

TABLE OF CONTENT

PROGRESS REVIEW INTRODUCTION	2
PROJECT SUMMARY.....	2
Project Description.....	2
Project Background	2
Project Objectives	3
Scope.....	4
Functional requirements	4
Non-functional requirements	5
Out of scope.....	5
PROJECT PROGRESS REVIEW.....	5
Progress To Date	5
Completed Work	7
Prototype screenshots	10
Work left to complete the project	13
Deliverables.....	14
Questions from the client.....	14
Suggestions from client	15
APPENDIX.....	16

PROGRESS REVIEW INTRODUCTION

This document is the synthesis of the client progress review for Group 1's CI/CD Pipeline project for the CS department of Swinburne Vietnam Alliance Program at HCMC (referred from now as the Client). The client progress review process was conducted in a direct meeting between the project team and the client's representative – Mr. Pham Thai Ky Trung (lecturer at the Client). Within the client review, the progress made in the first 5 weeks of the project, the agreement of project's scope and deliverables, and the planning for the remaining 6 weeks, are thoroughly discussed, all of which will be included in this document.

PROJECT SUMMARY

Project Description

The solution we are developing for the Client is a CI/CD pipeline for application development. The pipeline provides services such as Version Control Integration, Continuous Integration, Deployment, DevSecOps Integration and several report/monitoring features. The solution consists of various tools and technologies such as GitHub, git-secret, AWS, Terraform, ArgoCD, and Docker to build a robust and comprehensive CI/CD solution. Additionally, aside from the infrastructure, a detailed user guide and training materials will be provided to facilitate lecturers and students in using the new system accurately and effectively.

Project Background

The client – Computer Science (refer to as CS) department of Swinburne Vietnam Alliance Program at HCMC is a prestigious educational institution providing CS and IT courses offered by Swinburne University of Technology in Australia to Vietnamese university students. The CS department consists of around 10 experienced lecturers and professors with a student population of around more than 100 students varying from freshman to senior level. The CS and IT study curriculum designed by Swinburne University of Technology is thoroughly designed to equip students with real – working experiences through multiple project – based classes, which allowing students to develop, test, and deploy applications.

At the moment, the project-based classes are delivered in accordance with the traditional waterfall project development methodology. The students will work in group or individually to develop functionalities, interfaces of the application throughout the duration of the semester and only perform testing and deployment during the final weeks. This project development methodology is suitable for small and simple projects and allows students more time to conduct research, study about the technology prior to deployment. However, there are various problems regarding efficiency, collaboration, management, and security with the current methodology used by the Client, especially when the student population increases.

Currently, there is no standardized framework and infrastructure to help students maintain and update their projects' codes. This poses a significant problem of inefficient code management in groups and complicated projects. Moreover, it is difficult for developers to collaborate and keep track of the project's progress without a shared repository and management platform. A more important problem of the current system is the lack of continuous development and integration.

With only one deployment at the end of the semester, the students will have difficulties in debugging and visualizing the application in real production environment. Resolving issues after deployment can be costly and incredibly difficult, leading to reduced deliverable quality. Additionally, the students will not be able to receive valuable and continuous feedback from the lecturers to update and improve their projects without continuous development and integration. Finally, as the student population expands and the number of projects increases, it is difficult for lecturers and the infrastructure to manage student's application projects effectively and securely.

Therefore, the Client has hired our team to deliver a CI/CD (Continuous Integration/Continuous Development) pipeline to resolve the current problems. When the pipeline is completely delivered to the Client, students will have a standardized guidance and infrastructure such as code repository, production environment resources, security features and project management features to help them conduct their projects in a more efficient way. The students can effectively monitor the workload and the committed code from each team member. Moreover, they have an opportunity to constantly improve and adjust their applications if needed based on the lecture's feedback and performance on the production environment. The performance of test cases will also be set up automatic to support students to not get stuck on infrastructure – related problems. With the CI/CD solution, students can fully utilize their time to focus on and apply the topics introduced in their class without wasting time and resources in setting up infrastructure and redundant project management tasks.

Project Objectives

Per discussion with the representative of the Client – Mr. Pham Thai Ky Trung, we have reached an agreement and decided to make ***no new change*** to the project's objectives defined in the Initial Project Plan. The detailed objective are as follows:

- **Design the complete CI/CD pipeline**

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

- **Prepare infrastructure for the CI/CD pipeline**

Prepare and configure the required tools and infrastructure such as GitHub, Terraform Registry, DockerHub, ArgoCD, Kubernetes, and AWS services to fully support the CI/CD pipeline.

- **Implement the CI/CD pipeline**

Perform development and implementation of the real solution into the prepared infrastructure to ensure a smooth operation as intended in the planning.

- **Perform testing after implementation**

Perform technical testing and user acceptance testing (UAT) to ensure the system works as intended and the user experience quality is ensured.

- **Prepare User Guide and Training Document**

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

Scope

The goal of this project is to design and implement a secure Continuous Integration and Continuous Deployment (CI/CD) pipeline for application development. The key components include:

Functional requirements

A. DevOps CI/CD Pipeline

1. Version Control Integration

- Integrate the codebase with a Git-based version control system like GitHub to manage code changes and enable developer collaboration.
- Set up repositories for the main application, associated services, and infrastructure code.

2. Continuous Integration (CI)

- Implement automated build and testing processes using a CI tool like Jenkins to validate code changes.
- Perform unit tests, integration tests, and static code analysis to ensure application functionality and code quality.
- Provide feedback to developers on build and test results.

