

LAPORAN PRAKTIKUM TEKNOLOGI CLOUD COMPUTING

**DEPLOYMENT FRONTEND & BACKEND KE CLOUD RUN DAN APP
ENGINE SECARA CI/CD DENGAN MEMANFAATKAN CLOUD BUILD
PRAKTIKUM TEKNOLOGI CLOUD COMPUTING IF - G**



Disusun oleh

Nama : Kaifa Ahlal Katamsyi
NIM : 123220006

**PROGRAM STUDI INFORMATIKA
JURUSAN INFORMATIKA
FAKULTAS TEKNIK INDUSTRI
UNIVERSITAS PEMBANGUNAN NASIONAL "VETERAN"
YOGYAKARTA
2025**

HALAMAN PENGESAHAN

LAPORAN PRAKTIKUM

DEPLOYMENT FRONTEND & BACKEND KE CLOUD RUN DAN APP ENGINE SECARA CI/CD DENGAN MEMANFAATKAN CLOUD BUILD PRAKTIKUM TEKNOLOGI CLOUD COMPUTING IF - G

Disusun Oleh:

Kaifa Ahlhal Katamsyi

123220006

Telah diperiksa dan disetujui oleh Asisten Praktikum teknologi Cloud Computing

Pada tanggal :

Menyetujui.

Asisten Praktikum

Asisten Praktikum

Muhammad Rafli
NIM 123210078

Raditya Haikal
NIM 123210062

KATA PENGANTAR

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan praktikum Teknologi Cloud Computing ini dengan baik. Laporan ini disusun sebagai bagian dari pelaksanaan praktikum dan bertujuan untuk mendokumentasikan hasil kerja yang telah dilakukan dalam pengembangan aplikasi Web Service dengan tema Notes.

Penulis mengucapkan terima kasih kepada Bapak/Ibu Dosen pengampu mata kuliah Teknologi Cloud Computing, serta para Asisten Praktikum yang telah membimbing dan memberikan arahan selama pelaksanaan praktikum. Ucapan terima kasih juga penulis sampaikan kepada teman-teman yang telah membantu dalam penyelesaian tugas ini.

Semoga laporan ini dapat memberikan manfaat bagi pembaca dan dapat menjadi referensi dalam pengembangan sistem berbasis cloud computing. Penulis menyadari bahwa laporan ini masih jauh dari sempurna, oleh karena itu, kritik dan saran yang membangun sangat diharapkan.

Yogyakarta, 15 April 2025

Penulis

DAFTAR ISI

HALAMAN PENGESAHAN	2
HALAMAN PERSETUJUAN	2
KATA PENGANTAR	3
DAFTAR ISI	4
DAFTAR GAMBAR	5
BAB I	
PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	2
1.4 Manfaat	2
BAB II	
TINJAUAN LITERATUR	3
2.1 Google Cloud Run	3
2.2 Google Cloud Build	3
2.3 google App Engine	3
2.4 Continuous Integration dan Continuous Deployment (CI/CD)	4
BAB III	
METODOLOGI	5
3.1 Analisis Permasalahan	5
3.2 Perancangan Solusi	5
3.2.1 Konfigurasi CI/CD backend via Cloud Build	6
3.2.2 Deployment Frontend ke App Engine via Cloud Build	10
BAB IV	
HASIL DAN PEMBAHASAN	15
4.1 Hasil	15
4.2 Pembahasan	19
BAB V	
PENUTUP	20
5.1 Kesimpulan	20
5.2 Saran	21
DAFTAR PUSTAKA	22

DAFTAR GAMBAR

Gambar 3.2.1.1 konfigurasi cloudbuild.yaml (backend)	6
Gambar 3.2.1.2 konfigurasi URL Cloudrun ke frontend	7
Gambar 3.2.1.3 Buat Trigger backend	8
Gambar 3.2.1.4 Hubungkan trigger ke Repository Github	8
Gambar 3.2.1.5 Atur ke folder backend di branch main	9
Gambar 3.2.1.6 konfigurasi ke cloudbuild.yaml backend	9
Gambar 3.2.1.7 Gunakan service account Developer	10
Gambar 3.2.2.1 Membuat App. yaml	10
Gambar 3.2.2.2 Membuat Cloudbuild.Yaml frontend	11
Gambar 3.2.2.3 Buat Trigger Frontend	12
Gambar 3.2.2.4 Hubungkan trigger ke Repository Github	12
Gambar 3.2.2.5 Atur ke folder frontend di branch main	13
Gambar 3.2.2.6 konfigurasi ke cloudbuild.yaml frontend	13
Gambar 3.2.2.7 Gunakan service account Developer	14
Gambar 4.1.1 Run Trigger Backend	15
Gambar 4.1.2 isi backend yang diakses melalui URL	15
Gambar 4.1.3 Run trigger backend	16
Gambar 4.1.4 frontend berhasil di deploy ke app engine	16
Gambar 4.1.5 frontend menampilkan data catatan	17
Gambar 4.1.6 Tampilan Frontend Sebelum Diganti	18
Gambar 4.1.7 Ubah code di Github	18
Gambar 4.1.8 Tampilan Frontend Sesudah Diganti	18

BAB I

PENDAHULUAN

1.1 Latar Belakang

Seiring dengan meningkatnya kebutuhan akan efisiensi dan kecepatan dalam pengembangan perangkat lunak, penerapan otomatisasi dalam proses deployment menjadi aspek yang krusial. Pada tugas sebelumnya, proses deployment aplikasi baik frontend maupun backend masih dilakukan secara manual dengan bantuan Docker dan Cloud Shell. Meskipun pendekatan tersebut memberikan hasil yang fungsional, metode manual memiliki sejumlah keterbatasan, terutama dalam hal konsistensi, skalabilitas, serta potensi terjadinya kesalahan operasional.

