National College of Ireland

Securing the Speed: Balancing Security and Deployment Velocity in DevOps

MSc Research Project
In Cloud Computing

Ravina Mestry
Student ID: x22177264

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …….Ravina Mestry……

Student ID: ……x22177264……

Programme: …MSc in Cloud Computing…… **Year:** ……2023…….

Module: ……Research Project in Cloud Computing……..……

Supervisor: ……Vikas Sahni……..……

Submission Due Date: ……14/08/2023……..……

Project Title: Securing the Speed: Balancing Security and Deployment Velocity in DevOps

Word Count: ……6944…… **Page Count:** …20……..……..

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

Signature: ……Ravina Mestry……

Date: ……10/08/2023……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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

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

Securing the Speed: Balancing Security and Deployment Velocity in DevOps

Ravina Mestry
Student ID: x22177264

Abstract

The DevOps approach to software development emphasizes Continuous Integration and Continuous Delivery (CI-CD) to achieve faster release cycles and improved quality. However, security is often neglected in the pursuit of deployment velocity, leaving software applications vulnerable to cyberattacks. To address this challenge, organizations often incorporate multiple security tools into their DevOps pipelines. However, this raises the question of how many security tools should be used and what trade-offs exist between security and deployment velocity.

The research methodology employs an empirical approach, combining both quantitative and qualitative analyses. A sample CI-CD pipeline is constructed, integrating key security tools such as SonarQube, Super-Linter, OWASP (Open Web Application Security Project) ZAP, and Snyk, along with AWS EC2, GitHub Actions, Terraform, and Docker for scalable infrastructure. The evaluation of the research is centred on key performance metrics and indicators, including Mean Time to Build (MTTB), Mean Time to Change (MTTC), and vulnerabilities identified to measure the impact of security measures. Results demonstrate a slight increase in MTTB and MTTC, attributed to security checks. The number of vulnerabilities identified highlights the effectiveness of integrated security tools in enhancing application security. The findings empower organizations to deliver secure and high-quality software products, strengthening their competitive edge in the rapidly evolving software industry.

Keywords— DevOps, CI-CD pipeline, security, velocity

1 Introduction

Organizations are adopting DevOps because it enables faster and more frequent software deployments (Lwakatare, et al., 2019, p. 214). DevOps aims to increase the pace and automation of the software process. It utilizes automation to work with the development team, operations teams, and for continuous delivery. Faster development and delivery are accompanied with the crucial aspect of securing the CI-CD process. Continuous security integrates security on several aspects of CI-CD pipeline (Vehent, 2018, p. 8). This research discusses the importance of automating security to incorporate security effortlessly into CI-CD pipelines. Businesses that use a CI-CD strategy for application delivery identify velocity as a business goal (Deegan, 2020). This research aims to investigate the relationship between the number of security tools used in the DevOps pipeline and the security and efficiency of the CI-CD process. While using numerous tools can increase security, it can also complicate things and delay deployment. This research also discusses the trade-offs between enhancing security and maintaining a fast-paced CI-CD pipeline.

Layers of application security are used in continuous security as part of the CI-CD pipeline to find potential threats and vulnerabilities. Static code analysis, security testing,

container security, cloud and network security are some of the levels of security in the CI-CD pipeline. Security tools are progressively incorporated into the DevOps pipeline to assure the security and dependability of software applications. However, the effectiveness and speed of the pipeline could be impacted by the incorporation of various security solutions into the CI-CD process (Dyess, 2020). This paper examines different CI-CD pipeline stages and the security tools that go along with them at various integrated security stages. This research paper a comprehensive study that aims to explore the delicate balance between security and deployment velocity in DevOps. The paper delves into the integration of security practices at various stages of the CI-CD pipeline, analysing the impact on deployment speed and the effectiveness of security measures.

This research finds the trade-offs between safeguarding the efficiency of the CI-CD process and the security of applications by implementing the CI-CD pipeline with and without integrated security tools. The research methodology for this study adopts a data-driven and empirical approach. The research method encompasses both quantitative and qualitative analyses to provide a holistic understanding of the balance between security and efficiency. The research implementation phase focuses on the practical integration of security measures into the CI-CD pipeline. The evaluation is conducted applying the metrics studied in literature review to assess the efficiency and speed of the development process of the CI-CD pipeline. Moreover, experiments are done to determine how different numbers of tools affect the pipeline and related metrics to achieve balance.

This research analyses the effects of integrating multiple security tools in the DevOps pipeline and the potential impact on the speed of the pipeline as well as the efficiency of the CI-CD process. The paper emphasizes the significance of integrating security early in the development process, automating security testing, and fostering collaboration between development and security teams. This research also provides recommendations for optimizing the balance within the integration of security tools and velocity in the CI-CD pipeline. By identifying the optimal number of security tools and their appropriate placement within the CI-CD pipeline, organizations can effectively secure their software applications without sacrificing deployment speed. Ultimately, this research contributes to the development of a more secure and efficient DevOps process.

2 Related Work

DevOps is a software development methodology that combines operations (Ops) and software development (Dev) (Ahmed & C. Francis, 2019; Krief, 2019) to improve the effectiveness and efficiency of software development and deployment. Security is a major concern in DevOps due to the complexity and frequency of software releases (Rajapakse, et al., 2021, p. 2). The articles in this literature review were examined in relation to DevOps security challenges, security tools at various stages, and metrics to assess the DevOps pipeline.

2.1 Security and its challenges in DevOps

With a focus on quicker and more frequent releases, DevOps is becoming more and more well-known. However, this focus on speed can sometimes come at the expense of security

(Desai & T N, 2021, p. 1). Another difficulty that arises with speedy deliveries is how to get the security team to complete their tasks promptly and effectively (Leppanen et al., 2022, p. 8). DevOps emphasizes collaboration among the various (Rahman & Williams, 2016a) development team, operations team, and security team of an organization (Sánchez-Gordón & Colomo-Palacios, 2020, p. 2). Even though DevOps advocates for improved collaboration, faster delivery, and the elimination of defects, security is frequently overlooked. DevOpsSec is a tool that combines security with DevOps (Desai & N, 2021, p. 2-3; Larrucea, et al., 2019, p. 453-454) is essential because it upholds the fundamental principle of keeping security a priority and integrating security practices and measures into the DevOps cycle (Rajapakse, et al., 2022, p. 2; Sánchez-Gordón & Colomo-Palacios, 2020). The Security team is dedicated to making sure that the delivered software is thoroughly examined for vulnerabilities and threats while the DevOps team strives to move as quickly as possible (Dyess 2020, p. 16), by implementing the DevOpsSec solution, both departments productivity and outcomes are improved (Ahmed & C. Francis, 2019).

Moving security to the left (Bird, 2016; Mansfield-Devine, 2018, p. 16) earlier into the design, coding, and automated test cycles helps with the pace of the delivery rather than waiting until the system is developed, built, and then attempting to fit some security checks shortly before release. DevOpsSec emphasizes adding security to every step of the business process of DevOps, including requirement gathering, development, testing, deployment, monitoring, and maintenance, to achieve the desired results (Desai & N, 2021, p. 6). DevOpsSec incorporates security into all phases of the process by upholding compliance, identifying any weaknesses, and putting controls and monitoring in place (Bird, 2016), making security measures repeatable, dependable, and integrated throughout the entire development process (Desai & N, 2021).

Many businesses continue to believe that placing too much emphasis on security will slow down operations and harm their capacity to maintain a rapid pace (Dyess, 2020) (Ahmed & C. Francis, 2019, p. 179). Security procedures must be as quick as development procedures to maintain agility because speed is what distinguishes DevOps and DevOpsSec (Rajapakse et al., 2022, p. 7), as the primary goal of the DevOps methodology is to facilitate quick deliveries (Vehent, 2018). However, combining too many security measures in today's fast-paced environment can impede the CI-CD process (Dyess, 2020, p. 14), making it challenging to meet delivery deadlines for software (Rao, 2021).