3. Deployment

- Leverage Docker to build and publish container images to a registry like DockerHub.
- Automate the deployment of containerized services to a Kubernetes cluster using Terraform for consistent infrastructure provisioning.

4. Operations

- Use Terraform to provision the required cloud infrastructure as part of the CI/CD pipeline.
- Deploy the containerized application to a Kubernetes cluster, automating the deployment of Kubernetes manifests.

B. DevSecOps Integration

1. Version Control Integration

- Implement pre-commit hooks to prevent accidental commit of sensitive information.
- Integrate AWS Secrets Manager to manage access to credentials across services.

2. Continuous Integration

- Implement SAST (Static Application Security Testing) using Sonarqube to scan for security vulnerabilities.
- Leverage SCA (Source Composition Analysis) with GitHub to identify vulnerable libraries.
- Integrate OWASP ZAP to detect vulnerabilities in running applications.

3. Deployment

- Scan Docker images for vulnerabilities using tools like Clair to ensure container security.

4. Monitoring and Alerting

- Use Prometheus and Grafana for collecting, storing, and visualizing system metrics.
- Configure Prometheus to set up alerts based on predefined thresholds

- Integrate the alerting system with Telegram to notify the appropriate team members when issues arise.

Non-functional requirements

- Performance: includes high scalability, throughput, and minimal latency.
- Reliability: includes minimal downtime and the ability to tolerate failures
- Integrability: includes the compatibility with several tools and platforms
- Observability: includes the visualization of the pipeline's status and health metrics.

Out of scope

- Training for Cloud Services and Deployment Tools
- Marketing Plans and related Business Plans
- The maintenance, support, or management of third-party tools
- Further maintenance
- Additional functionalities (may require additional costs)

PROJECT PROGRESS REVIEW

Progress To Date

- Project Duration: From 06/05/2024 to 09/08/2024
- The project has been running for: 39 days
- Remaining: 55 days

Currently, our team has already completed the following phases of the project Gantt Chart plan.

- **Phase 1:** Project Preparation
- **Phase 2:** CI/CD Pipeline Design (we demonstrated several design diagrams in the review session with clients)

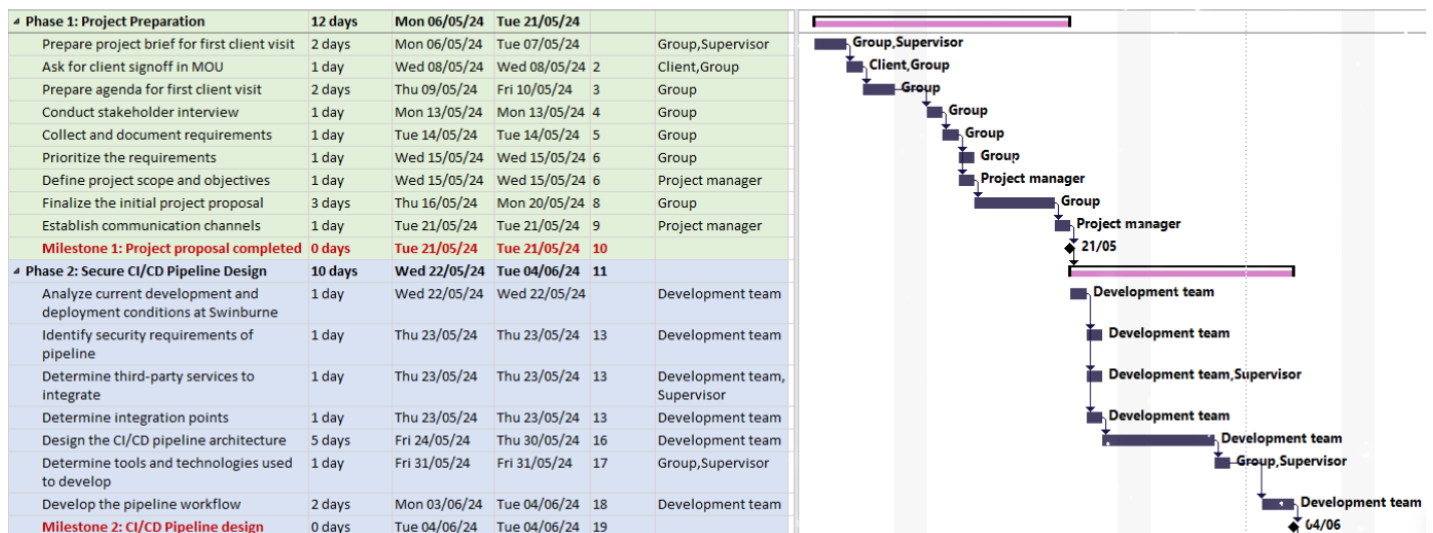


Figure 1: Gantt Chart – All tasks of Phase 1 and 2

We are performing tasks in Phase 3 - CI/CD pipeline development and have already finished Phase 4 – Interim Presentation (this Client Progress Review).

Initially, we planned to work on Phase 3 and Phase 4 in a sequence. However, as Phase 3 is the longest and most important phase, we could not finish it before the Interim presentation. We decided to modify our Gantt Chart to run Phases 3 and 4 simultaneously. First, we will work on Phase 3 for two weeks, then organize a presentation session to update clients' progress and gather their feedback to improve the product. After the Interim update, our team will continue with Phase 3 to finalize the CI/CD pipeline development.

