JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 39, 1117-1128 (2023)
DOI: 10.6688/JISE.202309_39(5).0007

Microservices-based DevSecOps Platform using Pipeline and Open Source Software*

WEN-TIN LEE+ AND ZHUN-WEI LIU
Department of Software Engineering and Management
National Kaohsiung Normal University
Kaohsiung, 80201 Taiwan
E-mail: {wtlee; 611077104}@mail.nknu.edu.tw

Continuous integration and continuous deployment (CI/CD) are best practices for automating the software development process. People leverage them to ensure rapid iteration and delivery of product development. The rapid lifecycle makes traditional security management vulnerable to its lack of agility, exposing the urgent need to put security into DevOps processes. Development, security, and operation, quoted as DevSec Ops, advocates shift-left security, promotes people to implant security best practices into all DevOps stages, and builds continuous security analysis, testing, and management with automation.

Based on CI/CD, this study defines continuous security practices and applies application security processes on a DevSecOps pipeline to implement shift-left security. The CodeHawk platform, based on the proposed secure pipeline and open source software, is developed to free the development team from testing manually, enable them to focus on development, gain the corresponding security assurance, and lower the operating costs. Experiments show that our DevSecOps pipeline design significantly improves the efficiency of the DevSecOps process.

***Keywords*:** DevOps, DevSecOps, continuous integration, continuous delivery, security testing, open-source software

1. INTRODUCTION

Continuous integration and continuous deployment (CI/CD) are best practices of DevOps to fill the communication gap between the development and operation teams. People used to take advantage of testing automation and version release to ensure fast delivery and bring DevOps culture to development teams. Implementing the DevOps process with the premise of automation can increase delivery speed and product quality. This workload and transformation in the software lifecycle expose the shortcomings of traditional security management. Especially when security awareness increases, so does the importance of automating security testing to be integrated with the DevOps process. Kurmar [1] categorizes the DevOps process into five parts: continuous planning, continuous development, continuous integration, continuous delivery, and continuous opera tion. With the growth of various open-source software (OSS), more and more programming languages are supported, making them easier to be integrated into the DevOps pro cess.

Containerized platforms allow users to pack applications into images that can be executed independently, improving the convenience of deployment. Since containers are independent and stand-alone, the logs within containers become harder to collect, so con-

Received November 1, 2022; revised December 7, 2022; accepted February 9, 2023.
Communicated by Lok Ka Man.
* This research was sponsored by Ministry of Science and Technology in Taiwan under grants MOST 110-2221-E-017-001-.

tinuous monitoring in the DevOps process will become problematic. Additionally, container orchestration is required if multiple containers are to be manipulated. Kubernetes is a centrally managed container orchestration application that can schedule or repair containers automatically. By applying the orchestration mechanism of Kubernetes, automation with high availability and expandability is realized. Containerized applications can then be managed in clusters for easier control during automated deployments.

This study investigates how security and test automation can be applied to CI/CD pipelines through the following four automated security activities: third-party software vulnerability scanning, static and dynamic application security testing, encrypted authentication during transmission, and security management. According to the designed DevSec Ops pipeline, we build a DevSecOps platform called CodeHawk using microservice architecture, OSS, and container technologies. The CodeHawk maintains the current working state of the pipeline in the event of a failure so that the development team can continue testing after revisions. It also detects incorrect settings and stops deployment jobs promptly to prevent time consumption caused by worthless deployments. QA engineers can then review all test results stored in the artifacts volume.

The remainder of this paper is organized as follows: Section 2 presents background knowledge and related work. Section 3 introduces security requirements and the design of the DevSecOps pipeline. Section 4 describes CodeHawk DevSecOps platform design and implementation. Section 5 conducts performance evaluations and explains the results. Finally, Section 6 concludes this study with future works.

2. RELATED WORK

This section introduces background information that significantly impacts this work, especially the research on DevOps, DevSecOps, continuous integration, continuous delivery, continuous testing maturity model [2], and security process automation.

According to IEEE Standard for DevOps [3], development and operation is a set of principles and practices which can offer methods and solutions to communicate and cooperate with stakeholders. It enables the project team to communicate efficiently with their clients and continuously improve in all aspects. As an extension of DevOps, the practices of DevSecOps adds security controls and advocates security shift-left, system design based on security concerns, and automated continuous security testing.

