policies and procedures that promotes programming excellence throughout the SDLC life cycle. Distinguishing boundaries between programming and conveyance disciplines allows for continuous modifications that improve quality and speed up advertising. A. Luz and F. Alih (2023) states that key principles of DevOps as follows:

1. Collaboration: DevOps encourages collaboration and communication among development, operations and other stakeholders throughout the software delivery process;
2. Automation: Automation plays a crucial role in DevOps, automating repetitive tasks such as code builds, testing, deployment and infrastructure provisioning to increase efficiency and reduce manual errors;
3. Continuous Integration (CI): CI involves automatically building and testing code changes as they are committed to the version control system, ensuring early detection of integration issues;
4. Continuous Delivery/Deployment (CD): CD extends CI by automating the deployment process, allowing organizations to deliver software changes to production rapidly and reliably;
5. Infrastructure as Code (IaC): IaC treats infrastructure configurations as code, enabling automated provisioning, management and scaling of infrastructure resources using code-based tools and practices; and
6. Monitoring and Feedback: DevOps focuses on ongoing observation of application performance and user feedback to pinpoint areas needing enhancement and guide future development initiatives.

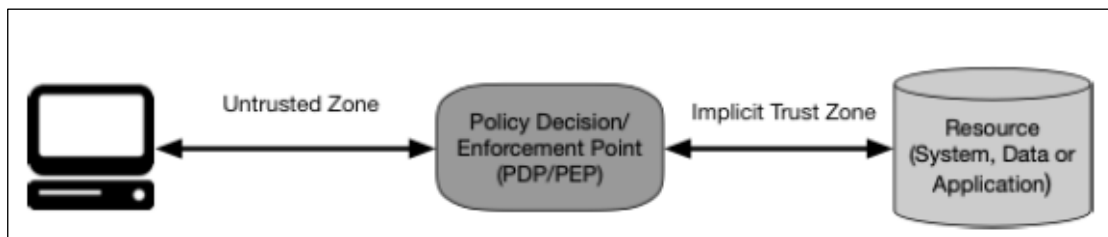


Figure 2. 1 Zero Trust Access

Source: Rose et al., 2020

In the DevOps methodology, security enforcement is a critical aspect, and Policy Decision Points (PDPs) and Policy Enforcement Points (PEPs) play a key role in implementing Zero Trust Architecture (ZTA) within CI/CD pipelines. Based on Figure 2.1 by Rose et. al, (2020), the zero-trust access begins with system should verify that the subject is genuine and the request is legitimate. The PDP/PEP makes an appropriate decision to grant the subject permission to utilize the resource. This means that zero trust pertains to two fundamental aspects which are authentication and authorization. In general, organizations must create and uphold flexible risk-oriented policies for resource access and establish a framework to guarantee that these policies are applied accurately and uniformly to each resource access request. This indicates that a business should not depend on assumed trustworthiness, where meeting a basic authentication standard leads to the assumption that all later resource requests are equally legitimate.

The “implicit trust zone” denotes a region where every entity is trusted to a minimum degree equivalent to the most recent PDP/PEP gateway. As an illustration, think about the passenger screening system at an airport. Every passenger goes through the airport security checkpoint (PDP/PEP) to reach the boarding gates. The terminal area is filled with passengers, airport staff, aircraft crew, all of whom are regarded as trusted individuals. In this model, the implicit trust zone refers to the boarding zone.

The PDP/PEP enforces a series of controls to ensure that all traffic outside the PEP maintains a uniform trust level. The PDP/PEP cannot enforce extra policies beyond its position within the traffic flow. In order for the PDP/PEP to be as precise as possible, the implicit trust zone needs to be minimized. Zero trust offers a framework of principles and ideas focused on bringing the PDP/PEPs nearer to the resource.

The concept involves clearly authenticating and authorizing all individuals, resources, and processes that comprise the organization. The typical situation includes

a business with a main headquarters and one or more remote sites that lack a physical network link owned by the enterprise. Workers at the remote site might lack a complete company-owned local network yet still require access to company resources to carry out their duties. The business might possess a Multiprotocol Label Switch (MPLS) connection to its headquarters network, yet it could lack sufficient bandwidth for all traffic or prefer that traffic intended for cloud-based applications/services does not pass through the enterprise HQ network. Similarly, workers might be telecommuting or situated remotely while utilizing company-owned or personal devices. In these situations, a company might want to allow access to certain resources while denying access or limiting actions regarding more sensitive resources. In this scenario, the Policy Enforcement (PE)/Policy Administrator (PA) is commonly offered as a cloud service (which typically ensures better availability and eliminates the need for remote employees to depend on enterprise infrastructure to reach cloud resources) with endpoint assets either equipped with an installed agent or accessing a resource portal. Hosting the PE/PA(s) on the enterprise local network might not be the most efficient since remote offices and staff need to route all traffic back to the enterprise network to access applications/services provided by cloud services.

Another scenario is a business that has on-site visitors or contracted service providers needing restricted access to company resources to perform their tasks. For instance, a company possesses its own internal applications/services, databases and resources. These involve services outsourced to providers who may sometimes be present on-site for maintenance. These guests and service providers will require network access to carry out their duties. A zero-trust organization could make this possible by granting these devices and any on-site service technician internet access while hiding enterprise assets. In this instance, the organization features a conference centre where guests engage with staff members. Once more, by utilizing a ZTA framework for SDPs, devices and users are distinguished, allowing them to access suitable enterprise resources. Campus visitors may use the internet but are unable to reach enterprise resources. They might not be able to identify enterprise services through network scans. In this scenario, the Policy Enforcement (PE) and Policy Administrator (PA) may be provided as a cloud service or on the local area network (LAN). The enterprise assets may possess an installed agent or obtain resources through a portal. The PA(s) guarantees that all non-enterprise assets (those without installed

agents or unable to connect to a portal) are blocked from accessing local resources but are allowed to access the internet.

Numerous businesses are a public service that may or may not require users to register. Such services might cater to the general public, a specific group of customers with a current business relationship, or a unique group of non-enterprise users like dependents of employees. In every instance, it is probable that the requested assets are not owned by the enterprise and the organization is limited in the internal cybersecurity policies it can implement. The organization is unable to tightly manage the condition of asset requests and public resources that are anonymous do not need credentials for access. Businesses can implement guidelines for registered public users, including customers and special users. If users must create or are given credentials, the organization may implement policies concerning password length, lifespan and may offer MFA as an option or necessity. Nonetheless, companies face restrictions in the policies they can apply for this category of user. Data regarding incoming requests can be valuable in assessing the status of public services and identifying potential attacks disguising themselves as genuine users. For instance, a portal for registered users is recognized to be accessed by signed-up customers utilizing a selection of popular web browsers. A rapid rise in access requests from unfamiliar browser types or recognized outdated versions might suggest an automated attack is occurring, prompting the enterprise to implement measures to restrict requests from these detected clients.

Zero-trust is a viable solution to the evolving security issues in software development (Rose et. al, 2020). By realizing the limitations of perimeter-based security models and embracing a Zero Trust mindset, organizations may foster a continuous security enhancement culture throughout the software development lifecycle. This proactive approach lowers the danger of both internal and external threats while blending in seamlessly with the collaborative and iterative nature of DevOps techniques. The study's background essentially emphasizes how important it is to include Zero Trust concepts into DevOps processes in order to strengthen CI/CD pipelines against new types of cyber threats. This establishes the framework for further investigation into how this integration might be accomplished in an efficient manner to improve the security posture and robustness of contemporary software delivery methods.

The traditional practice of treating security as a manual, post-development concern has proven to be a bottleneck for numerous companies striving to implement a Continuous Delivery (CD) pipeline (Conor Deegan, 2020). Currently, an imbalance in the research landscape regarding the integration of security into CI/CD exists. This project aims to address the methods for adding new zero trust techniques to the CI/CD pipeline, along with their advantages.

According to T.K. Gustavsson (2014), DevOps is a development methodology that aims to close the gap between development and operations stages. These DevOps tools are expected to fall into at least one of these categories, essential to critical aspects of the development and operation process (Pratibha Jha, 2018):

1. Programming — code development and evaluation, source code management tools, code integration;
2. Build — reliable connecting tools, create condition;
3. Examine — consistent evaluation tools that provide feedback on commercial risks;
4. Package — archives, application prior arrangement coordination;
5. Release — alteration of management, release approvals, release automation;
6. Design — framework configuration and management, Foundation as Code tools; and
7. Monitor — application operation tracking, end-user.

The quicker feedback loop is one advantage of DevOps. Software developers can produce more gratifying and even surprise products when they receive timely input from end users. This is accomplished through the use of a set of procedures and equipment referred to as continuous delivery and integration.

Researchers and experts in the field are observing a deficiency in the investigation and practical use of this methodology. This has to do with protecting the delivery pipeline and these quickly developed applications and micro-services. Numerous companies continue to employ conventional security procedures, with security teams operating independently of the DevOps team and conducting reviews. The boundaries between applications, infrastructures, and networks are changing, and security must adjust accordingly (Buyya et al., 2018).

Continuous deployment and continuous integration are two ways that DevOps promises to shorten the SDLC (Software Development Life Cycle) and streamline the design, development and implementation process. These benefits are detailed in the study of the preceding research report. The code produced is of higher quality when developers and operations collaborate and version releases occur far more frequently than a waterfall model. The software development and delivery process has undergone a paradigm shift with the introduction of Continuous Delivery. The set of techniques and tools that make this possible range widely in complexity and function within the process. The next section discusses the current state of security in CI/CD.