Before rescheduling

Phase 3 (Development) → Phase 4 (Presentation)

After rescheduling

Phase 3 [2 weeks] → Phase 4 (Interim Presentation) → Phase 3 [2 weeks]

Phase 3 Updated Progress

Done

1. Setup the CI server
2. Integrate Code versioning management
3. Prepare deployment scripts
4. Implement automated build
5. All tasks of Phase 4 – Interim Presentation

Not Done

1. Implement automated testing process
2. Implement security measures

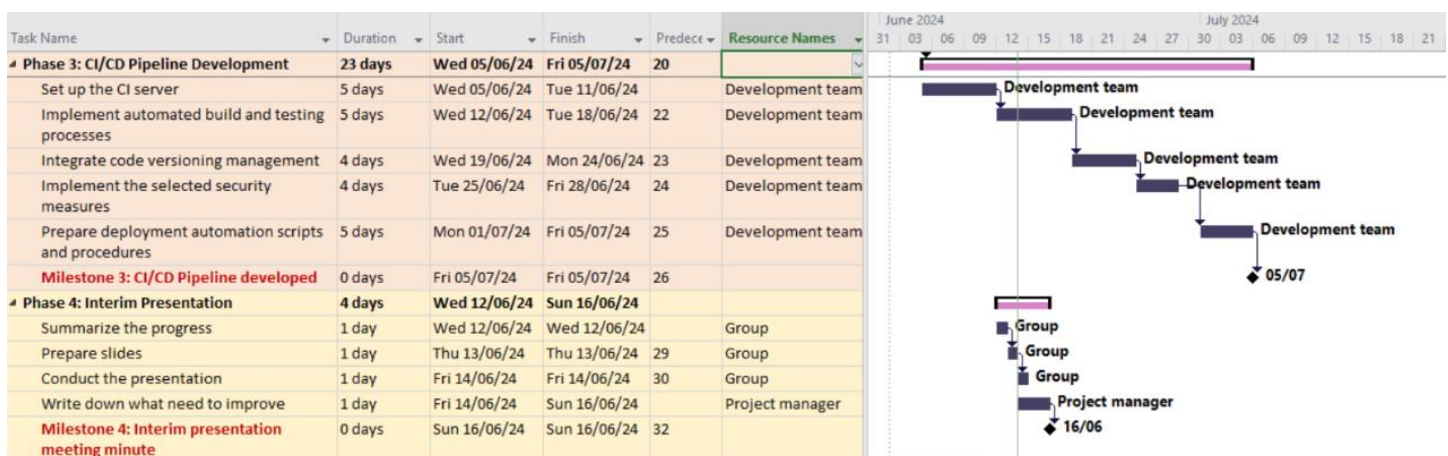


Figure 2: Gantt Chart – All tasks of Phase 3 and 4

Completed Work

We divide our CI/CD pipeline into 5 main parts: The Continuous Integration workflow, the Continuous Delivery workflow, The Infrastructure provisioning mechanism, the monitoring, alert, and visualization mechanism, and finally the Continuous Security Integration. Currently, we have finished our CI workflow and the infrastructure provisioning mechanism, as well as parts of the CD workflow, this includes:

CI Workflow

- Set up repositories for application codes and provision appropriate secrets for application development.
- Developed application codes and automated test cases.
- Application code containerization using Docker and pushing the docker image onto Docker Hub.
- Developed Application Infrastructure configuration for Docker Container using Kubernetes.

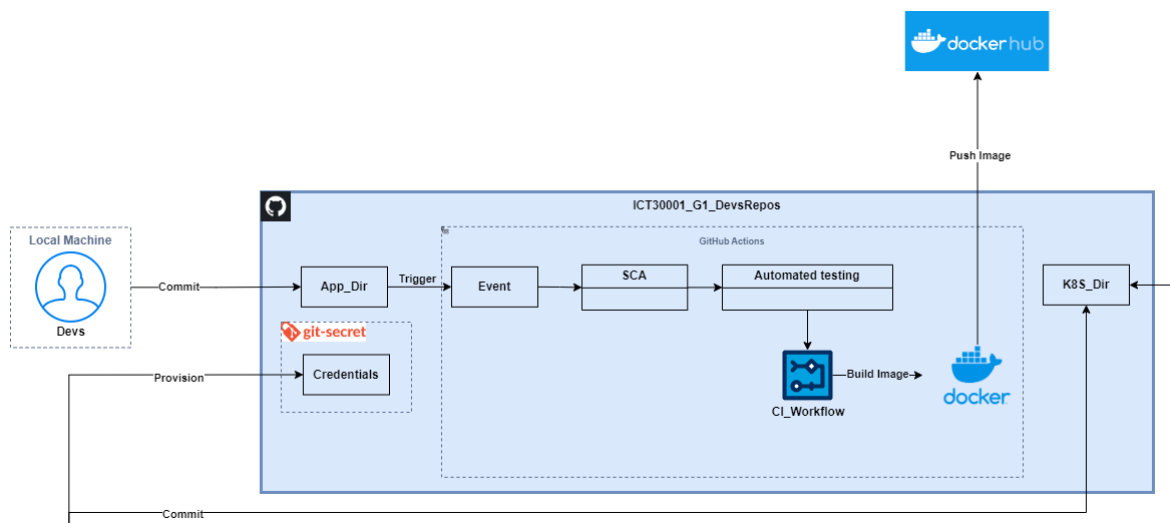


Figure 3: CI workflow

Infrastructure Provisioning mechanism

- Set up an AWS environment for the CI/CD pipeline.
- Set up infrastructure provisioning workflow to promote IaC.
- Set up Terraform Backend on AWS.
- Performed testing to confirm the use of Terraform to provision subsequent infrastructure.