Menindaklanjuti hal tersebut, pada tugas ini diimplementasikan mekanisme Continuous Integration dan Continuous Deployment (CI/CD) dengan memanfaatkan layanan Cloud Build dari Google Cloud Platform (GCP). Strategi ini bertujuan untuk mengotomatiskan seluruh proses build dan deployment, yang akan secara langsung dijalankan setiap kali terjadi perubahan pada kode sumber di branch utama repository GitHub.

Dalam implementasi ini, komponen backend dikemas dalam container dan dideploy secara otomatis ke Cloud Run, yang menyediakan lingkungan serverless dengan kemampuan skalabilitas tinggi dan manajemen infrastruktur yang minimal. Sementara itu, frontend dideploy ke App Engine, yang memungkinkan penyajian konten statis secara optimal melalui pendekatan Platform as a Service (PaaS).

Dengan pendekatan CI/CD ini, proses pengembangan dan pengelolaan aplikasi menjadi lebih efisien, terstruktur, dan andal. Otomatisasi ini juga mendukung praktik DevOps yang mendorong kolaborasi antara pengembangan dan operasional, serta memastikan bahwa aplikasi dapat dikelola dan dikembangkan secara berkelanjutan dan profesional.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan, terdapat beberapa permasalahan yang ingin diselesaikan dalam praktikum ini:

1. Bagaimana cara mengonfigurasi Cloud Build agar dapat melakukan build dan deployment backend ke Cloud Run secara otomatis?
2. Bagaimana mengatur pipeline CI/CD untuk frontend agar dapat ter-deploy ke App Engine saat terjadi perubahan di branch utama GitHub?
3. Bagaimana menyusun file konfigurasi `cloudbuild.yaml` yang mendukung proses CI/CD untuk masing-masing layanan (frontend dan backend)?

1.3 Tujuan

Tujuan dari praktikum ini adalah:

1. Menerapkan pipeline CI/CD menggunakan Cloud Build untuk mengelola proses build dan deployment backend ke Cloud Run secara otomatis.
2. Mengkonfigurasi deployment frontend ke App Engine melalui trigger otomatis berbasis push ke branch utama GitHub.
3. Menyusun dan mengintegrasikan file konfigurasi `cloudbuild.yaml` yang sesuai untuk kedua komponen aplikasi (frontend dan backend).
4. Memahami alur kerja CI/CD serta manfaatnya dalam mendukung efisiensi, kecepatan, dan konsistensi proses deployment aplikasi berbasis cloud.

1.4 Manfaat

Adapun manfaat yang diharapkan dari tugas praktikum ini adalah :

1. Memberikan pengalaman praktis dalam mengimplementasikan CI/CD menggunakan layanan Cloud Build.
2. Meningkatkan efisiensi dan reliabilitas dalam proses deployment aplikasi melalui otomasi.
3. Meminimalkan kesalahan manusia (human error) dalam pengelolaan dan penyebaran aplikasi ke cloud.
4. Mendukung pengembangan aplikasi yang berkelanjutan dan lebih profesional dengan pendekatan DevOps.

BAB II

TINJAUAN LITERATUR

2.1 Google Cloud Run

Google Cloud Run adalah layanan komputasi terkelola yang memungkinkan pengguna menjalankan container tanpa perlu mengelola server. Cloud Run secara otomatis menangani penskalaan, sehingga aplikasi dapat berjalan secara efisien dengan penggunaan sumber daya yang optimal. Dengan dukungan penuh untuk container berbasis Docker, Cloud Run menjadi solusi ideal untuk deployment backend dalam lingkungan cloud.

2.2 Google Cloud Build

Google Cloud Build adalah layanan dari Google Cloud Platform yang digunakan untuk melakukan proses Continuous Integration dan Continuous Delivery (CI/CD). Layanan ini memungkinkan pengguna untuk secara otomatis membangun, menguji, dan mendeploy aplikasi setiap kali terjadi perubahan pada repositori kode sumber, seperti GitHub atau GitLab. Cloud Build dapat menjalankan pipeline berdasarkan file konfigurasi (misalnya cloudbuild.yaml) yang berisi langkah-langkah spesifik untuk proses build dan deployment, mendukung berbagai bahasa dan framework secara fleksibel.

2.3 google App Engine

Google App Engine adalah platform serverless yang mendukung pengembangan dan deployment aplikasi web tanpa harus mengelola server. App Engine menyediakan dua jenis environment: Standard dan Flexible. Untuk aplikasi statis seperti front-end HTML/CSS/JS, App Engine Standard Environment memungkinkan penyajian cepat dengan konfigurasi minimal. App Engine juga mendukung deployment otomatis melalui integrasi dengan Cloud Build dan GitHub, memudahkan implementasi CI/CD.

2.4 Continuous Integration dan Continuous Deployment (CI/CD)

CI/CD adalah pendekatan DevOps yang mengotomatisasi integrasi kode dan proses deployment ke lingkungan produksi. Continuous Integration memastikan setiap perubahan kode yang dilakukan oleh pengembang secara rutin digabung dan diuji secara otomatis. Sementara itu, Continuous Deployment memastikan perubahan tersebut di-deploy ke lingkungan produksi dengan minim intervensi manual. CI/CD meningkatkan efisiensi pengembangan, mengurangi kesalahan manusia, dan mempercepat waktu rilis aplikasi.

BAB III

METODOLOGI

3.1 Analisis Permasalahan

Pada tugas sebelumnya, proses deployment dilakukan secara manual menggunakan Dockerfile dan dijalankan melalui Cloud Shell. Pendekatan tersebut meskipun efektif untuk memahami konsep dasar container dan deployment, namun masih memiliki keterbatasan dari segi efisiensi dan skalabilitas. Setiap kali ada perubahan kode, pengembang perlu melakukan build dan deploy ulang secara manual, yang membuka peluang kesalahan konfigurasi dan keterlambatan dalam proses pengembangan.