This literature review shows the significance of integrating security in the DevOps pipeline at various stages and emphasizes the challenges of integrating security in the DevOps pipeline. One of the most appealing aspects of DevOps is its emphasis on speed (Mansfield-Devine, 2018, p. 16). The gap in this research lies in the lack of comprehensive studies that specifically address the challenge of balancing security and deployment speed in DevOps. While there are existing research papers that discuss security practices and challenges in DevOps and the importance of continuous security monitoring, there is a need for empirical studies that explore the impact of integrating security measures on deployment speed. As velocity is the demanding factor in DevOpsSec (Leppanen, Honkaranta & Costin, 2022, p. 8), the research question aims on exploring the impact of

velocity on multiple security tools integrated in the CI-CD pipeline and analyse trade-offs in improving speed and efficiency of the CI-CD pipeline.

2.2 Security tools in DevOps

The DevOpsSec paradigm strongly supports and depends heavily on the use of tools (Ahmed & C. Francis, 2019, p. 179). Selecting the appropriate tools can be challenging (Rajapakse et al., 2022, p. 7). As a result, DevOpsSec has access to a wide range of tools at every stage. The security is integrated into CI-CD pipeline through plan, code commit, build, test and operation phases through Static application security testing (SAST), Container Security, Infrastructure as Code (IaC), API security, Dynamic application security testing (DAST), Incident response management and monitoring (Mao, et al., 2020; Kumar & Goyal, 2020).

One of the prevalent software practices in the DevOps culture is the automation of all tasks related to software development, including automating security on CI-CD pipelines (Rahman & Williams, 2016b, p. 73; Kumar & Goyal, 2020, p. 10; Larrucea et al. 2019, p. 454). Automation makes it easier to integrate continuous security practices and principles into the CI-CD pipeline (Jammeh, 2020). Web Application Security Testing (WAST) uses its user interface to automate web security test attacking (Rangnau, et al., 2020, p. 146) while static analysis tools are used to conduct code reviews and give feedback to the appropriate software developers as part of automating code reviews.

SAST tools enhance code security by automatically identifying vulnerabilities in the code (Bird, 2016; Jammeh, 2020) and automation of security testing executes testing tasks such as test case management, test monitoring, and control (Rahman & Williams, 2016, p. 73), and automates security attacks test (Bird, 2016). DAST, which includes penetration testing and exploits testing, examines an application's behaviour as it runs in test and pre-production environments (Kumar & Goyal, 2020, p. 13). Automation in monitoring uses automated tools to collect, report, and store system-related data for later analysis (Sojan, et al., 2021; Rahman & Williams, 2016b, p. 73) and also security overview, threat intelligence (Hsu, 2018), and incident management.

No.	Security at various stages	Security Tools
1	Code Commit	SAST Tools: SonarQube, Checkmarx, Veracode
2	Build	OWASP Dependency-Check, Snyk
3	Test	DAST Tools: OWASP ZAP or Burp Suite, IBM AppScan
4	Artifact Generation	Clair, Anchore, Semgrep, CodeQL, Twistlock
5	Deployment	Terraform Sentinel, AWS Config, Azure Security Center
6	Verification and Approval	Security Assertion Markup Language, OAuth tools
7	Monitoring and Observability	PagerDuty, VictorOps, Logstash, Datadog

Table 1: Security Tools for various stages of CI-CD Pipeline

Table 1 shows some examples of security tools for the various stages of the CI-CD pipeline (Rangnau, et al., 2020; Deegan, 2020; Kumar & Goyal, 2021; Jammeh, 2020). SAST tools secure coding (Yarlagadda, 2020, p. 3) without launching the program, examine the code and look for vulnerabilities with early detection of potential flaws, vulnerabilities, and code smells is made possible (Rajapakse, et al., 2022, p. 15-16). Later in the CI-CD cycle (Rajapakse, et al., 2021), DAST tools are used to secure and scan applications in real-time (Kumar & Goyal, 2020, p. 13). Utilizing IaC tools, businesses can deploy and manage infrastructure automatically (Kumar & Goyal, 2020, p. 12). API security tools are WAST aids in securing and monitoring API endpoints while CI-CD tools automate the build, test, and deployment of code (Rangnau, et al., 2020, p. 146) while tools for API security check authentication, validation, and error handling.

Lastly, due to the dynamic nature of the infrastructure, maintaining such infrastructures necessitates continuous monitoring in addition to other factors like security and infrastructure administration (Sojan, et al., 2021) and can manage security incidents and respond to security breaches with the aid of incident response and management tools (Yarlagadda, 2020, p. 3). Overall, the articles on security tools that have been reviewed emphasize the value of automation and integration of multiple security tools in adopting continuous security at various stages of the CI-CD process. Nonetheless, the studies focused on the technical aspects of security tools and practices without fully exploring the trade-offs and best practices to achieve an optimal balance between security and efficiency. The gap in the research is the absence of a systematic investigation into the practical implementation of security tools in the CI/CD pipeline and their effect on deployment velocity. By providing practical insights and recommendations, this research seeks to address the gap in existing research and contribute valuable knowledge to the field of DevOps security. Furthermore, this research aims to bridge this gap by evaluating their impact on deployment speed and effectiveness for which study on metrics is conducted in next subsection.

2.3 Metrics for Evaluation in DevOps

Measuring the security of software is a difficult task. The DevOps paradigm makes it even more difficult to measure security because of the frequent and continuous software releases. The lack of appropriate security metrics was one of the most frequently cited causes of these issues (Rajapakse, et al., 2022, p. 10). The new process is challenging to be evaluated using the conventional metrics currently in use to measure the productivity and effectiveness of the software development and delivery process (Deegan, 2020, p. 4). Data from the CI-CD pipeline that has been integrated with several security tools must be gathered and analysed, using a variety of metrics for security and deployment velocity efficiency. Thus, reviewing previous work on metrics is crucial for the research question.

Software metrics aim to identify and measure the key factors affecting software development (Lehtonen, et al., 2015, p. 18). Measuring pipeline performance is a crucial component of CI-CD pipeline for evaluation for which the identification of key metrics for assessing the effectiveness of the integrated security CI-CD. For assessing the effectiveness and quality of an automated delivery flow, numerous studies have put forth various

metrics. The paper by (Lehtonen, et al., 2015) explores different metrics that can be used to evaluate pipeline performance, such as development time, and deployment time while (Deegan, 2020, p. 12) highlights metrics for the velocity of the CI-CD pipeline. While (Kumar & Goyal, 2020) mentions metrics such as vulnerabilities identified and scan pass rate for measuring security as well metrics such as MTTC and change lead time to evaluate the agility aspect of the CI-CD pipeline. Identifying areas for improvement in the CI-CD pipeline and implementing changes that increase efficiency and reduce in errors is achievable by measuring with these metrics.

After reviewing, numerous studies have proposed various metrics for evaluating the performance and quality of an automated delivery flow Table 2 lists the metrics and their description (Kumar & Goyal, 2020; Deegan, 2020; Lehtonen, et al., 2015) which are important for evaluating the research question covering aspects of agility, velocity, security, and quality of the CI-CD pipeline. Finding a balance between security and deployment that maximizes security while minimizing any negative impacts on deployment velocity is thus necessary to ensure the efficient and rapid delivery of software. The metrics that are used to assess CI-CD pipelines are examined in this literature review to determine the efficiency and speed.