In "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation," book by Humble and Farley (2010) established the basis for comprehending CI/CD as essential elements of the DevOps pipeline. They emphasized the significance of automating the build, test and deployment procedures to attain quicker and more dependable software releases. Continuous Integration (CI) is a widely recognized practice in software development that encourages the regular merging of code updates into a common repository. Fowler and Foemmel (2006) characterized CI as a method that motivates developers to merge their work regularly, ensuring that each integration is validated by an automated build and testing procedure. The objective is to identify and resolve integration problems promptly, minimizing the likelihood of defects and enhancing the software's overall quality. Research conducted by Duvall, Matyas, and Glover (2007) in "Continuous Integration: Enhancing Software Quality and Minimizing Risk" further highlighted the importance of CI in ensuring a stable and healthy codebase.

Continuous Delivery (CD) builds upon the principles of CI by guaranteeing that the merged code is consistently in a deployable condition. Humble and Farley (2010) highlighted the significance of automating the deployment process to facilitate consistent and frequent releases to production. CD entails thorough automated testing and deployment automation, guaranteeing that code modifications can be deployed to production environments promptly and securely. In "Continuous Delivery: Overcoming Adoption Challenges," Chen (2015) identified major obstacles to CD adoption including organizational resistance and the intricacies of legacy systems, and offered strategies to tackle these issues.

Although the advantages of CI/CD pipelines are significant, the initial configuration can be intricate and require a lot of time (Chittala, 2024). Organizations need to allocate resources for appropriate tools, set up environments and create the essential infrastructure. This complexity can pose a major obstacle, particularly for smaller teams or organizations that have limited resources.

Implementing CI/CD practices frequently necessitates a change in culture among development teams. Engineers must adjust to updated workflows, tools and methods. The adaptation process can be difficult and might initially result in resistance or reduced productivity as team members familiarize themselves with new systems and practices.

2.3 CURRENT STATE OF SECURITY IN CI/CD

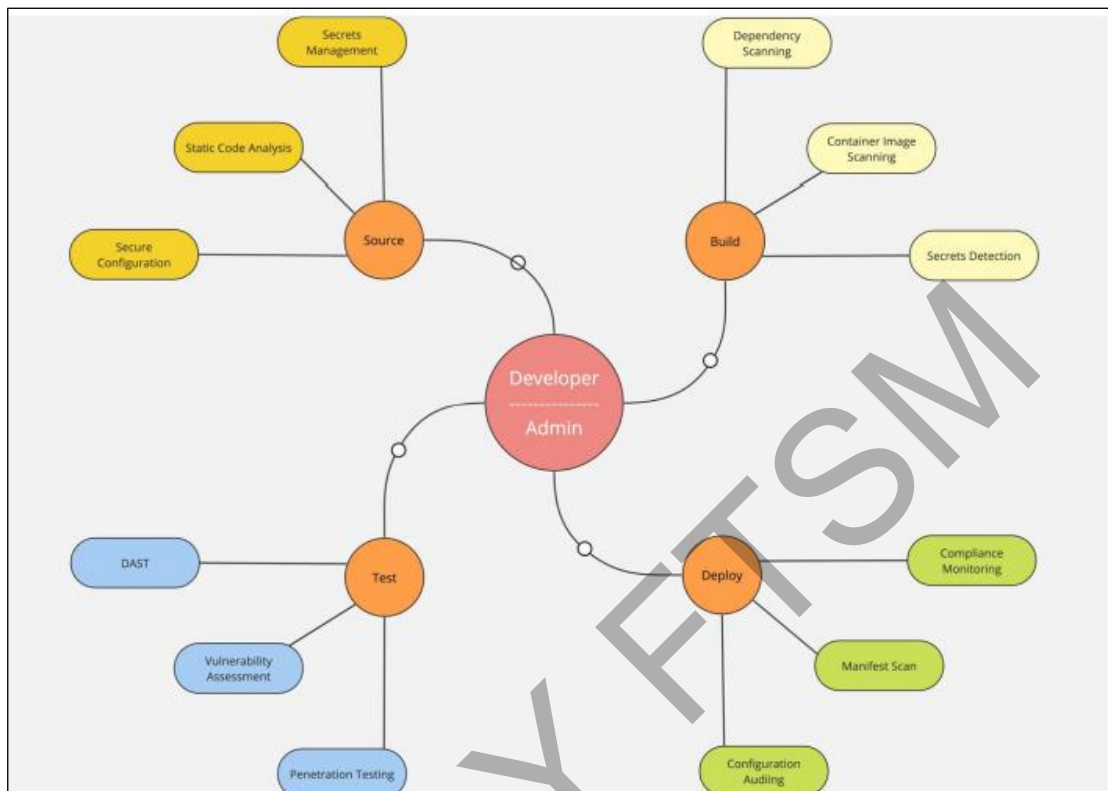


Figure 2. 2 Security Factors in CI/CD Pipelines

Source: Vighe, 2024

In the Source phase of the CI/CD pipeline, incorporating secrets management, static code analysis and secure configuration methodologies is crucial for enhancing security protocols. Secrets management guarantees that sensitive data, like API keys, passwords and cryptographic keys, is safely stored and retrieved within the source code repository. By effectively handling secrets, organizations reduce the risk of accidental exposure or unauthorized access to vital data throughout the development process. Furthermore, static code analysis tools are essential for examining the source code for security flaws, coding mistakes, and possible vulnerabilities. By examining code modifications as they are submitted to the version control system, these tools offer immediate feedback to developers, allowing them to quickly resolve security concerns before they spread across the codebase.

In the Build stage of the CI/CD pipeline, incorporating dependency scanning, container image scanning and secrets detection greatly improves security measures.

Dependency scanning tools carefully examine the application's dependencies, such as libraries and frameworks, to identify known vulnerabilities and licensing problems. By examining dependency manifests or package lock files throughout the build process, these tools offer vital information regarding security threats arising from outdated or vulnerable dependencies, allowing developers to quickly fix vulnerabilities prior to their inclusion in the application. Simultaneously, tools for scanning container images thoroughly assess these images for recognized vulnerabilities, misconfigurations and compliance breaches. With the rise of containerization, maintaining the integrity of container images is crucial for enhancing the security framework of the build process. By performing thorough scans during the construction phase, organizations proactively reduce security threats linked to the deployment of vulnerable or insecure container images. In addition, tools for detecting secrets are crucial for uncovering hardcoded secrets, including API keys and passwords, found in the application's code and configuration files. By actively identifying and removing hardcoded secrets throughout the build process, organizations strengthen their defences against possible security vulnerabilities and unauthorized entry to confidential data.

In the Test phase of the CI/CD pipeline, the integration of Dynamic Application Security Testing (DAST), vulnerability evaluation and penetration testing play an essential role in strengthening security protocols. DAST tools help assess the security of web applications by mimicking actual attack situations, enabling the discovery of vulnerabilities like injection issues, Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). Through dynamic evaluations of the application, organizations can uncover possible security weaknesses that might emerge exclusively during runtime, facilitating immediate remediation. Alongside DAST, vulnerability assessment tools thoroughly examine the application environment for recognized vulnerabilities, misconfigurations and inadequate security measures. By utilizing thorough scanning methods, these tools detect possible vulnerabilities in the application's infrastructure, libraries and configurations, enabling proactive mitigation strategies. Furthermore, penetration testing performed by ethical hackers entails mimicking cyberattacks to identify security vulnerabilities and leverage them to obtain unauthorized access. Through the simulation of actual attacks, penetration testing offers organizations important insights into their security status, enabling the detection of vulnerabilities that might escape automated scanning tools.

During the Deploy stage of the CI/CD pipeline, it is essential to incorporate compliance monitoring, manifest scanning and configuration auditing to enhance security measures. Compliance monitoring tools consistently evaluate the deployment environment to ensure conformity with regulatory requirements, industry standards and organizational policies. These tools offer immediate insight into compliance status and notify of any deviations or violations, allowing organizations to quickly rectify non-compliant configurations and maintain regulatory alignment. Simultaneously, manifest scanning tools carefully examine deployment manifests, package lock files or other specifications to detect vulnerabilities and security threats in the application's dependencies and external components. By examining dependency versions and established vulnerability databases, these tools provide information about possible security risks that may jeopardize the integrity of the deployment. Moreover, configuration auditing tools methodically assess the configuration settings and policies of the deployment environment to detect misconfigurations, inadequate security settings and departures from security best practices. Configuration auditing strengthens the deployment environment's resistance to potential threats by implementing secure configuration standards and identifying security vulnerabilities in the deployment infrastructure.

Based on study made by Chittala (2024), although CI/CD pipelines facilitate quicker deployments, there is a danger of emphasizing speed at the expense of quality. Organizations need to find a balance between quick implementation and comprehensive quality assurance. Achieving this balance necessitates thoughtful evaluation of testing methods, oversight practices and fallback protocols to guarantee that the heightened deployment pace does not jeopardize product stability or user satisfaction.

The development and security teams assume that these two teams work independently of one another and often do not coordinate when it comes to security. The security team finds potential security issues and, if necessary, returns the application to development, while the DevOps team delivers the produced application. Operations and development typically combine their efforts in a natural way. Conversely, it could appear that the development is the sole thing at odds with the procedures that guarantee an application's security. There may be a cultural reluctance to security in certain DevOps teams because of a concern about delaying the delivery

of applications. While management insists that there be no delays in operations, DevOps teams are simultaneously under intense pressure to release new versions of the application as frequently as feasible. It is a widely held belief that tackling security concerns during the initial phases of development would obstruct present and future endeavors. Additionally, some companies lack set duties regarding several aspects of security, primarily in the absence of a formal security staff. The likelihood of complex dangers materializing is increased, in addition to the potential that certain DevOps members may not have received the proper security training (Sheldon, 2019).