Masalah utama yang diidentifikasi dalam tugas ini adalah:

- Proses deployment backend dan frontend belum otomatis, sehingga tidak mendukung pengembangan berkelanjutan (CI/CD).
- Belum terdapat integrasi antara platform version control seperti GitHub dengan layanan deployment di Google Cloud.
- Proses deployment frontend dan backend tidak terpisah dan tidak responsif terhadap setiap perubahan pada masing-masing komponen.

Untuk menjawab tantangan tersebut, pada tugas ini dilakukan penerapan CI/CD dengan menggunakan Cloud Build untuk mengotomatisasi proses deployment frontend dan backend ke Cloud Run dan App Engine, berdasarkan setiap perubahan kode yang masuk ke branch utama GitHub.

3.2 Perancangan Solusi

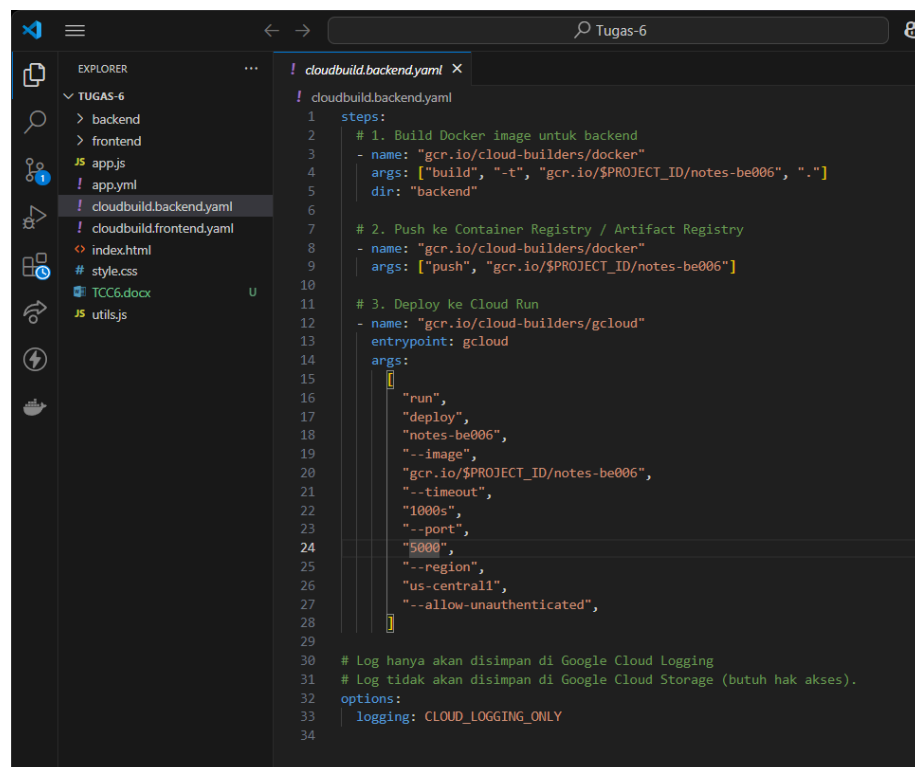
Untuk meningkatkan efisiensi deployment dan menjawab keterbatasan dari proses manual pada tugas sebelumnya, pada tugas ini dilakukan implementasi pipeline Continuous Integration dan Continuous Deployment (CI/CD) dengan memanfaatkan Cloud Build. Proses ini memungkinkan backend dan frontend untuk langsung ter-deploy ke Cloud Run dan App Engine secara otomatis setiap kali terjadi perubahan pada branch utama di GitHub.

Solusi ini dirancang dengan pendekatan berbasis konfigurasi file `cloudbuild.yaml` dan pemanfaatan trigger build dari Cloud Build. Tidak lagi menggunakan Dockerfile seperti pada tugas sebelumnya, karena build dan deployment dilakukan secara native oleh Cloud Build berdasarkan runtime environment dari GCP.

3.2.1 Konfigurasi CI/CD backend via Cloud Build

Langkah-langkah yang dilakukan untuk menerapkan CI/CD pada backend adalah sebagai berikut:

1. Membuat file konfigurasi `cloudbuild.yaml` pada direktori `/backend` untuk mendefinisikan proses build dan deploy ke Cloud Run.



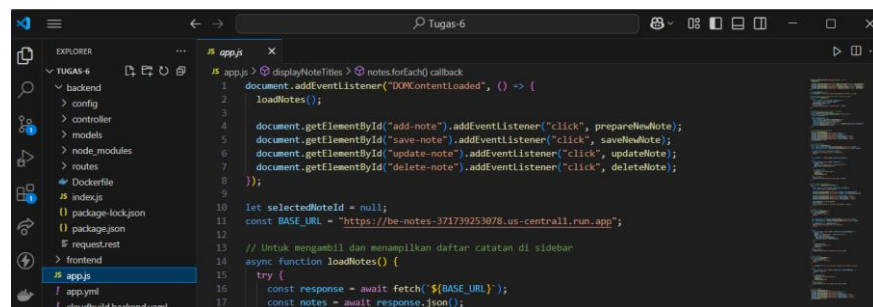
Gambar 3.2.1.1 konfigurasi `cloudbuild.yaml` (backend)