Continuous integration is when developers frequently push their work into the main branch [3], and all these changes will be built and tested automatically. Hence, operation and development engineers can cooperate and discover integration errors early, thus applying shift-left security to the project [4]. The goal of continuous delivery is that the entire project can have a stable version that can be deployed and ready for production at any time. The project should pass the required tests and push the tested code onto repositories. Deployment actions of when or which version to be published should be done manually by those authorized operation persons [4].

Certificate management is also essential when adopting open-source software. Angermeir *et al.* [5] pointed out that once projects have achieved automation and have lower coverage of security regulation, it is rare to see those projects integrate open-source software and do not value sensitive data management in their automated process. Roshan [6], Dupont

[7], Preira-Vale [8], Rahman [9] and their teams proposed 12 secret management implementations and mechanisms. Developers could establish authority management outside open-source software quickly and at a lower cost. Raza [10] mentioned the pros and cons of various cloud computing environments, pointing out that the private cloud has high security, privacy, stability, and controllability.

Throner [11] proposed a model-based DevOps environment upon the basis of Kubernetes, making use of the YAML configuration file to achieve fast deployment and version release, offering YAML configuration file reference for various deployment environments to lower the gitlab-runner workload. According to Silkin's [2] statement and evaluation, the continuous testing maturity model can be defined using three aspects, namely risks, quality, and cost. This maturity model implies an automated testing pipeline should at least include continuous unit testing, API testing, and dynamic testing. The test server for this study uses docker and Kubernetes to containerize the test services for better performance and load balancing, with communication between pipelines and machines via SSH.

NIST provided guidance for implementing DevSecOps primitives such as CI/CD pipelines for a reference platform hosting microservices-based applications with service mesh. The source code involved in the application environment is classified into five types: application code, application services code, infrastructure as code, policy as code, and observability as code. Their study describes what it takes to implement DevSecOps primitives for five types of source code. Conversely, our work focuses on developing a microservices-based DevSecOps platform by integrating various open-source software. This study describes in detail the security requirements, pipeline design, architectural design, implementation, and benefits of the CodeHawk DevSecOps platform.

3. DEVSECOPS SECURITY REQUIREMENTS AND PIPELINE DESIGN

Based on Kurmar's work [1], we analyze the challenges encountered in adopting open-source software as DevSecOps tools and collect security requirements for designing a DevSecOps pipeline. Table 1 shows the identified security requirements and the corresponding stages, which can also serve as a reference table for achieving the goals of continuous delivery and deployment in different stages. We increase the security level based on the security requirements during the build, test, release, deployment, and operation stages.

Table 1. Security requirements and its corresponding pipeline stages.

ID	Security Requirements	Stages
SR01	Security Requirement Analysis	Plan
SR02	Threat Modeling	Plan
SR03	Adaptive Security Architecture Decision	Code
SR04	Adaptive Security Architecture Design	Plan
SR05	Security use, misuse, abuse Test Cases	Plan
SR06	Code Review & Security Guidelines Linting	Code
SR07	Software Inventory Management	Build, Test
SR08	Version Control Security	Code, Build, Test, Release
SR09	Unit testing and Integration Testing Security	Test

SR10	Container Analysis and Infrastructure as Code	Test, Release
SR11	User Acceptance & Security Testing	Deploy, Operate
SR12	Artifact Repository Security Management	Build, Test, Release
SR13	Secret Management	Code, Build, Test, Release, Deploy
SR14	Infrastructure Provisioning and Orchestration	Release, Deploy
SR15	Application, System Logging	Operate, Monitor
SR16	Continuous Monitoring	Operate, Monitor
SR17	Security Incident Management	Monitor
SR18	Security Metric Measurement & Analysis	Operate, Monitor
SR19	Security Audit and Compliance	Operate, Monitor
SR20	Penetration Testing	Test, Operate
SR21	Static Application Security Testing	Test
SR22	Dynamic Application Security Testing	Test
SR23	Interactive Application Security Testing	Test
SR24	Continuous Vulnerability Scanning	Test, Operate
SR25	Security Patch Application	Build, Test, Operate
SR26	Infrastructure Hardening & Security Testing	Test, Deploy, Operate
SR27	Security Governance	ALL
SR28	Security Policy Enforcement	ALL
SR29	Access and Privilege Management	ALL
SR30	Integrated Minimal Common Processes, Methods, and Tools	ALL
SR31	Integrated Project Planning, Execution, and Monitoring	ALL

