JURNAL FASILKOM
Volume 14 No. 2 | Agustus 2024: 346-354

P-ISSN : 2089-3353
E-ISSN : 2808-9162

Integrasi *Infrastructure as Code* dengan *Continuous Integration/Continuous Deployment* di *Google Cloud Platform*

Wayan Deden Setyawan[1*], I Nyoman Piarsa[2], Putu Wira Buana[3]
[1,2,3]Program Studi Teknologi Informasi, Fakultas Teknik, Universitas Udayana
[1]deden.setyawan086@student.unud.ac.id*, [2]manpits@unud.ac.id, [3]wbhuana@it.unud.ac.id

Abstract

The integration of IaC (Infrastructure as Code) with CI/CD (Continuous Integration/Continuous Deployment) helps in producing software that has high quality and productivity. Tests carried out in the form of comparing manual and automatic deployment of infrastructure and assessing the effectiveness and efficiency of the deployment method. Testing to ensure that the infrastructure created is running properly is to do a simple application deployment. The research starts from creating an infrastructure design, configuring IaC Terraform, creating CI/CD scripts, deploying infrastructure manually and automatically, configuring applications along with CI/CD, and deploying applications. The average time required for the manual method is 13 minutes 34 seconds, while the average time required for the automatic method is 14 minutes 5 seconds. The effectiveness value obtained shows that both methods are successful in deploying infrastructure, while for the efficiency value, manual and automatic methods both have their own advantages and disadvantages. The Go - Gin application that was deployed took an average of 3 minutes 7 seconds, while the PHP - Laravel application took an average of 3 minutes 8 seconds. Applications that are deployed on the same infrastructure show a difference in time that is not much different and the difference in application configuration obtained is in the Dockerfile and the Kubernetes object file used.

Keywords: Continuous Integration/Continuous Deployment, Dockerfile, Infrastructure as Code, Kubernetes Object, Terraform

Abstrak

Integrasi IaC (*Infrastructure as Code*) dengan CI/CD (*Continuous Integration/Continuous Deployment*) membantu dalam menghasilkan perangkat lunak yang memiliki kualitas dan produktivitas yang tinggi. Pengujian yang dilakukan berupa perbandingan *deployment* infrastruktur secara manual dan otomatis dan menilai efektivitas dan efisiensi cara *deployment*. Pengujian untuk memastikan infrastruktur yang dibuat sudah berjalan dengan baik yaitu melakukan *deployment* aplikasi sederhana. Penelitian dimulai dari membuat desain infrastruktur, konfigurasi IaC *Terraform*, pembuatan *script* CI/CD, *deployment* infrastruktur dengan cara manual dan otomatis, konfigurasi aplikasi beserta CI/CD, dan *deployment* aplikasi. Rata-rata waktu yang dibutuhkan cara manual yaitu selama 13 menit 34 detik, sedangkan rata-rata waktu yang dibutuhkan cara otomatis yaitu selama 14 menit 5 detik. Nilai efektivitas yang diperoleh menunjukkan kedua cara tersebut berhasil dalam melakukan *deployment* infrastruktur, sedangkan untuk nilai efisiensinya, cara manual dan otomatis sama-sama memiliki kelebihan dan kekurangannya masing-masing. Aplikasi Go – Gin yang di *deploy* membutuhkan rata-rata waktu selama 3 menit 7 detik, sedangkan aplikasi PHP – Laravel membutuhkan rata-rata waktu selama 3 menit 8 detik. Aplikasi yang di *deploy* pada infrastruktur yang sama menunjukkan perbedaan waktu yang tidak jauh berbeda dan perbedaan konfigurasi aplikasi yang diperoleh yaitu pada *Dockerfile* dan *file kubernetes object* yang digunakan.

Kata kunci: *Continuous Integration/Continuous Deployment, Dockerfile, Infrastructure as Code, Kubernetes Object, Terraform*

1. Pendahuluan

Cloud computing dikenal dengan fleksibilitas dan biaya yang rendah. Teknologi ini merupakan salah satu bentuk upaya untuk mengurangi pengadaan dan pengelolaan infrastruktur yang besar [1]. Seiring dengan perkembangan teknologi *cloud computing*, proses pengembangan dan penerapan aplikasi di *cloud* juga banyak mengalami perubahan tidak hanya dari sisi skalabilitas dan keandalan, tetapi juga dari sisi integrasi dan delivery yang hanya membutuhkan waktu yang minimal. Salah satu *provider* yang menyediakan teknologi *cloud computing* yaitu *Google Cloud Platform*.

Google Cloud Platform menjadi salah satu *provider* teknologi *cloud* yang mendukung dalam proses pengembangan aplikasi. Proses pengembangan tersebut didukung oleh infrastruktur yang disediakan oleh *provider*. Pengadaan infrastruktur di *Google Cloud* dapat dilakukan dengan penerapan IaC (*Infrastructure as Code*), yaitu salah satu praktik pengembangan perangkat lunak yang memperlakukan infrastruktur sebagai kode [4]. Hal tersebut berarti infrastruktur didefinisikan dalam kode program yang dapat dikelola oleh alat dan proses yang sama seperti perangkat lunak.

Google Cloud Platform juga mendukung CI/CD (*Continuous Integration/Continuous Deployment*) dalam proses pengembangan perangkat lunak. CI/CD merupakan salah satu praktik pengembangan perangkat lunak, dimana CI/CD melakukan otomatisasi terhadap proses *deployment* dan membantu dalam pengecekan kualitas dan kinerja dari kode program yang dibuat dalam proses pengembangan aplikasi.

P-ISSN : 2089-3353
E-ISSN : 2808-9162

Penelitian sebelumnya dengan konteks IaC yaitu berjudul "Implementasi *Automation Deployment* pada *Google Cloud Compute* VM menggunakan *Terraform*" bertujuan untuk mempermudah melakukan inisialisasi banyak server di lingkungan *Cloud* dengan melakukan *automation* menggunakan *tool Terraform*. Kesimpulan yang dapat diambil yaitu konfigurasi *Terraform* dapat disesuaikan dengan kebutuhan serta kualitas dan performa *web server* pada *instance* vm *google cloud* sangat baik, mampu menangani beban yang cukup besar yaitu sekitar 40580 paket yang dikirimkan dengan waktu respon yang cepat dan stabil [3].