Konfigurasi pada file `cloudbuild.backend.yaml` dimulai dengan langkah pertama yaitu membangun image Docker dari source code yang berada di direktori `backend`. Proses ini menggunakan builder resmi Docker (*`gcr.io/cloud-builders/docker`*) dengan perintah `docker build` untuk membuat image dan menamainya dengan path registry

gcr.io/\$PROJECT_ID/notes. Langkah berikutnya adalah melakukan push terhadap image yang telah dibuat ke Google Container Registry atau Artifact Registry. Setelah image tersedia di registry, tahap ketiga adalah melakukan deployment ke layanan Cloud Run menggunakan perintah `gcloud run deploy`. Service backend diberi nama `notes`, dengan image yang berasal dari registry, dijalankan pada port 5000, dan dikonfigurasi agar dapat diakses secara publik tanpa autentikasi. Lokasi region yang digunakan dalam konfigurasi ini adalah `us-central1`. Seluruh log proses hanya akan dicatat melalui Google Cloud Logging tanpa penyimpanan tambahan ke Google Cloud Storage, untuk menghindari kebutuhan hak akses tambahan.

setelah itu lakukan deploy dengan membuat trigger di cloud build, setelah di deploy cloudbuild akan menghasilkan URL could run, dan Url tersebut akan dimasukan ke frontend Secara manual.

2. Setelah deployment, URL dari service Cloud Run dicatat untuk dikonfigurasi pada frontend.



```
1 document.addEventListener("DOMContentLoaded", () => {
2   loadNotes();
3
4   document.getElementById("add-note").addEventListener("click", prepareNewNote);
5   document.getElementById("save-note").addEventListener("click", saveNewNote);
6   document.getElementById("update-note").addEventListener("click", updateNote);
7   document.getElementById("delete-note").addEventListener("click", deleteNote);
8 });
9
10 let selectedNoteId = null;
11 const BASE_URL = "https://be-notes-371739253078.us-central1.run.app";
12
13 // Untuk mengambil dan menampilkan daftar catatan di sidebar
14 async function loadNotes() {
15   try {
16     const response = await fetch(`${BASE_URL}`);
17     const notes = await response.json();
```

Gambar 3.2.1.2 konfigurasi URL Cloudrun ke frontend

3. *Create* dan atur trigger di Cloud Build agar setiap push ke branch utama (main) pada folder backend/, dan langsung menjalankan

cloudbuild.backend.yaml. Namun sebelum itu perlu menghubungkan Repositori Github terlebih dahulu.

Google Cloud G-11 Search (/) for resources, docs, products, and more

Cloud Build / Triggers / Edit trigger: d336c3cb-0f0b-443c-a2bc-85ef840ee2d6

Dashboard
History
Repositories
Triggers
Settings

← Edit trigger Disable Delete

Source: [katamsyi/Teknologi-Cloud-Computing](#) [View triggered builds](#)

Name *
tugas6-006
Must be unique within the project's region

Region *
global (Global)

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

Gambar 3.2.1.3 Buat Trigger backend

Source

Repository generation

- ☒ 1st gen
☐ 2nd gen

Repository *
katamsyi/Teknologi-Cloud-Computing (GitHub App)

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

Branch *
^tugas-3\$

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

Gambar 3.2.1.4 Hubungkan trigger ke Repository Github

Branch *

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

☐ Invert Regex

Matches the branch: tugas-3

Included files filter (glob) 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

Gambar 3.2.1.5 Atur ke folder backend di branch tugas-3

Location

☒ Repository
katamsyi/Teknologi-Cloud-Computing (GitHub App)

☐ Inline
Write inline YAML

Cloud Build configuration file location *

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

Advanced

Gambar 3.2.1.6 konfigurasi ke cloudbuild.backend.yaml

Approval

Builds created by this trigger will require approval before they execute. Any user with a Cloud Build Approver role for the project can approve a build. [Learn more](#)

☐ Require approval before build executes

Build logs

Build logs will be visible to any GitHub user with read access to this repository.

☐ Send build logs to GitHub

Service account

Select a user-managed service account to use when executing a build with this trigger. [Learn more](#)

Service account *
371739253078-compute@developer.gserviceaccount.com



This service account may have very broad permissions by default. We strongly recommend selecting a service account with only the necessary permissions for this build's execution.

[Learn more](#)

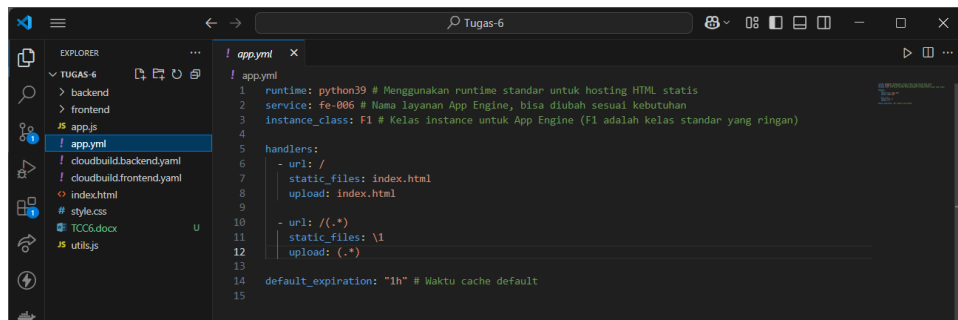
Gambar 3.2.1.7 Gunakan service account Developer

proses pengaturan Trigger sudah selesai, lanjut buat trigger dengan tekan button *Create*, Dengan Begitu Trigger Backend sudah selesai dibuat

3.2.2 Deployment Frontend ke App Engine via Cloud Build

Pada bagian frontend, proses deployment dilakukan secara otomatis ke App Engine dengan memanfaatkan Cloud Build dan file konfigurasi **cloudbuild.frontend.yaml**. Proses ini menjadi bagian dari pipeline CI/CD yang akan berjalan secara otomatis setiap kali terjadi perubahan pada source code di branch utama. langkah langkahnya sebagai berikut :

