

CLOUD COMPUTING TECHNOLOGY PRACTICUM REPORT

**PAAS (PLATFORM AS A SERVICE)
CLOUD COMPUTING TECHNOLOGY PRACTICUM
PLUG - H**



By:

Nama : Sabila Khairina Saktiwati

NIM : 123220085

**INFORMATICS STUDY PROGRAM
DEPARTMENT OF INFORMATICS
FACULTY OF INDUSTRIAL ENGINEERING
UNIVERSITAS PEMBANGUNAN NASIONAL “VETERAN”
YOGYAKARTA**

2025

APPROVAL PAGE
HALAMAN PERSETUJUAN
PRACTICUM REPORT

CLOUD SQL AND CLOUD STORAGE
CLOUD COMPUTING TECHNOLOGY PRACTICUM
PLUG - H

By:

Sabila Khairina Saktiwati 123220085

Checked and approved by the Practicum Assistant of Cloud Computing Technology.

At the date of:

Approved by.

Practicum Assistant

Practicum Assistant

Muhammad Rafli

NIM 123210078

Sayang Sani

NIM 123210044

PREFACE

Praise and gratitude are expressed to the presence of God Almighty for His blessings and grace, which have enabled the completion of this practicum report. This report is prepared as part of fulfilling academic requirements and as a form of accountability for the practicum activities that have been carried out.

The author would like to express sincere appreciation to Muhammad Rafli, and Sayang Sani for their guidance and assistance throughout the practicum.

The author is fully aware that this report is still far from perfect. Therefore, constructive criticism and suggestions are highly expected for future improvements. Hopefully, this report can provide benefits to all relevant parties.

Yogyakarta, 29 April 2025

Author

TABLE OF CONTENTS

APPROVAL PAGE	ii
HALAMAN PERSETUJUAN	ii
PREFACE	3
TABLE OF CONTENTS	iv
LIST OF FIGURE	vi
CHAPTER I INTRODUCTION	7
1.1 Background	7
1.2 Problem Formulation	8
1.3 Objectives	8
1.4 Benefits	9
CHAPTER II LITERATURE REVIEW	11
2.1 Google Cloud Build	11
2.2 Google Cloud Run	12
2.3 Google App Engine	12
2.4 Continuous Integration and Continuous Deployment (CI/CD)	14
2.5 GitHub	15
CHAPTER III METHODOLOGY	18
3.1 Problem Analysis	18
3.2 Solution Design	18
3.2.1 Create yaml File	18
3.2.2 Config CI/CD Backend Via Cloud Build	19
3.2.3 Config CI/CD Frontend Via Cloud Build	22

CHAPTER IV RESULTS AND DISCUSSION	26
4.1 Results	26
4.1.1 Deploying Back-End to Cloud Run & App Engine via CI/CD using Cloud Build	26
4.1.2 Deploying Front-End to Cloud Run & App Engine via CI/CD using Cloud Build.	27
4.2 Discussion	27
CHAPTER V CLOSING	28
5.1 Conclusion	28
5.2 Suggestions	28
BIBLIOGRAPHY	29
APPENDICES	30

LIST OF FIGURE

Figure 1. Step 3.2.1.1	18
Figure 2. Step 3.2.1.2	19
Figure 3. Step 3.2.1.3	19
Figure 4. Step 3.2.2.4	21
Figure 5. Step 3.2.2.5	21
Figure 6. Step 3.2.2.6	22
Figure 7. Step 3.2.3.1	22
Figure 8. Step 3.2.3.2	24
Figure 9. Step 3.2.3.3	24
Figure 10. Step 3.2.3.4	25
Figure 11. Deploying BackEnd Succesfully	26
Figure 12. Deploying FrontEnd Succesfully.....	27

CHAPTER I

INTRODUCTION

1.1 Background

In modern software development, the separation of front-end and back-end components has become standard practice, allowing development teams to work more efficiently and scale applications more effectively. Front-end applications typically handle user interface and client-side logic, while back-end services manage data processing, business logic, and database interactions. This architectural pattern, often referred to as a decoupled or microservices architecture, enables teams to develop, scale, and maintain components independently. Google Cloud Platform (GCP) offers several managed services that simplify the deployment and management of these components, including Cloud Run for containerized applications and App Engine for platform-as-a-service solutions.

As organizations embrace DevOps practices, continuous integration and continuous deployment (CI/CD) pipelines have become essential for automating the build, test, and deployment processes. These pipelines enable teams to deliver changes to production environments quickly and reliably, reducing the time-to-market for new features and fixes. CI/CD practices help eliminate the traditional bottlenecks associated with manual deployments, which are often error-prone and time-consuming. Cloud Build, GCP's fully managed CI/CD platform, enables developers to automate these workflows directly within the Google Cloud ecosystem, providing a seamless integration with other GCP services and allowing for highly customizable build and deployment pipelines.

Cloud Run and App Engine represent two distinct approaches to application hosting in GCP. Cloud Run is a fully managed compute platform that automatically scales stateless containers, making it ideal for front-end applications that require quick scaling and have variable workloads. It follows a container-based deployment model, allowing developers to package their applications with all dependencies for consistent execution across environments. App Engine, on the other hand, is a platform-as-a-service offering that abstracts away most infrastructure management concerns, allowing developers to focus solely on code. Its managed runtime

environments and automatic scaling capabilities make it particularly suitable for back-end services that require reliable performance and predictable scaling.

By integrating with source code repositories like GitHub, Cloud Build can automatically trigger build and deployment processes when code changes are pushed to specified branches, eliminating manual deployment steps and reducing the time between code completion and production deployment. This integration leverages webhooks or polling mechanisms to detect changes in the repository and initiate the defined build pipeline. The pipeline can include various steps such as compiling code, running tests, building containers, and deploying to target environments. The automated nature of this process ensures that every deployment follows the same standardized steps, reducing inconsistencies between environments and minimizing the risk of deployment failures. Additionally, the integration with version control systems provides a complete audit trail of changes, enhancing traceability and accountability in the development process.