Table 2: Metrics for evaluation of CI-CD Pipeline

No.	Metrics	Description
1	Deployment Mean Time	Time taken for deploying release in production
2	Deployment Frequency	Number of deployments in a certain period
3	Mean Time to Change	Time is taken to build, test and deploy the code change
4	Mean time to detect	Time is taken to detect the vulnerability
5	Mean time to Market	The average time is taken to initiate and release the feature in production
6	Vulnerabilities Identified	Number of vulnerabilities detected
7	Mean Time to Build	Average Time taken to build the code
8	Scan Pass Rate	Number of scans performed in a certain period

The most efficient way to collect and analyse important data using the metrics in Table 2 is to compare the CI-CD pipeline with multiple security tools integrated and examine the effects this has on the pipeline's velocity. The research questions below plan to explore the numerous tools and techniques available for implementing security in the DevOps pipeline at various stages, evaluating the CI-CD pipeline for multiple security tools using metrics researched and identifying best practices for achieving this balance.

RQ1: How does the integration of multiple security tools in the DevOps pipeline impact the security and efficiency of the CI-CD process?
RQ2: What are the trade-offs and best practices for achieving a balance between security and deployment velocity?

3 Research Methodology

In this research, the CI part is facilitated by a combination of powerful tools that streamline the development workflow and enhance the overall software quality as well as security by integrating security. The selected tools discussed in section 4 streamlines the development process, automate security testing, and ensure code quality throughout the development lifecycle. The CI-CD pipeline starts with AWS EC2, a scalable and reliable cloud infrastructure service provided by AWS EC2, the compute capacity to host the CI-CD environment and support the execution of build and test processes. The heart of the CI process is driven by GitHub Actions, a powerful automation platform integrated with GitHub repositories. GitHub Actions enabled to define workflows with push or pull requests. As a trigger event occurs, GitHub Actions automatically kicks off the defined CI workflows discussed in Section 5. EC2 instances are provisioned using Terraform, an IaC tool ensuring the consistent setup of the CI-CD environment, eliminating manual errors, and reducing deployment time provisioning AWS resources, including services required for the CI-CD environment.

Next, the CI-CD process performs code analysis using SonarQube, an advanced code quality and security analysis tool. SonarQube examines the codebase, identifies bugs, code smells, and security vulnerabilities, providing valuable insights to improve code quality and detailed reports to help address issues early in the development process. Super-Linter is then utilized to enforce consistent code styles and formatting guidelines across multiple programming languages. This ensures that the code adheres to the team's coding standards and maintains uniformity across the codebase. To monitor the CI-CD pipeline's performance, Datadog is integrated for real-time monitoring and observability. Datadog provides detailed insights into resource utilization, build times, and other key metrics to monitor the CI process's efficiency and performance.

In terms of security, Snyk is used for dependency scanning. Snyk identifies vulnerabilities in third-party libraries and dependencies, ensuring that the codebase remains secure from known security issues. Additionally, OWASP ZAP is employed for DAST performing automated security scans on the web application to identify potential vulnerabilities in real-time. To achieve consistency and portability, Docker is utilized for containerization. Docker allows for packaging the application and its dependencies into containers, ensuring that the same environment is used across different stages of the CI pipeline.

In terms of evaluation, analysing important metrics, such as MTTB and MTTC in order to determine the effects of incorporating security measures into the CI-CD pipeline. The objective was to locate any major temporal changes brought on by the employment of security technologies and procedures. To assess the effect of integrated security technologies on the application's overall security posture, the number of vulnerabilities discovered during security testing was also the goal. This goal was to demonstrate how well the integrated security technologies have supported the application's security.

In conclusion, the CI-CD part of the DevOps pipeline is a well-orchestrated integration of various tools, enabling automated build, code quality analysis, security testing, and

continuous monitoring. The seamless integration of these tools in designed pipeline enhances the software development process's efficiency, ensures code quality, and strengthens the security posture of the application and aims to finds the impact on the security and efficiency of the pipeline.

4 Design Specification

The resources that were used in the implementation of the research question are discussed in subsections below and details are provided in configuration manual.

4.1 Amazon Web Service (AWS)

AWS Elastic Compute Cloud (EC2) was utilized to host the CI-CD environment, providing scalable compute capacity for seamless integration. Terraform was employed to automate the provisioning, modifying, and destroying of AWS EC2 instances and set up a consistent and reproducible environment.

4.2 Docker

Docker was utilized for containerization, ensuring consistency across different environments, and simplifying the deployment process. Docker containers packaged the application and its dependencies, providing a consistent runtime environment throughout the pipeline.

4.3 GitHub and GitHub Actions

GitHub was used for collaboration and version control. GitHub Actions is a CI-CD platform that automates, modifies, and carries out the CI-CD workflows directly in the repository. GitHub Actions was used for automating, building, testing, and deploying the CI-CD pipeline. Docker delivered the software in packaged containers. GitHub Actions build and push Docker images which runs the application in the container.

4.4 Super-Linter

Super-Linter analyses the code in the repository against various linters for different programming languages, checking for code style issues and coding standards violations. With Super-Linter integrated into your GitHub Actions workflow, the consistent code styles were enforced which improve code quality across the repository, making it easier for to maintain a high standard of code throughout the software development process.

4.5 Security Tools

Below are the security testing tools that were used in the CI-CD pipeline at various stages.

4.5.1 SAST

SonarQube automated code review and consistently produced clean code. To assist in performing ongoing code checks of the project, it integrates into the current workflow and finds problems in the code. SonarQube was useful in locating and describing the problem

and explained why the code puts security at risk. It also offered advice on how to resolve the problem.

4.5.2 Container Security Scanning

Snyk was integrated for dependency scanning to identify security vulnerabilities in third-party libraries. This proactive approach helped address potential security risks before they became critical.

4.5.3 DAST

OWASP ZAP was used for DAST. It automatically scanned the web application for security vulnerabilities during runtime, providing real-time feedback to improve application security.

4.5.4 Network Security

AWS VPC (Virtual Private Cloud) protects web applications from common web exploits and provides a private network within the AWS cloud infrastructure where resources were launched securely. Security Group acts as a virtual firewall for AWS resources and control inbound and outbound traffic to instances.

4.5.5 Monitoring tools

Datadog was employed as a monitoring and observability tool to gain real-time insights into the CI-CD pipeline's performance. The tool provided metrics and dashboards to monitor resource utilization, build times, and application performance, enabling Cloud Security posture management.

5 Implementation

The research implementation includes two main workflows were designed and implemented: the Infrastructure Deployment Workflow and the Application Deployment Workflow. These custom workflows were defined to trigger events based on code changes, integrated with various tools to ensure security measures were met without compromising deployment velocity.

5.1 Infrastructure Deployment Workflow

The infrastructure deployment workflow was initiated using AWS EC2 as the cloud infrastructure service. Terraform, an IaC tool, was used to automate the provisioning of AWS EC2 instances and development, staging, and production environment resources. This ensured consistent and reproducible environment setups across all stages of the pipeline. For version control and continuous integration, GitHub Actions served as the central automation tool. As the infrastructure changes were pushed to the repository, GitHub Actions automatically initiated the workflow, streamlining the development cycle.

The Infrastructure Deployment Workflow used GitHub Actions to automatically deploy and configure the AWS EC2 instances and other infrastructure components to

development, staging, and production environments shown in Figure 1. The Infrastructure Deployment Workflow also included security measures. Snyk was used for dependency scanning to identify vulnerabilities in third-party libraries. It ensured that any potential security risks related to dependencies were addressed during the infrastructure setup. SonarQube early feedback helped to improve code quality and address potential security risks. Docker and Datadog agent are installed after the infrastructure is provisioned. Docker was utilized containerization, by application deployment workflow for packaging the application and its dependencies into containers.

Datadog was integrated into the Infrastructure Deployment Workflow for real-time monitoring and observability of the infrastructure. It provided metrics and dashboards to monitor resource utilization, ensuring the pipeline's performance and stability. Datadog also provided Cloud Security Posture Management showing the misconfigured resources, posture score and top issues to investigate according to the severity.