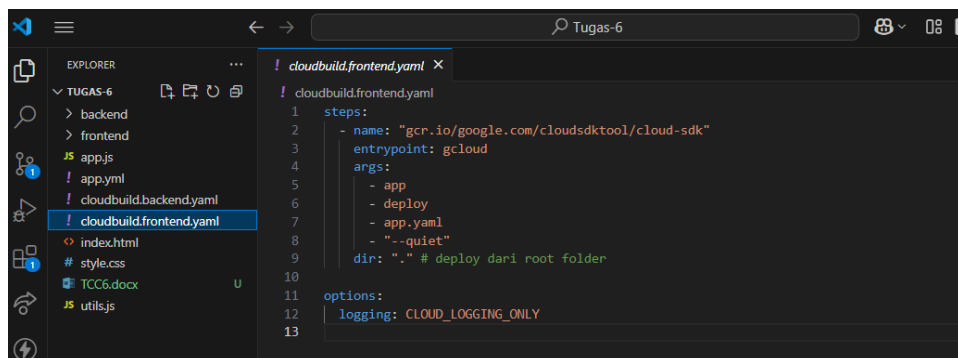
1. Membuat file app.yaml di direktori /frontend, yang berisi konfigurasi runtime dan pengaturan App Engine.



Gambar 3.2.2.1 Membuat App. yaml

File ini menggunakan runtime Python 3.9 yang mendukung penyajian file statis seperti HTML, CSS, dan JavaScript. Layanan diberi nama fe-006, dijalankan pada instance kelas F1 yang ringan, serta menggunakan handlers untuk mengatur agar seluruh file statis dapat dilayani dengan benar — dengan index.html sebagai beranda utama dan pola upload untuk file lainnya. Pengaturan default_expiration selama satu jam diterapkan untuk efisiensi caching.

2. Menyusun file cloudbuild.yaml pada folder /frontend untuk mendefinisikan langkah build dan deploy frontend secara otomatis.



Gambar 3.2.2.2 Membuat Cloudbuild.Yaml frontend

file cloudbuild.frontend.yaml digunakan untuk deployment frontend ke App Engine menggunakan mekanisme CI/CD yang lebih sederhana tanpa membangun Docker image. Pada file ini, langkah utama adalah menjalankan perintah gcloud app deploy yang akan membaca file app.yaml

di dalam direktori frontend. App Engine akan menggunakan konfigurasi tersebut untuk meng-host file statis frontend seperti HTML, CSS, dan JavaScript. Dengan menggunakan `--quiet`, proses deployment berlangsung secara non-interaktif. Seluruh aktivitas log juga diarahkan langsung ke Cloud Logging. Dengan pendekatan ini, setiap perubahan pada kode frontend yang di-push ke branch utama akan secara otomatis ter-deploy ke App Engine, memungkinkan pengembangan dan update antarmuka pengguna secara cepat, efisien, dan terpusat.

3. *Create* dan atur Trigger Cloud Build kemudian agar berjalan secara otomatis ketika terjadi push ke folder frontend/ pada branch utama. Dengan ini, deployment frontend dilakukan tanpa perlu campur tangan manual, menjadikan proses pengembangan aplikasi lebih cepat, efisien, dan mudah diatur. (prosesnya sama dengan pada saat create trigger backend)

History
Repositories
Triggers
Settings

Source: [katamsyl/Teknologi-Cloud-Computing](#) [View triggered builds](#)

Name *
tugas6-006-frontend
Must be unique within the project's region

Region *
global (Global)

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

Release Notes

Gambar 3.2.2.3 Buat Trigger Frontend

Source

Repository generation

☒ 1st gen

☐ 2nd gen

Repository *

katamsyi/Teknologi-Cloud-Computing (GitHub App)

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

Branch *

^tugas-3\$

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

☐ Invert Regex

Matches the branch: tugas-3

Included files filter (glob)

** ✕ 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](#)

Gambar 3.2.2.4 Hubungkan trigger ke Repository Github

Source

Repository generation

☒ 1st gen

☐ 2nd gen

Repository *
katamsyi/Teknologi-Cloud-Computing (GitHub App) ▼

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

Branch *
^tugas-3\$

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

☐ Invert Regex

Matches the branch: tugas-3

Included files filter (glob)
** 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](#)

Gambar 3.2.2.5 Atur ke folder frontend di branch tugas-3

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

katamsyi/Teknologi-Cloud-Computing (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](#)

Gambar 3.2.2.6 konfigurasi ke cloudbuild.frontend.yaml

Approval

Builds created by this trigger will require approval before they execute. Any user with a Cloud Build Approver role for the project can approve a build. [Learn more](#)

☐ Require approval before build executes

Build logs

Build logs will be visible to any GitHub user with read access to this repository.

☐ Send build logs to GitHub

Service account

Select a user-managed service account to use when executing a build with this trigger. [Learn more](#)

Service account *
371739253078-compute@developer.gserviceaccount.com



This service account may have very broad permissions by default. We strongly recommend selecting a service account with only the necessary permissions for this build's execution.

