CLIENT NAME
Computer Science Department
Swinburne Vietnam Alliance Program HCMC

Final Report and Implementation Strategy

Secure CI/CD Pipeline for Application Development

Version 1.0

Prepared by: GROUP 1

- Dang Vi Luan - 103802759
- Nguyen Linh Dan – 103488557
- Nguyen Duy Khang - 104056476
- Tran Bao Huy - 103505799

TABLE OF CONTENTS

Modification History

Table 1.0 Modification History

Date	Modifications	Reason	Version
26/07/2024	Initial version	Nil.	1.0

EXECUTIVE SUMMARY

The Computer Science department of Swinburne Vietnam Alliance Program at HCMC, now referred to as the Client, is an upcoming educational institution specializing in various domains, namely software development, data science, and cloud computing. The client was implementing the traditional waterfall software development methodology for developing and managing students' software projects. This posed significant challenges of reduced efficiency, limited flexibility, and collaboration, especially in complicated projects. Additionally, the students were not able to receive timely and constant feedback from the lecturers until the end of the project, causing difficulties in problem fixing or quality improvement. The conventional waterfall methodology prevents developers from developing and merging individual codes to the shared repository securely and effectively.

The scope of the project is categorized into functional and non-functional requirements. For functional requirements, our team would perform the implementation of a complete DevOps CI/CD pipeline that satisfied all clients' requirements and integrate DevSecOps features to enhance the security level of the pipeline. For non-functional requirements, our team would ensure the pipeline's performance was optimized with scalability and minimal latency. Overall, the final solution needs to achieve a high level of reliability, integrability, and observability besides satisfying the functional requirements.

By the completion of the project, our team has determined certain important findings. The CI/CD pipeline was able to satisfactorily meet all functional and non-functional requirements, improving the efficiency of students' projects by 50 to 70%. The improvement percentage is recorded when we tested to implement the built CI/CD pipeline to support a group of Swinburne students to deploy their project. However, to effectively and securely utilize the CI/CD pipeline, students need to be introduced and trained on the concepts of Agile methodology and CI/CD. Students without any prior IT experience, especially experience in DevOps, may find it challenging to apply to our CI/CD pipeline. To avoid the initial confusion, our team has prepared and provided the client with sufficient user documentation and instructions to set up and use the supporting tools integrated with the CI/CD pipeline.

Our team recommends that the Computer Science department at Swinburne - our client, should invest in hosting more workshops and training sessions on Agile methodologies and CI/CD implementation. This will be beneficial for the students to experience the most updated and widely used methodologies in the real industry, improving career readiness and work experience for Swinburne's students. Moreover, our team proposed several areas of improvement regarding the design, operation, and compliance of the current pipeline for the client in this report.

1. INTRODUCTION

1.1. Document overview

This document is the final report of the CI/CD pipeline project for the Computer Science department of the Swinburne Vietnam Alliance Program at HCMC.

The body of the document is structured into the following sections:

- Revisit the client's background and analyze the project inputs.
- Go through the development process and the applied methodology.
- Summarize all artifacts created throughout the project.
- Consider the change management and further improvements to the project.

1.2. Document objectives

The scope of the final report is to provide a thorough and complete overview of the project, including the client's situation, the analysis of users and the proposed solution, requirements, scope, methodology, final project outcome, and lessons learned. The final report does not include the full technical documentation or additional documents such as user documentation or usability testing, each is written in a separate document and attached in the Appendix.

1.3. About the project

The project is the implementation of a secured CI/CD pipeline with DevSecOps integration for the Client. Our team has successfully delivered all required deliverables from the complete pipeline to additional training documents. The main goal of this project is to create a robust and efficient system that addresses the challenges faced by Swinburne IT students in managing their source code manually. By implementing the CI/CD pipeline into a university department, we can create a more practical academic environment for Swinburne IT students to prepare for their future careers in professional IT businesses.

1.4. Document scope

The scope of the final report is to provide a thorough and complete overview of the project, including requirements, scope, methodology, what's achieved, and lessons learned. The final report does not include the details of the final solution package, technical documentation, or additional documents such as user documentation or usability testing.

1.5. Related documents

The related documents included in our final project package are as follows:

- Usability testing documentation
- User Manual
- Technical Solution
- Project Poster
- Client Project Sign-off
- Final Project Report (with Implementation Strategy described) [this report]

Please refer to these documents on Canvas or refer to the Appendix for more details.

1.6. Terminology

Please refer to Appendix 4 for a terminology glossary used in this report, and in the whole project.

2. PROJECT OVERVIEW

2.1. Client background

Established in 2021, Swinburne Vietnam Alliance Program - Ho Chi Minh Campus is a young and energetic educational institution. They have quickly proved themselves as a leading provider of high-quality international university programs in Vietnam. In 2019, they came to Hanoi for the first time with the first branch of a reputational Australian university. Swinburne Vietnam has placed a strong emphasis on motivating the development of the information technology and computer science fields.

In this project, we mainly focus on delivering high-quality outcomes to the Computer Science department at Swinburne Vietnam. Drawing on the reputation and resources of its parent institution, Swinburne Australia, the Computer Science department offers a wide range of bachelor programs in Vietnam to serve the evolving needs of the technology industry. The department's curriculum commits to providing students with a comprehensive education program that blends theoretical knowledge with practical projects. All courses cover new technologies and tools in different domains, such as software development, data science, and cloud computing. In the computer science department, students benefit from the guidance of IT professionals and improve their soft skills like problem-solving and collaborative spirit, qualities that the headhunters highly appreciate.

2.2. Client's current situation

The Computer Science department at Swinburne Vietnam (Ho Chi Minh City campus) is at the forefront of technological education and offers a learning curriculum of both foundational theory and hands-on practical labs. Students enrolled in the IT courses will have a chance to experience several software, tools, and programming languages, which will provide them with new skills and knowledge to prepare for the working environment.