Continuous integration runs through the build and testing stages. The earlier an integration error is detected, the higher the integration success rate of a project. Fig. 1 shows the pipeline steps designed from integration with open-source software. The entire DevSecOps testing pipeline can be divided into three phases: the testing phase, the stage phase, and the production phase. During the testing phase, an automatic integration pipeline is triggered after a commit is merged into the master branch of the project's git repository. The stage phase performs container security scanning and continuous testing. The pipeline will conduct image vulnerability scanning using Trivy [12]. For the production phase, the pipeline will trigger the production server to start its deployment. The software quality is enhanced by applying dynamic application testing in all three phases.

The pipeline utilizes Gitlab [13] and its work stage configuration through YAML files to integrate SonarQube [14] as the static code analysis tool. After passing the quality gate of static code analysis, the DevSecOps pipeline can enter the continuous deployment stage. The project will be deployed on the test server to conduct dynamic, dependency, unit, and API tests. Once this stage is passed, the latest committed code complies with security requirements. The pipeline will pack the code into an image, upload it to the Sonatype nexus repository [15], and trigger the stage server to work. As the Continuous Test Maturity Model recommends, the project's current version will be deployed on the nexus repository to complete container security checks and API testing. Once the image server testing process is complete, the final artifact will be pulled to the production server for deployment.

CI/CD workflow is the most critical element in DevOps. Hence the pipeline architecture design focuses on security and testing. The job should only enter the next stage when the required security activities are accomplished. In our architecture, the open-source tools are all deployed independently on containers, which can isolate the tools' execution environment. All jobs and data transactions run through an internal network to secure all hosts by insulating internal and external networks. In addition, OWASP Zap [16] is used for dynamic security testing and is deployed separately for continuous testing.

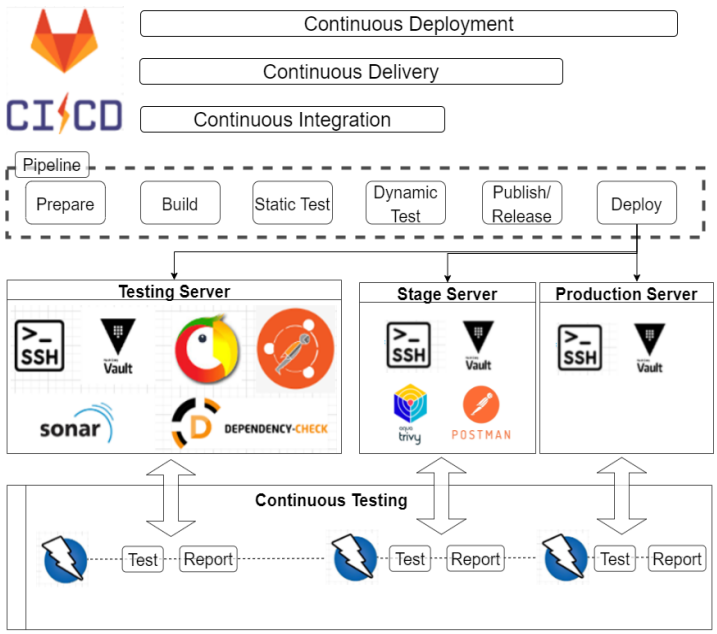

Fig. 1. CodeHawk DevSecOps pipeline design.

Authentication, authorization, and sensitive data control policies are easily overlooked during pipeline automation. To integrate secrets management into the DevSecOps pipeline, Codehawk integrates Hashicorp Vault [17] secrets database with Gitlab. During the initial infrastructure configuration, the SSH connection keys for the three servers and the Gitlab host are issued by Vault. The secret key and connection mechanism are secured, and the Vault server is isolated from all the other hosts.

4. CODEHAWK DEVSECOPS PLATFORM DESIGN AND IMPLEMENTATION

The Codehawk DevSecOps platform separates front-end and back-end and integrates RESTful API to implement a microservices architecture. Keycloak [18] is integrated as an authorization system for single sign-on and permissions policy management services. Fig. 2 shows the software architecture of the CodeHawk DevSecOps platform. There are three kinds of users: developers, project managers, and quality assurance engineers. When using

our portal website, users can access it from a single sign-in webpage, and CodeHawk will render the authorized content to their browser according to the corresponding authority. CodeHawk DevSecOps platform can strengthen code quality and security by integrating several open-source software, including Kanban software WeKan [19], Gitlab, SonarQube, OWASP Zap, Docker [20], Kubernetes [21], Sonatype, and ELK Stack [22].