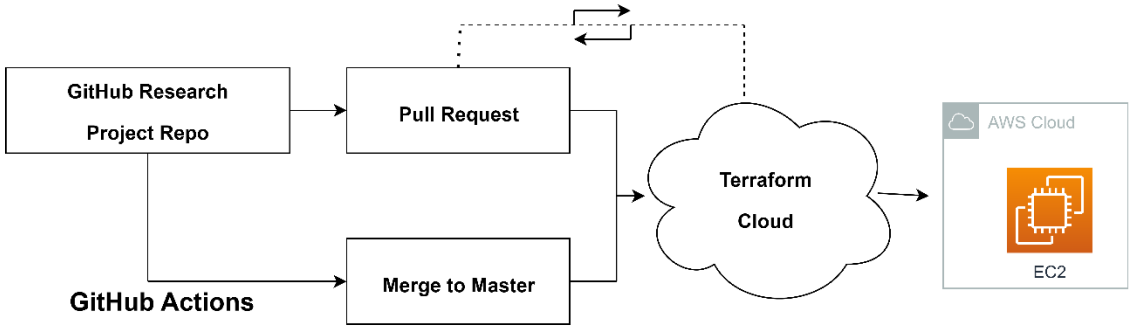


Figure 1: Infrastructure Deployment Workflow

5.2 The application deployment workflow

The application deployment workflow illustrated in Figure 2 was initiated after AWS EC2 instances were provisioned using Terraform and development, staging, and production environments were ready for application deployment. GitHub Actions served as the CI-CD automation tool, allowing for the definition of custom workflows triggered by code changes.

The Super-Linter tool was employed to enforce consistent code styles and coding standards across different programming languages. This not only improved code readability but also ensured that code adhered to the team's coding conventions. In terms of security, SonarQube and Super-Linter performed code analysis, providing early feedback to the developers. The SonarQube tool provided detailed reports and metrics to identify and address potential problems early in the development process. In terms of security, Snyk performed dependency scanning to identify vulnerabilities in third-party libraries, while OWASP ZAP conducted dynamic application security testing during runtime, providing real-time feedback on security issues. The continuous scanning feature of Snyk facilitated to stay updated on potential security risks. Docker containers packaged the application and its dependencies, ensuring a consistent runtime environment and simplifying the deployment process. The workflow shown in Figure 2 was structured to build and push the

image to the Docker Hub upon pull request on develop branch and deploy the Docker image to deployment environment upon merging with the develop branch. Similarly, when the code was merged to the master branch. Once the code was merged to master branch the Docker image was deployed to the staging environment, ensuring smooth deployment and security testing. The manual trigger was put in place to deploy the Docker image to production environment as shown in Figure 2.

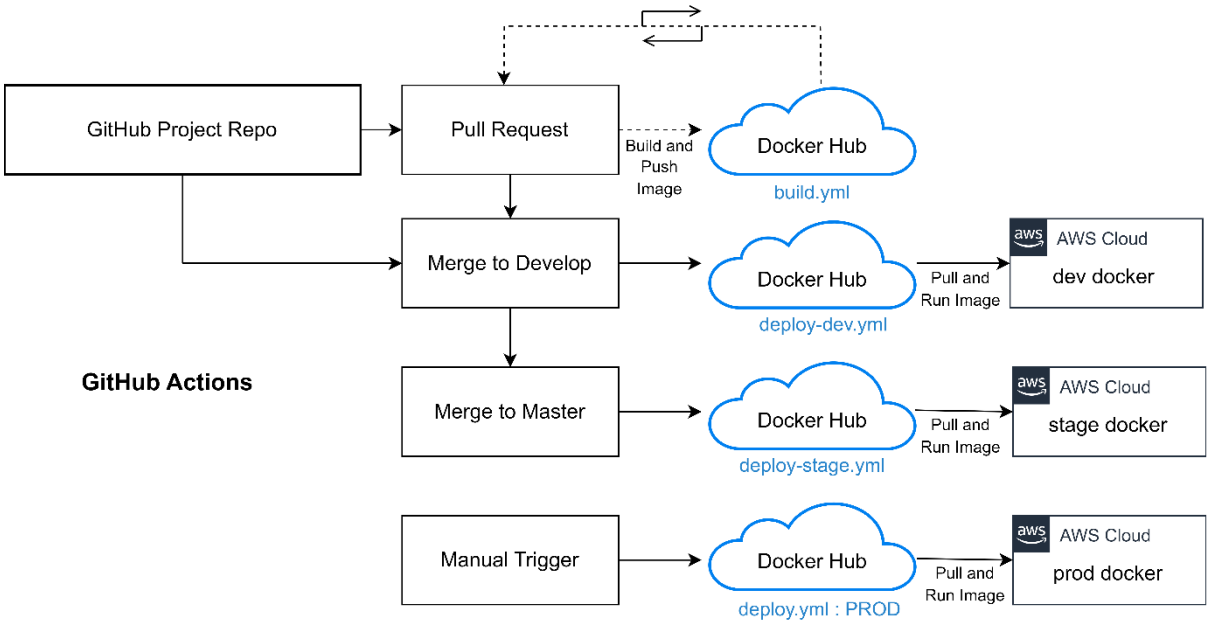
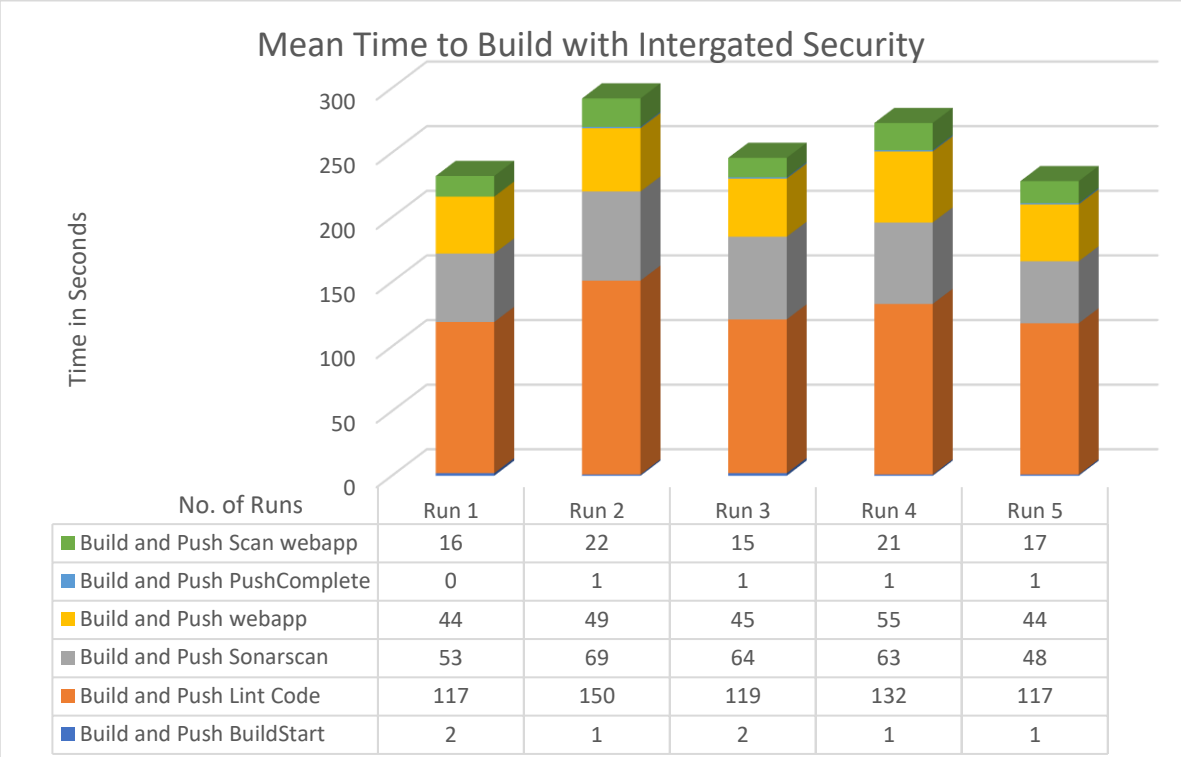


Figure 2: Application Deployment Workflow

The implementation involved carefully configuring and integrating these tools into the CI-CD pipeline to ensure smooth automation and security testing. The pipeline was structured to include stages for code compilation, static analysis, security testing, and containerization. The CI-CD pipeline's security measures were designed to detect and address vulnerabilities at various stages of the development process, from code writing to deployment. This proactive approach aimed to prevent security issues from propagating further down the pipeline. Furthermore, the implementation addressed how the pipeline would measure key metrics, such as deployment lead time, vulnerability identification, and mean time to change. These metrics were essential for evaluating the impact of the integrated tools on the overall security and deployment velocity. In conclusion, the design specification and implementation ensured the seamless integration of research resources in the CI-CD pipeline. The design's comprehensive approach emphasized security, code quality, and efficiency, contributing to the successful execution of the research project and its goal of achieving a balance between security measures and deployment velocity in DevOps practices.