A key component of the department's teaching approach is that they pay more attention to project-based learning than to theory alone. In the assignments, students are required to form a group and collaborate to develop real-world applications or websites. This collaborative approach not only fosters the development of technical expertise but also sharpens soft skills such as communication, problem-solving, and teamwork, which can help attract the attention of employers.

However, while the department's teaching style and lecture modules are innovative, students in the Computer Science program have faced challenges when it comes to applying the principles of the Software Development Life Cycle (SDLC) in their collaborative projects. Although the theory of SDLC has been taught in many courses, students find it difficult to integrate these methodologies into their team projects. As a result, students often rely on traditional and manual approaches to coding, merging code, and testing. Even this approach can lead to increased errors, inefficiencies, and inconvenience. The disconnect between the theory and practical application of SDLC is preventing students from fully understanding the development cycle and transferring the schoolwork into real-life experience.

To improve the current situation, the Computer Science department at Swinburne Vietnam (Ho Chi Minh campus) must explore new ways to effectively apply SDLC in students' projects to automate deployment, test processes in a more secure manner, and centralize source code management.

2.3. Project objectives

Per discussion with the client, our team has defined the following objectives for our CI/CD pipeline project:

Design the complete CI/CD pipeline

- Utilize the team's technical expertise and detailed discussions with the client to produce a CI/CD pipeline design that satisfies all the client's requirements.

Prepare infrastructure for the CI/CD pipeline

- Set up and configure the required tools and infrastructure such as GitHub, Terraform, Docker Hub, SonarQube, ArgoCD, Kubernetes, and AWS services to fully support the CI/CD pipeline.

Implement the CI/CD pipeline

- Develop and implement the actual solution into the prepared infrastructure to ensure a smooth operation as intended in the planning.

Perform testing after implementation

- Conduct technical testing and user acceptance testing (UAT) to ensure the system works as intended and the user experience quality is satisfactory.

Prepare User Guide and Training Documents

- Create comprehensive user guides and training materials to help the students get familiar with the Agile application development methodology and technical aspects of the CI/CD pipeline.

Gather feedback from the client

- Collect feedback from lecturers and professors to evaluate the effectiveness of the system and implement any additional changes if needed.

2.4. Solution characteristics

Automated CI/CD Pipeline

- Automates deployment process
- Reduces human error
- Ensures consistent and reproducible builds
- Integrates security practices

Five main components

- CI Pipeline
- CD Pipeline
- Monitoring Feature
- Infrastructure Provisioning Process
- Security Implementation

CI pipeline features

- Triggered by code commits to GitHub
- Utilizes GitHub Actions for automation
- Includes Software Composition Analysis
- Performs automated testing
- Containerizes code into Docker images
- Updates Kubernetes manifest files
- Pushes container images to Docker Hub

CD pipeline features

- Uses ArgoCD for deployment
- Tracks changes in Kubernetes manifest files
- Deploys to both local and remote (AWS EKS) environments
- Automatically reflects changes on worker nodes

Infrastructure provisioning

- Employs GitOps approach
- Uses Terraform for infrastructure as code
- Automates AWS infrastructure provisioning

Monitoring

- Integrates Prometheus and Grafana
- Provides real-time monitoring and visualization
- Enables proactive issue detection and analytics

Security implementation

- Credentials Security using GitHub Secrets
- Static Application Security Testing (SAST) using SonarQube
- Dynamic Application Security Testing (DAST) using OWASP ZAP

Web interfaces

- ArgoCD Web UI for application management and syncing
- Grafana Web UI for monitoring and visualization

Flexibility & Scalability

- Supports both local and remote environments
- Allows developers to use local environments for feature branches
- Uses remote server for main branch changes
- Utilizes AWS Elastic Kubernetes Service (EKS) for the remote environment

These characteristics collectively create a comprehensive, secure, and efficient CI/CD solution that automates deployment, ensures security, and provides robust monitoring capabilities. These features will not only facilitate the deployment of IT students' projects but also work as a basic prototype of the CI/CD pipeline in real companies.

2.5. User characteristics

2.5.1. General user characteristics

The target users of the proposed solution to address the challenge in the Computer Science department are the IT students. These students have several common characteristics and requirements that should be considered when developing the solution - a secure CI/CD pipeline, to support their collaborative coding projects at the university. To summarize the characteristics of the end-users, they are highly ambitious IT students with a strong desire to develop the practical skills and experiences that can help them have more opportunities in the competitive IT industry. The IT students are eager to apply the theoretical concepts they learned in the classroom to practical labs and projects. They are constantly looking for ways to improve their technical expertise and problem-solving abilities.

However, as described in the client's background section, these IT students are encountering a significant challenge when working on their coding projects. This challenge is preventing them from demonstrating

their excellent collaborative abilities and software development knowledge. Therefore, IT students need an innovative solution that can help them more effectively manage their deployment process based on the SDLC. This would reduce the manual effort and potential bugs that often arise in their team-based projects.

In the project, our team understood the demands of IT students at Swinburne Vietnam's Ho Chi Minh City campus to improve the quality of their deployment process, code management, and overall development workflow, so we decided that a secure CI/CD pipeline built using a DevSecOps methodology would be the best solution to address their needs. The IT students have expressed their needs for tools and workflows that can:

- Automate certain stages of the deployment process, such as building, testing, merging code, and packaging their projects.
- Manage the source code and encourage team collaboration.
- Provide a more transparent and visible view into the development process that allows team members to track progress and identify issues on time.
- Set up quickly and support the project development process. IT students appreciate a simple tool that allows them to focus on the core features of their projects rather than dealing with complex setup instructions.

In summary, our project outcome met the requirements of our end-users, the IT students. We also applied the developed CI/CD pipeline to a real project developed by a group of four students on campus. As a result, we can use the CI/CD pipeline to successfully manage the deployment of their project.