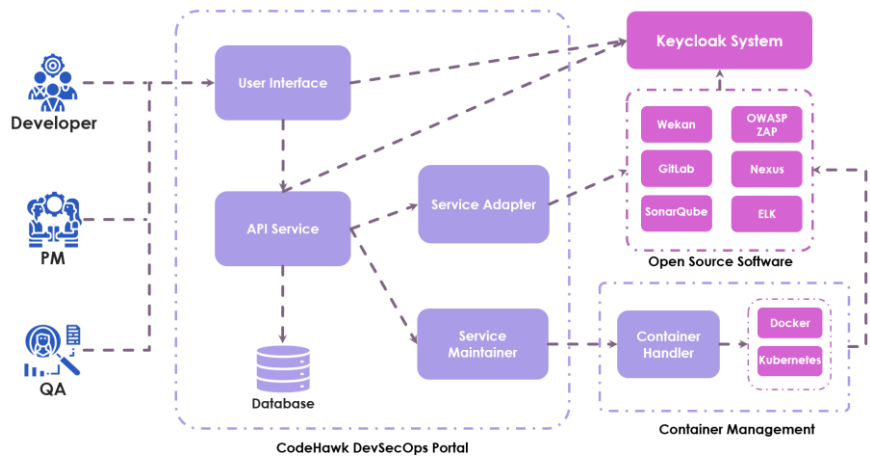

Fig. 2. Software architecture of CodeHawk DevSecOps platform.

After the user logs in, it will respond to the authorization content through the API Service and communicate with various system services through the Service Adapter. The component User Interface integrates all project information. The API Service component is connected to the KeyCloak system [18], which will direct authorized users to the project web page; the Service Adapter component uses multiple Restful APIs to link external Open source software. Since the information from different platforms is inconsistent, we use Anser, a microservices orchestration library, to integrate the data passed from the back-end DevOps open-source software, extract useful Meaning Data, and then respond to the front-end interface. The main task of the Service Maintainer component is to collect container information executed by Docker and Kubernetes and send it back to the front end. It is connected to ELK so the data can be more conveniently integrated into the user interface. Container Management uses Docker and Kubernetes as deployment and orchestration tools. The Container handler deploys the packaged image of the project as a Docker container, adds the container information to Kubernetes, and then uses Kubectl's features to improve the high availability of the entire system.

Fig. 3 shows the user interface of the CodeHawk Portal. It includes the login screen provided by KeyCloak after the user enters the CodeHawk platform; the overview of the orderService project; the project screen of Wekan; the Gitlab screen of the project and the execution results of the pipeline; static code analysis results of orderService project.

The users can sign in to the system, select or create a project, and set the connections with the required open-source software services, such as git repository Gitlab, Wekan, SonarQube, *etc.* The static code analysis results can then be generated each time the code is merged into the git repository.