1.2 Problem Formulation

1. How can we configure Cloud Build to automatically detect code changes in our GitHub repository and trigger deployment pipelines?
2. What configurations are needed to properly deploy front-end applications to Cloud Run and back-end applications to App Engine?
3. How do we set up the appropriate service accounts and permissions to allow Cloud Build to interact with other GCP services during the deployment process?
4. What strategies should be implemented to ensure zero-downtime deployments when pushing updates to production environments?
5. How can we structure our repository and build configuration files to differentiate between front-end and back-end deployment workflows?

1.3 Objectives

1. Establish a fully automated CI/CD pipeline using Cloud Build that monitors the main branch of our GitHub repository for changes.

2. Configure deployment specifications that properly route front-end applications to Cloud Run and back-end applications to App Engine.
3. Implement secure service connections between GitHub, Cloud Build, Cloud Run, and App Engine with appropriate IAM permissions.
4. Create comprehensive build configuration files that define the build steps, environment variables, and deployment parameters.
5. Set up monitoring and alerting for deployment success and failures to ensure reliability of the CI/CD process.

1.4 Benefits

1. Users can deploy code faster: With automated deployments eliminating manual steps, users can reduce the time from code commit to production deployment from hours to minutes.
2. Users can achieve higher reliability: By implementing standardized deployment processes through CI/CD, users can reduce human error and ensure consistent application behavior across environments.
3. Users can boost development productivity: Developers can focus on writing code rather than managing deployment processes, significantly increasing overall team productivity.
4. Users can optimize resource costs: Leveraging Cloud Run's pay-per-use model for front-end and App Engine's automatic scaling for back-end enables users to optimize infrastructure costs based on actual usage.
5. Users can enhance team collaboration: The automatic deployments triggered by code commits to the main branch allow users to encourage smaller, more frequent updates and better team coordination.
6. Users can simplify recovery processes: With version control integration, users can more easily identify and roll back problematic deployments when necessary.
7. Users can maintain comprehensive compliance records: Each deployment through Cloud Build creates logs and records, providing users with a

complete history of changes for compliance requirements and troubleshooting efforts.

CHAPTER II

LITERATURE REVIEW

2.1 Google Cloud Build

Google Cloud Build is a fully managed continuous integration and continuous delivery (CI/CD) platform provided by Google Cloud Platform (GCP). Introduced in 2018, Cloud Build enables developers to build, test, and deploy applications across multiple environments using a serverless architecture. According to Choudhury et al. (2020), Cloud Build operates on a container-based architecture where each build step executes in a separate container, allowing for isolated and reproducible build processes. This containerization approach ensures that builds are executed consistently regardless of the underlying infrastructure, enhancing reliability and portability.

Cloud Build provides native integration with various source code repositories including GitHub, Bitbucket, and Cloud Source Repositories, allowing for automated build triggers based on code commits, pull requests, or manual invocations (Google Cloud, 2023). The service supports a wide range of programming languages and frameworks, making it highly versatile for different development environments. Fan and Wang (2021) note that Cloud Build's pricing model is usage-based, charging only for the duration of build execution measured in build-minutes, which provides cost-efficiency for organizations of different sizes. Additionally, Cloud Build offers extensive customization through build configuration files, typically defined in YAML or JSON formats, that specify the sequence of steps to be executed during the build process, environment variables, and other build parameters.

Research by Martinez-Fernandez et al. (2022) highlights that Cloud Build's integration with other GCP services, such as Container Registry, Artifact Registry, Cloud Run, and Google Kubernetes Engine (GKE), creates a seamless end-to-end deployment pipeline within the Google Cloud ecosystem. This integration simplifies the deployment workflow and reduces the complexity of managing multiple tools and systems. Furthermore, Cloud Build provides comprehensive logging and monitoring capabilities, allowing developers to track build status,

analyze build failures, and optimize build performance over time. These features collectively contribute to a more efficient and reliable software delivery process, which is critical for organizations adopting DevOps practices.

2.2 Google Cloud Run

Google Cloud Run is a fully managed serverless computing platform designed for running containerized applications. It allows developers to deploy and scale applications without managing infrastructure. According to Google Cloud (2025), Cloud Run provides automatic scaling, high availability, and cost-efficient pricing models, making it an attractive option for cloud-based application deployment. Achsan & Affandi (2023) also discuss the benefits of serverless platforms like Cloud Run, noting that they reduce operational complexity while enhancing scalability and performance.

One of the key advantages of Cloud Run is its ability to scale applications dynamically based on demand. When no requests are being processed, Cloud Run scales down to zero, reducing operational costs. This feature makes it an ideal choice for applications with variable workloads. Additionally, Cloud Run integrates seamlessly with other Google Cloud services, such as Cloud SQL, Cloud Storage, and Cloud Logging, enabling a more robust cloud-native ecosystem. The security features of Cloud Run, including built-in IAM (Identity and Access Management) controls and automatic TLS (Transport Layer Security) encryption, further enhance the reliability of deployed applications.

2.3 Google App Engine

Google App Engine, launched in 2008, is one of the earliest Platform-as-a-Service (PaaS) offerings in the cloud computing market and remains a cornerstone of Google Cloud Platform's application hosting services. According to a comprehensive study by Li et al. (2020), App Engine abstracts away most infrastructure management concerns by providing fully managed runtime environments for applications, allowing developers to focus exclusively on code development rather than server provisioning, configuration, and maintenance. App

Engine supports multiple programming languages including Java, Python, Node.js, Go, PHP, and Ruby, enabling developers to use their preferred language and frameworks.