Businesses must protect customer information since they frequently manipulate it. If not, there might be a chance of hefty fines, allegations and most importantly a breach of such information that permanently harms a company's name. Regions and locations may have different laws and regulations, which makes data security compliance even more difficult. One such is the General Data Protection Regulation (GDPR), which was made mandatory in the European Union. Due to GDPR, there has been a significant increase in the number of reported data breaches. Organizations must notify authorities of such breaches three days after they are discovered.

Additionally, the meaning of protected data has been clarified, and the rule gives individuals more control over their data as well as additional limitations on how businesses can utilize it. Data protection during development and manipulation, including DevOps and related procedures, is becoming increasingly important to development businesses and their teams.

The agency verifies identity through either passwords or multi-factor authentication (MFA) with fixed access for entity identification. Agencies must guarantee and implement user and entity access to appropriate resources at the correct time for the suitable purpose while avoiding the provision of unnecessary access. Agencies ought to incorporate identity, credential and access management solutions throughout their organization whenever feasible to implement robust authentication, provide customized context-specific authorization and evaluate identity risk for agency users and entities. Agencies ought to combine their identity repositories and management frameworks when suitable, to improve understanding of organizational identities along with their related responsibilities and powers. Current solutions used is manually set up lifecycles (from creation to removal) and attribute assignments

(security and logging); rigid security policies and measures that tackle one aspect at a time with distinct reliance on outside systems; least privilege implemented solely during provisioning; isolated pillars of policy enforcement; manual reaction and mitigation deployment; and restricted correlation of dependencies, logs and telemetry.

2.3.1 Vulnerabilities

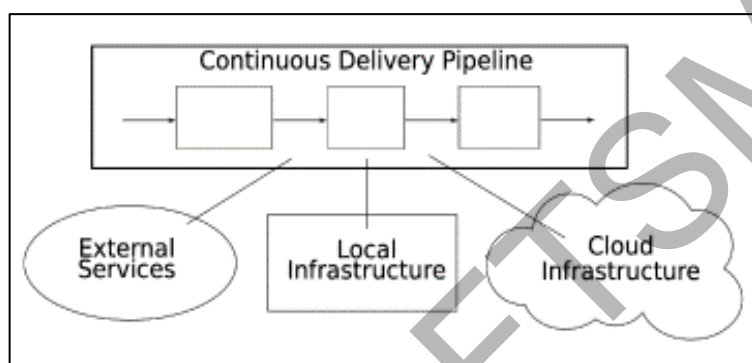


Figure 2. 3 Pipeline Dependencies

Source: Thomas et.al, 2018

The usual order of phases in the pipeline can differ from one activity to another. However, every pipeline consists of the typical phases - develop of the code, automated testing and ultimately the implementation. Different tools are developed for each stage and each consequently, businesses can utilize various options. A widely used open-source application (external services) for software development and testing is Jenkins (Mala, 2019). It might retrieve the source code from the repository (local infrastructure), execute automated tests, and generate and save a build image in a storage location or a storage place (cloud infrastructure). Amazon provides widely utilized storage solutions known as S3 (Simplified Storage Service) buckets (Atlassian, 2021). In the initial stage, automated tests need to be conducted. Tests typically fall into several primary categories - unit, integration, and comprehensive tests of various amounts. All of them are executed to guarantee the application offers the necessary functionalities with accurate results. Once all the tests are completed, the software is released into the environment. The final step may need manual authorization to avoid unwanted or erroneous deployments.

Multiple roles are portrayed inside the standard development procedure. Different access privileges to different areas of the pipeline must be granted to these jobs. For instance, until the changes have passed all automated tests and gone through the pipeline, the developer is unable to deploy to production directly (P.Rimba, 2015; Anastasov, 2022). It is possible and desirable for some jobs and releases to only be initiated by particular roles. Another illustration would be that the environments used for testing and production ought to be completely isolated from and independent of any necessary resources, such databases. A compromised or incorrectly designed pipeline may contain malicious or experimental code, which could lead to its collapse in production. The pipeline is frequently not built with these possibilities in mind.

The advantages that come with it may be compromised or maybe unavailable as a result of vulnerabilities in the pipeline. When that happens, it is critical to find, recognize, and address those. Closing security flaws improves pipeline dependability and stability, guaranteeing faster development, more frequent product increments, and shorter time to market (Christina Paule Thomas, 2019).

Another issue that could come up is when an employee undermines security by purposefully or inadvertently creating issues. This could be brought on, for instance, by ignorance of the subject matter being managed. Their awareness of security issues in that particular place might not be as high as it could be. Unencrypted connections between the pipeline's component parts may be utilized. For instance, a connection between a business server and an external provider that supplies the components or artifacts needed for the delivery pipeline, or a connection between plugins on an automation server like Jenkins.

The deployment environments for the pipeline's component parts may also be weak. Servers offering installed software may not be sufficiently secure due to unauthorized access or unreliable network connections. Alternatively, there can be the use of vulnerable versions of the components or external dependencies, which could result in the introduction of exploits or security flaws. The deployment environments for the pipeline's component parts may also be weak. Servers offering installed software

may not be sufficiently secure due to unauthorized access or unreliable network connections.

Alternatively, there may be the use of vulnerable versions of the components or external dependencies, which may result in the introduction of exploits or security vulnerabilities. Figure 2.1 illustrates how a pipeline typically depends on other services. When insecure tools are being used, security holes could be created by vulnerable pipeline setups that include commands for such tools.

Additionally, some team members might produce artifacts, pipeline scripts, or code changes that are vulnerable. Since the DevOps team is typically the only one who makes modifications to the pipeline, these changes frequently go unreviewed.

2.3.2 Legacy systems

The main obstacle preventing most firms from embracing DevOps is undoubtedly the move from old systems and the re-building of apps to use DevOps architecture or shifting them to cloud processing. Rebuilding the development team and altering internal procedures are also necessary to conform to the new paradigm. This include adding new team members, modifying team roles and implementing new tools (Aleksandrova, 2019). A significant obstacle to DevOps integration is security, which puts the business at risk of security problems. The primary cause is that the security team lacks DevOps integration and is unaware of the technologies and techniques being employed to enhance the development processes. In an effort to expedite the process and disregard essential scans and code checks, DevOps frequently prioritizes speed over security. This creates vulnerabilities and makes it difficult for the security team to monitor several security flaws.

Adopting DevOps can help accomplish rapid software deployment, which may be the first security challenge (Mohan, 2016). When companies begin to overwork this process and deploy too frequently, for example, security testing may be disregarded, which could lead to the introduction of new vulnerabilities. The selection of faulty or use case-specific automated tools can have unfavourable effects. For instance, an automated monitoring tool that is too complicated for a small software development

business could be expensive and difficult to set up correctly, potentially resulting in security flaws (Mohan, 2016).

The organization may be vulnerable to assaults from both inside and outside the corporation if access controls are used in an imprecise manner and are not audited or updated to reflect current circumstances (Mohan, 2016). Using containers as environments creates a whole new set of potential security flaws, including exposed networks, file systems and kernels. These cannot be protected by conventional methods, which shield entire systems rather than just certain applications. As a result, the containers might be exposed to side channels, misconfiguration, kernel exploits, attacks on shared host resources and other threats including data leakage (Mohan, 2016).

When human testing is preferred over an automated technique and the automated approach is not used frequently, a security issue may occur. According to Rafi (2020), manual approaches are more likely to result in errors and faults than correctly configured automated solutions. Despite its limitations, manual security testing is still a potential concern. It is not as efficient as automated testing. When determining potential DevOps problems, the absence of automated testing is a category in itself (Rafi, 2020). Another crucial factor influencing the overall security level is the cooperation between the security and DevOps teams.

There may be misconceptions and undervaluation of situation, specific critical security factors when this coordination is disorganized (Rafi, 2020). The absence of secure coding standards may be another element raising security vulnerabilities (Rafi, 2020). This could indicate that developers are not well-versed in those standards, which would explain why the code they write does not adhere to the essential security criteria. The disregard for change management and security during rapid changes in project development could be detrimental (Rafi, 2020). The security requirements must be updated to reflect the most recent changes in the circumstances. A security risk element for the project may be represented by trusted inputs of various kinds (Rafi, 2020).

Unauthorized access to the project's codebase or an external dependency could introduce a potential software vulnerability. When handling secure data, such as user accounts or database passwords, the volume of information may lead to security issues (Rafi, 2020).

When management does not prioritize security or when nobody is accountable for it, security may be in jeopardy (Tomas, 2019). There are several reasons to not want to prioritize security, one of which could be the claim that security adds no new functionality. The code in the production environment can be impacted by inadequate role separation, such as normative developers (Mohan, 2018), which is related to the previously described ambiguous access limits.

2.4 THREATS AND RISKS IN THE EXISTING CI/CD PROCESS

Threat modelling serves as the essential foundation for creating secure software. If one does not comprehend the threats they face in an organized manner, building a secure operational environment and software will not be achievable (Krishnan, 2007). It goes without saying that threats increase alongside the advancement of technology and delivery methods. A SANS survey (State of Application Security: Closing the Gap, 2015) shows that threat assessment (also known as threat modelling) ranks as the second most important application security practice (following penetration testing) for developing secure web applications. Therefore, threat modelling is a proactive security approach that organizations ought to implement.