Penelitian kedua dalam konteks IaC yaitu berjudul "Implementasi *Web Server* Menggunakan *Infrastructure as Code Terraform* Berbasis *Cloud Computing*" bertujuan untuk membandingkan kualitas *instance web server* yang dibangun secara otomatis menggunakan *Infrastructure as Code* (*Terraform*) dengan *web server* yang dibangun secara manual. Kesimpulan yang dapat diambil yaitu perbandingan waktu untuk membuat *instance web server* 12 menit 19 detik, rata-rata nilai *throughput* 0.750 Mbit/s, *packet loss* 0.00%, selisih nilai *delay* 0,0035 ms, selisih nilai parameter *jitter* yaitu 0,004 ms, parameter CPU *usage*, didapatkan dengan selisih rata-rata di seluruh pengujian yaitu 5,899% [5].

Penelitian ketiga dalam konteks IaC yaitu berjudul "*Infrastructure as Code* (IaC) menggunakan OpenStack untuk Kemudahan Pengoperasian Jaringan *Cloud Computing* (Studi Kasus: Smart City di Provinsi Bali)" bertujuan untuk mengatasi lamanya waktu pengoperasian dalam pengimplementasian *cloud computing* pada Bali *Smart Island*, melakukan otomatisasi pada pengoperasian *cloud computing* serta minimalisasi jumlah perintah yang digunakan untuk mempersingkat waktu pengoperasian pada *cloud computing*. Kesimpulan yang dapat diambil yaitu implementasi IaC berbasis *OpenStack* untuk orkestrasi pada jaringan *cloud computing* dapat menyelesaikan permasalahan lamanya waktu pemrosesan dan operasional pada jaringan cloud computing. Waktu rata-rata yang diperoleh dari proses *deployment* hingga *node* (*stack*) terbentuk menggunakan *Neo*-CLI adalah sebesar 9,1 detik [9].

Penelitian keempat dalam konteks IaC yaitu berjudul "*Provisioning Google Kubernetes Engine Cluster* dengan Menggunakan *Terraform* dan *Jenkins* pada Dua *Environment*" bertujuan untuk membahas penyediaan *cluster Google Kubernetes Engine* (GKE) menggunakan *Terraform* dan *Jenkins* di dua lingkungan untuk menyoroti pentingnya penyediaan *server* dalam sistem berskala besar dan kebutuhan untuk lingkungan pengembangan dan produksi. Kesimpulan yang dapat diambil yaitu keberhasilan penyediaan cluster GKE di lingkungan pengembangan dan produksi dan sistem yang diteliti dapat secara efektif merampingkan penyediaan *server* untuk tim *DevOps* [8].

Penelitian selanjutnya yaitu dalam konteks CI/CD yaitu berjudul "Implementasi *Continuous Integration* dan *Continuous Delivery* Pada Aplikasi myITS *Single Sign On*" bertujuan untuk untuk melakukan otomatisasi dalam proses *delivery* dan *deployment* dari pengembangan aplikasi myITS SSO sehingga *developer* bisa fokus pada kode aplikasi dan aplikasi dapat dirilis dengan cepat dan sudah melalui serangkaian tes pengujian tanpa perlu membuat aplikasi menjadi lambat karena proses *deploy* yang manual. Kesimpulan yang dapat diambil yaitu implementasi CI/CD berjalan dengan baik dengan ditandai proses *pull*, *build*, *push*, dan *deploy* berjalan otomatis setelah pengembang melakukan pembaharuan terhadap repositori [7].

Penelitian kedua dalam konteks CI/CD yaitu berjudul "Implementasi *Continuous Integration* dan *Continuous Deployment* pada Aplikasi *Learning Management System* di PT. Millennia Solusi Informatika" bertujuan untuk menerapkan konsep CI/CD (*Continuous Integration/Continuous Deployment*) ke aplikasi LMS (*Learning Management System*) yang merupakan salah satu aplikasi yang dikembangkan oleh PT. Millennia Solusi Informatika sehingga *developer* bisa berfokus pada aplikasi yang dikembangkan dan setiap perilisan aplikasi dapat dilakukan dengan cepat dan sudah melalui serangkaian pengujian. Kesimpulan yang dapat diambil yaitu penerapan konsep CI/CD memudahkan tim pengembang dan operasional bekerja secara praktis. Otomatisasi pada tahapan CI/CD ini memungkinkan kesalahan yang terjadi oleh manusia dapat terhindarkan dikarenakan seluruhnya telah otomatis dilakukan oleh mesin [10].

Penelitian ketiga dalam konteks CI/CD yaitu berjudul "*Deploying an Application to Cloud Platform Using Continous Integration and Continous Delivery*" bertujuan untuk membahas penyebaran aplikasi ke *platform cloud* menggunakan *continuous integration* and *continuous delivery*. Kesimpulan yang dapat diambil yaitu CI/CD dapat menyederhanakan proses pengembangan dan memastikan pengiriman aplikasi perangkat lunak yang lancar dan efisien [2].

Berdasarkan penelitian-penelitian sebelumnya yang dilampirkan diatas, sejauh ini belum ditemukan penelitian yang membahas mengenai integrasi antara IaC dengan CI/CD. IaC dan CI/CD merupakan dua pendekatan dalam proses pengembangan perangkat lunak dan manajemen infrastruktur. IaC memungkinkan untuk mendefinisikan atau pengadaan infrastruktur menggunakan kode yang dapat dieksekusi, sehingga infrastruktur dapat dibangun, diperbarui, dan ditingkatkan dengan cara yang konsisten dan terdokumentasi menggunakan bahasa pemrograman atau konfigurasi terstruktur seperti YAML atau JSON. CI/CD mengotomatiskan proses pembangunan, pengujian, dan penerapan perangkat lunak ke lingkungan produksi [6]. CI berfokus pada penggabungan kode dan memastikan perubahan kode terintegrasi dengan baik dengan kode yang sudah ada,

P-ISSN : 2089-3353
E-ISSN : 2808-9162

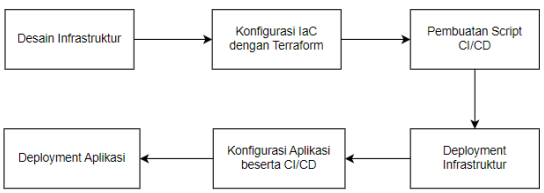
sedangkan CD memastikan pengiriman otomatis dan cepat ke lingkungan produksi dengan mengurangi intervensi manual.