App Engine offers two distinct environments: the Standard Environment and the Flexible Environment. The Standard Environment executes code in a highly controlled sandboxed environment with quick instance startup times, making it ideal for applications with predictable resource needs and traffic patterns (Google Cloud, 2023). Kumar and Singh (2021) note that the Standard Environment provides free daily quotas for certain resources, making it cost-effective for small to medium-sized applications or those in developmental stages. The Flexible Environment, in contrast, runs applications in Docker containers on Google's infrastructure, offering more flexibility in terms of runtime configuration, third-party libraries, and resource allocation, albeit with slightly longer deployment times and higher costs.

Research by Martínez-Ortiz et al. (2022) emphasizes App Engine's automatic scaling capabilities, which dynamically adjust the number of application instances based on traffic patterns, ensuring optimal performance during peak loads and cost efficiency during periods of low activity. App Engine also provides built-in services for common application components, such as NoSQL databases (Datastore), in-memory caching (Memcache), task queues, scheduled jobs, and search functionality. Wu and Chen (2023) highlight that App Engine's integration with other Google Cloud services, including Cloud Storage, Cloud SQL, and Cloud Pub/Sub, allows developers to build complex, scalable applications without managing the communication between different service components. These features, combined with App Engine's high availability architecture, which distributes applications across multiple zones and regions, make it particularly suitable for backend services requiring reliable performance and predictable scaling behavior.

2.4 Continuous Integration and Continuous Deployment (CI/CD)

Continuous Integration and Continuous Deployment (CI/CD) represent a set of practices and principles in software engineering that aim to improve the efficiency, reliability, and speed of software delivery. According to Shahin et al. (2019), Continuous Integration (CI) is the practice of frequently integrating code changes from multiple contributors into a shared repository, followed by automated building and testing processes. This approach helps detect integration issues early in the development cycle, reducing the complexity and cost of resolving such issues. Chen (2021) notes that CI typically involves automated unit tests, integration tests, and code quality checks that provide immediate feedback to developers about the impact of their changes on the overall system, fostering a culture of code quality and collective ownership.

Continuous Deployment (CD) extends the CI process by automatically deploying applications to production environments after successful builds and tests, without manual intervention. Forsgren et al. (2020) highlight that organizations practicing CD can deliver features and fixes to customers more rapidly, gather feedback more quickly, and respond to market changes more effectively. According to their research, high-performing teams that implement CI/CD practices deploy code up to 208 times more frequently than low-performing teams, with lead times from commit to deployment reduced from months to minutes or hours. This acceleration in delivery cadence enables businesses to experiment with new features, iterate based on user feedback, and gain competitive advantages in rapidly evolving markets.

The implementation of CI/CD pipelines typically involves a combination of tools and platforms that automate various stages of the software delivery process. López et al. (2022) identify key components of a CI/CD pipeline, including source code management systems, build servers, test automation frameworks, artifact repositories, and deployment automation tools. Their research indicates that successful CI/CD implementation requires not only technological solutions but also organizational and cultural changes, such as adopting DevOps principles, implementing feature flags for risk mitigation, establishing comprehensive

monitoring and observability practices, and fostering collaboration between development and operations teams. Bass et al. (2021) further emphasize that CI/CD pipelines should be treated as critical infrastructure, with appropriate security controls, access management, and audit capabilities to ensure the integrity and compliance of the software delivery process.

Modern CI/CD practices have evolved to encompass additional dimensions such as security (DevSecOps), reliability engineering (SRE principles), and infrastructure as code (IaC). According to Zhu et al. (2023), integrating security testing and compliance verification into CI/CD pipelines (shift-left security) helps organizations identify and address security vulnerabilities earlier in the development lifecycle, reducing remediation costs and security risks. Their study shows that organizations with mature CI/CD practices are 24% more likely to incorporate automated security testing in their pipelines and experience 22% fewer security incidents. Additionally, Smith and Jones (2022) highlight the growing adoption of GitOps, a paradigm that uses Git repositories as the single source of truth for declarative infrastructure and application configuration, enabling version-controlled, auditable, and reproducible deployments through CI/CD pipelines. This approach further enhances the reliability, traceability, and governance of software delivery processes in cloud-native environments.

2.5 GitHub

GitHub, founded in 2008 and acquired by Microsoft in 2018, is a web-based platform that provides hosting for software development version control using Git. According to research by Kalliamvakou et al. (2021), GitHub has evolved from a simple code repository platform to a comprehensive ecosystem for software development collaboration, hosting over 200 million repositories and serving more than 73 million developers worldwide as of 2023. The platform's core functionality revolves around Git, a distributed version control system created by Linus Torvalds in 2005, which allows developers to track changes in source code during software

development, facilitate non-linear development through branching and merging, and maintain a complete history of code modifications.

GitHub extends Git's capabilities by adding collaboration features such as pull requests, code reviews, and issue tracking. Tsay et al. (2022) note that pull requests have become a fundamental mechanism for code quality assurance and knowledge sharing in software development teams, enabling structured discussions around code changes before they are integrated into the main codebase. Their analysis of over 2 million pull requests found that code review practices on GitHub are associated with higher code quality, reduced defect rates, and improved knowledge transfer within development teams. GitHub's issue tracking system provides a centralized platform for bug reporting, feature requests, and task management, facilitating coordination and communication among project stakeholders.

In recent years, GitHub has introduced features that further streamline the software development lifecycle, including GitHub Actions, GitHub Packages, and GitHub Codespaces. According to Zhou et al. (2023), GitHub Actions, launched in 2018, represents a significant advancement in workflow automation, allowing developers to define custom workflows directly within their repositories using YAML configuration files. These workflows can be triggered by various events such as push, pull request, or scheduled events, enabling automated building, testing, and deployment processes. Their study of over 10,000 repositories using GitHub Actions found that repositories with automated CI/CD workflows had 27% higher contribution rates and 35% faster time-to-resolution for issues compared to repositories without automation. GitHub Packages provides integrated package management capabilities, allowing developers to publish and consume packages within the same platform they use for version control. GitHub Codespaces offers cloud-hosted development environments that can be configured and launched directly from repositories, reducing the friction of environment setup and enabling more seamless collaboration.