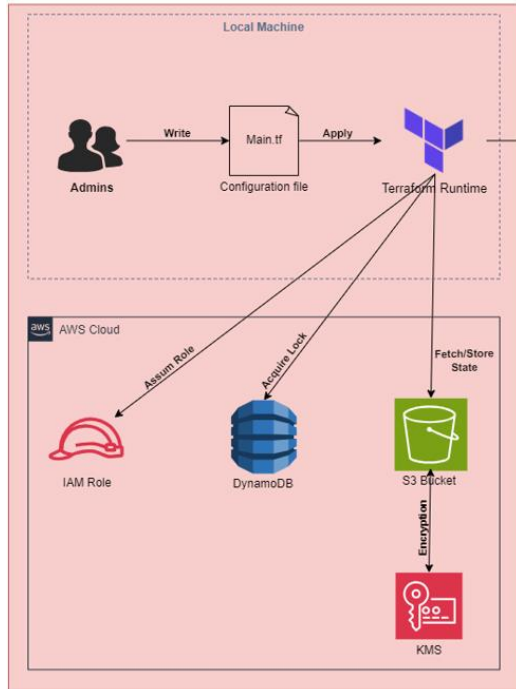


Figure 4: Terraform Backend

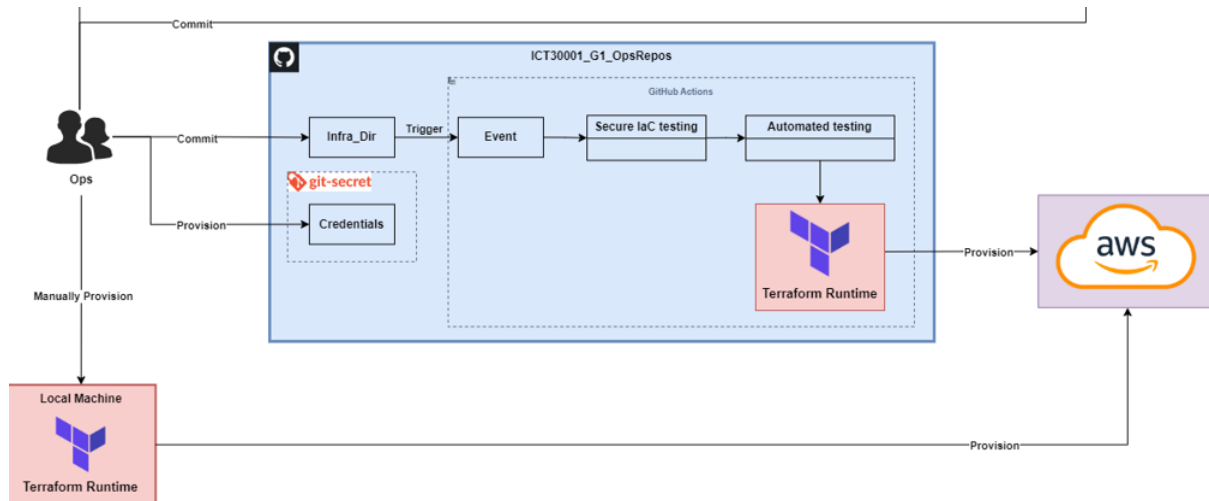


Figure 5: Infrastructure provision workflow

CD workflow

- Set up Argo CD as an automation server and sync the state of application infrastructure between Argo CD and Kubernetes manifest files on GitHub.
- Deployed test application to confirm that Kubernetes can pull Docker Image and deploy on its pods.
- Performed testing to confirm that Argo CD can sync the state of Kubernetes manifest files and Kubernetes cluster.