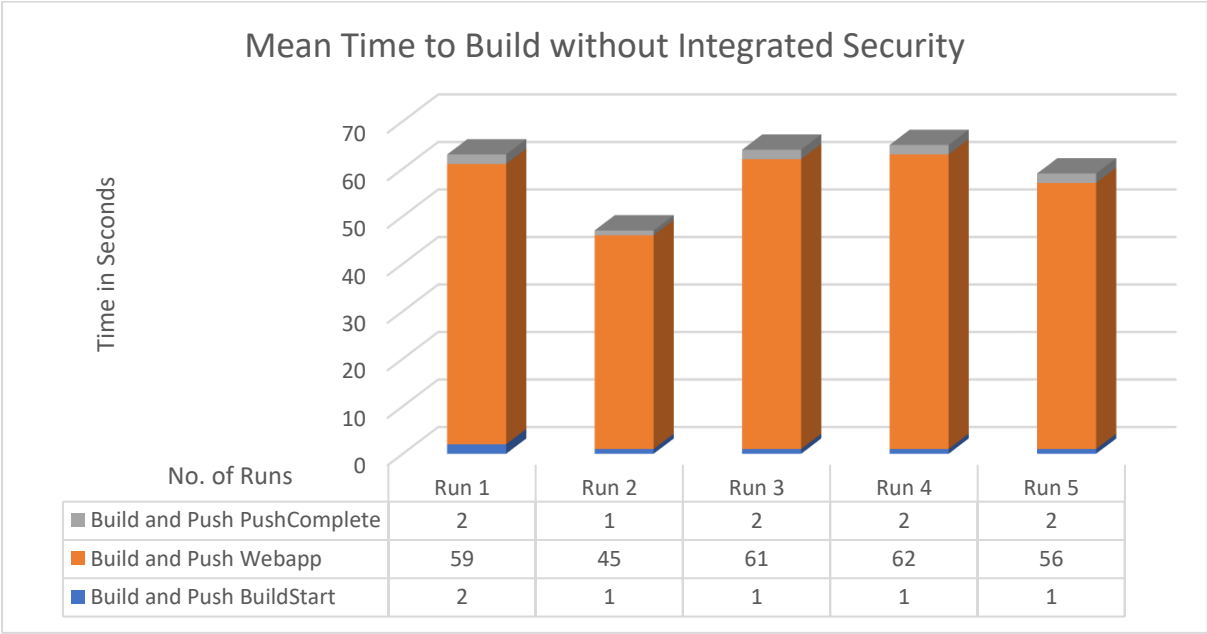
6 Evaluation

The evaluation of the research project involved analysing key metrics, including MTTB for measuring aspect of velocity and MTTC for measuring aspect of agility and vulnerabilities identified for measuring aspect of security, to understand the impact of integrating security measures in the CI-CD pipeline.



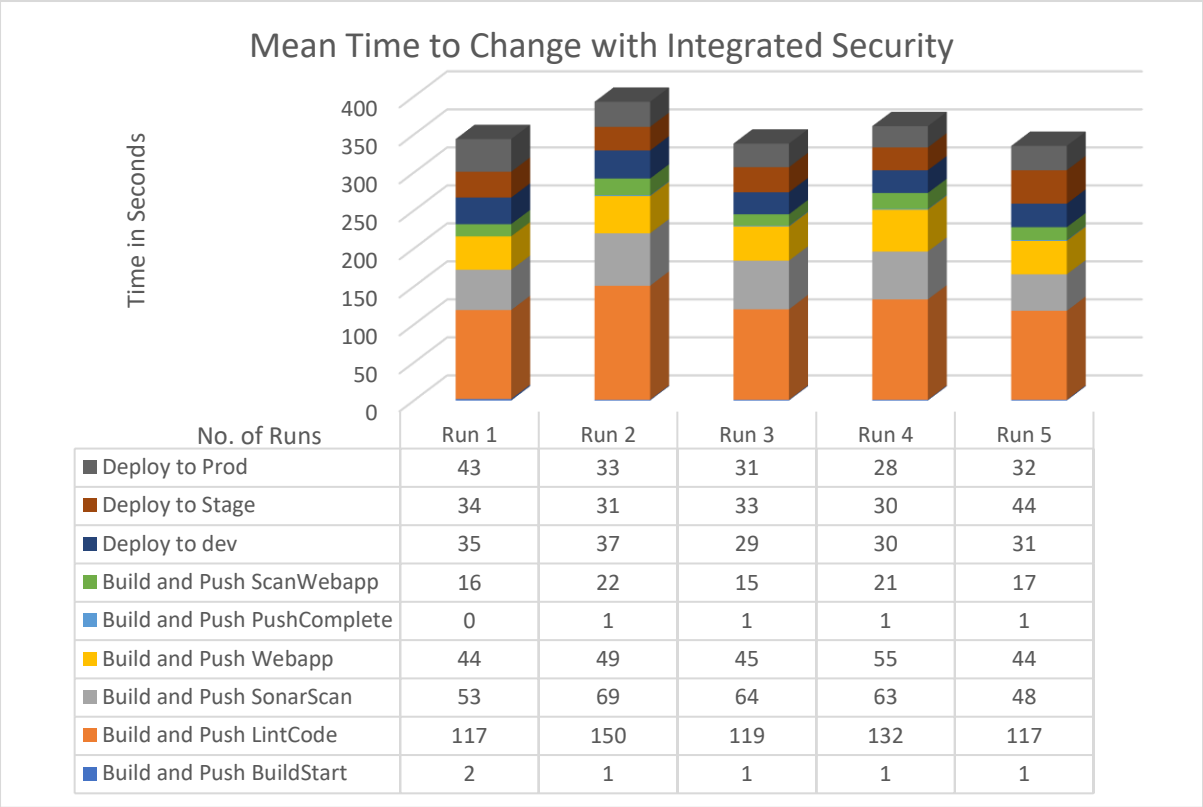
No. of Runs	Run 1	Run 2	Run 3	Run 4	Run 5
Build and Push Scan webapp	16	22	15	21	17
Build and Push PushComplete	0	1	1	1	1
Build and Push webapp	44	49	45	55	44
Build and Push Sonarscan	53	69	64	63	48
Build and Push Lint Code	117	150	119	132	117
Build and Push BuildStart	2	1	2	1	1