2.5.2. User Characteristics for Secure CI/CD Pipeline Access

Analyzing and understanding what the end-users know, who they are, and what their needs are, is the first step to ensuring we can choose the appropriate tools to develop a project solution that will be comfortable and friendly for the end-users.

- Technical level

Most IT students understand software development tools and programming languages. However, they do not focus much on the package deployment and release processes. For this reason, our team decided to develop the CI/CD pipeline using the DevSecOps methodology. The goal was not only to support students to learn about the deployment via their projects but also to help the students better protect their projects. We used ArgoCD and Docker to support the deployment. ArgoCD provides a user-friendly interface and additional security controls, which allows IT students with less command-line experience to use the pipeline effectively.

- Collaborative nature

The students often work in teams on their coding projects; they need secure access and permission management within the CI/CD pipeline. Therefore, our team chose to manage the source code with a GitHub repository and use GitHub Secrets to store the credentials. When team members commit their code, the CI/CD pipeline will automatically synchronize the changes with the current codebase and reflect them in the project's code repository.

- Security Awareness

While IT students are skilled in software and tools, they may not have a deep understanding of security. Therefore, our team integrated the pipeline with additional security measures to ensure the students' projects

are protected. This includes features like scanning the source code with SonarQube, security scanning with OWASP ZAP, using Prometheus and Grafana to monitor the pipeline's performance, and sending alerts when any anomalies are identified.

- Consistency needs

Most IT students prefer working on a consistent system with clear workflows instead of a system with frequent changes or disruptions. Before deployment, the project's source code is packaged into a Docker image file and then deployed to a Kubernetes cluster. This approach allows the IT students to bundle all dependencies and source code together in a standardized format to ensure a consistent deployment process. Furthermore, a load balancer has been integrated into ArgoCD to maintain high availability as the number of student projects using the CI/CD pipeline grows.

2.6. Beneficial aspects

Automation and Security Integration:

- Automates deployment, reducing errors and manual work
- Embeds security practices throughout the pipeline

Flexible environment support:

- Accommodates both local and cloud-based (AWS EKS) deployments
- Enables feature branch development locally and main branch changes remotely

Comprehensive monitoring:

- Utilizes Prometheus and Grafana for real-time insights and proactive issue detection

Infrastructure as Code and GitOps:

- Manages infrastructure with Terraform, enabling version control
- Implements GitOps for streamlined operations and collaboration

Containerization and Orchestration:

- Uses Docker for consistent application packaging
- Leverages Kubernetes for scalable container management

Continuous Integration and Delivery:

- Implements full CI/CD pipeline for rapid, frequent, and reliable deployments

Visualization and Management Tools:

- Provides intuitive interfaces (ArgoCD, Grafana) for deployment tracking and monitoring

2.7. General constraints

Budget limitations:

- No initial budget allocated
- Prioritization of free, open-source tools
- Potential compliance issues in later stages due to tool selection

Tight timeframe:

- 12-week project duration
- Requires efficient project management and resource allocation

Limited resource availability:

- Team skilled in solution architecture, DevOps, security, and auditing
- Lack of dedicated developers for web or desktop applications
- May require additional human resources consultancy

Infrastructure constraints:

- Swinburne's Computer Science department lacks established infrastructure
- Absence of specific regulations and documentation for student projects
- Heavy reliance on team assumptions and experience

Compliance uncertainty:

- Potential future challenges with regulatory requirements
- Possible need for specific tools or licenses not initially budgeted

3. DEVELOPMENT METHODOLOGY

3.1. Requirement summary

3.1.1. Functional requirements

A. DevOps CI/CD Pipeline

A1. Version Control Integration

- Integrate the codebase with a version control system using Git for managing source code changes and enabling developer collaboration.
- Set up repositories for the main application, associated services, and any necessary infrastructure code using GitHub, Terraform Registry, and DockerHub.

A2. Continuous Integration (CI)

- Implement automated build and testing processes to ensure codebase integrity.
- Use ArgoCD to automatically trigger builds upon code commits or merges.
- Incorporate unit tests, integration tests, and static code analysis to validate application functionality and code quality.
- Provide developers with feedback on the success or failure of build and test processes.

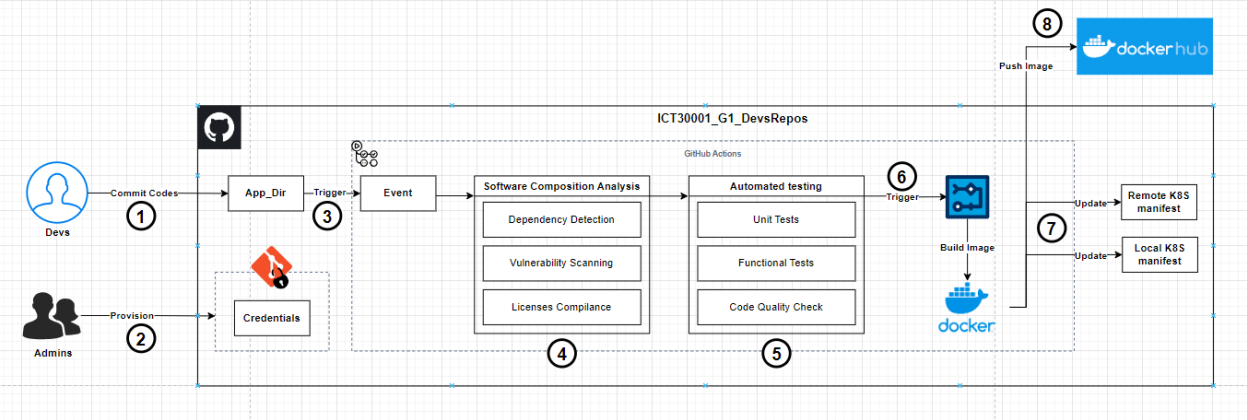

Figure 1: Overall CI flow

A3. Deployment

- Integrate Docker with the CI/CD pipeline to build and push container images to a registry, maintaining traceability through versioning.
- Implement Dockerfiles for services and components and configure the CI process to automatically build and publish Docker images.