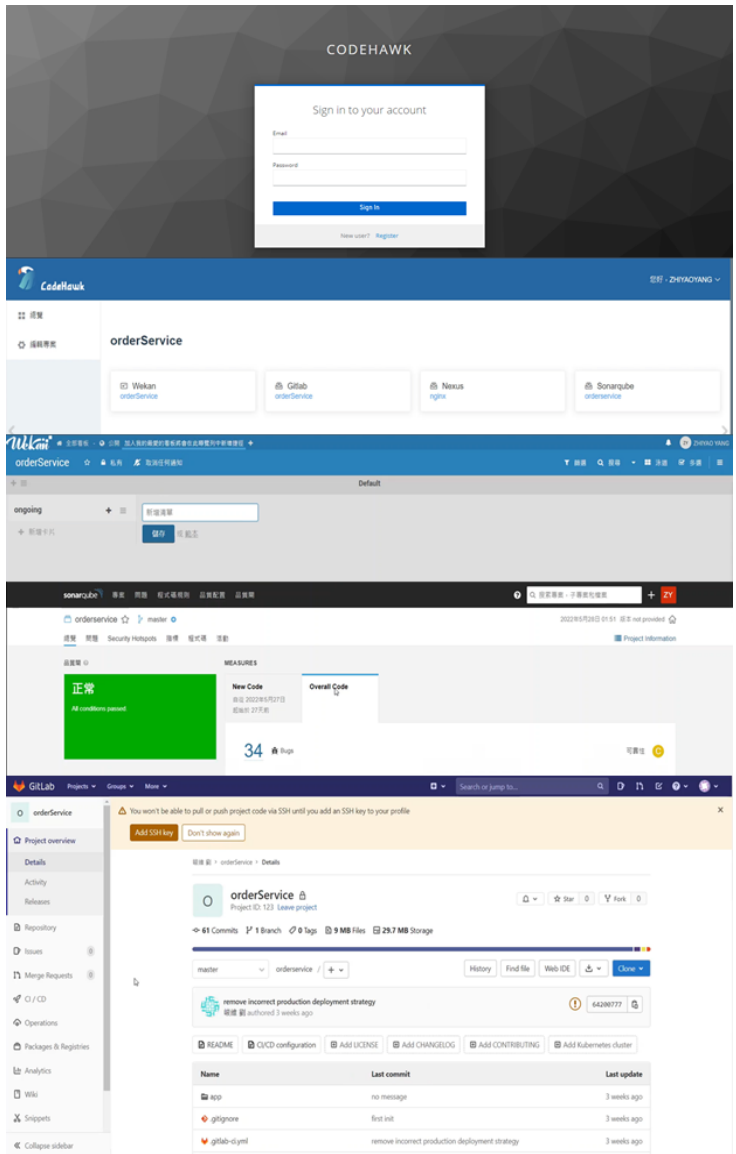

Fig. 3. CodeHawk portal user interface.

5. EVALUATION RESULTS

CodeHawk integrates open-source software, including Gitlab, SonarQube, OWASP Dependency-Check [23], Postman [24], Sideex, Zap, and Unit Test tools, in the CodeHawk platform to enable shift-left testing, and increase successful integration rates. CodeHawk uses static code analysis to filter unworthy commits through a strict quality gate configuration for shift-left security and continuous testing. This study uses continuous testing maturity model [2] to assess CodeHawk's automated testing.

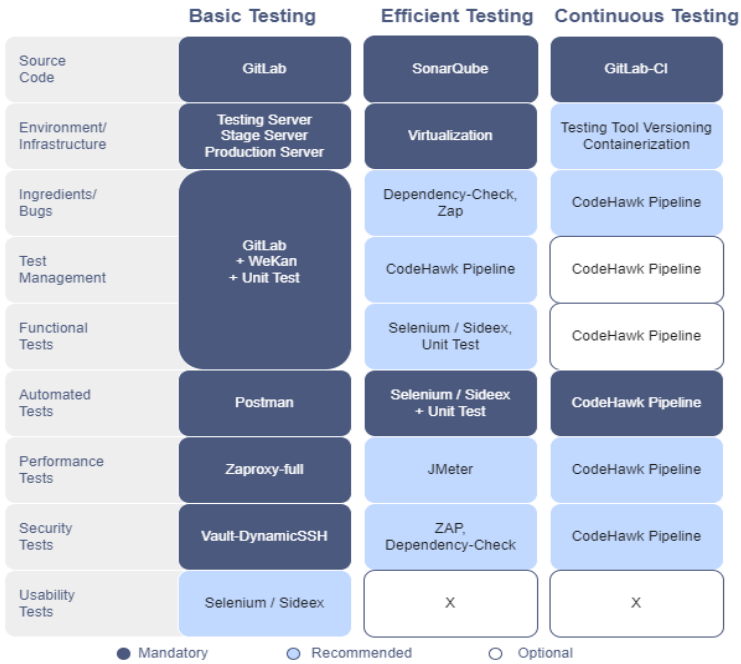

Fig. 4. Open-source software corresponds to continuous testing maturity model [2].

Fig. 4 shows the correspondence between the proposed pipeline design and the continuous test maturity model. Automated testing is classified into basic, effective, and continuous testing; For source code management, Gitlab is used for version control of source code, and Gitlab-CI integrates and automates SonarQube for automatic static code testing. Environment and infrastructure refer to where the software itself is placed and executed. In the basic test, the test environment is divided into three parts, the test, the staged, and the production environment. Virtualization technology allows users to execute desired tests in an environment with minimal resources. Since emerging attack methods and dependent package vulnerabilities are unpredictable risks, testing tool versioning and containerization are recommended for test environment management and rapid deployment.