Figure 3: MTTB with Integrated Security Tools



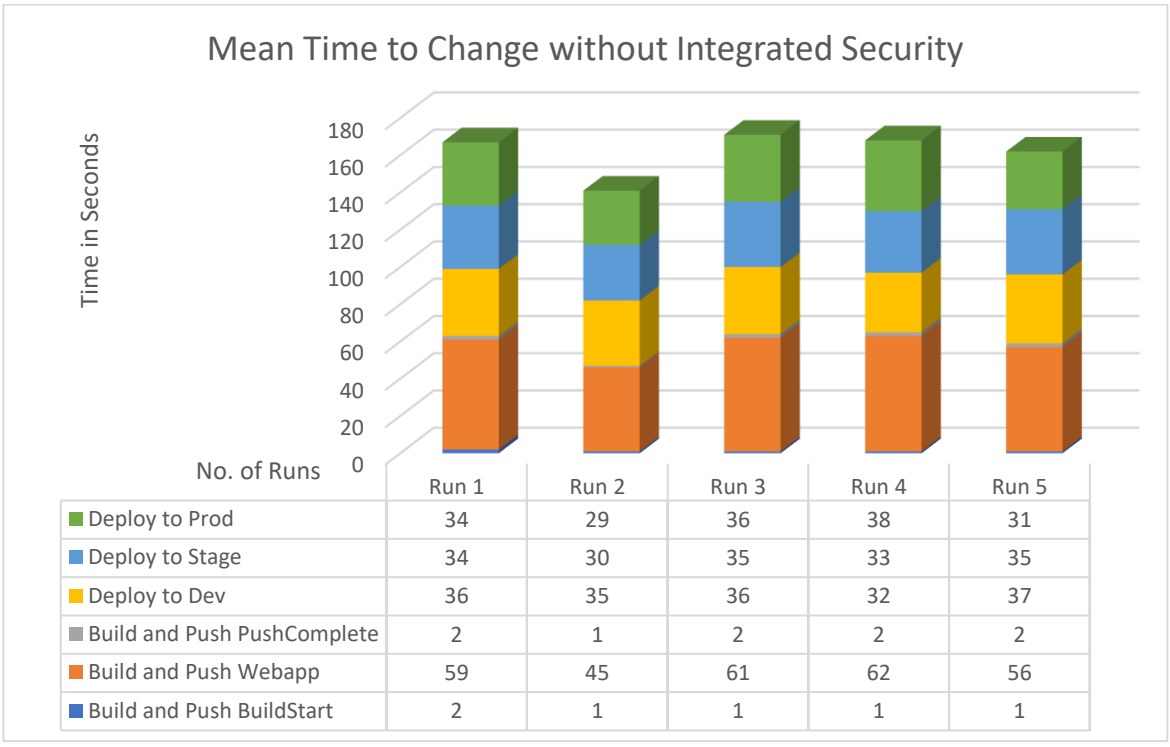
No. of Runs	Run 1	Run 2	Run 3	Run 4	Run 5
Build and Push PushComplete	2	1	2	2	2
Build and Push Webapp	59	45	61	62	56
Build and Push BuildStart	2	1	1	1	1

Figure 4: MTTB without Integrated Security Tools



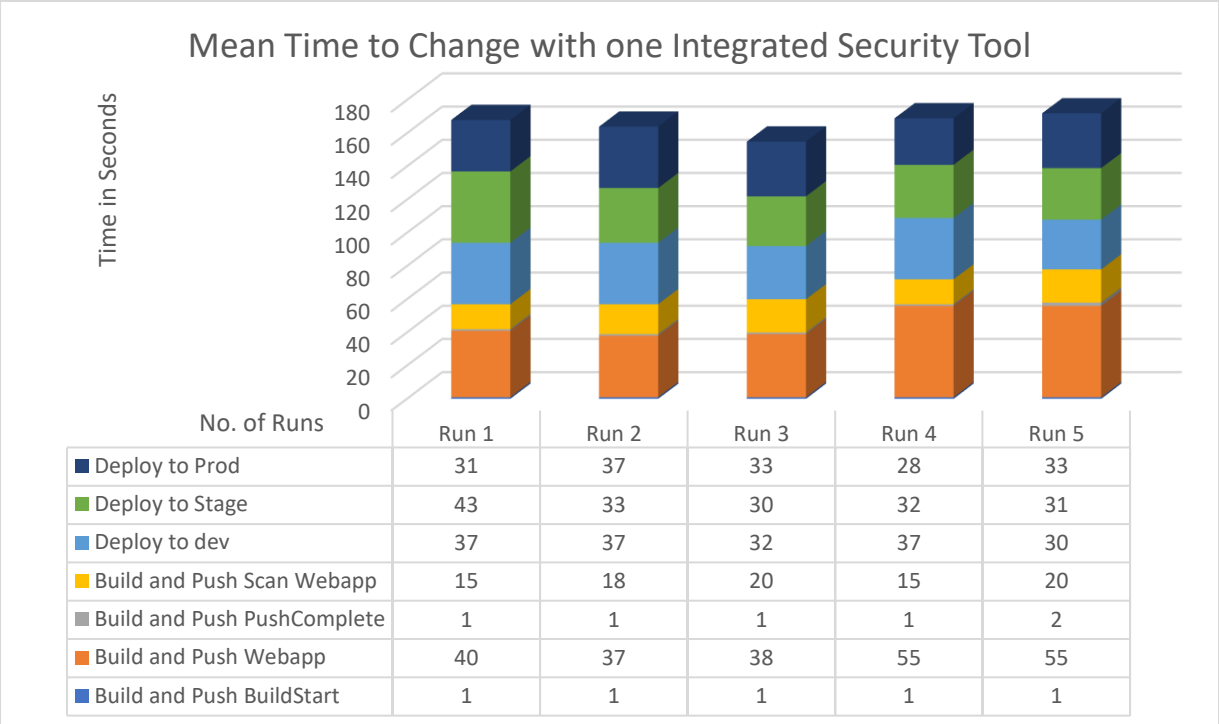
No. of Runs	Run 1	Run 2	Run 3	Run 4	Run 5
Deploy to Prod	43	33	31	28	32
Deploy to Stage	34	31	33	30	44
Deploy to dev	35	37	29	30	31
Build and Push ScanWebapp	16	22	15	21	17
Build and Push PushComplete	0	1	1	1	1
Build and Push Webapp	44	49	45	55	44
Build and Push SonarScan	53	69	64	63	48
Build and Push LintCode	117	150	119	132	117
Build and Push BuildStart	2	1	1	1	1

Figure 5: MTTC with Integrated Security Tools



No. of Runs	Run 1	Run 2	Run 3	Run 4	Run 5
Deploy to Prod	34	29	36	38	31
Deploy to Stage	34	30	35	33	35
Deploy to Dev	36	35	36	32	37
Build and Push PushComplete	2	1	2	2	2
Build and Push Webapp	59	45	61	62	56
Build and Push BuildStart	2	1	1	1	1

Figure 6: MTTC without Integrated Security Tools



	Run 1	Run 2	Run 3	Run 4	Run 5
Deploy to Prod	31	37	33	28	33
Deploy to Stage	43	33	30	32	31
Deploy to dev	37	37	32	37	30
Build and Push Scan Webapp	15	18	20	15	20
Build and Push PushComplete	1	1	1	1	2
Build and Push Webapp	40	37	38	55	55
Build and Push BuildStart	1	1	1	1	1