Kedua pendekatan tersebut, yaitu IaC dan CI/CD masing-masing sudah dijelaskan dan terbukti bahwa keduanya sangat membantu para *developer* dalam proses manajemen infrastruktur dan *delivery* perangkat lunak ke lingkungan produksi. Maka dari itu, seperti yang dijelaskan diatas, penelitian ini akan melakukan pengujian dengan melakukan integrasi IaC dengan CI/CD di lingkungan *Google Cloud Platform*. Pengujian tersebut bertujuan untuk memastikan IaC dan CI/CD dapat menciptakan lingkungan pengembangan yang handal yang tentunya akan lebih mempermudah para *developer* dalam proses pengembangan perangkat lunak karena mengkombinasikan dua hal yang sebelumnya sudah mempermudah pekerjaan yang ada.

Pada penelitian ini, konfigurasi IaC akan dibuat menggunakan *Terraform* dan untuk CI/CD akan dibuat menggunakan *CircleCI*. Terdapat 3 buah skenario pengujian yang akan dilaksanakan, yaitu *deployment* infrastruktur dengan cara manual (*Terraform*) dan otomatis (*Terraform* terintegrasi *CircleCI*), *deployment* aplikasi *Go – Gin*, dan *deployment* aplikasi *PHP – Laravel*. Skenario pengujian pertama akan membandingkan waktu yang diperlukan untuk *deployment* infrastruktur dengan cara manual dan otomatis untuk menilai cara yang memiliki efektivitas dan efisiensi yang lebih besar. Skenario pengujian kedua dan ketiga akan menguji kualitas dari infrastruktur yang dibuat sekaligus membandingkan waktu *deployment* dari aplikasi dan perbedaan konfigurasi dari aplikasi yang di *deploy* pada infrastruktur yang sama.

2. Metode Penelitian

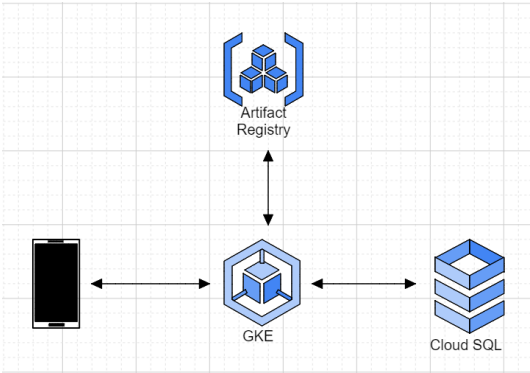
Metode yang digunakan pada penelitian ini memberikan langkah-langkah yang jelas untuk setiap proses atau tahapan yang akan dilaksanakan pada penelitian ini. Hal tersebut bertujuan untuk memberikan arahan mengenai penelitian yang dilaksanakan, memastikan kesinambungan antar proses yang dikerjakan, dan menghasilkan output yang jelas. Gambar 1 berikut merupakan alur penelitian dari penelitian yang akan dikerjakan



Gambar 1. Alur Penelitian

2.1. Desain Infrastruktur

Langkah awal dalam penelitian yang akan dilakukan yaitu membuat desain dari infrastruktur yang akan dibuat dan digunakan oleh aplikasi. Desain ini diperlukan untuk perencanaan atau perancangan sumber daya *cloud* sesuai dengan keperluan. Desain yang dibuat akan saling terhubung satu sama lain untuk membuat lingkungan yang terorganisir. Gambar 2 berikut merupakan desain infrastruktur yang akan menjadi acuan dalam. pembuatan *script Terraform* untuk pengadaan infratruktur tersebut di *Google Cloud*.



Gambar 2. Desain Infrastruktur

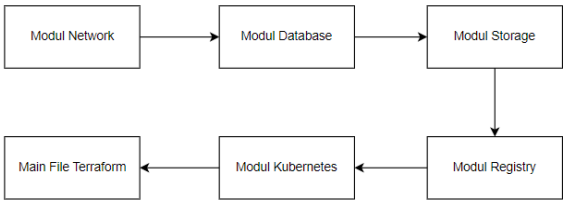
Infrastruktur yang dibangun akan berupa GKE (*Google Kubernetes Engine*), *Artifact Registry*, dan *Cloud* SQL. GKE akan menjadi tempat dilakukannya *deploy* aplikasi, *Artifact Registry* akan menjadi tempat untuk menyimpan *docker image* dari aplikasi, dan *Cloud* SQL akan menjadi tempat untuk penyimpanan data dari aplikasi (data *user* dan lain sebagainya), dimana SQL *instance* yang akan digunakan yaitu *PostgreSQL*.

Infrastruktur GCP lainnya yang akan dibuat selain 3 infrastruktur sebelumnya yaitu VPC (*Virtual Private Cloud*) dan *Cloud Storage*. VPC akan menjadi infrastruktur utama yang berperan sebagai pondasi karena digunakan sebagai *networking* oleh infrastruktur sebelumnya dan *Cloud Storage* akan menjadi tempat untuk menyimpan *Terraform state* secara online yang bertujuan untuk memberikan akses *tool* CI/CD yang digunakan nantinya ke *Terraform state* untuk otomatisasi pembaruan infrastruktur.

2.2. Konfigurasi IaC dengan *Terraform*

Langkah kedua yaitu membuat konfigurasi *Terraform* untuk pengadaan infrastruktur yang akan digunakan di *Google Cloud Platform* sesuai dengan desain infrastruktur yang telah dibuat pada tahapan sebelumnya. Konfigurasi yang dibuat dibagi menjadi beberapa modul dengan tujuan pemisahan logika dan fungsi infrastruktur ke dalam unit-unit terpisah sehingga memudahkan pengguna untuk mengelola dan menggabungkan sumber daya GCP secara modular dan memfasilitasi pembangunan dan skalabilitas yang lebih baik, dimana hal tersebut tentunya didukung oleh *Terraform*. Gambar 3 berikut merupakan alur pembuatan konfigurasi IaC dengan *Terraform* yang dibagi menjadi beberapa modul.