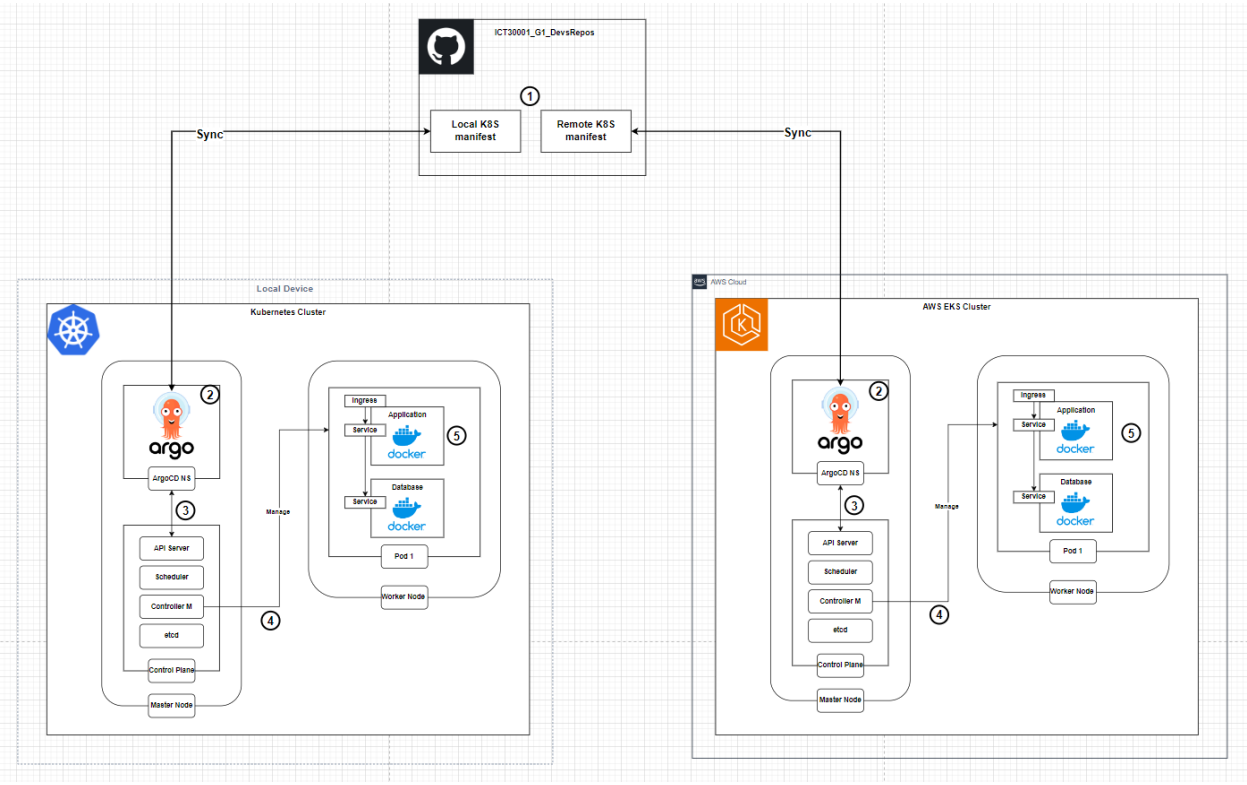

Figure 2: Overall CD flow

A4. Operation

- Use Terraform to provision required cloud infrastructure, integrating Terraform code into the CI/CD pipeline for consistent provisioning across environments.
- Deploy the containerized application to a Kubernetes cluster, automating Kubernetes manifest deployments as part of the CD process.

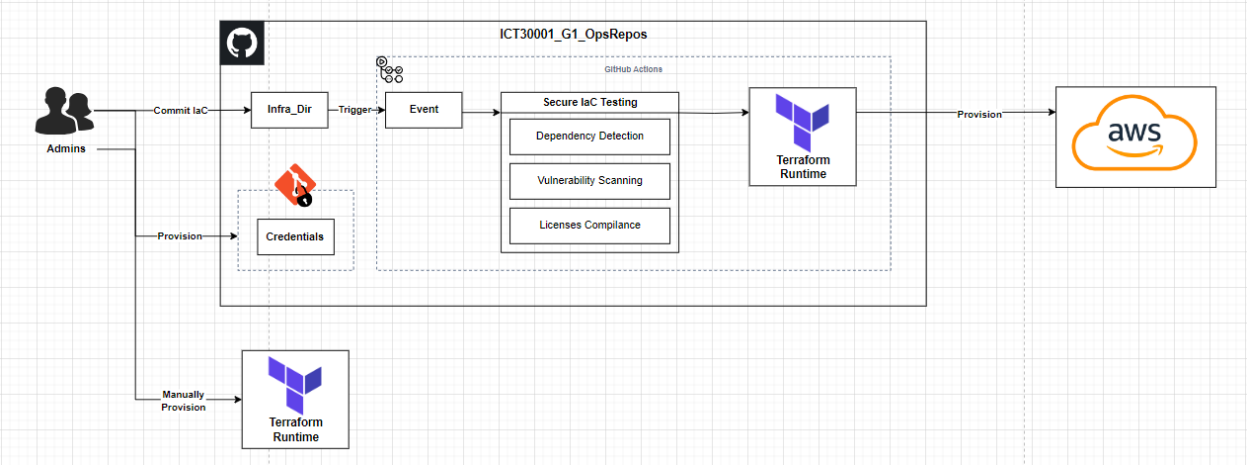

Figure 3: Infrastructure provisioning using Terraform

B. DevSecOps Integration

B1. Credentials security

Managing credentials securely is crucial to protecting sensitive information such as API keys, passwords, and other secrets. Implementing tools like "Git Secrets" can prevent accidental commits of sensitive information by scanning for known secret patterns and blocking commits that contain them. This ensures that credentials are never exposed in version control repositories, reducing the risk of security breaches.