The STRIDE method for threat modelling was created by Loren Kohnfelder and Praerit Garg in 1999 (Shostack, 2014). This method aids in counting threats according to characteristics of the attacks. Threats are listed by examining each attack characteristic and the related security aspect affected. Think about a Software as a Service (SaaS) application that handles payroll processing for different companies. The application sends and saves confidential information like employee salary details. The system additionally connects with its client's network for authentication and to gather Human Resource Management System (HRMS) data for processing payments. For every one of these attack characteristics, there exists a collection of security themes compromised using STRIDE, as shown in the Table 2.1:

Table 2. 1 Threat Enumeration using STRIDE

Attack Property	Security Theme
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information Disclosure	Confidentiality
Denial-of-Service	Availability
Elevation of Privilege	Authorization

Source: Sriram Krishnan, 2017

The STRIDE model, which encompasses Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), and Elevation of Privilege, serves as a comprehensive framework for identifying potential security threats within systems. Implementing targeted mitigation strategies can effectively address these threats. For instance, Spoofing threats, which involve impersonation of entities, can be countered by employing multi-factor authentication (MFA) and robust identity management systems, ensuring that only authenticated users gain access. Tampering, referring to unauthorized alterations of data or code, can be mitigated through the use of cryptographic integrity checks and secure code repositories, safeguarding the integrity of the system's components. To prevent Repudiation, where actions are denied by users, implementing comprehensive logging and audit trails provides verifiable records of all activities, ensuring accountability. Information Disclosure risks, which pertain to unauthorized access to sensitive data, can be minimized by enforcing strict access controls and data encryption, thereby protecting data confidentiality. Denial of Service (DoS) attacks, aiming to disrupt service availability, can be addressed by establishing redundant systems and rate-limiting mechanisms to maintain service continuity. Lastly, Elevation of Privilege threats, where users gain unauthorized higher-level access, can be mitigated by implementing the principle of least privilege and conducting regular access reviews, ensuring users have only the permissions necessary for their roles. Studies have demonstrated that applying such mitigation strategies effectively reduces the risks associated with each STRIDE category, thereby enhancing the overall security posture of systems (Sriram Krishnan, 2017).

In the context of software delivery processes, the confidentiality, integrity, and availability of these processes can be jeopardized by a variety of security threats within the continuous integration and delivery (CI/CD) pipelines. This section discusses the risks and possible security threats within the CI/CD pipeline from a comprehensive threat modeling review to comprehend possible weaknesses and the effects they may have on the pipeline. The following are some typical dangers to CI/CD pipeline security:

1. **Code Injection:** Via variety of techniques, such as inserting malicious scripts or instructions into source code repositories or build scripts, attackers may try to introduce malicious code into the CI/CD pipeline (Vighe, 2024). Remote code execution (RCE) and SQL injection are two examples of code injection vulnerabilities that can result in system compromise, data breaches or unauthorized access;
2. **Dependency Confusion:** This is the process by which harmful or unauthorized third-party dependencies are used by CI/CD pipelines as a result of attackers taking advantage of flaws in dependency management systems (Vighe, 2024). Attackers can introduce security flaws and jeopardize software development by changing package metadata or introducing malicious packages into repositories;
3. **Insecure Configuration:** Pipelines may be exposed to security concerns due to misconfigurations in CI/CD tools, platforms, or environments (Vighe, 2024). Unnecessary services that are enabled, default settings with lax security measures, and excessively lax access controls are examples of insecure configurations. These setup errors provide attackers the ability to escalate privileges, obtain unauthorized access, and run arbitrary code inside the pipeline;
4. **Sensitive Data Exposure:** Inadequate security measures or inappropriate handling of secrets can cause CI/CD pipelines to unintentionally reveal sensitive information, including credentials, API keys, or proprietary data (Vighe, 2024). Attackers could get or intercept private information that has been disclosed via communication channels, artifacts, or logs, which could result in data breaches or illegal access to resources;

5. Inadequate Access restrictions: Inadequate access restrictions in the CI/CD pipeline can give unauthorized users or bad actors access to private information, the ability to elevate privileges, and the ability to modify pipeline activities (Vighe, 2024). Unauthorized access can be made easier and insider threats are more likely to occur when there are insufficient authentication measures, inappropriate role-based access controls (RBAC) or no multi-factor authentication (MFA);
6. Insecure Integration Points: Attackers might compromise the pipeline by taking advantage of vulnerabilities in the integration points between CI/CD pipeline components, such as build servers, deployment tools, and source code repositories (Vighe, 2024). Attackers can carry out attacks, alter pipeline behaviour, or exfiltrate sensitive data by taking advantage of flaws in webhooks, data exchange protocols, or API endpoints;
7. Inadequate Security Testing: Applications may become exploitable if thorough security testing isn't done at every stage of the CI/CD pipeline (Vighe, 2024). Inadequate testing can result in security vulnerabilities going undiscovered and untreated. It can also involve poor code analysis, insufficient vulnerability scanning, or no penetration testing at all;
8. Denial of Service (DoS): Attackers may use denial-of-service attacks against pipeline infrastructure or components in an effort to interfere with the operation of the CI/CD pipeline (Paule et.al, 2018). Attackers can cause performance degradation, sabotage software delivery processes, or even stop pipeline services by flooding pipeline resources with unsolicited requests or malicious traffic;
9. Phishing and social engineering: Phishing and social engineering attacks directed at individuals working on CI/CD pipeline operations have the potential to jeopardize pipeline security (Paule et.al, 2018). Attackers may deceive users into divulging private information, including login passwords or access tokens, or into carrying out illicit acts within the pipeline by using phishing emails, pretexting, or impersonation tactics; and

10. Supply Chain Attacks: The security of CI/CD pipelines may be impacted by supply chain attacks that target software dependencies, tools, or infrastructure elements (Paule et.al, 2018). In order to add malicious code, backdoors, or exploitable vulnerabilities into software builds, attackers may breach trusted dependencies, alter software artifacts, or infiltrate supply chain networks, jeopardizing the integrity and security of the pipeline.

Organizations may improve the security posture and resilience of their continuous integration and delivery (CI/CD) pipelines by recognizing these security threats using the STRIDE model and putting in place the necessary security measures. This will reduce risks and protect software delivery processes from potential attacks.

2.5 ACCESS CONTROL SOLUTIONS

Security vulnerabilities major threats to the dependable operation of business processes and may have a detrimental impact on business value, including reputation or profits (Cavusoglu, 2004). As a result, companies are consistently allocating additional resources to safeguarding business resources (Ekelhart, 2009). The objective of a cybersecurity assessment is to evaluate the current state of cybersecurity for an evaluated asset to determine how well the asset meets particular security goals. It involves assessing assets in relation to their cybersecurity. criteria, considering the possible hazards, outcomes of dangers, and associated expenses. By implementing the protective measures within the system, the aim of the cybersecurity assessment is to assess the proper execution and functioning of controls, as well as their sufficiency and effectiveness in fulfilling security requirements specifications of the system (Chapple, 2018; Leszczyna, 2021). This section will discuss the existing access control solutions.

2.5.1 Defense in Depth (DiD)

According to Gajbhiye et.al (2024), Defense in Depth is an established security approach that safeguards data and systems through multiple layers. This method has been applied in cybersecurity to create multiple security layers to minimize the likelihood of a sole point of failure (NIST, 2022). Defense in Depth encompasses physical, network, application, and endpoint layers. security tiers, as per NIST (2022).

This method enhances resilience by providing multiple obstacles to violations and reducing the likelihood of an attack (Shinder, 2019). Defense in Depth demands accuracy synchronization of security protocols and technology, rendering it complex and demanding on resources.

Defense in Depth (DiD) comprises a collection of design principles that are broadly used in industry for managing risks (Squillante et.al, 2015). Principles of DiD influence the fundamental structure of the system being designed and incorporating them is not straightforward at a later stage in the design process without incurring significant redesign expenses. Consequently, it is essential to conduct an early evaluation of the overall security structure. The fundamental idea of DiD is that numerous systems are established in position to manage various possible scenarios that vary from standard functioning to the handling of incidents. These systems are organized in DiD tiers. The responsibilities at every DiD level are conducted by systems organized as autonomous and successive defense lines. If one defensive line breaks down, the next defense line aims to avoid or lessen the effects of the initial failure. Safety measures and crisis scenarios, facility design (protection divisions), access management, and crisis response associations.

According to Papakonstantinou et.al (2024), there are four concepts in DiD that are crucial to highlight in the context of this document involving redundancy, physical separation, functional independence, and variety as follows:

1. **Redundancy:** Denotes the presence of multiple methods for carrying out a necessary function (International Electrotechnical Commission, 2015). A frequent illustration is the "m out of n" format, in which a minimum of m out of the total n items needs to be operational to retain the capacity to carry out the necessary task;
2. **Physical Separation:** Pertains to the division of systems or elements from each other through the use of appropriate barriers, spacing or shape, or mixtures of them (International Atomic Energy Agency, 2007; Stuk, 2013). Functional isolation is described in (International Atomic Energy Agency, 2007) as “the

avoidance of effects arising from the functioning or malfunctioning of a single circuit or system on a different one;

3. **Functional Independence:** Refers to a state where occurs when the necessary tasks of a system are accomplished successfully. Functions do not rely on any actions, including breakdowns for standard functioning, of an alternative system, or based on any signals, information or data obtained from the alternative system (International Atomic Energy Agency, 2007). In this context, the aim of functional isolation is to minimize dependencies (e.g. data transfers, coordination problems, frequent cause failures) among functions rather than system parts. The methods can be entirely functional (without signals), software-oriented (protocols) or tangible (unidirectional connections) or lacking signal paths); and
4. **Variety:** Generally, diversity denotes the state of being made up of various components. The IAEA (International Atomic Energy Agency, 2007) defines diversity as “the existence of two or more overlapping systems or elements to execute a defined task, in which the various systems or elements possess distinct characteristics in order to lessen the likelihood of common cause failure, encompassing standard mode malfunction. Numerous forms of diversity exist regarding the operational principles, design techniques, organizations and technologies for implementation (Nuclear Regulatory Commission, 2010). The aim of diversity is to enhance system reliability.