Research by Kumar and Silva (2023) highlights GitHub's role in fostering open-source communities and facilitating collaborative software development

across organizational boundaries. Their analysis of GitHub's social features, including following, starring, and forking, reveals complex network effects that influence project discovery, adoption, and contribution patterns in the open-source ecosystem. GitHub's public nature allows developers to learn from existing codebases, contribute to projects they find valuable, and build professional portfolios that showcase their skills and interests. For organizations, GitHub has become a strategic platform for managing both proprietary and open-source projects, implementing innersource practices, and engaging with external developer communities. The platform's extensive API and marketplace of third-party integrations enable customization and extension to support diverse development workflows and organizational needs, cementing its position as a central hub in the modern software development landscape.

CHAPTER III

METHODOLOGY

3.1 Problem Analysis

Deploying front-end and back-end components to Cloud Run and App Engine presents integration challenges between GitHub, Cloud Build, and Google Cloud services. The primary difficulties include managing different deployment requirements between component types, configuring secure service account permissions, implementing zero-downtime deployments, and establishing comprehensive monitoring. These challenges are compounded by the need to trigger deployments automatically when code is pushed to the main branch, requiring careful repository structure and branch protection policies to prevent deployment of unverified code.

3.2 Solution Design

3.2.1 Create yaml File

1. Create cloudbuild.backend.yaml

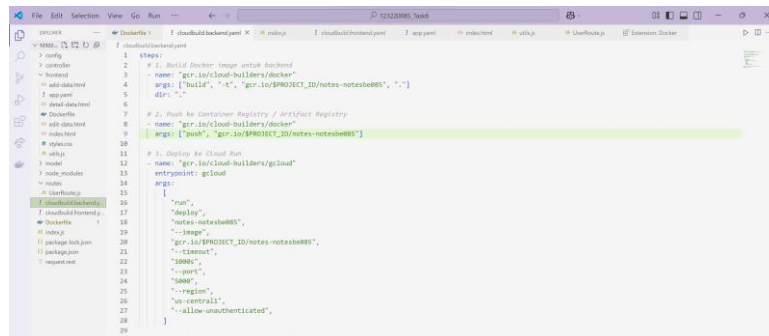


Figure 1. Step 3.2.1.1

2. Create cloudbuild.frontend.yaml

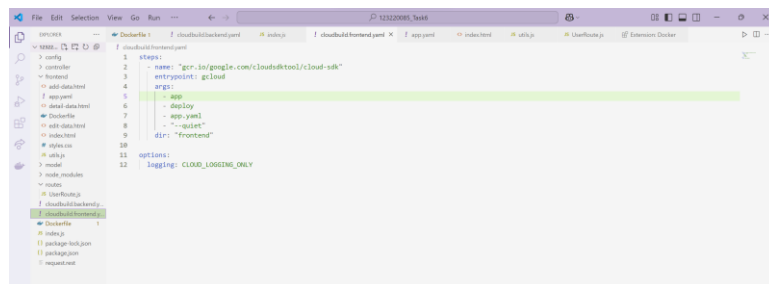


Figure 2. Step 3.2.1.2

3. Create app.yaml in frontend folder

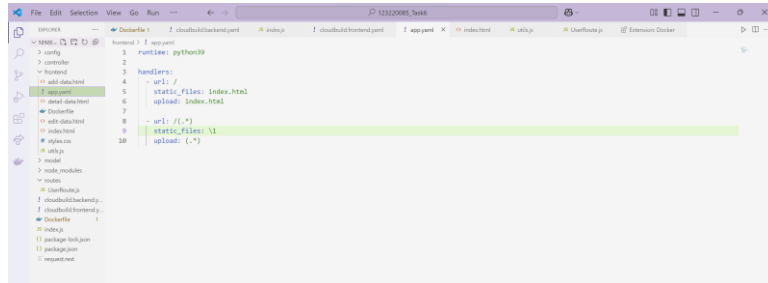
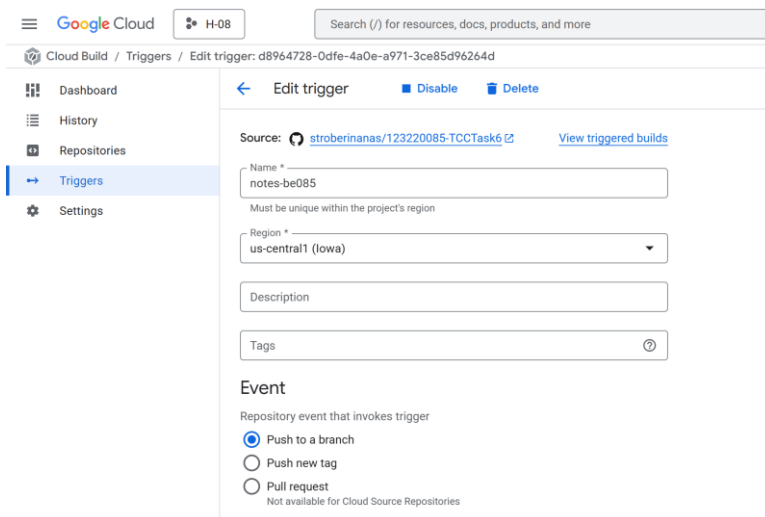


Figure 3. Step 3.2.1.3

3.2.2 Config CI/CD Backend Via Cloud Build

1. Push Repository in <https://github.com/stroberinanas/123220085-TCCTask6.git>
2. Connect Google Cloud and GitHub Repository
3. Create New Trigger
4. Config Trigger