B2. Static Application Security Testing (SAST)

Static Application Security Testing (SAST) involves analyzing source code for security vulnerabilities before the code is compiled. Tools like SonarQube can be integrated into the CI/CD pipeline to perform automated static code analysis.

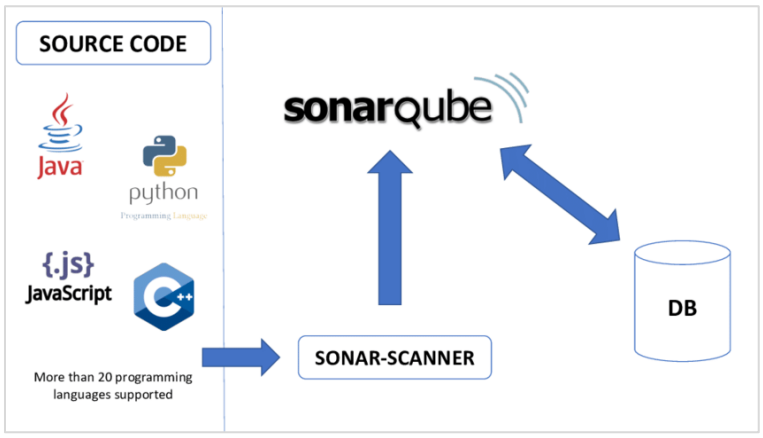

Figure 4: Sonarqube for SAST

SonarQube scans the codebase for common security flaws, such as SQL injection, cross-site scripting (XSS), and buffer overflows, providing detailed reports and recommendations for remediation.

B3. Dynamic Application Testing (DAST)

Dynamic Application Security Testing (DAST) involves testing running applications for security vulnerabilities by simulating external attacks.

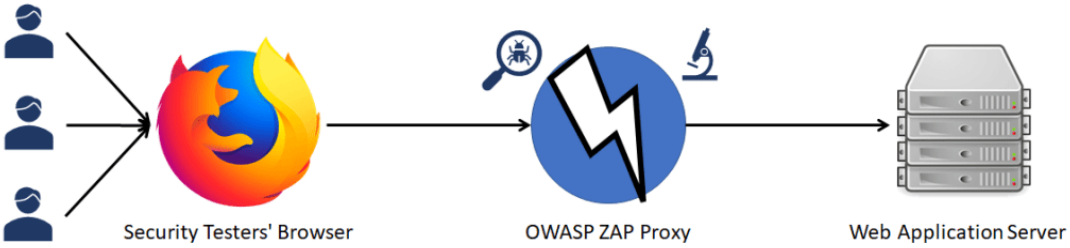

Figure 5: OWASP ZAP for DAST

OWASP ZAP (Zed Attack Proxy) is a powerful tool that can be used for DAST. It can be integrated into the CI/CD pipeline to automatically scan deployed applications for vulnerabilities, such as insecure configurations, weak authentication mechanisms, and potential entry points for attacks.

B4. Monitoring and Alerting

Effective monitoring and alerting are essential for maintaining the health and security of the CI/CD pipeline and the deployed applications.

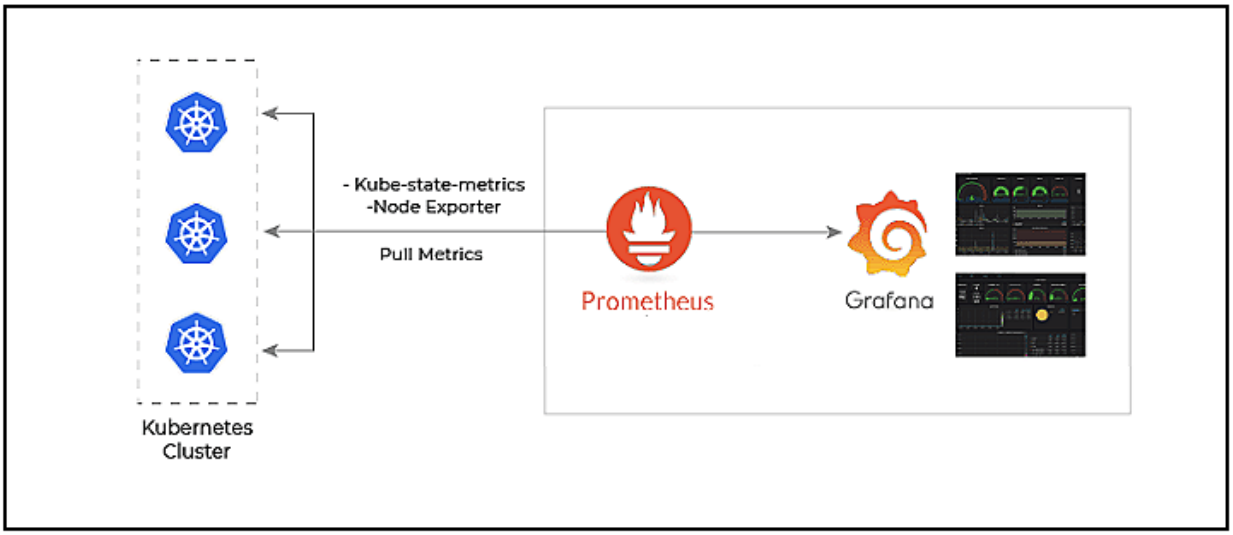

Figure 6: Prometheus and Grafana for monitoring

Prometheus and Grafana can be used to collect and visualize metrics from the application and infrastructure components. Prometheus scrapes metrics from various sources, providing real-time data on system performance, resource utilization, and application behavior. Grafana can then be used to create custom dashboards and visualizations with insights into the overall system health.