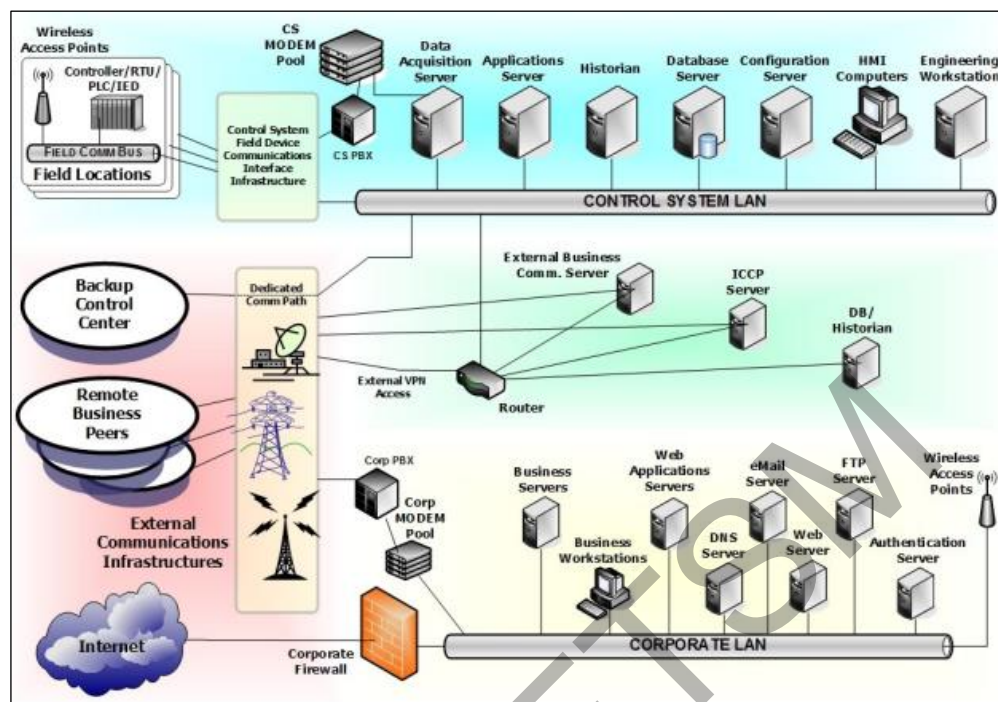


Figure 2. 4 Defense in Depth (DiD) Architecture

Source: Idaho National Laboratory, 2006

This architecture offered methods for data exchange, data collection, information transfer and various commercial activities. Nonetheless, the safety of any specific system was grounded in the reality that few, if any, comprehended the complex structure or the operational functions of the resources within the control system LAN. By linking these networks and integrating IT elements into the management domain of the system, security issues emerge as follows (Idaho National Laboratory, 2006):

1. Growing reliance on automated systems and control technologies;
2. Unsecure connection to outside networks;
3. Utilization of technologies that have identified vulnerabilities;
4. Lack of justification for cyber security in control system settings;
5. Security in control system technologies is often restricted, and if any security features are present, they are activated if the administrator knows about them;
6. Communication protocols in control systems lack security features; and
7. A significant quantity of open-source information exists about control, system setup and procedures.

The operational security of control systems has traditionally been characterized by the industry as the degree of the system's reliability to function. The complete separation from the outside and the organization's use of an untrusted network enabled it to lower the degree of communications. The security risks to operations were linked to physical entry to a facility or production area. Therefore, the majority of data exchanges within the information infrastructure necessitated restricted approvals or safety supervision. The acquisition took place in a secure setting where every communication was deemed reliable. If a command or instruction was transmitted through the network, it was expected to reach its destination and execute the permitted task, since only approved personnel had entry to the system.

2.5.2 MITRE Att&ck Framework

The scientific community has concentrated on creating models for patterns and methods of cybersecurity breach from documented occurrences in an effort to predict enemy conduct, strategic methods, and organized nefarious activities (Straub, 2020; Khan et.al, 2018). In this context, a widely recognized adversary model is the one suggested by MITRE ATT&CK, which addresses the who, how, and reasons behind cyberattacks a digital framework (Georgiadou, 2021). MITRE ATT&CK is an extensive body of knowledge foundation of opponent strategies and methods grounded in actual insights into threats related to cybersecurity. It has gained broad acceptance within the research community and industry and has been applied in various areas, including behavioral analysis development and adversary emulation (Georgiadou, 2021).

The MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) Framework offers a comprehensive and practical database of adversarial tactics, techniques, and methods (Georgiadou,2021). This method has been widely embraced by both suppliers and corporate clients in the sector. The MITRE ATT&CK framework serves as a detailed matrix of tactics and techniques utilized by threat hunters, red team members, and defenders to more effectively categorize attacks and evaluate organizational risks (Georgiadou,2021).

MITRE ATT&CK was launched in 2013 to record and classify tactics, techniques, and procedures (TTPs) used by adversaries after a compromise, specifically

targeting Microsoft Windows systems to enhance the identification of malicious activities (Strom,2018; Strom et.al, 2018). Throughout the years, ATT&CK has grown considerably, exploring various platforms and technologies, transforming into a repository of cyber adversary behavior and a classification system for adversarial actions throughout their lifecycle. It is currently utilized as both a playbook for emulating adversaries and as a way to identify analytic coverage and defense deficiencies within target networks (Strom, 2017). The ATT&CK behavioral model relies on several fundamental elements:

1. **Tactics:** Indicating the tactical goal of the opponent in executing an attack. It effectively tackles the "why" (Strom,2018; MITRE Corporation, 2016). Tactics act as contextual classifications for specific techniques and encompass common, advanced representations of actions taken by adversaries during an attack, including data exfiltration, privilege escalation, and evasion of defenses (Strom et.al, 2018);
2. **Techniques:** Outlining the methods through which opponents reach tactical objectives by executing an action. To put it differently, they explore the "how" and, in certain situations, the "what" an opponent achieves by taking an action (Al-Shaer et.al, 2020; Strom, 2018). There can be various methods or techniques to reach tactical goals, resulting in several techniques within each tactical category (Strom, 2018);
3. **Sub-techniques:** Illustrating more detailed methods through which adversaries reach tactical objectives at a more granular level than techniques (Strom, 2018; MITRE Corporation, 2016);
4. **Procedures:** Assigning the particular execution that the adversary employs for methods or sub-methods (MITRE Corporation, 2016). They are utilized to characterize real-world application of techniques or sub-techniques, demonstrating various extra behaviors in how they are executed (Al-Shaer et.al, 2020; Strom et.al, 2018); and
5. **Mitigations:** Outlining the measures that could deter adversaries from accomplishing their tactical goals through the application of particular methods. Mitigations focus on the "what actions to take" regarding the TTPs (Tactics, Techniques, and Procedures) issue (Caimi et.al, 2020).

2.5.3 Cloud Security Alliance (CSA)

The Cloud Security Alliance (CSA) outlines seven fundamental threat modeling processes in their Report on Cloud Threat Modeling (Tran et.al, 2023). This simulation will adhere to the mentioned references procedures to illustrate how the artifact can be efficiently utilized. Although various methods are available, it is crucial to recognize that the core nature of threat modeling the procedure remains unchanged. The following are the sequence and specifics might differ, but the fundamental principles and objectives stays unchanged:

1. Determine security goals for threat modeling;
2. Define the range of the evaluation;
3. Decomposition of system/application;
4. Recognize and evaluate the possible dangers;
5. Recognize flaws and deficiencies in the system and create elements;
6. Create and rank strategies and safeguards; and
7. Convey and generate a call to action.

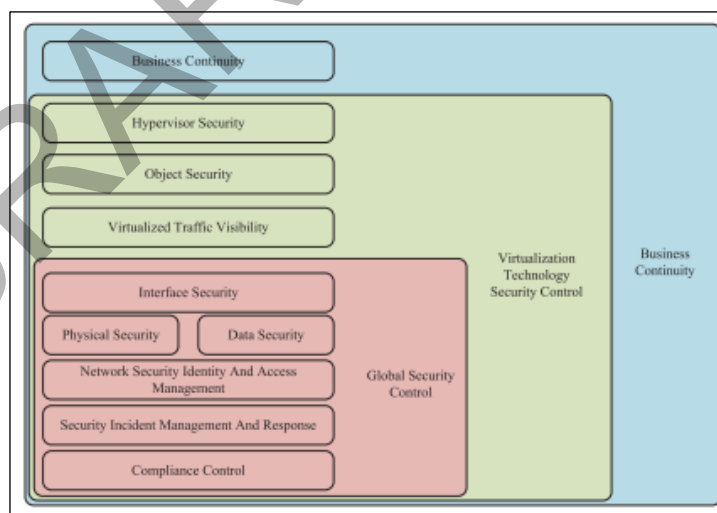


Figure 2. 5 CSA Cloud Platform Security Architecture

Source: Chen et.al, 2022

Possible security threats posed by the highly demanding capabilities of the cloud platform. The cloud platform enables intensive management of IT resources and

significantly enhances the effectiveness of resource deployment and utilization (Chen et.al, 2022). Nevertheless, the accumulation of resources will render the original singular system and application ineffective, leading to the emergence of numerous systems, various applications, and potentially the unavailability of the entire IT infrastructure, alongside other security issues, which are vital considerations in cloud platform design and development. In 2019, the Cloud Security Alliance (CSA) published the top ten threats analysis report for cloud computing which has received significant citation and acknowledgment within the industry. The CSA cloud security architecture model is illustrated examines the cloud platform's security architecture from three perspectives: global security management, virtualization technology security measures, and business continuity.