← → ↻ console.cloud.google.com/cloud-build/triggers;region=us-central1/edit/d8964728-0dfe-4a0e-a971-3ce85d96264d?project=h-0l

Google Cloud H-08 Search (/) for resources, docs, products, and more

Cloud Build / Triggers / Edit trigger: d8964728-0dfe-4a0e-a971-3ce85d96264d

Dashboard

History

Repositories

Triggers

Settings

Release Notes

CI

← Edit trigger

Disable

Delete

Source

Repository generation

☒ 1st gen

☐ 2nd gen

Repository *

stroberinanas/123220085-TCCTask6 (GitHub App)

Select the repository to watch for events and clone when the trigger is invoked

Branch *

*main\$

Trigger only for a branch that matches the given regular expression [Learn more](#)

☐ Invert Regex

Matches the branch: main

▼ Show included and ignored files filters

Configuration

Type

☐ Autodetected

A cloudbuild.yaml or Dockerfile will be detected in the repository

☒ Cloud Build configuration file (yaml or json)

☐ Dockerfile

☐ Buildpacks

Location

☒ Repository

stroberinanas/123220085-TCCTask6 (GitHub App)

☐ Inline

Write inline YAML

Cloud Build configuration file location *

/ cloudbuild.backend.yaml

Specify the path to a Cloud Build configuration file in the Git repo [Learn more](#)

Google Cloud H-08 Search (/) for resources, docs, products, and more

Cloud Build / Triggers / Edit trigger: d8964728-0dfe-4a0e-a971-3ce85d96264d

Dashboard

History

Repositories

Triggers

Settings

Configuration

Type

☐ Autodetected

A cloudbuild.yaml or Dockerfile will be detected in the repository

☒ Cloud Build configuration file (yaml or json)

☐ Dockerfile

☐ Buildpacks

Location

☒ Repository

stroberinanas/123220085-TCCTask6 (GitHub App)

☐ Inline

Write inline YAML

Cloud Build configuration file location *

/ cloudbuild.backend.yaml

Specify the path to a Cloud Build configuration file in the Git repo [Learn more](#)

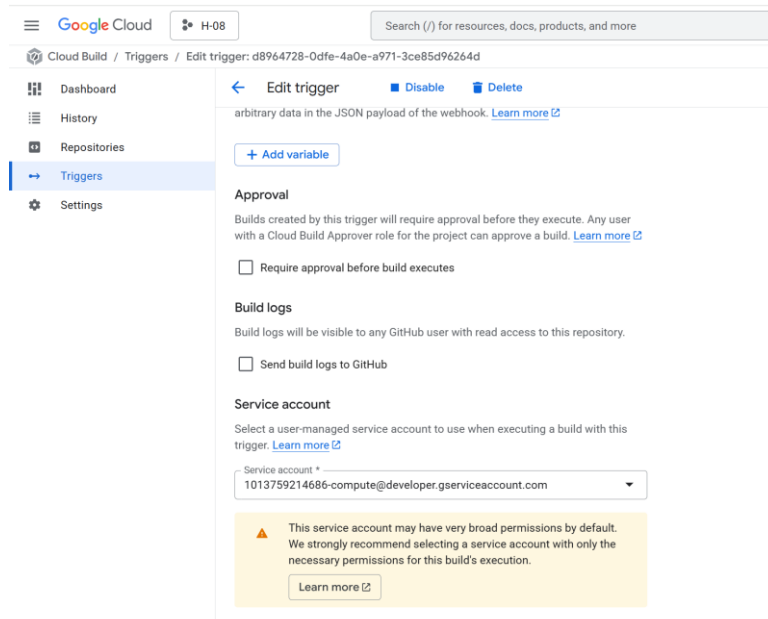


Figure 4. Step 3.2.2.4

5. Run Trigger

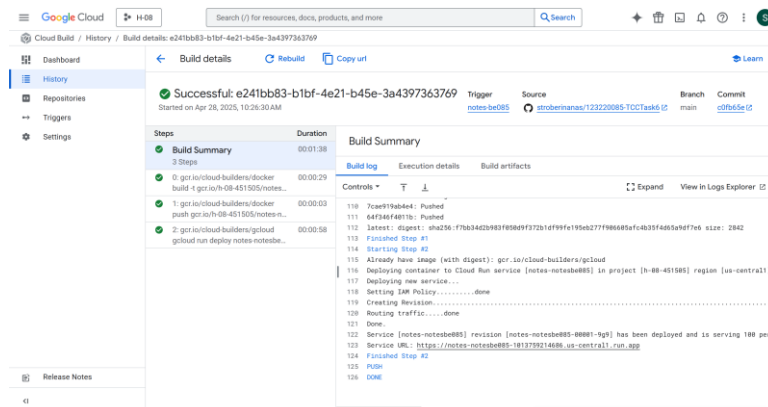
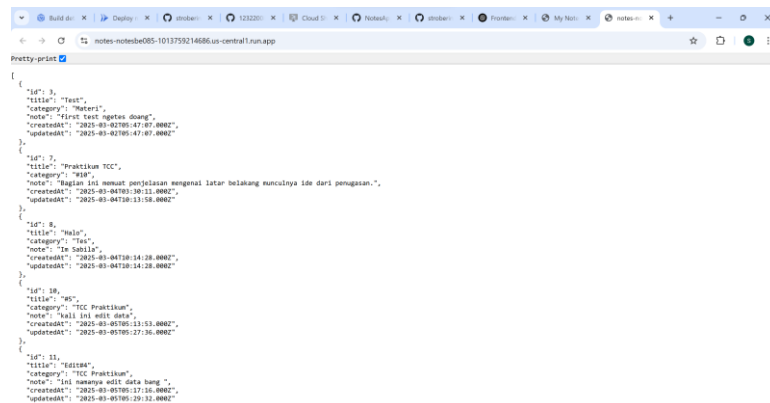