3.1.2. Non-functional requirements

Performance

- **Scalability**: The CI/CD pipeline must handle increasing workloads as the number of developers and the size of the codebase grows, supporting horizontal scaling as needed.
- **Throughput**: The pipeline should efficiently process a high volume of builds, tests, and deployments, minimizing delays and bottlenecks.
- **Latency**: Ensure minimal latency in the build, test, and deployment stages to provide quick feedback to developers and reduce the overall development cycle time.

Reliability

- **Availability**: The CI/CD pipeline should be highly available with minimal downtime, implementing redundancy and failover mechanisms for continuous operation.
- **Fault Tolerance**: Design the pipeline to tolerate failures and recover gracefully without data loss or significant service interruption.

Integrability

- **Tool Compatibility**: Ensure compatibility with various development tools and platforms, including version control systems (e.g., GitHub), CI tools (e.g., ArgoCD), and container registries (e.g., DockerHub).

Observability

- **Metrics Collection**: Use Prometheus to collect detailed metrics on pipeline performance, application health, and infrastructure status.
- **Visualization**: Integrate Grafana for creating visual dashboards to monitor metrics and system health in real time.
- **Alerting**: Configure Prometheus and integrate with Telegram for real-time alerting on predefined thresholds and anomalous behaviors to ensure timely incident response.

3.1.3. Out of scope

- Training for cloud services: Knowledge transfer and documentation on using the CI/CD pipeline will be provided, but extensive training related to cloud services and additional features on integrated services (e.g., advanced AWS features, in-depth Terraform training) is not included.
- Marketing and User Adoption Plans: Developing and executing marketing strategies or user adoption plans to promote the new CI/CD pipeline within the organization is not included.
- Third-party tools and services: While various third-party tools (e.g., ArgoCD, GitHub, Docker, Terraform) will be integrated, ongoing management, maintenance, and support for these tools are out of scope. This includes troubleshooting issues directly related to these tools and their updates.
- Bug fixing and additional functionalities: The project focuses solely on integrating and deploying applications via the CI/CD pipeline. Bug fixing or development of additional functionalities for the applications themselves are out of scope, including addressing application code bugs and adding new features.

3.2. Description of the development process

The development process for the CI/CD pipeline involves several key stages, each designed to ensure the final solution is robust, secure, efficient, and user-friendly. Below is a overview of each stage:

3.2.1. Project proposal preparation

- Stakeholder meeting: Engage with stakeholders to understand their needs and expectations.
- Requirement analysis: Document the functional and non-functional requirements for the CI/CD pipeline.
- Project plan: Develop a detailed project plan, including timelines, milestones, and resource allocation.
- Proposal document: Create a comprehensive project proposal that outlines the project's purpose, scope, deliverables, and expected outcomes.

Please refer to Project Proposal in Appendix 1 for a complete project proposal.

3.2.2. The CI/CD development process

The CI/CD development process is divided into several sub-stages to ensure a systematic approach:

Version Control Integration:

- Set up version control systems using Git.
- Create repositories for the main application, associated services, and infrastructure code on platforms like GitHub, Terraform Registry, and DockerHub.

Continuous Integration (CI):

- Implement automated build and testing processes.
- Use ArgoCD for triggering builds upon code commits or merges.
- Incorporate unit tests, integration tests, and static code analysis tools.

Deployment:

- Integrate Docker for building and pushing container images.
- Implement Dockerfiles for services and configure the CI process to automatically build and publish Docker images.

Operation:

- Use Terraform to provision cloud infrastructure.
- Deploy containerized applications to Kubernetes clusters, automating the deployment of Kubernetes manifests.

DevSecOps Integration:

- Implement security measures such as pre-commit hooks, AWS Secrets Manager, SonarQube, OWASP ZAP, and Docker image scanning tools.
- Establish monitoring and alerting systems using Prometheus, Grafana

Please refer to Technical Solution in the Appendix 2 for a detailed explanation of the CI/CD pipeline development process.

3.2.3. Progress review sessions

From the internal side, our team conducted sprint review sessions to ensure the project stayed on track and met its objectives. These sessions often cover the following contents:

- Status updates: Team members provide updates on their progress and any challenges faced.
- Milestone reviews: Evaluate the completion of key milestones and deliverables.
- Feedback sessions: Stakeholders and team members provide feedback and suggestions for improvement

For external project activities, we conducted a progress update meeting with the clients based on the Agile methodology. However, given the relatively low complexity of our project, we decided to perform a half-project review in week 6, instead of the typical sprint review with the clients. This decision was made due to the limited time availability of both the clients and our team. The update session with our clients covered the following topics:

- Features demo: We showed the functionality that had been developed up to that point.
- Update on the project timeline and backlog: We reviewed the current state of the project timeline and updated the project backlog accordingly.
- Q&A session: to clarify any questions from the clients and gather their feedback.
- Outlining the next steps: We discussed the upcoming backlog items that would be developed in the next phase of the project.

Please refer to the Progress Review report in Appendix 3 for the summary of our progress review session with clients.

3.2.4. Other documentation artifacts preparation

a. User manual

We wrote a user manual to guide end-users on how to set up and use the CI/CD pipeline to deploy their projects effectively. This manual includes:

- Introduction: Overview of the CI/CD pipeline and its components.
- Setup instructions: Step-by-step guide for setting up and configuring the pipeline.
- Tutorial videos: Detailed instructions on how to perform common tasks, such as committing code, triggering builds, and deploying applications.

Please refer to User manual in Appendix 4.

b. Usability testing report

Usability testing is conducted to ensure the CI/CD pipeline is user-friendly and meets the needs of its users. This involves:

- Test scenarios: Develop realistic scenarios to test various aspects of the pipeline.
- User feedback: Collect feedback from end-users on the ease of use, performance, and functionality.
- Improvements: Make necessary adjustments and improvements based on user feedback.