Security threats associated with the dynamic resource allocation capability of the cloud platform. The development of the cloud platform has transformed the conventional IT landscape because of the stringent reliability demands of dual-machine backup, virtual machines frequently experience dynamic migration, leading to fluctuations in the protection boundary (Chen et.al, 2022). Conventional security devices are unable to implement flexible security protection policies during virtual machine migration. The virtual machine cannot be secured post-migration, posing significant risks to the underlying business.

Based on the study made by Chen et.al (2022), risks associated with network security in cloud platform virtualization. The operational mode of virtual machines differs from hardware servers. Data traffic traveling east-west between business virtual machines is only observable within the cloud platform and communicates through virtual switching networks. Conventional security protection tools like firewalls, intrusion prevention systems, anti-virus gateways and auditing tools cannot detect the traffic between business virtual machines, leading to their inability to offer effective protection within the cloud environment. Next, Chen et.al (2022) states that risks to data security on cloud platforms. The cloud platform consistently houses business data in cloud computing servers and the aggregation of cloud computing resources increases the concentration of security risks.

2.5.4 Zero Trust Concept

A security strategy called Zero Trust Architecture (ZTA) was created to address cyberthreats and the possibility of security breaches meanwhile the Zero Trust (ZT) paradigm postulates the absence of a trusted perimeter. Users and devices are granted the minimum level of access privileges (Rose et. Al, 2020). In a Zero Trust environment, users can never access company resources without first being vetted and granted permission. ZTA can consequently help to improve visibility and analytics across the network of the firm. According to a survey by M. Bowen (2022), 83% of 1,300 security and risk specialists who answered a survey by Ericom Software believed that zero trust is a successful security strategy. Jabbari et al. (2016) define DevOps as a development process that prioritizes quality assurance, delivery with automated deployment utilizing a set of development methods, and teamwork and communication. Its objective is to close the gap that exists between operations and development. Compared to traditional security testing, which typically occurs after development and impedes the continuous delivery and integration of new features and applications, this iteration of security testing is different. The goal of this initiative is to encourage the software delivery cycle to include the security testing procedure. An automated method to security testing is necessary for an automated pipeline for software delivery and integration.

As the name implies, ZTA abandons the perimeter strategy and places all of its confidence in the network. Assuming that the network is infiltrated, security can adopt a more sophisticated strategy by limiting access to network resources and establishing strict authentication and authorization guidelines to grant particular access based on characteristics unique to individual users and devices. According to the "least-privilege access" paradigm, ZTA only gives people and devices access to the programs, services, and information that are strictly required for them to perform their jobs inside an organization. An organization can distribute its resources and data more precisely and swiftly grant or deny access to a user when they take on changing responsibilities by basing access determination on "role." Role-Based Access Control (RBAC) streamlines user permission management by organizing them into roles, whereas Zero Trust guarantees that access is continually validated and restricted to what is essential (MicroAge, 2024). The first line of defence will still be perimeter security, but relying

too much on it may raise the possibility of data interception and network breaches. By identifying network intruders at the next level of access and comparing the identities of users and devices to the corresponding authorizations, ZTA may close security gaps around the perimeter by ensuring that access is authorized properly and securely.

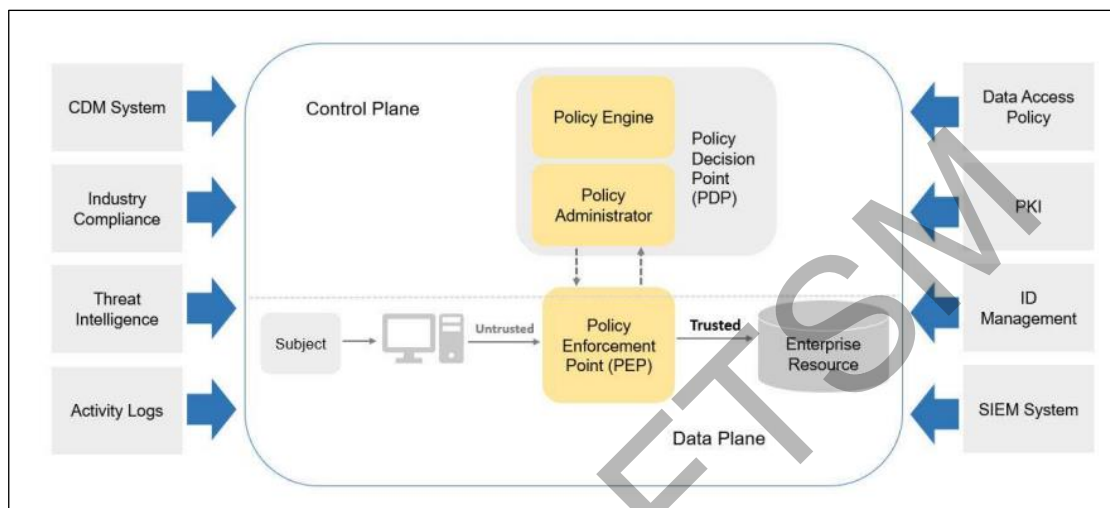


Figure 2. 6 Zero Trust Framework

Source: Rose Scott, 2022

ZTA creates more granular access controls to those particular resources by refocusing attention from a network's perimeter to its individual apps and services. "Micro segmentation" is the term for this approach to encircling apps and services with security, which enables more specialized access control and security than is possible with typical perimeter security. The security of data packets entering and leaving a ZTA system becomes even more crucial when the dependence on network perimeter security decreases. Therefore, one of the most essential conditions of any ZTA is the encryption of those data packets, which ought to be a basic feature of any decent security architecture and data transfer (including protocols like SSL and TLS). Because it provides explicit verification of users and devices for each application and service as opposed to implicit verification, which presumes safety and authorizes access once a user or device enters the network, the authentication and authorization process is also essential to ZTA's success.

Based on study made by Rose Scott (2022), the elements illustrated in Figure 2.6 by logical functions, therefore it is feasible that several elements can perform one logical function in a distributed way, or one solution can accomplish multiple logical tasks. The roles are outlined in NIST SP 800-207 (2020), but they are briefly summarized below:

1. Policy Engine (PE): The central component of a ZTA implementation that assesses and processes resource access requests. The PE depends on data from multiple sources (access logs, threat intelligence, endpoint health and network ID authentication);
2. Policy Administrator (PA): The implementation role of the PE. The PA's function is to create, sustain and end sessions in the data plane between the subject and the resource. The PA, PE, and PEP interact over a distinct set of channels, either logically or physically, referred to as the control plane. The control plane is utilized to set up and configure the data plane channels that transmit application traffic;
3. Policy Enforcement Point (PEP): The element that applications and endpoints will connect with to obtain access authorization to a resource. The PEP is tasked with collecting data for the PE and adhering to the guidelines provided by the PA to initiate and conclude communication sessions. All communications on the data plane between enterprise resources should be overseen by a PEP; and
4. Data feeds (left and right): Often referred to as policy information points (PIPs). These do not constitute the "core" functional ZTA components but serve to assist the PE. These feeds encompass the collection of formalized policies, identity and endpoint characteristics, environmental elements and historical information utilized by the PE to make resource access choices.

According to Kurt Delbene (2019), three basic steps must be followed for ZTA to be implemented at the network's application and service level which consist of:

Authentication:

1. Verify the user;
2. Verify the device; and

Access Privileges:

3. Check authorizations.

Based on each layer's unique characteristics, a set of compliance tests are used to complete these three layers of verification. These compliance checks can cover anything from user behaviour patterns to device encryption, and they can grow in scope as additional data about users and devices is gathered. The user or device will be classified as "non-compliant" and access will be denied if any of these checks are not completed. Even while it is best to use updated software systems to maintain a uniform set of compliance checks throughout a network and organization, networks that are distributed and rely on outdated hardware and software, like the Department of Defence, can nonetheless apply this crucial component of zero trust.

It is possible to design systems with built-in compliance checks that force users and devices interacting with them to follow predetermined rules. Users and devices beyond the purview of those services may be impacted by this in turn, and they must meet those standards in order to be granted access. In this sense, a ZTA can function as a virus by compelling people or devices belonging to third parties to comply with certain requirements in order to be granted access (Kurt Delbene, 2019). This is because they will already have some norms of compliance to base their development on, those third parties will find it easier to eventually adopt zero trust into their own systems and services. Iterative compliance measurements are both possible and desirable. Depending on the sensitivity of the applications and services in question and the intended access limitations, ZTA can start with a basic set of identity and device health checks and gradually build a more sophisticated set.

According to Shyo Prakash et al. (2024), there are four basic components of zero trust:

1. Identity and Access Management (IAM): Zero Trust makes use of IAM as the foundation for allowing those with legitimate and authentic requests entry into restricted regions;

2. Device security: Strict examination for compliance with security policy enforcement and fundamental endpoint protection mechanisms is necessary for any device attempting to access system-wide or organizational resources;
3. Micro-segmentation: Networks are divided into smaller segments to limit access to sensitive data and lateral migration of vulnerabilities; and
4. Constant monitoring involves keeping an eye on network traffic, device health, and user activity in order to spot irregularities and possible security breaches.