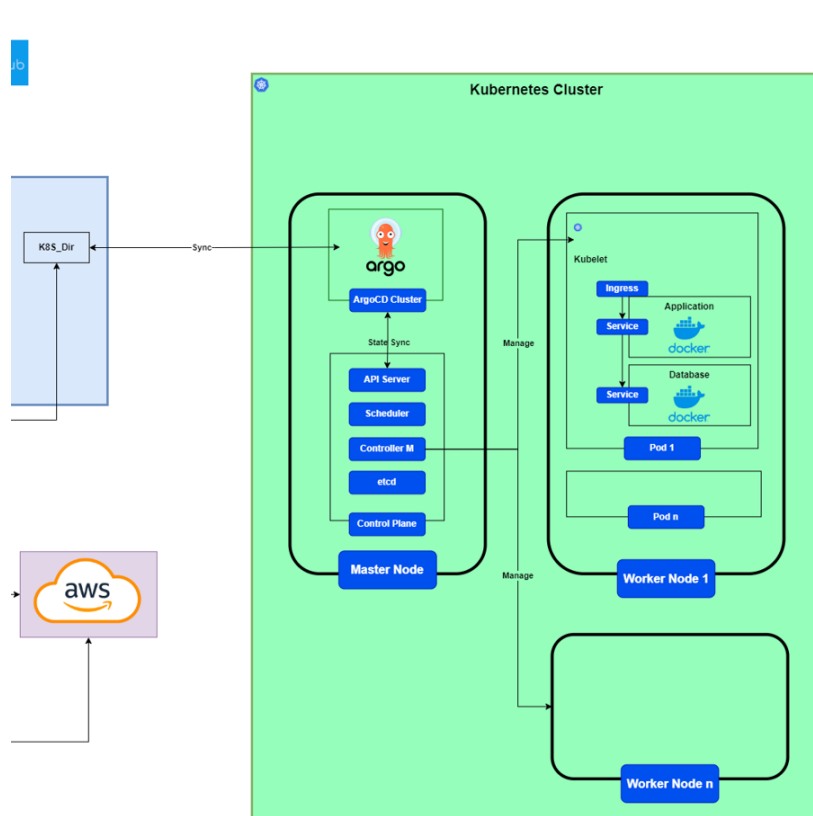


Figure 6: Current CD workflow

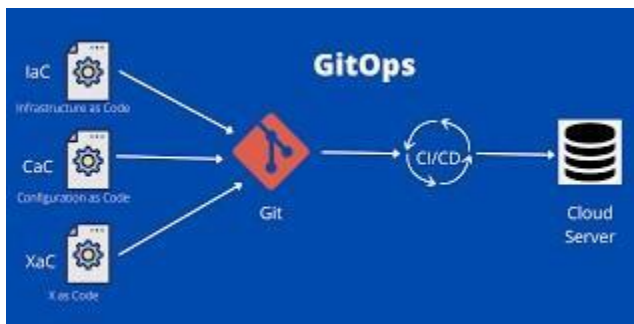


Figure 7: GitOps Methodology

Additionally, we also run the current workflow by changing 3 mains codes parts to confirm our workflow is adhered to the GitOps methodology:

- Changing the application code and confirming that changes in the code are automatically reflected on the Docker Image and the Application hosted on Kubernetes pods according to the CI workflow.
- Changing the Infrastructure as Code (IaC) using Terraform and confirming that the infrastructure is automatically modified according to the Terraform code using the Infrastructure as Code workflow.
- Changing the Configuration files of Kubernetes and confirming that the application infrastructure is automatically modified by Argo CD and the application is scaled accordingly without manual configuration.

Prototype screenshots

In this section, the current progress of the CI workflow, Infrastructure provisioning mechanism, and part of the CD workflow will be presented.

CI Workflow

Repository for application code with GitHub Action as CI tool to test and build image onto Docker Hub.

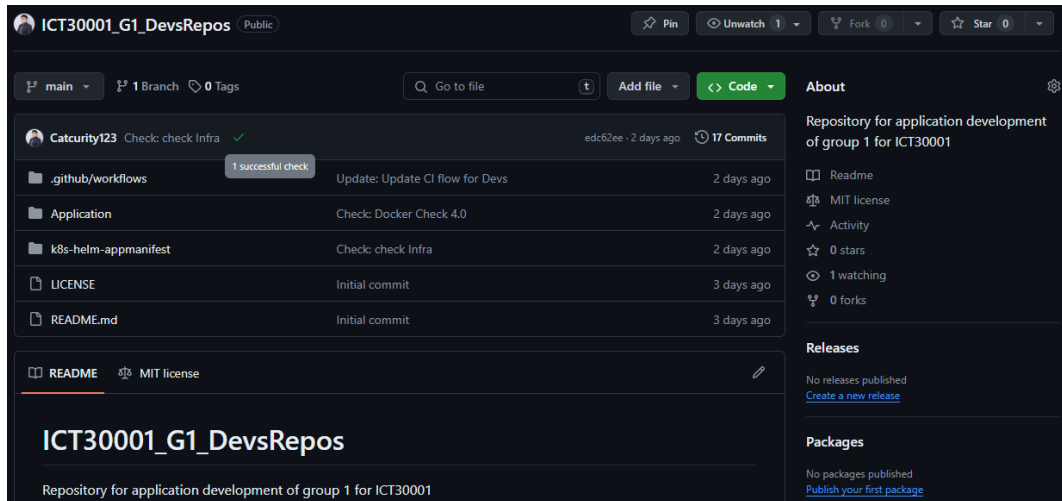


Figure 8: GitHub repository



Figure 9: CI workflow to check, build, and upload application image on Docker Hub

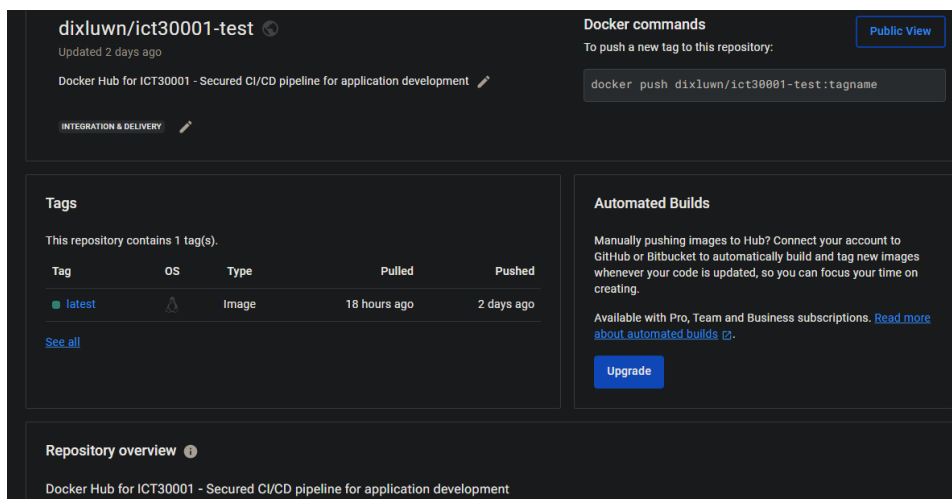
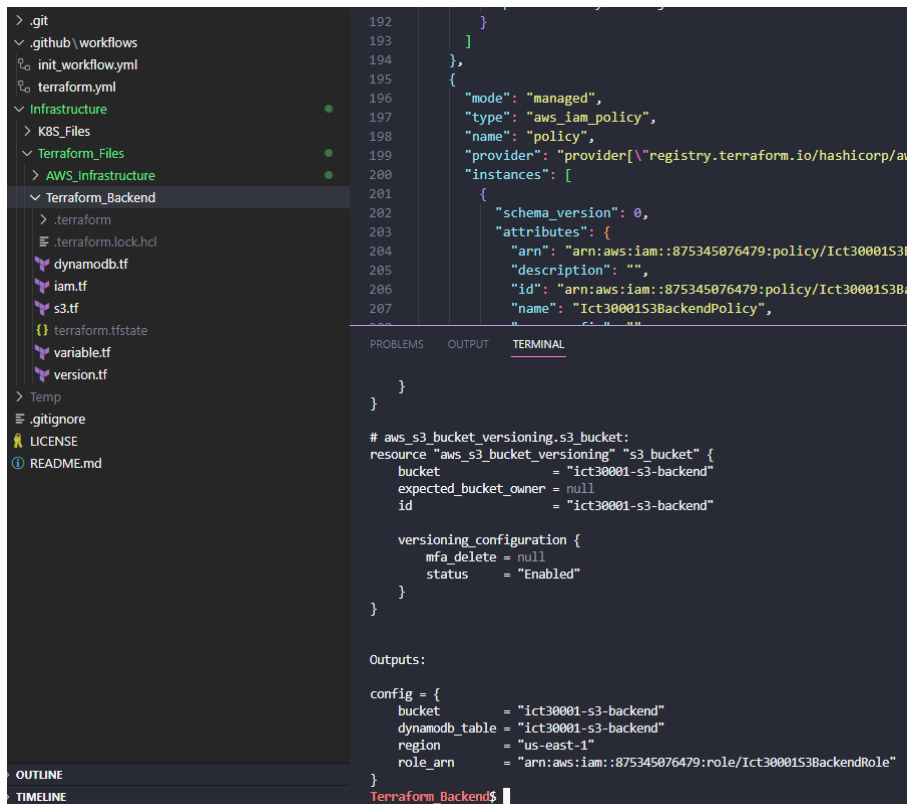