P-ISSN : 2089-3353
E-ISSN : 2808-9162



Gambar 3. Alur Pembuatan Infrastruktur GPC dengan IaC *Terraform*

Alur tersebut dimulai dengan pembuatan modul *network* terlebih dahulu dikarenakan pada modul ini terdapat infrastruktur utama yang berperan sebagai pondasi yang akan digunakan pada modul selanjutnya. Modul selanjutnya yang akan dibuat yaitu modul *database*, modul *storage*, modul *registry*, dan modul *kubernetes*. Total terdapat 5 buah modul yang akan dibuat dan kelima modul tersebut akan digunakan dengan cara memanggil masing-masing modul ke *main file* dari *Terraform*.

a. Modul *network*: *resource* GCP yang berkaitan dengan jaringan.
b. Modul *database*: *resource* GCP yang berkaitan dengan penyimpanan data aplikasi, seperti data *user*.
c. Modul *storage*: *resource* GCP yang berkaitan dengan penyimpanan data yang berbentuk *file.*
d. Modul *registry*: *resource* GCP yang berkaitan dengan penyimpanan *image* aplikasi.
e. Modul *kubernetes*: *resource* GCP yang berkaitan dengan *kubernetes*.
f. *Main file Terraform*: tempat pemanggilan semua modul yang telah dibuat.

Setiap modul-modul yang dibuat memiliki 2 file didalamnya yaitu *main.tf* dan *variables.tf*. *main.tf* sebagai tempat untuk mendefinisikan *resouce* GCP yang akan dibuat dan *variables.tf* sebagai tempat untuk mendefinisikan variabel-variabel yang digunakan di *main.tf* dan variabel yang dibuat dapat memiliki *default value*. Namun, khusus untuk modul *network* terdapat 1 *file* tambahain yaitu *outputs.tf*, dimana *file* ini digunakan sebagai tempat untuk mendefinisikan *value* atau nilai dari *resouce* modul *network* sehingga dapat digunakan pada modul-modul lainnya. Hal tersebut dikarenakan modul *network* ini bisa dikatakan sebagai pondasi untuk seluruh infrastruktur yang akan dibuat.

Sama halnya dengan modul-modul sebelumnya, *main file Terraform* juga terdapat *file* yang sama, yaitu *main.tf* dan *variables.tf*. Namun, pada *main file Terraform* terdapat 1 *file* yang berbeda dan tidak ada pada modul-modul yang telah dibuat. *File* tersebut yaitu *terraform.tfvars* yang digunakan untuk mendefinisikan variabel *key* dan *value* yang akan digunakan pada *variables.tf* yang tidak memiliki *default value*. Hal tersebut dikarenakan biasanya variabel *key value* yang didefinisikan di *file* ini bersifat rahasia dan krusial untuk *project* GCP yang digunakan yang sama halnya dengan *file environtment* variabel yang biasa digunakan dalam proses pengembangan aplikasi.

main.tf di main file terdapat pendefinisian beberapa hal yaitu *backend* yang digunakan sebagai penyimpanan *terraform state*, *provider* yang digunakan beserta dengan versinya, pengaktifan *services* yang diperlukan, dan pemanggilan modul-modul yang telah didefinisikan sebelumnya beserta dengan masing-masing argumen yang diperlukannya. Argumen yang didefinisikan yaitu variabel dari masing-masing modul yang tidak mempunyai *default value*.

2.3. Pembuatan *Script* CI/CD

Langkah ketiga yaitu membuat *script* CI/CD untuk otomatisasi integrasi dan *deployment* terhadap konfigurasi *Terraform* yang telah dibuat pada tahapan sebelumnya. *Script* CI/CD tersebut dibuat sesuai dengan kebutuhan dan ketentuan yang diberikan oleh *CircleCI*, *tool* CI/CD *open source* yang digunakan. *Script* CI/CD tersebut disimpan di dalam *folder* yang sama di konfigurasi *Terraform* yang telah dibuat sebelumnya dan dibuatkan *folder* khusus untuk untuk *script* tersebut. Sesuai dengan dokumentasi dari CircleCI, *script* CI/CD harus berada di dalam *folder* *.circle* dan nama filenya diberi nama *config.yml*.

Script CI/CD yang dibuat memiliki 2 buah *jobs* yang didefinisikan, yaitu *plan* dan *apply*, dimana masing-masing *jobs* memiliki langkah-langkah tersendiri. Kedua *jobs* tersebut memilki keterkaitan satu sama lain, dimana untuk *jobs plan* terdapat proses persiapan dan pada *jobs apply* terdapat proses implementasi dari persiapan sebelumnya. Hasil dari *jobs plan* akan diteruskan ke *jobs apply*, sehingga hal tersebut dapat mengurangi langkah-langkah yang sifatnya berulang. Konfigurasi tersebut juga terdapat *workflows* yang berfungsi untuk menentukan alur dari eksekusi *jobs* yang yang didefinisikan sebelumnya.

2.4. *Deployment* Infrastruktur

Langkah keempat yaitu melakukan *deployment* infrastruktur dengan IaC *Terraform*. *Deployment* tersebut dibagi menjadi 2 cara, yaitu cara manual dan otomatis. Sebelum melakukan *deployment* infrastruktur secara keseluruhan, diperlukan melakukan *deployment* untuk infrastruktur *Cloud Storage* terlebih dahulu dikarenakan infrastruktur ini diperlukan untuk menyimpan *Terraform state* secara online yang berperngaruh terhadap *deployment* dengan cara otomatis nantinya.

Deployment dengan cara manual dan otomatis dapat dilanjutkan apabila infrastruktur *Cloud Storage* sudah dibangun. *Deployment* cara manual dapat dilakukan secara langsung dengan mengetikkan perintah *Terraform* melalui terminal. Perintah *Terraform* yang dimaksud yaitu *terraform init* (inisialisasi *Terraform*), *terraform plan* (membuat rencana eksekusi), dan *terraform apply* (membangun infrastruktur). Gambar 4 berikut merupakan hasil dari menjalankan perintah *Terraform* tersebut.