2.5.5 Discussion

The ideas of Defense in Depth (DiD), Cloud Security Alliance (CSA), MITRE ATT&CK Framework and Zero Trust Framework pertain to cybersecurity however, each has distinct objectives and concentrates on various elements of security. An overview of their distinctions as follows:

Table 2. 2 Summary Frameworks

	Defense in Depth (DiD)	Cloud Security Alliance (CSA)	MITRE ATT&CK Framework	Zero Trust Framework
Concept	Utilizes various layers of protective mechanisms to safeguard data and systems.	Security measures to protect cloud environments and cloud-based services.	Adversary tactics, techniques, and procedures (TTPs) that respond to cyberattacks by providing a comprehensive matrix of attack vectors and techniques.	Assumes that no one, whether inside or outside the network, should be trusted by default
Focus Area	<ul style="list-style-type: none"> i. Multiple Layers of Protection; and ii. Redundancy and Resilience 	<ul style="list-style-type: none"> i. Cloud-Specific Risks; and ii. Integration with Existing Security Policies. 	<ul style="list-style-type: none"> i. Adversary Behavior Mapping; and ii. Threat Detection and Response 	<ul style="list-style-type: none"> i. Continuous Authentication and Authorization; and ii. Granular Access Control
Components	<ul style="list-style-type: none"> i. Network security (firewalls, intrusion detection systems); ii. Identity and access management (IAM), multi-factor authentication (MFA); iii. Data encryption, monitoring, and real-time threat detection; and iv. Physical security controls 	<ul style="list-style-type: none"> i. Data protection and encryption; ii. Identity and access management (IAM) for cloud environments; iii. Cloud workload protection and security posture management; and iv. Secure software development lifecycle (SDLC) for cloud-based applications 	<ul style="list-style-type: none"> i. Tactics: Objectives of an attack (initial access, execution, persistence); ii. Techniques: Specific methods attackers use (phishing, lateral movement, privilege escalation); iii. Procedures: Steps or tools used to execute techniques; iv. Multi-Factor Authentication (MFA) & Identity Verification; and v. Secrets management tools and secure API authentication. 	<ul style="list-style-type: none"> i. Identity and access management (IAM); ii. Multi-factor authentication (MFA); iii. Micro-segmentation and network segmentation; and iv. Continuous monitoring and logging

To summarize, the chosen access control solution is Zero Trust as it is protecting every access point and resource throughout the infrastructure, whether on-premises or in the cloud. Zero Trust is the most compatible to integrate in DevOps, which typically includes regular modifications to infrastructure, system, and applications and aligns effectively with DevOps CI/CD pipelines, enabling automatic enforcement of access policies based on identity and context. Micro-segmentation in Zero Trust can assist DevOps by ensuring that if a single system is breached, the impact is limited, stopping lateral movement across the network. Since DevOps entails regular changes, automation, and various access points (from developers to automated CI/CD tools), Zero Trust guarantees that only authenticated users and devices can reach essential resources. This integrates effortlessly with ongoing identity verification and minimal access rights, rendering it very appropriate for DevOps.

For an IT company that is implementing DevOps, the ideal method would involve a framework and security protocols that emphasize continuous integration/continuous deployment (CI/CD), automated security measures, and scalable defense. Therefore, the Zero Trust Framework has been chosen for this project. The next section will discuss the possible technology solutions.

2.6 POSSIBLE TECHNOLOGY SOLUTIONS AND BEST PRACTICES TO SECURE DEVOPS

This section will discuss the possible technology and best practices since the solutions are decided in the previous section. To guarantee greater security, automation should generally be implemented in as many processes as possible (Mohan, 2016). These procedures could include code reviews, continuous deployment pipelines, monitoring, and code scanning, for instance. When it comes to automation, one potential strategy to combat security threats is to automate tests to find potential non-compliances in crucial areas, including the code base and its external dependencies (Mohan, 2016). Another crucial element is to outline the security policies and follow these established guidelines (Mohan, 2016). To be effective, the security configuration of the employed tools, roles, groups, firewalls, and other elements must be monitored and kept up to date (Mohan, 2016). Security can be increased by utilizing the dynamic security scanning system which consists of four tiers to it (Mohan, 2016):

1. Pre-authentication: Public attack surface scanning;
2. Post-authentication: Auto re-login detection, user role management, logout, and session maintenance;
3. Backend: A separate examination of the different application layers; and
4. Scan: Workflows to the target application.

A method put out by (Larrucea, 2019) is based on the NIST cybersecurity framework and suggests the actions that must be done in order to secure DevOps. The following are the steps:

1. Determine the security assessment, consider, and add it. This activity involves risk assessment;
2. Protect: Think through and carry out particular measures (during the integration phase);
3. Detect: Identifying any risks during the integration stage;
4. React by acting against each possible threat; and
5. Recover: If a security vulnerability arises, consider earlier versions of the source code.

It is crucial to forbid any unauthorized access while maintaining a record of all requests for sensitive data in order to address the issue with access limits. This is how some irregularities can be traced back, and potential weak points can be examined and addressed (Mohan, 2016). Securing Linux container environments must go hand in hand with their use. This could entail utilizing the protection features built into Docker itself, such as the container management daemon (Mohan, 2016), or it could entail utilizing a LiCShield Framework, which safeguards hosts by limiting access to containers and container management daemons so they can only carry out tasks approved for the testing environment.

Consider using threat modeling to identify organization and project specific threats, assess their seriousness, and assign suitable countermeasures when security is a crucial component (Rafi, 2020). In order to prevent team members from having a limited understanding of secure coding standards, it is crucial to ensure their

compliance. One such solution may be to offer a course that thoroughly explains these requirements (Rafi, 2020). Adoption of some broad security rules is related to coding standards (Mohan, 2018). It is imperative that team members are aware of these rules and implement them in their work.

In order to employ the isolated software running environments with the role-specific privileges and to achieve greater role privilege separation, the environments must be separated (Mohan, 2018). As stated in security challenges, it is advised to use a minimum of three environments: one each for development, testing, and production.

2.6.1 Existing Frameworks

In order to minimize any conflicts between the development and security team members later on, it is ideal to automate the development lifecycle from the start. In this manner, all possible issues are resolved swiftly and affordably (Subramonian, 2019). To secure the DevOps environment, a variety of commercial or open-source frameworks such as Open Policy Agent, Trivy and HashiCorp Vault are available for free.

2.6.2 WhiteSource

Vulnerabilities in open source code can be fixed with this program. It notifies consumers of potential dangers via the pipeline. This framework is compatible with more than 200 programming languages (Subramonian, 2019). Transitive dependencies are among the open source components in an application that Whitesource can identify as insecure. Additionally, it ranks the effectiveness of the most vulnerable functionality. According to a journal (Open Source Security), this results in a 70% decrease in vulnerability warnings, which aids the development team in more effectively prioritizing issues that need to be fixed. Subsequently, it offers functionalities like vulnerability remediation, complete trace analysis, identifying the code's susceptible functionality, and delineating the application's use of the vulnerability. Additionally, it might automatically open issue tickets for recently installed or found vulnerable components.

2.6.3 Zed Attack Proxy

The framework Zed Attack Proxy (ZAP) is open-source and free. The Open Web Application Security Project (OWASP) is the organization that created and maintains this penetration testing tool. This broad and adaptable framework's primary function is to test web applications by employing the "man-in-the-middle" proxy approach. Various roles, including developers, testers, and security specialists, can use it.

2.6.4 AquaSecurity

This technology integrates completely automated security testing and policy-driven controls early on, enabling enterprises to automate safe application development and deployment in DevOps pipelines (Automate DevSecOps, 2020). The Aqua framework's ability to thoroughly analyse serverless functions and container images for known vulnerabilities, malware, embedded secrets, OSS (open-source software) license problems, and configuration concerns is one of its primary features. Based on customized policies, it might stop unauthorized images and functionalities from being distributed in the environment, preventing operational mistakes, image sprawl, or rogue deployments.

2.6.5 |Code|AI

In order to prevent assaults, the CodeAI framework fixes security flaws in source code by utilizing AI and machine learning to identify potential problems in new code (Sen, 2021). According to an online source ([CODE]AI: Smart Defensive Coding Application For Devops), deep learning is used to create code that is educated on actual faults and remedies. A deep learning code model is trained with the patterns stored as feature vectors, and the model learns to predict errors in fresh code. Following bug detection, the framework can additionally use program transformation schemas that are obtained from open-source software bug-fixing commits to address issues.

2.6.6 Prisma Cloud

This solution makes it easy to integrate security controls into CI/CD pipelines and automate security throughout the whole application lifetime, protecting certain DevOps

operations in the process. It is an answer for cloud security management. permits a unified set of security features for the apps, data, network, compute, storage, users, and Platform as a Service components of the cloud-native technology stack. The most comprehensive cloud native platform, it offers the capacity to keep constant security and compliance management, recognizes and stops threats, and stops suspicious activity. It can make use of automated risk prioritization and continual vulnerability intelligence throughout the development process and cloud stack.

2.6.7 SaltStack

For security operations teams, Saltstack is an intelligent automation and collaboration tool. It allows for the provisioning, setting up, and management of cloud or on-premise infrastructures at a large scale, hence enabling infrastructure automation (Manage and secure your digital infrastructure). Security automation, which may enforce continuous compliance and patch vulnerabilities, is also a component of this program. Custom security policies can be created, defined, and automatically applied to all projects.

It can then do ongoing searches to find the aforementioned vulnerabilities wherever in the system. Finally, it can employ autonomous policy enforcement to initiate a remediation workflow or automatically repair infractions so that the team can flag and prioritize resolving the issue.

2.6.8 IriusRisk

It is a threat modelling tool that could help developers understand the application's planned features, technical architecture, and security environment. It teaches users about the technological architecture, planned features, and security context of the application (IriusRisk Threat Modeling Platform) via an architectural diagram and adaptive questionnaires powered by a system.