Figure 5. Step 3.2.2.5

6. Open the URL to Check is it Connect or Not

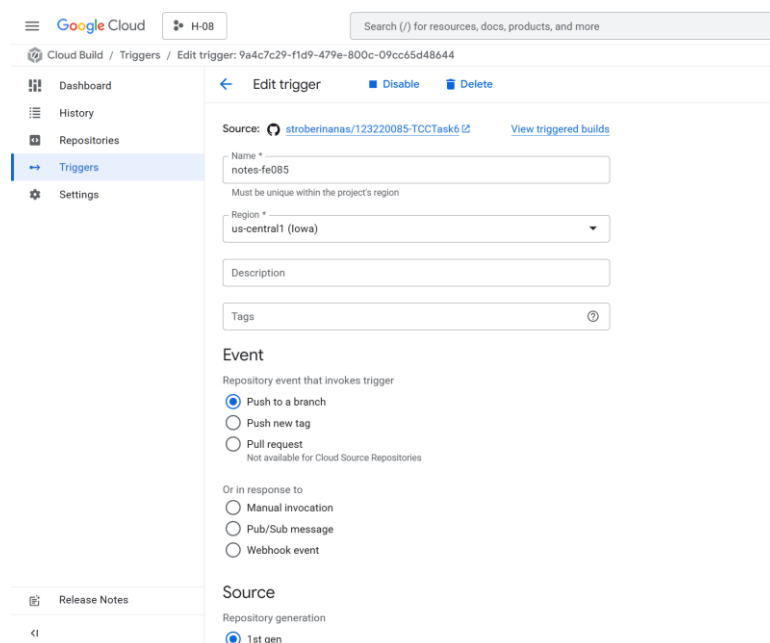


```
[{"id": 3, "title": "Test", "category": "Materi", "note": "Jring test notes doang", "createdat": "2025-03-02T05:47:07.000Z", "updatedat": "2025-03-02T05:47:07.000Z"}, {"id": 7, "title": "Praktikum TCC", "category": "PR", "note": "Bagian ini memuat penjelasan mengenai latar belakang munculnya ide dari penggunaan.", "createdat": "2025-03-04T03:10:11.000Z", "updatedat": "2025-03-04T03:11:58.000Z"}, {"id": 8, "title": "Halo", "category": "Tes", "note": "Ya Sabila", "createdat": "2025-03-04T03:14:19.000Z", "updatedat": "2025-03-04T03:14:19.000Z"}, {"id": 10, "title": "PS", "category": "TCC Praktikum", "note": "Halo ini edit data", "createdat": "2025-03-05T05:13:53.000Z", "updatedat": "2025-03-05T05:27:36.000Z"}, {"id": 11, "title": "EditData", "category": "TCC Praktikum", "note": "ini namanya edit data bang", "createdat": "2025-03-05T05:17:16.000Z", "updatedat": "2025-03-05T05:29:32.000Z"}]
```

Figure 6. Step 3.2.2.6

3.2.3 Config CI/CD Frontend Via Cloud Build

1. Create New Trigger for Frontend



Google Cloud H-08 Search (/) for resources, docs, products, and more

Cloud Build / Triggers / Edit trigger: 9a4c7c29-f1d9-479e-800c-09cc65d48644

Dashboard History Repositories **Triggers** Settings

← Edit trigger Disable Delete

Source: stroberinanas/123220085-TCCTask6 View triggered builds

Name * notes-fe085
Must be unique within the project's region

Region * us-central1 (Iowa)

Description

Tags

Event

Repository event that invokes trigger

☒ Push to a branch
☐ Push new tag
☐ Pull request
Not available for Cloud Source Repositories

Or in response to

☐ Manual invocation
☐ Pub/Sub message
☐ Webhook event

Source

Repository generation

☒ 1st gen

Release Notes

Figure 7. Step 3.2.3.1

2. Config Trigger and Connect to Repository GitHub that was connected before

Google Cloud

H-08

Search (/) for resources, docs, products, and more

Cloud Build / Triggers / Edit trigger: 9a4c7c29-f1d9-479e-800c-09cc65d48644

Dashboard

History

Repositories

Triggers

Settings

Edit trigger

Disable

Delete

Source

Repository generation

☒ 1st gen
 ☐ 2nd gen

Repository *

stroberinanas/123220085-TCCTask6 (GitHub App)

Select the repository to watch for events and clone when the trigger is invoked

Branch *

*main\$

Trigger only for a branch that matches the given regular expression [Learn more](#)

☐ Invert Regex

Matches the branch: main

Included files filter (glob)

frontend/** X glob pattern example: src/**

Changes affecting at least one included file will trigger builds

Ignored files filter (glob)

Changes only affecting ignored files won't trigger builds

[^ Hide included and ignored files filters](#)

Google Cloud

H-08

Search (/) for resources, docs, products, and more

Cloud Build / Triggers / Edit trigger: 9a4c7c29-f1d9-479e-800c-09cc65d48644

Dashboard

History

Repositories

Triggers

Settings

Edit trigger

Disable

Delete

Configuration

Type

☐ Autodetected

A cloudbuild.yaml or Dockerfile will be detected in the repository

☒ Cloud Build configuration file (yaml or json)
 ☐ Dockerfile
 ☐ Buildpacks

Location

☒ Repository

stroberinanas/123220085-TCCTask6 (GitHub App)

☐ Inline

Write inline YAML

Cloud Build configuration file location *

/ cloudbuild.frontend.yaml

Specify the path to a Cloud Build configuration file in the Git repo [Learn more](#)