The corresponding open-source software includes Gitlab, Wekan, unit test tool, and Postman API testing tools for software bugs, test management, functional tests, and automated tests. We recommend testing with OWASP Dependency-Check and Zap to secure dependency packages and software to find more software bugs and security issues. For the test management according to the design of the CodeHawk security-first pipeline, test management can be more efficient and automated. For functional automation testing, in addition to the inspection of traditional test cases, it is also recommended to add automated script testing tools such as Selenium or Sideex, so that the project can conduct simulated user testing in an earlier development cycle.

The performance test uses the full scanning of the Zap test tool to test the performance of the software application. Further, it is recommended to use JMeter for automated performance testing to analyze software performance under stress or load testing. For building a reliable automated security testing script or pipeline, we have integrated Vault's dynamic

SSH key function to make the communication between servers more secure. It is also recommended to add Zap and Dependency-Check in the security testing process to eliminate security vulnerabilities of third-party software. Finally, for usability testing, automated testing tools such as Selenium and Sideex can significantly reduce testing costs through script writing and recording.

Fig. 5 shows the failed execution result of the designed pipeline. The pipeline will stop automatically to prevent unworthy deploys and halt the recent job. After correction, developers can continue this testing pipeline and press the play button next to the stage name to continue execution. Not only can the test results affect the job's status, but wrong commands can also cause the pipeline to stop. As shown in Fig. 6, a failed deployment configuration command triggers the pipeline to pause zap testing in the production server and warns developers to make changes.

Fig. 7 shows the successful execution of the entire pipeline. The pipeline design improves code quality and security by integrating automated unit testing, API testing, and penetration testing.

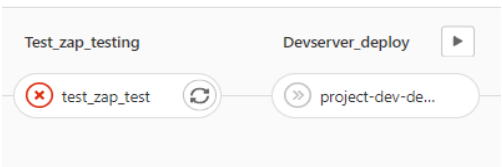

Fig. 5. Failed pipeline execution result.

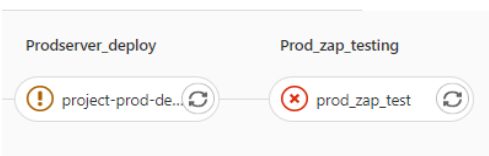

Fig. 6. Failed deployment configuration.

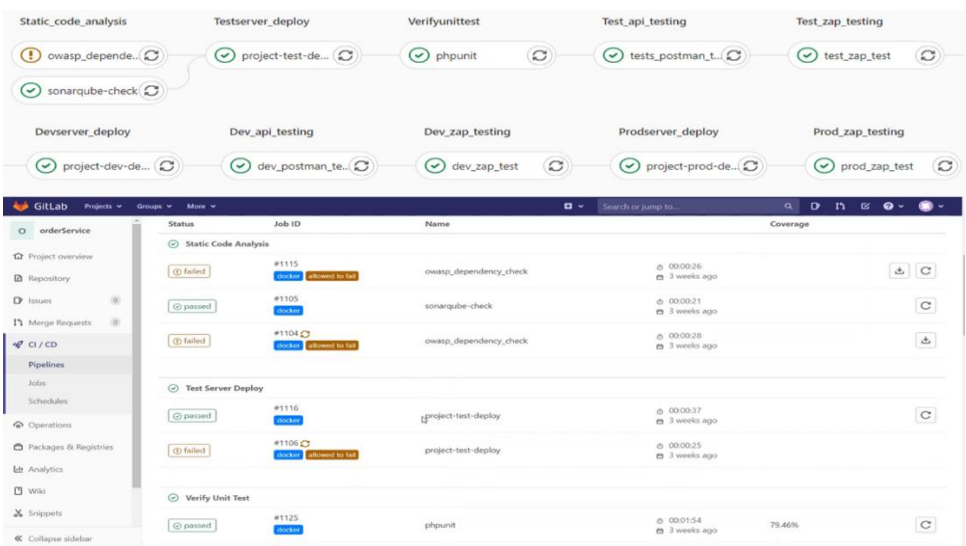

Fig. 7. The pipeline execution results.