Figure 10: Application image is automatically uploaded onto Docker Hub

Infrastructure Provisioning mechanism

Terraform's IaC successfully provisioned AWS infrastructure and the state is stored onto S3 Bucket.



The screenshot shows a code editor with a file explorer on the left and a terminal on the right. The file explorer shows a project structure with folders like .github, init_workflow.yml, terraform.yml, Infrastructure, K8S_Files, Terraform_Files, AWS_Infrastructure, Terraform_Backend, .terraform, .terraform.lock.hcl, dynamodb.tf, iam.tf, s3.tf, terraform.tfstate, variable.tf, version.tf, Temp, .gitignore, LICENSE, and README.md. The terminal shows the output of a Terraform command, including the configuration for an AWS IAM policy and an AWS S3 bucket with versioning enabled. The terminal output is as follows:

```
192     }
193   ],
194 },
195 {
196   "mode": "managed",
197   "type": "aws_iam_policy",
198   "name": "policy",
199   "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
200   "instances": [
201     {
202       "schema_version": 0,
203       "attributes": {
204         "arn": "arn:aws:iam::875345076479:policy/Ict30001S3BackendPolicy",
205         "description": "",
206         "id": "arn:aws:iam::875345076479:policy/Ict30001S3BackendPolicy",
207         "name": "Ict30001S3BackendPolicy",
208         "path": ""
209       }
210     }
211   ]
212 }
213
214 # aws_s3_bucket_versioning.s3 bucket:
215 resource "aws_s3_bucket_versioning" "s3 bucket" {
216   bucket = "ict30001-s3-backend"
217   expected_bucket_owner = null
218   id = "ict30001-s3-backend"
219
220   versioning_configuration {
221     mfa_delete = null
222     status = "Enabled"
223   }
224 }
225
226 Outputs:
227
228 config = {
229   bucket = "ict30001-s3-backend"
230   dynamodb_table = "ict30001-s3-backend"
231   region = "us-east-1"
232   role_arn = "arn:aws:iam::875345076479:role/Ict30001S3BackendRole"
233 }
```