2.6.9 Discussion

The ideas of WhiteSource, Zed Attack Proxy, AquaSecurity, |CODE|AI, Prisma Cloud, SaltStack and IriusRisk adds great value to secure DevOps processes however, each has distinct objectives and concentrates on various elements of security. An overview of their distinctions and how they enhance one another as follows:

Table 2. 3 Summary Technologies Solutions

Tool	Concept	Focus Area	Zero Trust Relevance
WhiteSource	Securing third-party dependencies.	Dependency Security	Codebase integrity and dependency trust
Zed Attack Proxy (ZAP)	Web application and API security.	Runtime application security	Detecting runtime vulnerabilities
AquaSecurity	Securing containerized environments.	Container and Kubernetes security	Micro-segmentation for containers
CODE AI	Static code analysis	Code	AI
Prisma Cloud	End-to-end security in cloud environments.	Cloud-native security	Holistic multi-cloud security
SaltStack	Automating secure infrastructure setups	Infrastructure automation	Consistent infrastructure security
IriusRisk	Secure-by-design development processes.	Threat modelling	Proactive risk mitigation

Prisma Cloud is a perfect option for IT firms adopting Zero Trust strategies within their DevOps processes. It smoothly integrates with CI/CD pipelines, enabling security assessments to be incorporated throughout the software development process. Prisma Cloud guarantees that Zero Trust policies are implemented early in the development cycle and stopping misconfigurations before deployment. Moreover, Prisma Cloud offers real-time threat identification via machine learning, guaranteeing that vulnerabilities and threats are recognized and addressed. Prisma Cloud is built for cloud-native architectures and secures various environments, constantly tracks API activity and implements detailed access controls, guaranteeing that only validated and permitted requests are handled. Its scalability, adaptability, and integrated compliance capabilities position it as an excellent option for organizations aiming to safeguard their DevOps pipelines while upholding Zero Trust.

2.7 INTEGRATION STRATEGIES TO IMPLEMENT ZERO TRUST IN DEVOPS WORKFLOW

2.7.1 Integrating Security Into CI/CD Pipelines

In a thorough study of the effects of continuous security on a DevOps SDLC, Kumar et al. (2020) offer a methodology for this adoption of security. The research covering the ideas, resources, and techniques for putting such an approach into practice in a real-world business setting is cited in the paper. The identification of the business goals that drive the adoption of CI/CD and the key metrics that will be used to assess the effectiveness of a successful implementation of CI/CD and continuous security make this paper important to the study being done for this project. A key commercial goal for businesses using the CI/CD software development methodology is identified as velocity to develop information security frameworks and propose integration of zero trust strategies within current CI/CD process. According to Kumar (2020), there are several important measures that can be utilized to gauge how well CI/CD principles are being applied.

Among these are the following: mean time to market, average deployment time, post-approval mean time taken to deploy a release in production, frequency of deployment, and number of deployments in production in a particular period. If the velocity of any CI/CD installation was positive for these critical criteria, it might be considered successful. This study will add distinctive value to the field of research by attempting to quantify the effectiveness of CI/CD with continuous security using the key metrics found in the study, even though it is influenced by the work of Kumar and others in the area of modelling a pathway to adoption to continuous security.

This will provide an even greater platform for future studies to learn even more about the performance implications of security in the context of continuous integration and delivery (CI/CD).

The development and deployment processes used by many organizations are undergoing a paradigm shift. The DevOps methodology has greatly accelerated the time to market for both new applications and feature additions to already-existing ones. But

as Mohan et al. (2018) highlight in their research, there is still a lot of scepticism about the advantages of combining DevOps and security. The report on IBM's journey to combining their security team and procedures with their DevOps methodology for development and deployment makes up the paper. This project will benefit from a real-world example or case study to better understand the ramifications of including security procedures into the DevOps development pipeline. The information gleaned from this report will facilitate a project that is aware of the typical issues and roadblocks associated with this procedure.

Developers and security experts are two teams typically kept apart in a traditional software development environment and collaborate as part of the process of integrating security into the DevOps development process. The goal of this study is to ascertain whether integrating security procedures into the development process through automation can help close the gap between these two teams.

One of the main research directions of this project is to identify the procedures, tools, and ideas that would enable a successful integration of security into a DevOps application/services delivery paradigm. Papers like Kumar et al. (2020) will be helpful in identifying these aspects in this way. This study examines the various challenges associated with implementing a "continuous security" approach into the application delivery process by dissecting each component of a continuous integration/continuous delivery (CI/CD) process, starting with its concepts, enablers, essential components, and workflow.

2.8 SUMMARY

To summarize, the connection between CI/CD and Zero Trust is a crucial domain of interest for both practitioners and researchers to ensure that DevOps practices are effective. This chapter acts as a roadmap for examining the current knowledge and establishes the guidelines for the following chapters which will examine in greater depth the possible enhancements in DevOps practices following the strengthening of Zero Trust within the approach.

CHAPTER III

METHODOLOGY

3.1 INTRODUCTION

This section outlines the approach utilized to carry out this study. The methodology involves collecting information and data to draw conclusions. At the conclusion of this chapter, a complete explanation of the research methods used is provided to obtain valid results. The components such as research design, data collection techniques, the formulation of survey questions and data analysis techniques are examined in detail.

3.2 RESEARCH DESIGN

This research utilized a mixed-methods approach that integrated both qualitative and quantitative techniques to achieve a more thorough outcome efficiently. For qualitative approaches, the analysis of literature reviews pertinent to various fields of study, such as the present software development methodology, the current security status in CI/CD, threats and vulnerabilities in the existing CI/CD workflow, access control measures, technologies and best practices, as well as integration strategies for implementing zero trust within DevOps practices. For quantitative approaches, the interviews conducted with various literature reviews to gather insights on the significance of security and protective strategies in carrying out online investigations. The interview sessions also serve as an appropriate platform to determine the needs for personnel and infrastructure to implement the recognized measures.

The interviews for the quantitative approach will be designed to offer a comprehensive understanding of how Zero Trust is incorporated into DevOps practices. The individuals participating in this study are chosen from industry experts, DevOps

professionals and security analysts. This survey questionnaire will examine in depth the present condition of CICD pipelines and the typical risks. By integrating information from the literature review and adding insights from domain experts, this study developed a comprehensive improved workflow aimed at more secure DevOps practices. Following the developed work process, a preparatory checklist for DevOps review was methodically organized.

3.3 DATA COLLECTION METHODS

To create an improved workflow for a more secure DevOps analysis, data is gathered through qualitative and quantitative techniques and examined to achieve the necessary results that will be required to attain the goals. In this study, two methods for data collection will be employed, specifically literature review and expert interviews.

3.3.1 Literature Review

A literature review is essential in research to understand the existing scenario and pinpoint gaps and challenges in the topic. Various types and methodologies of literature reviews can be performed to acquire current knowledge in a field and aid researchers in addressing gaps in understanding on that subject. Snyder (2019) carried out a comparative analysis of the three methods for literature review: systematic, semi-systematic and integrative review. Summary of the comparison of these methods is presented in Table 3.1.

Table 3. 1 Approaches to Literature Reviews

Review Type	Purpose	Strategy	Features
Systematic Review	Synthesis and Comparison of Evidence	Systematic	Quantitative, detailed for particular field, guides practice.
Semi-Systematic Review	Summary of the research field and monitor developments within the field.	Systematic/Non-Systematic	Quantitative or qualitative, pinpoints themes or research voids, constructs a theoretical framework or presents a background of the discipline.
Integrative Review	Literature analysis to create viewpoints or concepts.	Non-Systematic	Qualitative, integrates concepts from various disciplines, emphasizes developing new frameworks or theories through the critique of existing ideas.

Source: Snyder, 2019

A semi-systematic review is the selected method of literature review for this study. This kind of review can typically be utilized to pinpoint themes, theoretical viewpoints, and other qualitative data regarding a specific topic. The Semi-systematic review is equally significant for creating a theoretical framework and a research strategy for a field (Snyder 2019). Zunder (2021) employed a semi-systematic review in his work titled “A semi-systematic literature review, identifying research opportunities for more sustainable, receiver-led inbound urban logistics flows to large higher education institutions”. He characterized the semi-systematic review as a literature review that employs a systematic method for surveying and selecting literature, combined with a narrative approach that permits “plurality of knowing” based on readers' interpretations. This enabled the literature review to be clear regarding literature searching while also possessing the possibility of being extensive. The literature review performed involves three (3) key steps, which are:

1. Outlining Literature Review;
2. Conducting Literature Review; and
3. Reporting Literature Review.

a. Outlining Literature Review

The primary aim of this research is to tackle the security issue linked to the present DevOps practices in the private sector and suggest a new workflow for performing a more secure DevOps practice. Therefore, these four (4) electronic databases were chosen to gather information on multiple subject areas. pertaining to the goals, alongside Google Scholar for finding grey literature.

1. ScienceDirect (<https://www.sciencedirect.com/>);
2. Semantic Scholar (<https://www.semanticscholar.org/>);
3. ResearchGate (<https://www.researchgate.net/>); and
4. IEEE Xplore (<https://ieeexplore.ieee.org/Xplore/home.jsp>).

This literature review was carried out by establishing the criteria for including and excluding the literature under analysis. The publication was limited to the year 2010 and beyond to pinpoint the most recent literature. To create a thorough literature review, the kinds of publications examined encompass review articles and research papers. Any publications, guidelines, and expert perspectives deemed pertinent to reader assessment are likewise examined. The reference lists of each selected article were examined manually to look for additional potentially eligible publications. Irrelevant publications were removed during the first evaluation of the titles and abstracts. Following the preliminary screening, the complete texts of potentially suitable publications underwent additional review to assess their eligibility. Table 3.2 provides a summary of the different criterion types along with their descriptions.