The testing results and artifacts are automatically put into a directory for developers to access. This study conducts the experiment by comparing the performance between the proposed pipeline and manual testing. Both test cases use the same project and run on a host with the same specs. Fig. 8 shows each step's time comparison between manual testing and the CodeHawk pipeline. Traditional testing methods require all other tools to be pre-installed and checked. The proposed pipeline can reduce time costs and ensure system security.

Steps	Manual Testing	CodeHawk Pipeline
Static analysis	2 min 5 sec	47 sec
testServer_deploy	7 min 37 sec	37 sec
Unit test	1 min 24 sec	1 min 46 sec
API test (Testing Server)	1 min 40 sec	29 sec
Dynamic test (Testing Server)	2 min 34 sec	43 sec
Container Vulnerability Scanning	5 min 58 sec	37 sec
API test (Image Server)	48 sec	23 sec
Dynamic test (Image Server)	1 min 35 sec	44 sec
ProdServer_deploy	5 min 20 sec	40 sec
Dynamic test (Production Server)	1 min 44 sec	42 sec

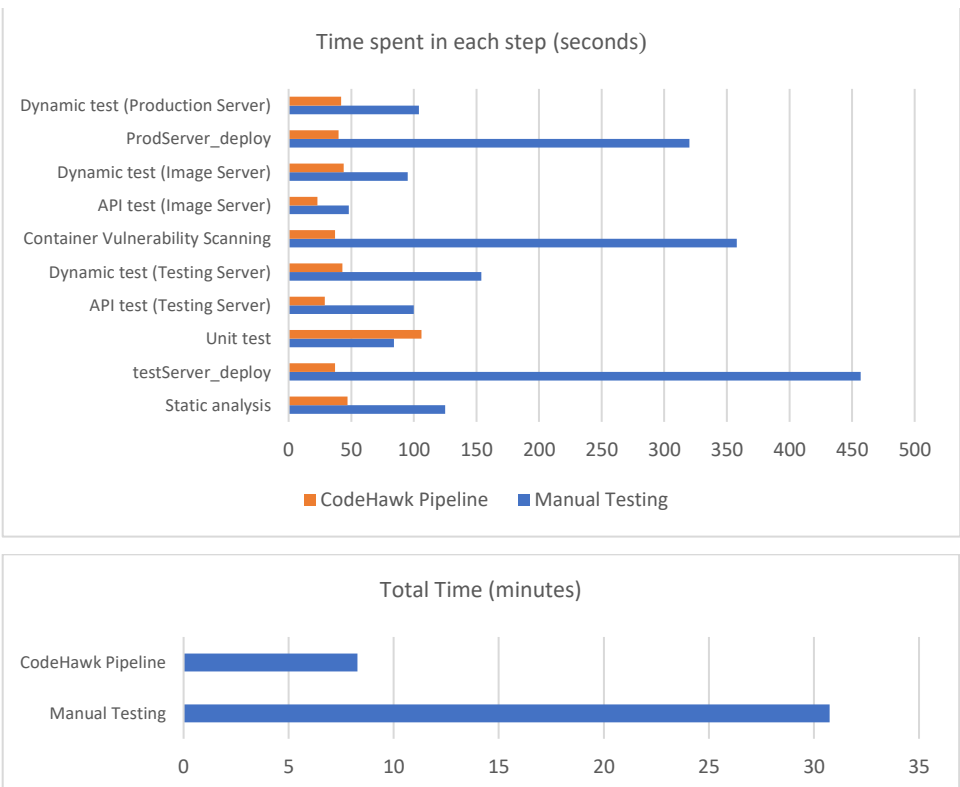

Fig. 8. Time comparison between manual testing and CodeHawk pipeline.

6. CONCLUSION

DevSecOps is an extension of DevOps that aims to automate security management, integrate security into the testing process, make the rapid development iteration more robust, and make automation no longer criticized for security issues.