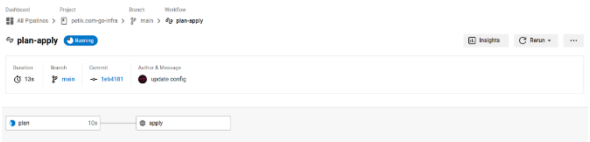
P-ISSN : 2089-3353
E-ISSN : 2808-9162

Gambar 4. Hasil Menjalankan Perintah *Terraform Init*, *Plan*, dan *Apply*

Disisi lain yaitu cara otomatis, *deployment* cara ini memerlukan tahapan yang berbeda dari cara manual sebelumnya. *Deployment* cara otomatis memerlukan pengubahan penyimpanan *state* yang digunakan pada konfigurasi *Terraform* yang awalnya *local* (*default*) ke *Cloud Storage* yang telah dibuat sebelumnya. Hal ini dikarenakan pada *deployment* cara otomatis, *CircleCI* memerlukan akses ke *Terraform state* untuk melakukan otomatis pembangunan infrastruktur. Singkatnya, *Terraform state* merupakan sebuah konfigurasi *file* yang di *generate* otomatis oleh *Terraform* yang berfungsi untuk memetakan sumber daya *cloud* ke *file* konfigurasi, melacak metadata, dan meningkatkan kinerja untuk infrastruktur yang besar. Gambar 5 merupakan gambaran singkat mengenai perubahan penyimpanan state yang digunakan oleh konfigurasi *Terraform*.

Gambar 5. Hasil Menjalankan Perintah Pengubahan Penyimpanan *State Terraform*

Deployment cara otomatis dapat dilanjutkan dengan melakukan *push* konfigurasi *Terraform* tersebut ke *remote repository* (*Github*). Apabila sudah di *push*, secara otomatis *script* CI/CD yang dibuat akan berjalan dan akan membangun infrastruktur sesuai dengan konfigurasi *Terraform* yang dibuat. Gambar 6 merupakan gambaran mengenai CI/CD *pipeline* yang telah dibuat sesudah melakukan push konfigurasi *Terraform* ke *Github*.



Gambar 6. Hasil CI/CD *Pipeline Push* Konfigurasi *Terraform*

Pipeline tersebut akan menjalankan *jobs* sesuai dengan yang didefinisikan dengan urutan yang sesuai pada *workflows* di *script* CI/CD yang telah dibuat. Terdapat 2 *jobs*, yaitu *plan* dan *apply*. *Plan* berisikan proses persiapan sampai dengan *terraform init* dan *plan*, sedangkan *apply* berisikan proses *terraform apply*.

2.5. Konfigurasi Aplikasi beserta CI/CD

Langkah kelima yaitu melakukan konfigurasi terhadap *Dockerfile* dan *file kubernetes object* yang digunakan oleh aplikasi serta membuat script CI/CD untuk aplikasi. Aplikasi yang di *deploy* ada 2, yaitu aplikasi *Go – Gin* dan aplikasi *PHP – Laravel*. Kedua aplikasi tersebut dibuatkan masing-masing konfigurasi *file kubernetes object* seperti *deployment.yaml* dan *service.yaml*. Pada *file deployment.yaml* berisikan konfigurasi untuk *deployment* dari aplikasi sesuai dengan kebutuhan dari aplikasi itu sendiri, sedangkan *file service.yaml* berisikan konfigurasi untuk memberikan akses ke *public* terhadap aplikasi yang telah di *deploy*.

Sama halnya dengan *deployment* infrastruktur cara otomatis sebelumnya, *script* CI/CD untuk masing-masing aplikasi dibuat menggunakan *CircleCI*. *Script* CI/CD tersebut disimpan pada tempat yang sama seperti sebelumnya dikarenakan hal tersebut suatu hal yang wajib seperti yang telah dijelaskan pada dokumentasi CircleCI.

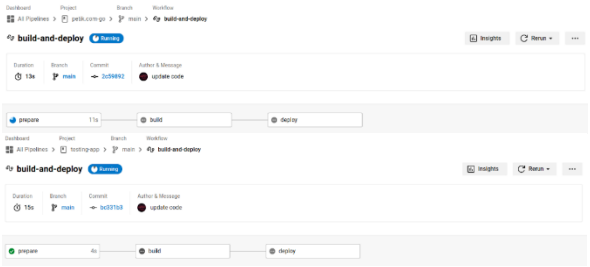
2.6. *Deployment* Aplikasi

Langkah terakhir yaitu melakukan *deployment* terhadap aplikasi dengan memanfaatkan *script* CI/CD yang telah dibuat pada tahapan sebelumnya. Namun sebelum itu, diperlukan pembuatan *kubernetes secret* terlebih dahulu dikarenakan hal tersebut berisi *credentials* aplikasi yang diperlukan dan melakukan *deployment* aplikasi secara manual terlebih dahulu. *Deployment* tersebut dapat dimulai dari pembuatan *image* dari aplikasi, kemudian *push image* tersebut ke infrastruktur *Artifact Registry* yang telah dibuat sebelumnya, dilanjutkan dengan membuat *kubernetes secret*, dan terakhir yaitu menjalankan perintah *kubectl*

P-ISSN : 2089-3353
E-ISSN : 2808-9162

untuk *deployment* aplikasi ke *Google Kubernetes Engine*. *Deployment* cara manual ini diperlukan karena pada *script* CI/CD yang dibuat terdapat perintah untuk melakukan *restart* dengan tujuan mendapatkan aplikasi yang terbaru sesuai dengan perubahan yang dilakukan.

Deloyment aplikasi dengan memanfaatkan *script* CI/CD dapat dilanjutkan dengan melakukan *push* perubahan kode program ke *remote repository* (*Github*) yang digunakan. Apabila sudah di *push*, secara otomatis *script* CI/CD yang dibuat akan berjalan dan akan memperbarui aplikasi sesuai dengan perubahan yang dilakukan. Gambar 7 berikut merupakan gambaran mengenai CI/CD pipeline yang terlah dibuat sesudah melakukan *push* perubahan kode program aplikasi ke *Github*.



Gambar 7. Hasil CI/CD *Pipeline Push* Konfigurasi *Terraform*

Sama halnya dengan *pipeline deployment* infrastruktur sebelumnya, kedua *pipeline* tersebut akan menjalankan *jobs* sesuai dengan yang didefinisikan dengan urutan yang sesuai pada *workflows* di *script* CI/CD yang telah dibuat. Terdapat 3 *jobs*, yaitu *prepare*, *build*, dan *deploy*. *Prepare* berisikan proses persiapan, *build* berisikan proses untuk membuat *image* aplikasi, dan *deploy* berisikan proses *deployment* aplikasi di *Google Kubernetes Engine*.