Figure 7: MTTC with One Integrated Security Tool

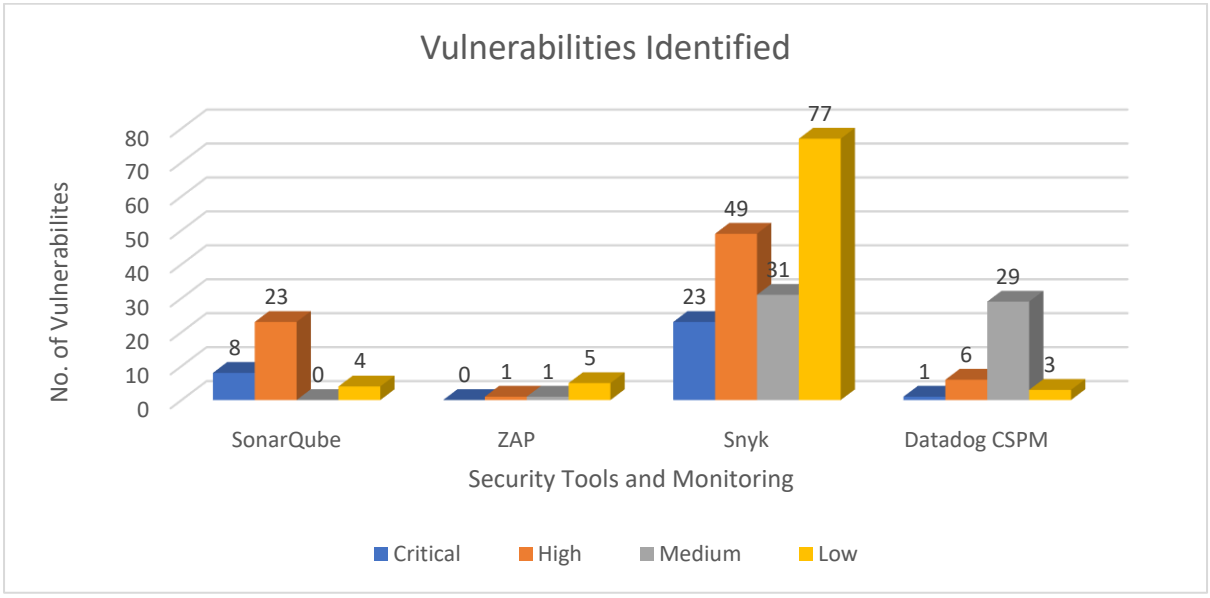


Figure 8: Vulnerabilities Identified by Security Tools and Monitoring

6.1 Discussion

The evaluation involves applying the metrics studied in literature review and study the impact of multiple security tools in the CI-CD pipeline. the analysis of key metrics to measure agility, velocity, and security in the CI-CD pipeline. Two crucial metrics, MTTB

and MTTC, were used to assess the efficiency and speed of the development process. Additionally, the number of vulnerabilities identified during security testing were measured to evaluate the impact of integrated security tools on the overall security posture of the application.

Mean Time to Build (MTTB)

MTTB measured the average time taken to build and compile the application code to the successful build completion from the initial pull request to merged in develop branch and master branch, shorter MTTB signifies faster development cycles and increased responsiveness to code changes.

During the evaluation, the MTTB was calculated for both the standard CI-CD pipeline and the pipeline with integrated security tools shown in Figure 3. The pipeline with integrated security experienced a significant increase in MTTB compared to the standard pipeline without integrated security shown in Figure 4. This can be attributed to the additional security testing and code analysis processes incorporated into the pipeline. While the increase in MTTB indicates improved security measures, it also signifies that the development cycle may take a little longer to ensure code quality and security.

Mean Time to Change (MTTC)

MTTC measured the average time taken from the successful build completion to the deployment of the application into the development, staging and production environments. This metric evaluated the speed of deployment and is a critical factor in measuring deployment velocity.

During the evaluation, the MTTC was substantially increased in the pipeline with integrated security tools displayed in Figure 5 compared to the standard pipeline displayed in Figure 6. The streamlined security testing and automated vulnerability scanning contributed to quicker identification and resolution of security issues, resulting in faster deployments. This upsurge in MTTC indicates that integrating security measures into the CI-CD pipeline increase the delay but can, in fact, enhance it by preventing security-related delays.

During the evaluation of MTTC with one Security Tool shown in Figure 7, it was observed that integrating Snyk into the CI-CD pipeline resulted in a slight increase in MTTC compared to the scenario with no security tools. The additional time taken can be attributed to the dependency scanning process. SonarQube provides crucial feedback to developers during the coding phase, it may not directly impact the work of release engineers in the deployment process and like SonarQube, Super-Linter's direct influence on the work of release engineers in the deployment phase is minimal. In contrast, Synk plays a more significant role in enhancing security during the deployment process by identifying vulnerabilities in third-party libraries. This proactive approach helps release engineers address security risks and avoid potential security incidents in the production environment. While this increase in MTTC signifies an enhanced security posture, it also reflects that the security checks introduced an additional step in the deployment process.

Vulnerabilities Identified

In addition to agility and velocity metrics, the evaluation focused on measuring the number of security vulnerabilities identified during the security testing phase. The pipeline with integrated security tools consistently detected and addressed a higher number of vulnerabilities. This outcome highlights the effectiveness of the integrated security tools in bolstering the security of the application.

The increased number of vulnerabilities identified in the security integrated pipeline demonstrated in Figure 8 shows the proactive nature of security measures, preventing potential security breaches in the production environment. By addressing vulnerabilities early in the development process, teams can significantly reduce the risk of security incidents and associated downtimes.

Overall, the evaluation revealed a significant shift in time with the integrated security tools. This shift indicates a balanced approach between security measures and deployment velocity, highlighting the importance of integrating security using automation into the CI-CD pipeline.

6.2 Trade-offs and best practices for achieving a balance between Security and Deployment Velocity

Trade-offs

Time-to-Market vs. Security: One of the most significant trade-offs is between time-to-market and security. Organizations often prioritize quick releases to stay ahead in the competitive market. However, rushing deployments can compromise security measures, leading to potential vulnerabilities. Striking the right balance involves implementing security practices without significantly impacting release time.

Automation vs. Manual Processes: Automation expedites the development process and enhances efficiency. However, some security aspects may require manual intervention, such as code reviews. Balancing automation with manual processes ensures thorough security testing while maintaining deployment velocity.

Comprehensive Testing vs. Deployment Speed: Comprehensive security testing, including static analysis, dynamic testing, and dependency scanning, is essential to identify vulnerabilities. However, extensive testing can slow down deployments as demonstrated in evaluation of the research. Optimized testing process ensures comprehensive security without hampering deployment speed.

Complexity vs. Maintainability: Introducing multiple security tools can increase pipeline complexity. While these tools enhance security, managing them can become challenging. Striking a balance involves choosing the most effective tools while considering ease of maintenance and operability.

Best Practices:

Shift-Left Security: Emphasize early security testing in the development cycle. Developers should be actively involved in identifying and addressing security issues before code commits. This reduces the chances of vulnerabilities reaching the later stages of the pipeline as studied in literature review and implemented in research methodology.

Continuous Security: Adopt a continuous approach to security testing throughout the CI-CD pipeline. Automated security tools continuously monitor and analyse code changes, providing real-time feedback to developers. This ensures timely identification and resolution of security issues.

Selective Security Tools: Carefully selecting security tools that align with the organization's specific security requirements. Avoid unnecessary replication of functionality and focus on tools that deliver the most use for the given context.

Security as Code: Integrate security practices into the CI-CD pipeline as code. This includes versioning security configurations and treating security-related infrastructure as code to enable repeatability and consistency.

Continuous Learning and Improvement: Encourage a culture of continuous learning and improvement in security practices. Regularly review and update security processes and tools to stay ahead of evolving threats.

Collaboration and Communication: Encourage collaboration between the development, operations, and security teams to ensure that everyone is in agreement with the security goals. Effective communication can lead to better decision-making and problem-solving.

Security Training: Security training to all team members involved in the development and deployment process as knowledgeable teams are better equipped to identify and address security concerns effectively.

Security Metrics: Define and measure key security metrics to track the effectiveness of security measures and identify areas for improvement.

Continuous Feedback Loop: Establishing a continuous feedback loop to gather insights from users on security aspects. This feedback can inform improvements in security practices and contribute to a more secure and user-centric result.

7 Conclusion and Future Work

This research highlights speed as a significant factor for DevOpsSec. This research aimed on integrating multiple security tools in the DevOps pipeline and examine the impact of the security and efficiency of the CI-CD process and also provide trade-off between the number of security tools used and the speed of the pipeline. This research objective was to find a balance that maximizes security while minimizing any negative impacts on deployment velocity. Through a thorough literature review, this research discussed various stages of security integrated and listed various security tools at various stages of the pipeline and explored metrics for evaluating the efficiency and agility of the CI-CD pipeline.