Please refer to Usability testing report in Appendix 5.

c. Business-related documentation

For the business-related documentation, we began with a comprehensive investigation of the business context and environment, analyzing current processes and conducting market research to benchmark against industry best practices. Initial meetings with key stakeholders, including project sponsors, business analysts, and end-users, were held to gather detailed requirements and understand expectations

Regular follow-up meetings throughout the project ensured ongoing stakeholder engagement and alignment with business goals. Based on the gathered information, we drafted a detailed project proposal outlining the project's purpose, scope, goals, deliverables, timelines, and resource allocation. As the project progressed, we developed regular progress reports providing updates on milestones, challenges faced, and steps taken to address them

Upon project completion, a comprehensive business report, including an executive summary, detailed findings, analysis, and recommendations, was documented. These business-related documents also included the internal reviews for accuracy and completeness, followed by stakeholder reviews for input and feedback.

Please refer to Business Plan and Communication Plan in Appendix 7 & 8.

d. Technical solution report

For the technical solution reports, the process began with team internal meetings, starting with a kick-off meeting to align on the project's technical goals, timelines, and individual responsibilities, followed by daily stand-ups to discuss progress and address blockers.

Solution architects in our team held architecture planning meetings to design the overall architecture of the CI/CD pipeline, making decisions on tools, technologies, and integration strategies. Regular design review sessions ensured that the solution was scalable, secure, and aligned with best practices.

Testing phases included automated unit and integration tests using CI tools, security testing with SonarQube and OWASP ZAP, and performance testing to ensure system efficiency under various conditions. Continuous integration was practiced, with code changes integrated and tested continuously, and automated deployment pipelines set up for consistent and repeatable deployments.

Detailed technical specifications were documented, including system architecture diagrams, component descriptions, and configuration details. Operational guides and security and monitoring documentation were created to assist the operations team. Peer reviews by team members ensured technical accuracy and completeness, and solution architects reviewed the documents to ensure alignment with overall architecture and design principles.

Please refer to Technical Solution report in Appendix 2.

e. Implementation strategy

The implementation strategy provides a comprehensive approach for deploying the CI/CD pipeline into the production environment. This strategy ensures that all aspects of the deployment are planned, executed, and reviewed to guarantee a seamless transition and effective operation.

The key components of this strategy include organizing stakeholder meetings, requirement validation, knowledge assessment, deployment planning, migration strategy, monitoring and support, and post-implementation review.

Step 1: Organize meeting with the stakeholders

The implementation process begins with a series of meetings with key stakeholders to:

- Understand expectations: Gather detailed information about their expectations, concerns, and specific requirements for the CI/CD pipeline.
- Define objectives: Clearly define the project's objectives, scope, and success criteria.
- Identify key roles: Identify the roles and responsibilities of all stakeholders, including project managers, developers, operations staff, and security teams.

Step 2: Requirement Validation

Next, a thorough validation of the requirements is conducted to ensure all needs are accurately captured:

- Review requirements: Re-examine all functional and non-functional requirements, including security, performance, reliability, and integrability aspects.
- Confirm resources: Ensure that all necessary resources, tools, and infrastructure components are available and meet the specified requirements.
- Identify gaps: Identify any gaps in the current setup that need to be addressed before deployment.

Step 3: Knowledge Assessment

Assess the current knowledge and skill levels of the development and operations staff:

- Skill evaluation: Evaluate the team's familiarity with CI/CD concepts, tools, and processes.
- Training needs: Identify any training needs and develop a plan to address them. This might include hands-on workshops, online courses, or documentation review.
- Knowledge transfer: Plan sessions for knowledge transfer and hands-on training to ensure that the team can effectively operate and maintain the CI/CD pipeline.

Step 4: Deployment Plan

Develop a detailed deployment plan to guide the implementation process:

- Timelines: Define clear timelines for each phase of the deployment, including preparation, execution, and review.
- Responsibilities: Assign specific responsibilities to team members, ensuring accountability for each task.
- Risk management: Identify potential risks and develop mitigation strategies to address them.

Step 5: Migration Strategy

Outline the steps for migrating existing projects and codebases to the new CI/CD pipeline:

- Assessment: Evaluate the current state of projects and codebases to determine migration requirements.
- Planning: Develop a migration plan that includes a step-by-step process for moving projects to the new pipeline, minimizing downtime and disruptions.
- Execution: Implement the migration plan, ensuring thorough testing and validation at each stage.

Step 6: Monitoring and Support

Establish robust monitoring processes and support mechanisms to ensure smooth operation post-deployment:

- Monitoring setup: Configure monitoring tools like Prometheus and Grafana to track the performance and health of the pipeline and applications.
- Alerting: Set up alerting systems to notify the team of any issues or anomalies, enabling prompt response.
- Support mechanisms: Provide ongoing support through knowledge sharing sessions, user manuals, and a helpdesk to address any issues that arise.

Step 7: Knowledge Sharing and User Manual

After deployment, ensure that all users are well-informed and capable of using the new CI/CD pipeline:

- User manual: Develop a comprehensive user manual that includes setup instructions, usage guidelines, troubleshooting tips, and best practices.
- Training sessions: Conduct training sessions and workshops to familiarize users with the new system.
- Continuous support: Offer continuous support and updates to ensure users remain proficient and confident in using the pipeline.

Step 8: Post-Implementation Review

Conduct a thorough review after implementation to assess the performance and gather feedback:

- Review session: Hold a review session one sprint after deployment to evaluate the success of the implementation.
- Performance assessment: Analyze the performance of the CI/CD pipeline, identifying any issues or areas for improvement.
- Feedback collection: Gather feedback from stakeholders and users to understand their experiences and gather suggestions for further enhancements.
- Future improvements: Use the feedback to plan future improvements and optimizations to the CI/CD pipeline.

4. ACCEPTANCE CRITERIA

The acceptance criteria for the CI/CD pipeline project must ensure that the developed CI/CD pipeline is robust, secure, efficient, and user-friendly.