3. Hasil dan Pembahasan

Hasil yang didapatkan dari pengujian yang telah dilakukan dibagi menjadi 2, yaitu *deployment* infrastruktur dan *deployment* aplikasi. Kedua hasil tersebut memiliki pembahasannya masing-masing.

3.1. *Deployment* Infrastruktur

Hasil pengujian yang didapatkan berdasarkan pengujian yang dilakukan untuk *deployment* infrastruktur GCP yaitu berupa perbandingan waktu, efektivitas, dan efisiensi *deployment* infrastruktur dan integrasi IaC dengan CI/CD. Berikut merupakan penjelasan lebih lanjut mengenai hal yang telah disebutkan sebelumnya.

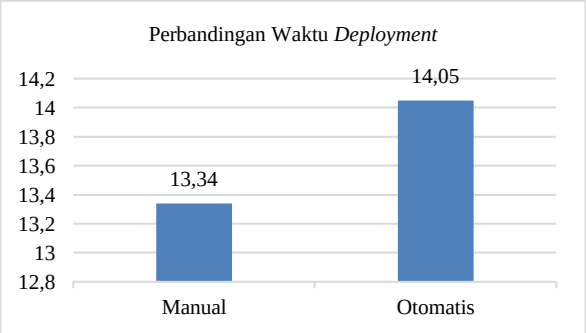
3.1.1. Perbandingan Waktu, Efektivitas, dan Efisiensi *Deployment* Infrastruktur

Waktu *deployment* infrastruktur yang didapatkan terbagi menjadi 2, yaitu waktu *deployment* dengan cara manual dan otomatis. Masing-masing cara yang digunakan untuk *deployment* tentunya mendapatkan waktu yang berbeda. Tabel 1 berikut merupakan hasil yang didapatkan berdasarkan pengujian yang telah dilakukan. Pengujian tersebut dilakukan sebanyak 5 kali untuk masing-masing cara dengan mengulangi proses yang sama.

Tabel 1.Waktu *Deployment* Infrastruktur

Manual	Otomatis
13 menit 47 detik	13 menit 39 detik
14 menit 8 detik	14 menit 8 detik
13 menit 39 detik	13 menit 47 detik
13 menit 44 detik	14 menit 37 detik
13 menit 34 detik	14 menit 18 detik

Perbandingan waktu *deployment* yang dijadikan perbandingan adalah waktu *deployment* dari cara manual dan otomatis berdasarkan hasil pengujian yang didapatkan pada tahapan sebelumnya. Gambar 8 berikut merupakan hasil perbandingan rata-rata waktu *deployment* infrastruktur berdasarkan hasil pengujian yang didapatkan.



Gambar 8. Perbandingan Waktu *Deployment* Infrastruktur

Perbandingan rata-rata waktu *deployment* infrastruktur tersebut menunjukkan bahwa cara manual lebih cepat dibandingkan cara otomatis, dimana cara manual rata-rata membutuhkan waktu selama 13 menit 34 detik, sedangkan cara otomatis rata-rata membutuhkan waktu selama 14 menit 5 detik. Perbedaan antara kedua cara tersebut yaitu sebesar 31 detik.

Penilaian efektivitas dan efisiensi *deployment* infrastruktur dinilai dari waktu perbandingan yang telah didapatkan sebelumnya. Efektivitas tersebut dinilai berdasarkan berhasilnya dalam melakukan *deployment*, sedangkan efisiensi dinilai dari waktu dan kemudahan dalam melakukan *deployment*. Tabel 2 berikut merupakan hasil penilaian efektivitas dan efisiensi *deployment* infrastruktur dengan cara manual dan otomatis.

Tabel 2.Efektivitas dan Efisiensi *Deployment* Infrastruktur

	Efektivitas	Efisiensi
Manual	Baik, dikarenakan berhasil melakukan *deployment* sesuai dengan konfigurasi *Terraform* yang dibuat.	Baik dari sisi waktu, dikarenakan waktu yang dihasilkan untuk proses *init*, *plan*, dan *apply* lebih cepat dibandingkan cara otomatis. Kurang dari sisi kemudahan, dikarenakan memerlukan intervensi dari pengguna dalam proses pelaksanaan perintah *Terraform*

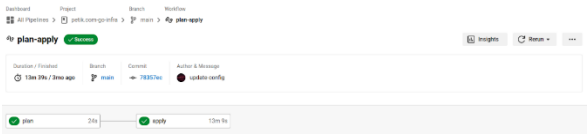
P-ISSN : 2089-3353
E-ISSN : 2808-9162

		sehingga penyesuaian menjadi lambat
Otomatis	Baik, dikarenakan berhasil melakukan deployment sesuai dengan konfigurasi *Terraform* yang dibuat.	Kurang dari sisi waktu, dikarenakan waktu yang dihasilkan dalam proses *init*, *plan*, dan, *apply* lebih lama dibandingkan cara manual. Baik, dikarenakan tidak memerlukan intervensi dari pengguna dan memudahkan dalam melakukan kolaborasi bersama *developer* lain.

Berdasarkan tabel 2 diatas, cara manual dan otomatis dari sisi efektivitasnya berhasil untuk melakukan *deployment*, sedangkan dari sisi efisiensi, kedua cara tersebut sama-sama memiliki kelebihan dan kekurangannya masing-masing, dimana jika dilihat dari sisi waktu cara manual lebih cepat, namun apabila dilihat dari sisi kemudahan cara otomatis lebih mudah. Cara manual lebih cepat dikarenakan cara ini tidak memerlukan checkout *repository project* dan persiapan *environment CircleCI* seperti cara otomatis yaitu pada *jobs* di konfigurasi CI/CD yang dibuat, sedangkan cara otomatis lebih mudah dikarenakan cara ini tidak memerlukan intervensi dari pengguna dan hal tersebut memudahkan dalam proses kolaborasi bersama dengan *developer* lain.

3.1.2. Integrasi IaC dengan CI/CD

Integrasi IaC dengan CI/CD menunjukkan hasil positif, dimana integrasi tersebut berhasil dilakukan dengan memanfaatkan *platform CircleCI*. Hal tersebut ditunjukkan oleh CI/CD *pipeline* pada gambar 9 berikut yang menunjukkan bahwa *deployment* infrastruktur secara otomatis berhasil dan sudah selesai dilakukan.