Through the research method and implementation of the CI-CD pipeline, this research provides the best practices for achieving a balance between security and deployment velocity in DevOps including continuous monitoring and improving the security posture of the application. Overall, this research contributes to the knowledge in the field of DevOpsSec by providing insights into the effective use of security tools and their impact on the CI-CD process. It highlights the importance of balancing security and deployment velocity and provides practical recommendations for achieving this balance.

The evaluation involved applying the metrics researched in literature review and study the impact of multiple security tools in the CI-CD pipeline. The metrics used are mean time to change, vulnerabilities identified, and Mean time to Build which measured the aspects of agility, velocity, and security in the CI-CD pipeline. Additionally, the balance is achieved by conducting experiments to measure the impact of varying numbers of tools on the pipeline and the corresponding metrics. By measuring and tracking these metrics, the effectiveness of the pipeline is evaluated, and improvements is made to optimize the pipeline for efficiency and security. By selecting a subset of tools and evaluation metrics, the research provides insights into how to achieve a balance between security and deployment velocity in DevOps.

The future work for this research lies in exploring advanced security tools, Artificial Intelligence/Machine Learning based security, dynamic security orchestration, and the impact of security on customer experience. Additionally, enhancing security training for developers, automated remediation mechanisms, and industry-specific security guidelines are areas that hold promise for further research. By exploring these avenues, the research can provide supplementary contribution to the continuous improvement of security practices in DevOps and ensure a balanced approach that prioritizes both security and deployment velocity.

References

Ahmed, Z. & C. Francis, S., 2019. *Integrating Security with DevSecOps: Techniques and challenges.* International Conference on Digitization (ICD), cited by 24, DOI: 10.1109/ICD47981.2019.9105789

Bird, J., 2016. *DevOpsSec.* O'Reilly Media, Inc., URL: https://learning.oreilly.com/library/view/devopssec/9781491971413/ [Accessed 18 March 2023].

Deegan, C., 2020. *Continuous Security; Investigation of the DevOps Approach to Security.* School of Computing, National College of Ireland.

Desai, R. & T N, N., 2021. *Best Practices for Ensuring Security in DevOps: A Case Study Approach.* Journal of Physics: Conference Series, Volume 1964, Issue 4, article id. 042045, Cited by 3, DOI: 10.1088/1742-6596/1964/4/042045

Dyess, C., 2020. *Maintaining a balance between agility and security in the cloud.* 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED), Cited by 1, URL: https://www.sciencedirect.com/science/article/pii/S1353485820300313

Hsu, T., 2018. *Hands-On Security in DevOps.* Published by Packt Publishing, URL: https://learning.oreilly.com/library/view/hands-on-security-in/9781788995504/ [Accessed 12 March 2023]

Jammeh, B., 2020. *DevSecOps: Security Expertise a Key to Automated Testing in CI-CD Pipeline.* Department of Computing and Informatics Bournemouth University Poole, England, cited by 4.

Krief, M., 2019. *Learning DevOps.* Chapter 1, Published by Packt Publishing, URL: https://learning.oreilly.com/library/view/learning-devops/9781838642730/ [Accessed 22 March 2023]

Kumar, R. & Goyal, R., 2020. *Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC).* Volume 97, University School of Information, Communication and Technology (USIC&T), Guru Gobind Singh (GGS) Indraprastha University, New Delhi 110078, India, Cited by 46, URL: https://doi.org/10.1016/j.cose.2020.101967

Kumar, R. & Goyal, . R., 2021. *When Security Meets Velocity: Modeling Continuous Security for Cloud Applications using DevSecOps.* Part of the Lecture Notes on Data Engineering and Communications Technologies book series (LNDECT,volume 59), cited by 6, URL: https://doi.org/10.1007/978-981-15-9651-3_36

Larrucea, X., Berreteaga, A. & Santamaria, I., 2019. *Dealing with Security in a Real DevOps Environment.* Communications in Computer and Information Science book series (CCIS,volume 1060), Cited by 8, URL: https://doi.org/10.1007/978-3-030-28005-5_35

Lehtonen, T., Suonsyrja, S., Kilamo, T. & Mikkonen, T., 2015. *Defining Metrics for Continuous Delivery and Deployment Pipeline.* Tampere University of Technology, Tampere, Finland, Symposium on Programming Languages and Software Tools, Cited by 32, URL: http://ceur-ws.org/Vol-1525/paper-02.pdf

Leppanen, T., Honkaranta, A. & Costin, A., 2022. *Trends for the DevOps Security. A Systematic Literature Review.* Conference: Business Modeling and Software Design: 12th International Symposium, BMSD 2022At: Fribourg, Switzerland, Cited by 1, DOI: 10.1007/978-3-031-11510-3_12, URL: https://www.researchgate.net/publication/361696077

Lwakatare, L. E. et al., 2019. *DevOps in practice: A multiple case study of five companies.* Information and Software Technology Volume 114, October 2019, 217-230, Cited by 139, URL: https://doi.org/10.1016/j.infsof.2019.06.010

Mansfield-Devine, S., 2018. *DevOps: finding room for security.* Network Security, Volume 2018, Issue 7, cited by 39, URL: https://doi.org/10.1016/S1353-4858(18)30070-9

Mao, R. et al., 2020. *Preliminary Findings about DevSecOps from Grey Literature.* IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), cited by 42, DOI: 10.1109/QRS51102.2020.00064

Mohan, V. & Othmane, L. b., 2016. *SecDevOps: Is It a Marketing Buzzword?.* 2016 11th International Conference on Availability, Reliability and Security, cited by 122, DOI: 10.1109/ARES.2016.92

Rahman, A. A. U. & Williams, L., 2016a. *Security Practices in DevOps.* HotSos '16: Proceedings of the Symposium and Bootcamp on the Science of Security, URL: https://doi.org/10.1145/2898375.2898383

Rahman, A. A. U. & Williams, L., 2016b. *Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices.* 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED), Cited by 95, cited by 25,
URL: https://doi.org/10.1145/2896941.2896946

Rajapakse, R. N., Zahedi, M., Babar, M. A. & Shen, H., 2022. *Challenges and solutions when adopting DevSecOps: A systematic review.* Information and Software Technology, Volume 141, January 2022, 106700, Cited by 21, URL: https://doi.org/10.1016/j.infsof.2021.106700

Rajapakse, R., Zahedi, M. & Babar, M. A., 2021. *An Empirical Analysis of Practitioners' Perspectives on Security Tool Integration into DevOps.* ESEM '21: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement,
URL: https://doi.org/10.1145/3475716.3475776

Rangnau, T., Buijtenen, R. v., Fransen, F. & Turkmen, F., 2020. *Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI-CD Pipelines.* IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC), Cited by 25,
URL: https://doi.org/10.1109/EDOC49727.2020.00026

Rao, M., 2021. *How to integrate automated AST tools in your CI-CD pipeline.* Synopsys,
URL: https://www.synopsys.com/blogs/software-security/integrating-automated-ast-tools/ [Accessed 01 February 2023].

Sánchez-Gordón, M. & Colomo-Palacios, R., 2020. *Security as Culture: A Systematic Literature Review of DevSecOps.* ICSEW'20: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, cited by 45, URL: https://doi.org/10.1145/3387940.3392233

Sojan, A., Rajan, R. & Kuvaja, P., 2021. *Monitoring solution for cloud-native DevSecOps.* IEEE 6th International Conference on Smart Cloud (SmartCloud), cited by 4,
DOI: 10.1109/SmartCloud52277.2021.00029

Vehent, J., 2018. *Securing DevOps.* Manning Publications,
URL: https://learning.oreilly.com/library/view/securing-devops/9781617294136/ [Accessed 28 May 2023].

Yarlagadda, R. T., 2020. *DevOps for Better Software Security in the Cloud.* Chapter 2, SSRN Electronic Journal, Department of Information Technology, USA,
URL: https://www.researchgate.net/publication/350157883