Figure 11: Terraform's Backend

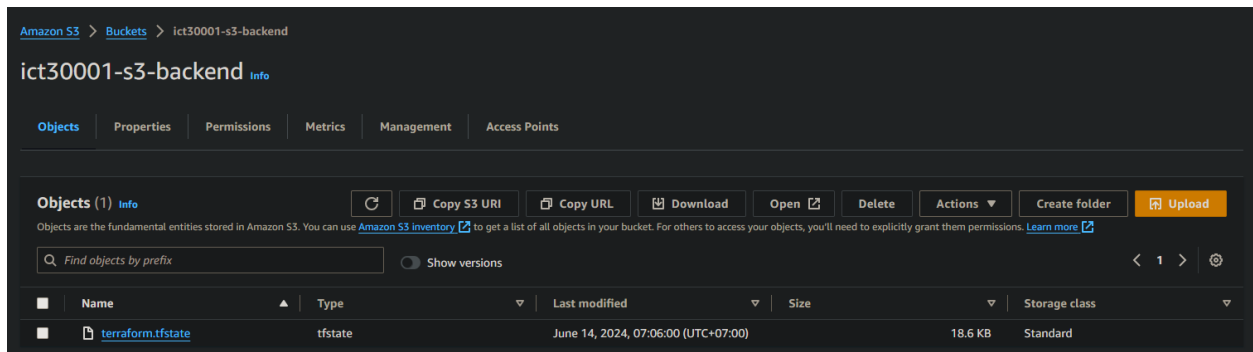


Figure 12: Terraform's state file is stored on AWS S3

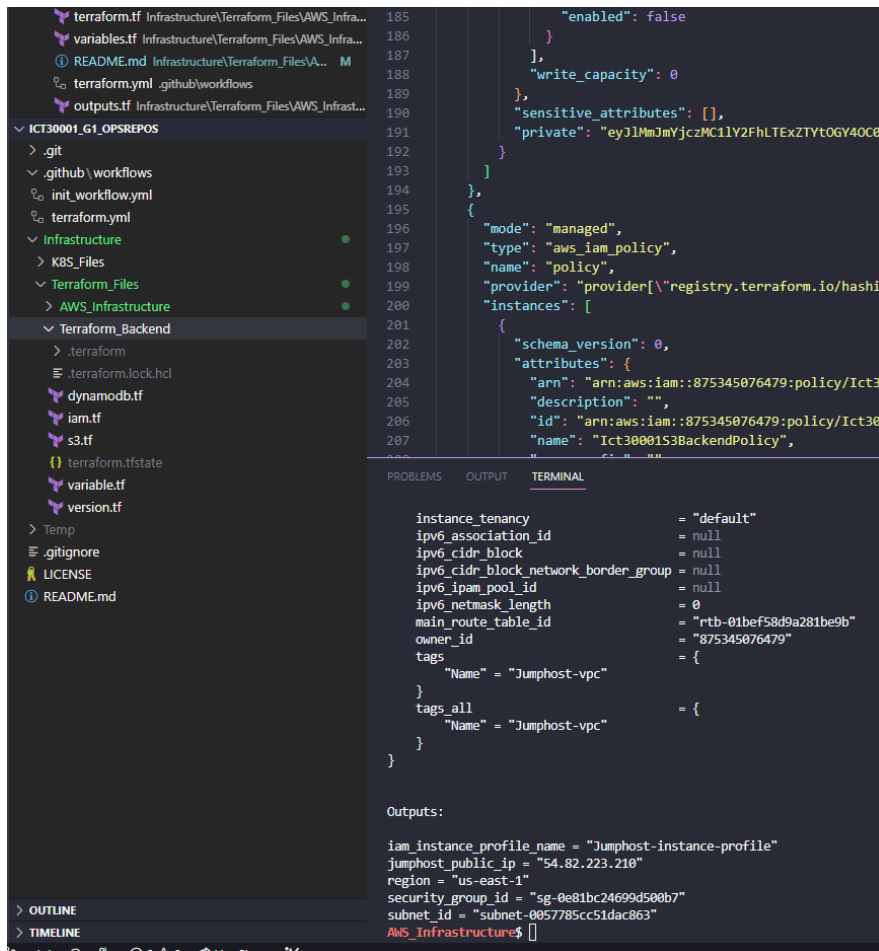


Figure 13: Infrastructure provisioning using Terraform

CD Workflow

Argo CD successfully syncs the state of application infrastructure between the Development Repository with Kubernetes and makes modification whenever Kubernetes manifest is changed.

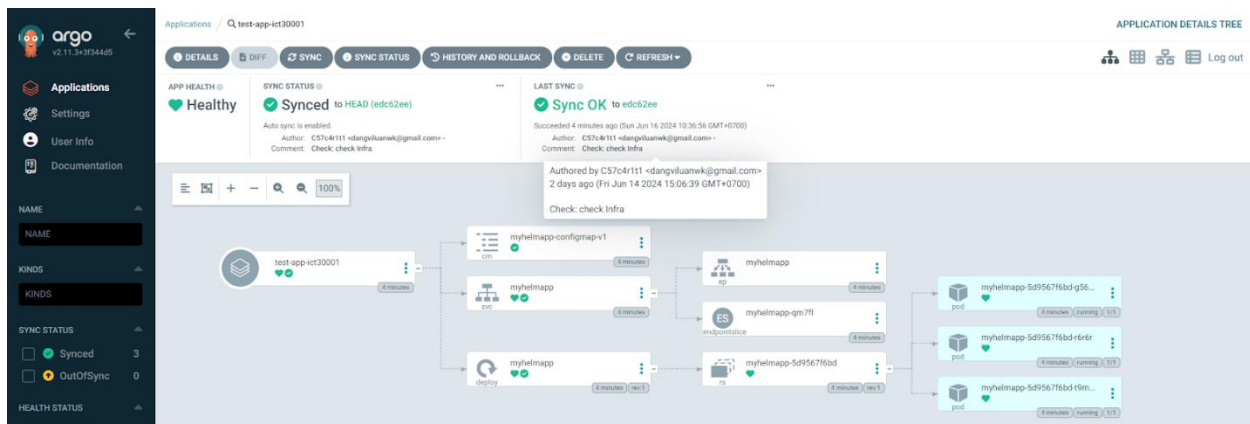


Figure 14: Argo CD successfully synced the state between Kubernetes and GitHub manifest files