About the functional requirements, the final CI/CD pipeline will pass the acceptance criteria if:

- It integrates with Git, GitHub, Terraform Registry, and DockerHub, automates builds and tests via ArgoCD, uses Docker for container management, and employs Terraform and Kubernetes for cloud infrastructure and deployments.
- The final pipeline is secured with DevSecOps measures, including git-secrets, AWS Secrets Manager, SonarQube, OWASP ZAP, and DockerHub image scanning.
- The pipeline performance is monitored via Prometheus, Grafana.

About the non-functional requirements, the final CI/CD pipeline will pass the acceptance criteria if it ensures:

- Scalability
- Minimal latency
- High availability

- Fault tolerance
- Compatibility with development tools.
- Observability is ensured through Prometheus and Grafana.

About the documentation, at least the following documents must be covered:

- Usability testing report
- User manual
- Technical solution report
- Business plan
- Implementation strategy with a post-implementation review

5. FUTURE CONSIDERATIONS

5.1. Further recommendations

To enhance the CI/CD pipeline, we recommend full integration on the cloud using AWS services. Specifically, utilizing AWS Elastic Kubernetes Service (EKS) for managing Kubernetes clusters and AWS Elastic Container Registry (ECR) for storing and managing Docker container images will provide robust scalability, security, and ease of management. This transition will streamline operations and leverage AWS's powerful cloud infrastructure for better performance and reliability.

5.2. Change management

- **For changes belong to the deploying projects (technical changes):**

It is recommended that implementing a professional CI engine like Jenkins or Harness can significantly improve the efficiency and capabilities of the CI/CD pipeline. Jenkins, with its extensive plugin ecosystem, and Harness, with its focus on continuous delivery and deployment, offer advanced features that can automate and optimize the build, test, and deployment processes. Managing these changes will involve careful planning, including evaluating the current system, defining migration strategies, training the team, and ensuring seamless integration with existing workflows.

- **For changes belong to the CI/CD pipeline improvements:**

Before making any changes to the CI/CD pipeline, you should have a plan to backup the existing data and configurations. Having a backup will allow you to revert to the previous state if needed, minimizing the risk of disruption. The backup items include configuration files, scripts, and any other critical assets that are part of the current pipeline.

All changes made to the CI/CD pipeline should be documented with:

- Details of the changes
- The reasons for the changes
- The testing result
- The rollback plan

+ <u>For minor changes:</u>

- Ensure that the changes are thoroughly tested in a non-production environment before applying them to the live pipeline.
- Communicate the changes to the relevant teams and have a rollback plan in case any issues arise.

+ For major changes:

- The business *(Computer Science department in the current context)* should have a Business Continuity Plan.
- BCP should outline the steps to be taken to ensure that the deployment process is not disrupted, or when the pipeline changes take many hours to complete.
- BCP also includes details on how to maintain the availability of the existing pipeline during the update process, and a clear rollback strategy in case of any issues.

5.3. Process improvement

To maintain and enhance the effectiveness of the CI/CD pipeline, further training for operations staff is recommended. This training should focus on new tools and processes introduced, ensuring the team is proficient in managing and troubleshooting the pipeline. Moreover, promoting knowledge sharing between operations staff and developers will foster a collaborative environment, enhancing the overall efficiency and innovation within the team. Regular workshops, documentation updates, and knowledge transfer sessions should be conducted to keep everyone aligned and informed.

5.4. Design improvements

Several design improvements can be made to optimize the CI/CD pipeline in the future:

- Modular architecture: Break down the CI/CD pipeline into modular components that can be independently updated and scaled. This will enhance maintainability and flexibility, allowing individual parts of the pipeline to evolve without disrupting the entire system.
- Pipeline as code: Implement the CI/CD pipeline configuration as code using tools like Jenkins Pipeline or Harness Configuration-as-Code (CaC). This approach ensures consistency, version control, and easier collaboration on pipeline configurations.
- Automated rollbacks: Introduce automated rollback mechanisms to quickly revert to the last known good state in case of deployment failures. This will minimize downtime and ensure business continuity.
- Enhanced security measures: Integrate additional security checks and balances, such as automated vulnerability scanning, security compliance audits, and encryption for data in transit and at rest, to strengthen the pipeline's security posture.
- Scalability enhancements: Design the pipeline to dynamically scale based on workload demands, leveraging AWS's auto-scaling features to ensure optimal resource utilization and cost efficiency.

6. CONCLUSION

The comprehensive final project report has provided an overview of the various aspects involved in the development of the Secure CI/CD Pipeline, which was created to support the deployment of IT student projects at the Swinburne Vietnam Alliance Program's Ho Chi Minh City campus. We have gone through the client's background, revisited their current challenges, and reviewed the project objectives. The end-user characteristics were also analyzed to better understand the demands of the target audience. The main body of this report provides a detailed explanation of our development methodology and process. Moreover, other necessary artifacts, such as user documentation, testing reports, plans, and strategies, are also covered. Our team has included a section on further considerations to discuss potential improvements that can be implemented in the CI/CD pipeline. This reflects our belief that technology should always strive for innovation.

By implementing the CI/CD pipeline solution, the report has successfully demonstrated how the proposed recommendations can effectively address and solve the main challenges faced by IT students. The consistent and reliable system with containerization and Kubernetes-based deployment will provide a secure environment for the students to focus on their projects and avoid having everything done separately or working with a decentralized development infrastructure. Overall, our CI/CD pipeline solution aligns closely with the needs and characteristics of IT students. The solution will empower IT students to develop and deploy their projects effectively throughout their academic journey at Swinburne Vietnam.

APPENDIX

1. Project Proposal
2. Technical Solution report
3. Progress Review report
4. User Manual
5. Usability Testing report
6. Project Poster
7. Business Plan
8. Communication Plan
9. CI/CD Pipeline Source Code