Buildpacks

Buildpacks

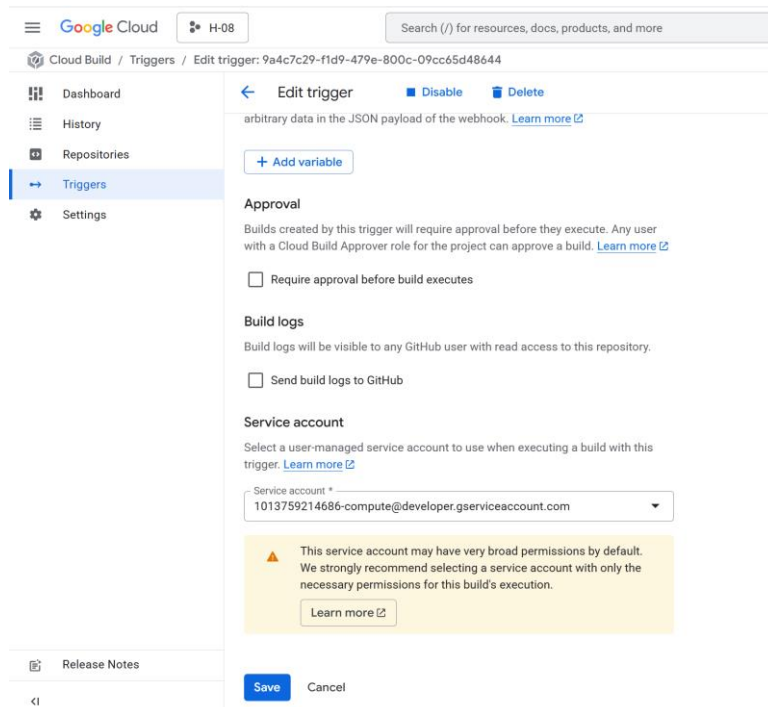


Figure 8. Step 3.2.3.2

3. Run Trigger

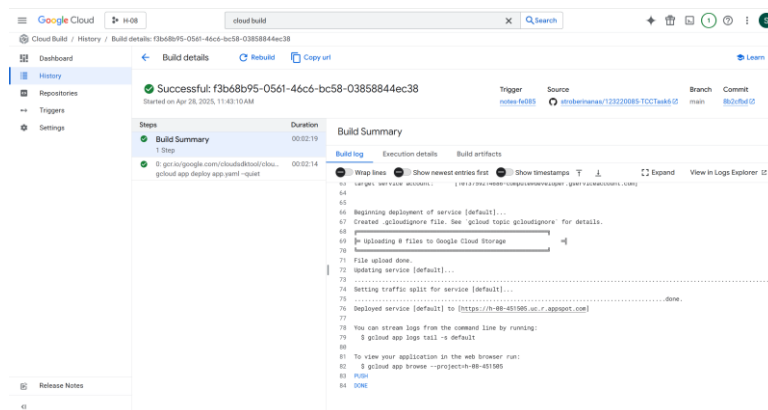


Figure 9. Step 3.2.3.3

4. Open the URL to Check Is it Connected or Not

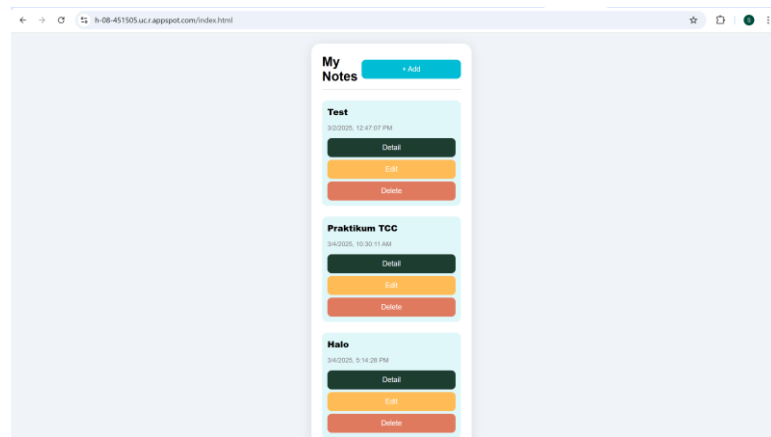


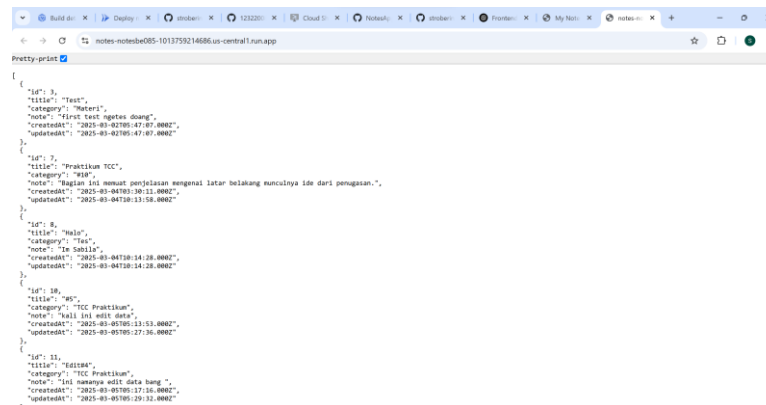
Figure 10. Step 3.2.3.4

CHAPTER IV

RESULTS AND DISCUSSION

4.1 Results

4.1.1 Deploying Back-End to Cloud Run & App Engine via CI/CD using Cloud Build



```
{
  "id": 3,
  "title": "Test",
  "category": "Materi",
  "note": "First test ngates doang",
  "createdAt": "2025-03-04T01:47:07.000Z",
  "updatedAt": "2025-03-04T01:47:07.000Z"
},
{
  "id": 3,
  "title": "Praktikum TCC",
  "category": "Materi",
  "note": "Bagian ini memuat penjelasan mengenai latar belakang munculnya ide dari program.",
  "createdAt": "2025-03-04T01:30:11.000Z",
  "updatedAt": "2025-03-04T01:31:08.000Z"
},
{
  "id": 8,
  "title": "Halo",
  "category": "Tas",
  "note": "Ini adalah tas",
  "createdAt": "2025-03-04T01:14:18.000Z",
  "updatedAt": "2025-03-04T01:14:18.000Z"
},
{
  "id": 10,
  "title": "Halo",
  "category": "TCC Praktikum",
  "note": "Ini adalah data",
  "createdAt": "2025-03-04T01:11:53.000Z",
  "updatedAt": "2025-03-04T01:27:36.000Z"
},
{
  "id": 11,
  "title": "Editasi",
  "category": "TCC Praktikum",
  "note": "Ini memuat edit data yang",
  "createdAt": "2025-03-04T01:17:10.000Z",
  "updatedAt": "2025-03-04T01:29:32.000Z"
}
```