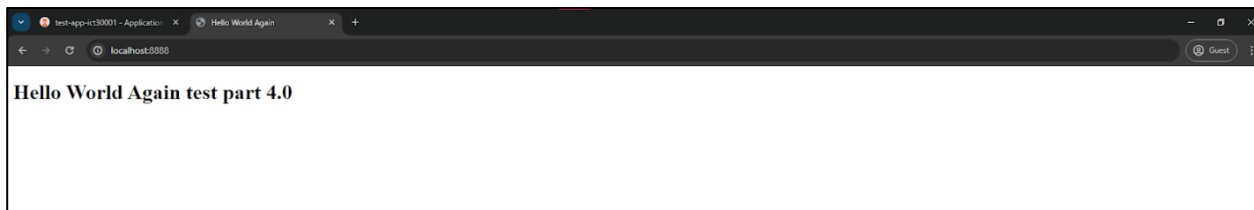


Figure 15: Application image successfully runs on Kubernetes pods

```
ICT30001_G1_DevsRepos$ kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d
dCuZ58dRXhqPp5A9ICT30001_G1_DevsRepos$ kubectl port-forward service/myhelmapp 8888:80 -n dev &
[1] 1675
ICT30001_G1_DevsRepos$ Forwarding from 127.0.0.1:8888 -> 80
Forwarding from [::1]:8888 -> 80
Handling connection for 8888

ICT30001_G1_DevsRepos$ kubectl get all -n dev
NAME                                READY    STATUS    RESTARTS   AGE
pod/myhelmapp-5d9567f6bd-g56bp      1/1     Running   0           5m36s
pod/myhelmapp-5d9567f6bd-r6r6r      1/1     Running   0           5m36s
pod/myhelmapp-5d9567f6bd-t9mg8      1/1     Running   0           5m36s

NAME                                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
service/myhelmapp                   NodePort    10.100.138.133 <none>         80:31011/TCP   5m36s

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/myhelmapp           3/3      3              3            5m36s

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/myhelmapp-5d9567f6bd 3          3          3        5m36s
ICT30001_G1_DevsRepos$
```

Figure 16: Pods running locally

Work left to complete the project

We have completed the implementation of the CI flow with the exception that the application is currently deployed to a local-host K8S cluster.

Remaining items to complete the project our plan for the 4 following weeks is as follow:

Week 7:

- Test deployment of application on real EKS cluster with static IP output so real users can access the application
- Implement pre-commit hooks to prevent accidental commit of sensitive information.
- Integrate AWS Secrets Manager to manage access to credentials across services.
- Integrate Github Actions secrets to manage access across platforms.

Week 8

- Implement SAST (Static Application Security Testing) using Sonarqube to scan for security vulnerabilities.
- Leverage SCA (Source Composition Analysis) with GitHub to identify vulnerable libraries.
- Integrate OWASP ZAP to detect vulnerabilities in running applications.
- Implement Snyc to scan for code vulnerabilities.

Week 9

- Scan Docker images for vulnerabilities using tools like Clair to ensure container security.

- Ensure security for the container image repository.

Week 10

- Use Terraform to provision the required cloud infrastructure as part of the CI/CD pipeline.
- Deploy the containerized application to a Kubernetes cluster, automating the deployment of Kubernetes manifests.

Week 11

- Finalize Documentation and arrange a Knowledge Transfer session with the customer.
- Finalize completion report to be ready for customer sign-off.

Week 12

- Final Presentation with the clients.
- Archive project assets & close the project.

Deliverables

- **Secured CI/CD Pipeline Design:** A comprehensive design that integrates version control, continuous integration, deployment, and DevSecOps components using various tools and technologies. This will establish a robust and secure development and deployment pipeline, as illustrated in the provided diagram.
- **Test Report:** A detailed report documenting the testing procedures and outcomes conducted on the CI/CD pipeline. This will provide insights into the effectiveness and reliability of the pipeline in ensuring code integrity and security.
- **Project Completion Report:** A comprehensive report summarizing the project's objectives, activities, achievements, and outcomes. This will serve as formal documentation of the project's completion and include insights into the challenges faced, lessons learned, and recommendations for future improvements.
- **Setup Documentation and User Knowledge Transfer:** Documentation outlining the setup procedures and configurations for the key services involved in the CI/CD pipeline. Additionally, user knowledge transfer documentation will be provided to facilitate efficient utilization of the pipeline by relevant stakeholders.

**** Deliverables (up to now)**

- CI/CD prototype diagrams
- CI flow and CD flow
- Some parts of the CI/CD Pipeline

Questions from the client

Q1: What are the differences between the project deliverable – a CI/CD pipeline, and functions on Github that allow pushing and managing code securely?

Answer

Although some functionality that helps manage and securely push code is already available on GitHub's CI/CD pipeline, the CI/CD pipeline still provides better effectiveness in managing the project's source code because the CI/CD pipeline can do more than GitHub. GitHub can only manage the code but cannot automatically perform tasks such as running test cases when code is committed. On the other hand, the CI/CD pipeline also creates a connection between GitHub and other deployment tools such as Kubernetes, Docker, etc. If there is no CI/CD pipeline, these tools will only work separately, and the user will have to manually configure different tools to serve the project deployment.

Q2: How can the CI/CD pipeline be used for different source code languages (PHP, C#, etc.)?

Answer

Docker containers can be used to enable the CI/CD pipeline to handle different projects with different code languages and technologies. Each project has its own Docker containers configured to serve a specific language and framework. Docker containers can increase the consistency of the deployment process across different projects because they do not depend much on the underlying technologies.

Suggestions from client

S1: Pay more attention to market research to understand the demands of end-users.

- The end users are the target audience for project information, so understanding their needs is crucial.
- Emphasis should be placed on market research to evaluate if the project can address end users' problems.
- The project should be continuously improved to be more user-friendly.
- Identifying what the end users know and do not know about the project and providing additional information for the gaps can help enhance their understanding.

S2: The project deliverable should only stop at the Testing phase.

- Risk to Current Business Assets: Deploying directly to PROD could potentially damage or disrupt the existing production systems and assets, which could have serious consequences for the business.
- Additional Project Costs: The migration and deployment process to the PROD environment may require additional effort, resources, and project costs that were not initially included.

APPENDIX

Meeting minute: [Link](#)

Presentation materials: [Link](#)

Evidence of the Client Progress Review