Gambar 9. Hasil CI/CD *Pipeline* IaC *Terraform*

3.2. Deployment Aplikasi

Hasil pengujian yang didapatkan berdasarkan pengujian yang dilakukan sebelumnya untuk *deployment* aplikasi yaitu berupa perbandingan waktu *deployment* aplikasi, perbedaan konfigurasi aplikasi, dan kesimpulan mengenai kualitas infrastruktur yang telah dibuat. Berikut merupakan penjelasan lebih lanjut mengenai hal yang telah disebutkan sebelumnya.

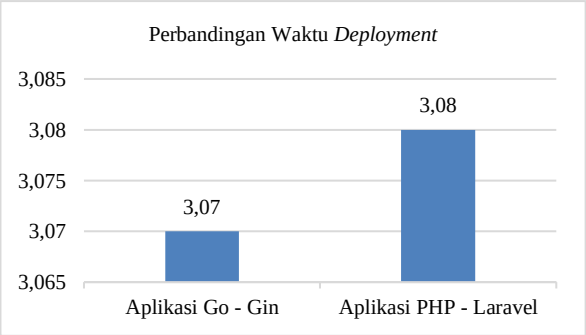
3.2.1. Perbandingan Waktu Deployment Aplikasi

Waktu *deployment* aplikasi yang didapatkan terbagi menjadi 2, yaitu waktu *deployment* aplikasi *Go – Gin* dan *PHP – Laravel*. Masing-masing aplikasi yang di *deploy* tentunya mendapatkan waktu yang berbeda walaupun di *deploy* pada infrastruktur yang sama. Tabel 3 berikut merupakan hasil yang didapatkan berdasarkan pengujian yang telah dilakukan. Pengujian tersebut dilakukan sebanyak 5 kali untuk masing-masing aplikasi.

Tabel 3.Waktu *Deployment* Aplikasi

Go – Gin	*PHP – Laravel*
3 menit 17 detik	3 menit 20 detik
3 menit 24 detik	2 menit 37 detik
3 menit 24 detik	3 menit 10 detik
2 menit 55 detik	3 menit 16 detik
2 menit 36 detik	3 menit 18 detik

Perbandingan waktu *deployment* yang dijadikan perbandingan adalah waktu *deployment* dari aplikasi *Go – Gin* dan aplikasi *PHP – Laravel* yang didapatkan pada tahapan sebelumnya. Gambar 10 berikut merupakan hasil perbandingan rata-rata waktu *deployment* aplikasi berdasarkan hasil pengujian yang didapatkan.



Gambar 10. Perbandingan Waktu *Deployment* Aplikasi

Perbandingan waktu *deployment* tersebut menunjukkan bahwa aplikasi *Go – Gin* lebih cepat dibandingkan aplikasi *PHP – Laravel*, dimana aplikasi *Go – Gin* rata-rata membutuhkan waktu selama 3 menit 7 detik, sedangkan aplikasi *PHP – Laravel* rata-rata membutuhkan waktu selama 3 menit 8 detik. Perbedaan rata-rata waktu antara kedua aplikasi tersebut yaitu sebesar 1 detik.

3.2.2. Perbedaan Konfigurasi Aplikasi

Kedua aplikasi yang telah di *deploy* menggunakan *platform* yang sama yaitu GKE mendapatkan perbedaan ketika dilakukan *deployment*. Perbedaan yang didapatkan yaitu pada *Dockerfile* dan *file kubernetes object* yang digunakan. Hal tersebut dikarenakan setiap aplikasi memiliki konfigurasi *Dockerfile* yang berbeda-beda sesuai dengan bahasa pemrograman yang digunakan oleh aplikasi. *File kubernetes object* yang digunakan juga berbeda terutama pada bagian *deployment.yaml*, dimana pada aplikasi *Go – Gin* memerlukan *mount environment variable* sedangkan aplikasi *PHP – Laravel* tidak memerlukan hal tersebut.

3.2.2. Infrastruktur GCP

Aplikasi *Go – Gin* dan *PHP – Laravel* yang telah di *deploy* pada *platform* yang sama yaitu GKE berhasil dilakukan. Hal tersebut menunjukkan bahwa infrastruktur yang dibuat menggunakan IaC *Terraform* sudah berjalan dengan baik, dimana infrastruktur yang

P-ISSN : 2089-3353
E-ISSN : 2808-9162

dibuat sudah sesuai dengan desain yang telah dibuat sebelumnya.

4. Kesimpulan

Kesimpulan dari penelitian "Integrasi *Infrastructure as Code* dengan *Integration/Continuous Deployment* di *Google Cloud Platform*" berdasarkan analisis hasil pengujian yang didapatkan yaitu sebagai berikut:

a. Rata-rata waktu perbandingan *deployment* infrastruktur menggunakan IaC dengan cara manual lebih cepat dibandingkan dengan cara otomatis. Cara manual rata-rata membutuhkan waktu selama 13 menit 34 detik, sedangkan cara otomatis rata-rata membutuhkan waktu selama 14 menit 5 detik. Perbedaan waktu antara kedua cara tersebut yaitu sebesar 31 detik.
b. Nilai efektivitas yang diperoleh menunjukkan kedua cara tersebut berhasil dalam melakukan *deployment* infrastruktur, sedangkan untuk nilai efisiensinya, cara manual dan otomatis sama-sama memiliki kelebihan dan kekurangannya masing-masing, dimana jika dilihat dari sisi waktu cara manual lebih cepat, namun apabila dilihat dari sisi kemudahan cara otomatis lebih mudah. Cara manual lebih cepat dikarenakan cara ini tidak memerlukan *checkout repository project* dan persiapan *environment CircleCI* seperti cara otomatis yaitu pada *jobs* di konfigurasi CI/CD yang dibuat, sedangkan cara otomatis lebih mudah dikarenakan cara ini tidak memerlukan intervensi dari pengguna dan hal tersebut memudahkan dalam proses kolaborasi bersama dengan *developer* lain.
c. Integrasi yang dilakukan berhasil yang dibuktikan dengan CI/CD *pipeline* yang diperoleh.
d. Aplikasi yang dilakukan *deployment* mendapatkan perbandingan waktu sebesar 4 detik dan menunjukkan bahwa *deployment* aplikasi *Go – Gin* lebih cepat dibandingkan aplikasi *PHP – Laravel*, walaupun perbedaan waktunya tidak terlalu jauh. Aplikasi *Go – Gin* rata-rata membutuhkan waktu selama 3 menit 7 detik, sedangkan aplikasi aplikasi *PHP – Laravel* rata-rata membutuhkan waktu selama 3 menit 8 detik, dimana kedua aplikasi tersebut di *deploy* pada infrastruktur GKE.
e. Perbedaan konfigurasi yang diperoleh yaitu pada konfigurasi *Dockerfile* dan *file kubernetes object* yang digunakan. Hal tersebut dikarenakan setiap aplikasi memiliki konfigurasi *Dockerfile* yang berbeda-beda sesuai dengan bahasa pemrograman yang digunakan oleh aplikasi. *File kubernetes object* yang digunakan juga berbeda terutama pada bagian *deployment.yaml*, dimana pada aplikasi *Go – Gin* memerlukan *mount environment variable* sedangkan aplikasi *PHP – Laravel* tidak memerlukan hal tersebut. Berhasilnya *deployment* aplikasi tersebut juga menunjukkan bahwa infrastruktur yang dibuat sebelumnya berjalan dengan baik.