[Learn more](#)

Gambar 3.2.2.7 Gunakan service account Developer

proses pengaturan Trigger sudah selesai, lanjut buat trigger dengan tekan button *Create*, Dengan Begitu Trigger Frontend sudah selesai dibuat

BAB IV

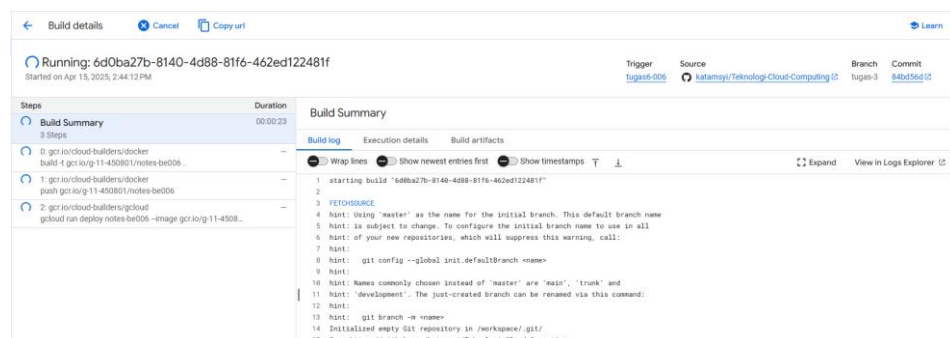
HASIL DAN PEMBAHASAN

4.1 Hasil

Implementasi CI/CD deployment pada aplikasi Notes berhasil dilakukan dengan memanfaatkan layanan Google Cloud Build, Cloud Run, dan App Engine. Proses deployment backend dan frontend dapat berjalan secara otomatis setelah perubahan di-push ke branch utama GitHub, sesuai dengan konfigurasi dan trigger yang telah ditetapkan.

1. Backend berhasil Di-deploy ke Cloud Run

Backend aplikasi Notes berhasil di-deploy ke Cloud Run dan dapat diakses melalui URL yang disediakan oleh Cloud Run dengan endpoint /user. Namun Sebelum itu Kita perlu Run trigger backend yang sudah dibuat.



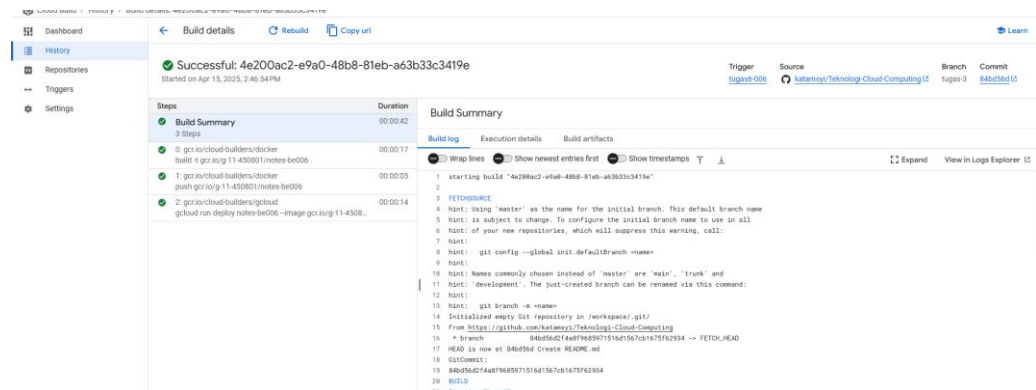
Gambar 4.1.1 Run Trigger Backend



Gambar 4.1.2 isi backend yang diakses melalui URL

2. Frontend Berjalan Otomatis ke App Engine.

Pada sisi frontend, pipeline juga berjalan otomatis berdasarkan file `cloudbuild.frontend.yaml`. Tanpa proses build Docker, Cloud Build langsung menjalankan perintah `gcloud app deploy`, yang membaca konfigurasi pada `app.yaml`. Hasilnya, file frontend berhasil dideploy ke App Engine dan tersedia melalui domain default App Engine.



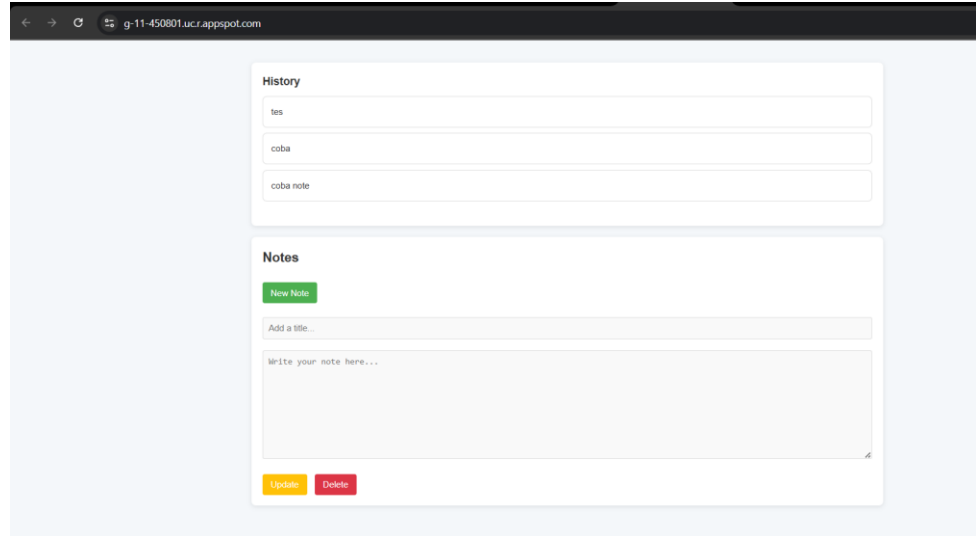
Gambar 4.1.3 Run trigger backend

Services Delete Edit Ingress Setting							
<input type="checkbox"/> Service	Versions	Labels	Dispatch routes	Ingress	VPC access name	VPC egress setting	Last version deployed
<input type="checkbox"/> default	6			All			Apr 15, 2025, 2:50:27 PM by 371739253078-compute@developer.gserviceaccount.com

Gambar 4.1.4 frontend berhasil di deploy ke app engine

3. Koneksi Antara Frontend dan Backend Berhasil

Frontend yang berjalan di App Engine berhasil melakukan request API ke backend yang dihosting di Cloud Run. Ini menunjukkan bahwa komunikasi antar layanan yang berjalan di lingkungan terpisah (App Engine dan Cloud Run) berlangsung dengan baik. URL backend diatur sesuai dengan endpoint publik dari Cloud Run.