This study develops a microservices-based DevSecOps platform called CodeHawk by integrating various open-source software. The security requirements, pipeline design, architectural design, implementation, and benefits of the CodeHawk. The proposed CodeHawk pipeline and platform allow testing throughout the project's lifecycle in different stages and environments by adding secret management, unit testing, API testing, static analysis, and dynamic testing to continuous testing, continuous delivery, and continuous deployment. Only grant commits with the required code quality can advance to the next stage. It also can detect and halt unworthy deployment. Moreover, testing results will be collected into a unique project directory for users to access. Experiment results show that the proposed pipeline shortens the testing time compared to manual testing. Compared to the maturity model proposed by Silkin [6], the proposed DevSecOps pipeline has achieved an advanced level which is valuable for researchers or engineers working on establishing DevSecOps infrastructure. Since the tools utilized by the CodeHawk platform are not load-balanced, future work would delve into performance issues by applying container orchestration mechanisms.

REFERENCES

1. R. Kumar and R. Goyal, "Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC)," *Computers & Security*, Vol. 97, 2020, p. 101967.
2. G. Silkin, O. Bodrov, G. Ovechkin, I. Bodrova, and S. Baranova, "Continuous testing maturity model," in *Proceedings of the 10th Mediterranean Conference on Embedded Computing*, 2021, pp. 1-4.
3. "IEEE standard for DevOps: Building reliable and secure systems including application build, package, and deployment," *IEEE Std 2675-2021*, 2021, pp. 1-91.
4. T. Rangnau, R. Buijtenen, F. Fransen, and F. Turkmen, "Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines," in *Proceedings of IEEE 24th International Enterprise Distributed Object Computing Conference*, 2020, pp. 145-154.
5. F. Angermeir, M. Voggenreiter, F. Moyón, and D. Méndez, "Enterprise-driven open source software: A case study on security automation," in *Proceedings of IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice*, 2021, pp. 278-287.
6. R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," *Information and Software Technology*, Vol. 141, 2022, p. 106700.
7. S. Dupont *et al.*, "Incremental common criteria certification processes using DevSec Ops practices," in *Proceedings of IEEE European Symposium on Security and Privacy Workshops*, 2021, pp. 12-23.
8. A. Pereira-Vale, G. Márquez, H. Astudillo, and E. B. Fernandez, "Security mechanisms used in microservices-based systems: A systematic mapping," in *Proceedings*

of XLV Latin American Computing Conference, 2019, pp. 01-10.
9. A. Rahman, F. L. Barsha, and P. Morrison, "Shhh!: 12 practices for secret management in infrastructure as code," in *Proceedings of IEEE Secure Development Conference*, 2021, pp. 56-62.
10. M. Raza, "Public vs private vs hybrid: Cloud differences explained," *BMC Blogs V*, 2023.
11. S. Throner *et al.*, "An advanced DevOps environment for microservice-based applications," in *Proceedings of IEEE International Conference on Service-Oriented System Engineering*, 2021, pp. 134-143.
12. Aqua Security, "Trivy," https://github.com/aquasecurity/trivy, 2023.
13. GitLab BV., "GitLab," https://gitlab.com/gitlab-org/gitlab, 2023.
14. Sonar Source, "SonarQube," https://www.sonarsource.com/products/sonarqube/, 2023.
15. Sonartype, "Sonatype nexus repository," https://www.sonatype.com/new/products/nexus-repository, 2023.
16. ZAP Dev Team, "OWASP zed attack proxy (ZAP)," https://www.zaproxy.org/, 2023.
17. Hashicorp, "Hashicorp vault," https://www.vaultproject.io/, 2023.
18. RedHat, "Keycloak," https://www.keycloak.org/, 2023.
19. WeKan Team, "WeKan – Open source Kanban," https://wekan.github.io/, 2023.
20. D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, Vol. 2014, 2014, No. 239.
21. The Kubernetes Authors, "Kubernetes," https://kubernetes.io/, 2023.
22. Elasticsearch B.V., "ELK stack," https://www.elastic.co/what-is/elk-stack, 2023.
23. The OWASP Foundation, "OWASP dependency-check," https://owasp.org/www-project-dependency-check/, 2023.
24. Postman, Inc., "Postman," https://www.postman.com/, 2023.

Wen-Tin Lee (李文廷) received his Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2008. Lee is currently an Associate Professor and Chair in the Department of Software Engineering and Management at National Kaohsiung Normal University. His research interests include software engineering, service-oriented computing, and deep learning.

Zhun-Wei Liu (劉峻維) is currently a master student in the Department of Software Engineering and Management at National Kaohsiung Normal University, Taiwan. His research interests include software engineering, microservice architecture, DevOps, web programming, and container technology.