Dari kesimpulan tersebut, penelitian ini diharapkan bisa untuk diterapkan dalam proses pengembangan aplikasi secara nyata, dikarenakan dapat mempermudah *developer* dalam proses kolaborasi dan mengurangi terjadinya kesalahan pada manusia.

Terkait dengan penelitian selanjutnya yang sekiranya bisa dilanjutkan berdasarkan penelitian ini yaitu pengujian *deployment* infrastruktur dapat menggunakan *tool* IaC lainnya seperti *Chef*, *Ansible*, *Puppet*, *Vagrant*, dan lain sebagainya. Sedangkan untuk *provider cloud computing* yang digunakan bisa selain GCP, yaitu AWS dan *Microsoft Azure*. Hal tersebut dikarenakan setiap *tools* IaC dan *provider cloud computing* yang digunakan memiliki cara dan dokumentasinya tersendiri untuk membangun sebuah infrastruktur. Sebagai contoh IaC menggunakan *Terraform* pada GCP tentunya memiliki perbedaan konfigurasi dengan IaC menggunakan *Terraform* pada AWS. Walaupun *tool* yang digunakan sama, namun *provider* yang digunakan berbeda sehingga konfigurasi yang dihasilkan tentunya akan berbeda juga walaupun maksud dan tujuannya sama. Kemudian untuk tool CI/CD bisa menggunakan *Github Action*, *TravisCI*, *Jenkins*, dan *tool* CI/CD *open source* lainnya. Alasannya kurang lebih sama seperti alasan pada *tool* IaC dan *provider cloud computing* sebelumnya, dimana setiap *tools* CI/CD memiliki cara dan dokumentasinya tersendiri dan *tools* tersebut pasti berbeda satu sama lain sehingga konfigurasi yang dihasilkan tentunya akan berbeda walaupun maksud dan tujuannya sama. Selain itu diharapkan juga apabilan nanti melakukan perbandingan cara manual dan otomatis dalam deployment infrastruktur, uji coba yang dilakukan bisa lebih banyak untuk memastikan cara otomatis jauh lebih unggul dibandingkan cara manual baik itu dari sisi efektivitas dan efisiensi.

Daftar Rujukan

[1] Alanda, A., Mooduto, H. A., & Hadelina, R. (2022). Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures. *JITCE (Journal of Information Technology and Computer Engineering)*, *6*(02), 50–56
[2] Albaihaqi, M. F., Wilda, A. N., & Sugiantoro, B. (2020). Deploying an Application to Cloud Platform Using Continous Integration and Continous Delivery. *Proceeding International Conference on Science and Engineering*, *3*(April), 279–282
[3] Gustian, D., Fitrisia, Y., Purwantoro E.S.G.S3, S., & Novayani, W. (2023). Implementasi Automation Deployment pada Google Cloud Compute VM menggunakan Terraform. *INOVTEK Polbeng - Seri Informatika*, *8*(1), 50
[4] Hasan, M. M., Bhuiyan, F. A., & Rahman, A. (2020). Testing Practices for Infrastructure as Code. *LANGETI 2020 - Proceedings of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next-Generation Testing, Co-Located with ESEC/FSE 2020*, 7–12
[5] Hidayat, Y., & Arifwidodo, B. (2021). Implementasi Web Server Menggunakan Infrastructure As Code Terraform Berbasis Cloud Computing. *Format Jurnal Ilmiah Teknik Informatika*, *10*(2), 192
[6] K.Janani, K.Anuhya, V.L.Manaswini, V.Likitha, B.Suneetha, & T.Vignesh. (2022). Analysis of CI/CD Application in Kubernetes Architecture. *Mathematical Statistician and Engineering Applications*, *71*(No.4), 11091–11097
[7] Parama, R. A., Studiawan, H., & Akbar, R. J. (2022). Implementasi Continuous Integration dan Continuous Delivery

P-ISSN : 2089-3353
E-ISSN : 2808-9162

Pada Aplikasi myITS Single Sign On. *Jurnal Teknik ITS*, *11*(3)

[8] Pramadika, O., & Chandra, D. W. (2023). Provisioning Google Kubernetes Engine (GKE) Cluster dengan Menggunakan Terraform dan Jenkins pada Dua Environment. *JIPI (Jurnal Ilmiah Penelitian Dan Pembelajaran Informatika)*, *8*(2), 597–606

[9] Pratama, I. P. A. E. (2021). Infrastructure as Code (IaC) Menggunakan OpenStack untuk Kemudahan Pengoperasian Jaringan Cloud Computing (Studi Kasus: Smart City di Provinsi Bali) Infrastructure as Code (IaC) Using OpenStack for Ease of Operation of Cloud Computing Network (Case Study. *Jurnal Ilmu Pengetahuan Dan Teknologi Komunikasi*, *23*(1), 93–105

[10] Wahyu, A. P., & Noviantama, I. G. (2021). Implementasi Contionous Integration Dan Continous Deployment Pada Aplikasi Learning Management System Di Pt. Millennia Solusi Informatika. *Jurnal Ilmiah Teknologi Infomasi Terapan*, *8*(1), 183–186