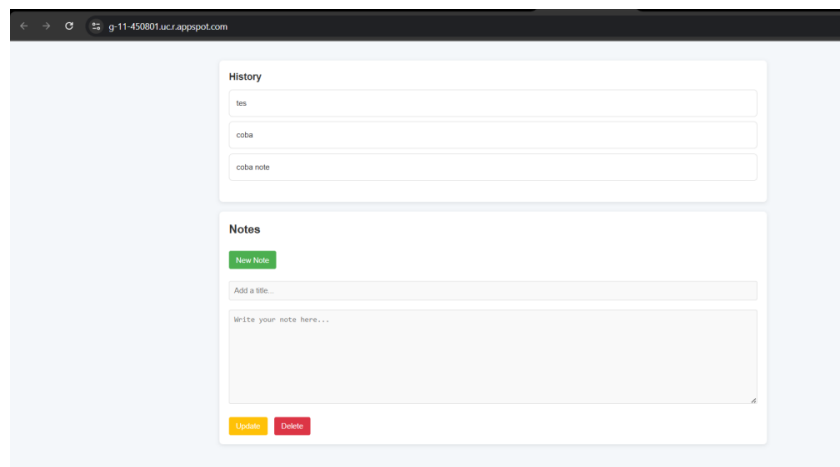
Gambar 4.1.5 frontend menampilkan data catatan

Terlihat jika Front end Dapat mengambil dan menampilkan data catatan dari backend

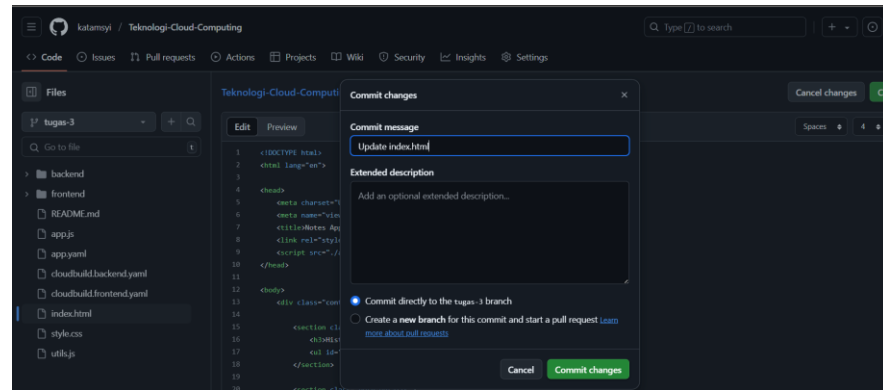
4. CI/CD berhasil diterapkan

Proses CI/CD pada proyek ini berhasil diterapkan dengan baik, dibuktikan dengan otomatisasinya alur build dan deployment setiap kali terjadi perubahan kode di repository GitHub. Dengan memanfaatkan layanan Cloud Build, aplikasi secara otomatis melakukan build ulang dan deployment baik untuk backend ke Cloud Run maupun frontend ke App Engine.

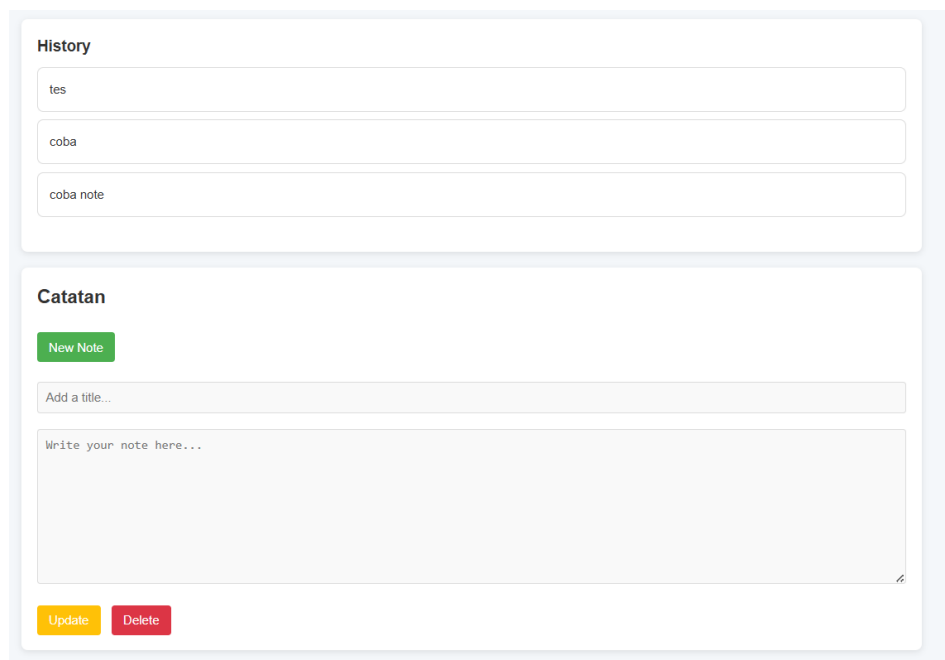
untuk pengetesan disini saya mencoba mengganti bagian frontend yang awalnya tertulis “Notes”, diubah menjadi “Catatan”.



Gambar 4.1.6 Tampilan Frontend Sebelum Diganti



Gambar 4.1.7 Ubah code di Github



Gambar 4.1.8 Tampilan Frontend Sesudah Diganti

Dapat terlihat Judul pada Frontend yang awalnya “Notes” berubah menjadi “Catatan”, dengan ini dapat disimpulkan CI/CD berhasil Diterapkan.