Figure 11. Deploying BackEnd Successfully

The image above shows the successful deployment of the back-end service for the Notes application to Google Cloud Run and App Engine using a CI/CD pipeline with Cloud Build. The deployment was automatically triggered when the latest code was pushed to the main branch of the GitHub repository. The JSON response confirms that the API is live and functioning, serving stored note data through a public endpoint, validating the success of the automated back-end deployment process.

4.1.2 Deploying Front-End to Cloud Run & App Engine via CI/CD using Cloud Build.

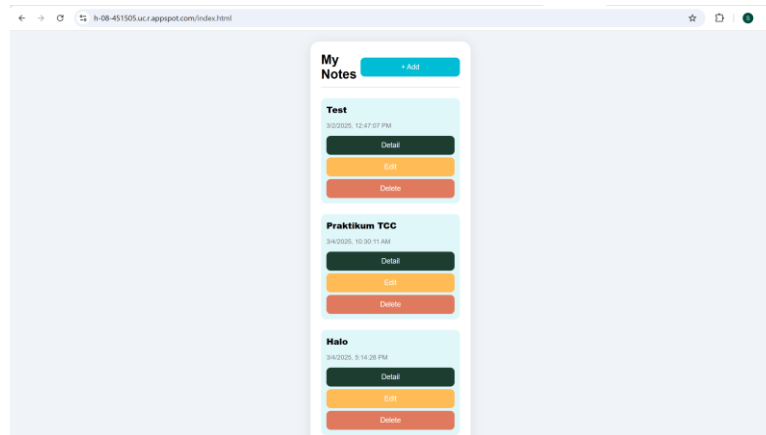


Figure 12. Deploying FrontEnd Successfully

The screenshot illustrates the successful deployment of the front-end interface of the Notes application to Google Cloud Run and App Engine via CI/CD using Cloud Build. Triggered by a code push to the main branch, the automated deployment ensured that the updated front-end was immediately available online. The interface displays dynamic note entries, demonstrating a fully operational connection to the back-end and confirming that the deployment pipeline executed as intended.

4.2 Discussion

The strategic decision to deploy front-end applications to Cloud Run and back-end services to App Engine leverages each platform's unique strengths—Cloud Run's container flexibility and cost-effective scaling for stateless components, and App Engine's managed environment for services requiring persistent resources. Implementing CI/CD through Cloud Build eliminates manual deployments, reduces human error, and accelerates the feedback cycle. However, this approach introduces complexity in configuration management across platforms and requires strict code review processes to ensure only properly tested changes reach production. Cost optimization benefits are available but require ongoing monitoring and tuning of resource allocations.

CHAPTER V

CLOSING

5.1 Conclusion

Overall, the deployment process successfully leveraged cloud and containerization technologies to build a scalable and portable application. Cloud Run facilitated efficient BE hosting without the need for infrastructure management, while Docker provided consistency in the FE environment. Despite the challenges faced, the final deployment ensures smooth application performance and proper interaction between BE and FE.

5.2 Suggestions

Organizations should implement a well-structured repository with clear separation between front-end and back-end code, hierarchical Cloud Build configurations, and least-privilege service accounts. Security should be prioritized through Secret Manager integration and vulnerability scanning at multiple pipeline stages. For reliability, implement environment parity with staging environments mirroring production, canary deployments for critical updates, and comprehensive observability through Google Cloud's operations suite. Advanced optimizations should include multi-level caching, regular resource adjustments based on usage patterns, and preview environments for feature branches, creating a data-driven development process focused on continuous improvement.

BIBLIOGRAPHY

- [1] Nguyen, H. T. (2023). A Comprehensive CI/CD Pipeline and Google Cloud Deployment for Web Application.
- [2] Ivanov, O. (2021). Development of CI/CD platform deployment automation module for group software development.
- [3] Prathvi Poojari. (2022, May 3). *CI/CD with Cloud Build for deploying applications in Cloud Run*. Medium; Searce. <https://blog.searce.com/ci-cd-with-cloud-build-for-deploying-applications-in-cloud-run-294b833c83a0>
- [4] *Deploying to App Engine*. (2024). Google Cloud. <https://cloud.google.com/build/docs/deploying-builds/deploy-appengine>
- [5] *Deploying frontend of an application using Google Cloud Run and Cloud Build from remote repository - Unanimous: Elevating Success Through Expert IT Solutions*. (2023, October 30). UnanimousTech: Elevating Success through Expert IT Solutions. <https://unanimoustech.com/2023/10/30/deploying-frontend-of-an-application-using-google-cloud-run-and-cloud-build-from-remote-repository/>
- [6] Dheeraj Panyam. (2024, April 5). *How To Set Up CI/CD Pipelines & Monitoring For Google App Engine (GAE)*. D3V Technology Solutions | Cloud Services | Google Cloud Partner. <https://www.d3vtech.com/insights/how-to-set-up-cicd-pipelines-monitoring-for-google-app-engine-gae/>

APPENDICES

- [1] <https://github.com/stroberinanas/123220085-TCCPract.git>
- [2] <https://github.com/stroberinanas/123220085-TCCTask6.git>