4.2 Pembahasan

Hasil praktikum ini menunjukkan bagaimana pendekatan modern dalam deployment aplikasi dapat dioptimalkan menggunakan layanan cloud dan automasi CI/CD. Berbeda dari pendekatan manual pada tugas sebelumnya, kali ini proses deployment dibuat otomatis, lebih efisien, dan minim potensi kesalahan manusia.

Penggunaan Cloud Build memungkinkan proses deployment backend dan frontend berjalan secara otomatis setiap kali terdapat perubahan pada source code. Hal ini memberikan manfaat besar dalam konteks pengembangan berkelanjutan (continuous delivery), karena developer cukup melakukan push ke GitHub, dan pipeline akan menangani seluruh proses build hingga deploy.

Backend aplikasi dideploy ke Cloud Run, platform serverless yang dapat menyesuaikan skala secara otomatis tanpa perlu mengelola server. Sementara frontend di-deploy ke App Engine, layanan GCP yang dirancang untuk menjalankan aplikasi web statis secara efisien. Pemisahan ini menunjukkan bagaimana arsitektur modern dapat menggabungkan berbagai layanan cloud sesuai kebutuhan aplikasi.

Dengan adanya file konfigurasi cloudbuild.yaml dan app.yaml, seluruh proses deployment menjadi transparan, terstandarisasi, dan mudah untuk dikelola ulang atau diperluas. Keberhasilan integrasi antara frontend dan backend juga menunjukkan bahwa komunikasi antar layanan yang berjalan di platform berbeda dapat dilakukan dengan lancar melalui konfigurasi endpoint yang tepat.

Secara keseluruhan, praktikum ini berhasil menunjukkan penerapan CI/CD yang efektif dan modern untuk mengelola aplikasi berbasis web, serta mengilustrasikan kekuatan layanan Google Cloud dalam mendukung skenario deployment aplikasi yang profesional dan skalabel.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan implementasi dan pengujian yang telah dilakukan, proses deployment aplikasi Notes berhasil diotomatisasi secara menyeluruh menggunakan layanan Google Cloud. Backend berhasil dideploy ke Cloud Run dan frontend ke App Engine melalui pipeline CI/CD yang dikonfigurasi dengan Cloud Build. Hal ini membuktikan bahwa arsitektur aplikasi terpisah (decoupled) antara frontend dan backend dapat dikelola dan di-deploy secara efisien dengan memanfaatkan layanan cloud-native.

Cloud Run terbukti efektif dalam menangani backend karena kemampuannya untuk otomatis scale berdasarkan trafik, serta kemudahan integrasi dengan container image dari Artifact Registry. Sementara App Engine menjadi solusi ideal untuk deployment frontend statis karena kesederhanaan konfigurasinya melalui app.yaml, serta kemudahan akses tanpa perlu pengelolaan infrastruktur.

Penggunaan Cloud Build sebagai inti dari pipeline CI/CD memungkinkan proses deployment menjadi lebih cepat, konsisten, dan minim intervensi manual. Setiap perubahan pada kode sumber yang dikirimkan ke branch utama akan secara otomatis terdeteksi dan dideploy sesuai konfigurasi, memastikan aplikasi selalu berada dalam kondisi terbaru dan siap digunakan.

5.2 Saran

Untuk pengembangan lebih lanjut dan penerapan skala produksi, beberapa hal berikut dapat dipertimbangkan:

1. Penerapan Tahap Testing dalam CI/CD

Untuk meningkatkan keandalan, sebaiknya pipeline CI/CD tidak hanya fokus pada build dan deploy, tetapi juga menambahkan tahap automated

testing (unit test, integration test) sebelum deployment dilakukan. Ini akan mencegah kesalahan muncul di produksi.

2. **Penerapan Versi dan Rollback**

Manfaatkan fitur *revision control* di Cloud Run dan *versioning* di App Engine agar lebih mudah melakukan rollback jika terdapat kesalahan pada versi terbaru aplikasi.

3. **Monitoring dan Logging Lanjutan**

Gunakan Google Cloud Operations Suite (d/h Stackdriver) untuk memantau performa aplikasi secara real-time, serta menangani error dan performa lambat sebelum berdampak ke pengguna.

4. **Penggunaan Secrets Manager**

Simpan variabel sensitif seperti API key atau kredensial database di Google Secret Manager agar lebih aman dan tidak tersimpan secara hardcoded dalam source code.

5. **Pemanfaatan Infrastructure as Code (IaC)**

Pertimbangkan penggunaan Terraform atau Deployment Manager untuk mengelola infrastruktur secara terstandarisasi dan *reproducible*, terutama jika proyek berkembang menjadi lebih kompleks.

DAFTAR PUSTAKA

Google Cloud. (2024). Cloud Build Documentation. Diakses dari <https://cloud.google.com/build/docs>

Google Cloud. (2024). Cloud Run Documentation. Diakses dari <https://cloud.google.com/run/docs>

Google Cloud. (2024). App Engine Standard Environment Documentation. Diakses dari <https://cloud.google.com/appengine/docs/standard>

Google Cloud. (2024). Deploying from GitHub using Cloud Build triggers. Diakses dari <https://cloud.google.com/build/docs/automating-builds/github>

Google Cloud. (2025). Cloud Logging Overview. Diakses dari <https://cloud.google.com/logging/docs>

Google Cloud. (2025). Service Configuration (app.yaml) for App Engine. Diakses dari <https://cloud.google.com/appengine/docs/standard/python3/config/appref>

GitHub Docs. (2025). Managing GitHub Actions and CI/CD pipelines. Diakses dari <https://docs.github.com/en/actions>