IMPLEMENTASI DEPLOYMENT APLIKASI WORDPRESS MENGGUNAKAN LAYANAN KUBERNETES CLUSTER PADA GOOGLE CLOUD PLATFORM

Disusun Oleh:

N a m a : Ahmad Kusumo Haryo

NIM : 19523120

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2023

HALAMAN PENGESAHAN DOSEN PEMBIMBING

IMPLEMENTASI DEPLOYMENT APLIKASI WORDPRESS MENGGUNAKAN LAYANAN KUBERNETES CLUSTER PADA GOOGLE CLOUD PLATFORM

TUGAS AKHIR JALUR MAGANG

Disusun Oleh:

N a m a	: Ahmad Kusumo Haryo
NIM	: 19523120

Yogyakarta, 21 Juli 2023

Pembimbing,

(Chandra Kusuma Dewa, S.Kom., M.Cs., Ph.D.)

HALAMAN PENGESAHAN DOSEN PENGUJI

IMPLEMENTASI DEPLOYMENT APLIKASI WORDPRESS MENGGUNAKAN KUBERNETES CLUSTER PADA GOOGLE CLOUD PLATFORM

TUGAS AKHIR JALUR MAGANG

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 21 Juli 2023

Tim Penguji

Chandra Kusuma Dewa, S.Kom., M.Cs., Ph.D. _____

Anggota 1

Elyza Gustri Wahyuni, S.T., M.Cs. _____

Anggota 2

Sri Mulyati, S.Kom., M.Kom. _____

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Ahmad Kusumo Haryo

NIM : 19523120

Tugas akhir dengan judul:

IMPLEMENTASI DEPLOYMENT APLIKASI WORDPRESS MENGGUNAKAN KUBERNETES CLUSTER PADA GOOGLE CLOUD PLATFORM

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 21 Juli 2023

(Ahmad Kusumo Haryo)

HALAMAN PERSEMBAHAN

Alhamdulillahirobbil 'alamin, sholawat serta salam tercurahkan kepada Rasulullah Muhammad SAW sebagai pedoman serta tauladan bagi jutaan umatnya. Tugas akhir ini merupakan persembahan untuk kedua orang tua yang telah mendukung dalam doa, moril, materil. Serta kepada dosen pembimbing, Bapak Chandra Kusuma Dewa yang telah memberikan masukkan dan arahan serta waktu dalam proses bimbingan tugas akhir ini. Kemudian kepada para sanak saudara, kerabat, dan keluarga yang memberikan doa dan juga dukungannya. Serta semua guru, asatidz, dan masyaikh yang telah membimbing, memberikan ilmu dan nasihat berharga. Tidak lupa juga kepada teman – teman yang selalu membantu dalam memberikan masukkan dan juga dukungan serta motivasi dalam pembuatan tugas akhir ini. Terima kasih atas segalanya, semoga selalu menjadi pahala jariyah bagi kalian dan mendapatkan berkah dan ridho Allah SWT, *aamiin ya rabbal 'alamin*.

HALAMAN MOTO

“Hard times, make great people”.
- Unknown

“Apabila engkau melihat seseorang mengunggulimu dalam masalah dunia, maka unggulilah dia dalam masalah akhirat”
- Hasan Al Bashri

“Every choice comes with a consequence. Once you make a choice, you must accept responsibility. You cannot escape the consequences of your choices, whether you like them or not.”
- Roy T. Bennet

“Ya Allah, aku tidak memohon kepada-Mu untuk mengubah takdirku.
Aku hanya memohon kepada-Mu agar Engkau memberikanku kekuatan, agar aku, mampu.”
- AK Haryo

“Tidak peduli betapa kuatnya dirimu, jangan pernah mengatasi semuanya sendirian, sebaliknya kau pasti akan gagal.”
- Uchiha Itachi

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatuh.

Alhamdulillahil karimirrahman, 'allamal quran, khalaqal insan, 'allamahul bayan. Puji dan syukur atas kehadirat Allah SWT yang telah melimpahkan rahmat dan karunia kepada para hamba-Nya. Shalawat serta salam dipanjatkan atas kehadirat Nabi Muhammad Saw yang telah membimbing umat manusia dari keterpurukan menuju kejayaan. Pada kesempatan kali ini izinkan saya untuk menyampaikan terima kasih banyak kepada pihak-pihak yang telah membantu dalam pembuatan laporan ini sebagai syarat tugas akhir penjaluran magang, sekaligus sebagai syarat kelulusan untuk memperoleh gelar sarjana Program Studi Informatika Fakultas Teknologi Industri Universitas Islam Indonesia. Pihak-pihak yang terkait adalah sebagai berikut:

1. Allah SWT yang telah memberikan rahmat dan karunia-Nya sehingga laporan ini dapat selesai dengan baik.
2. Nabi Muhammad Saw yang selalu menjadi panutan dan contoh tauladan bagi umatnya.
3. Kedua orang tua yang selalu memberikan doa, dukungan moril, dan dukungan materil.
4. Bapak Prof. Fathul Wahid, S.T., M.Sc., selaku rektor Universitas Islam Indonesia.
5. Bapak Prof. Dr. Ir. Hari Purnomo, M.T., selaku dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
6. Ketua Program Studi Informatika, Bapak Dr. Raden Teguh Dirgahayu, S.T., M.Sc, selaku ketua program studi Informatika Universitas Islam Indonesia.
7. Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D., selaku Ketua Program Studi Informatika Program Sarjana Fakultas Teknologi Industri Universitas Islam Indonesia.
8. Dosen pembimbing, Bapak Chandra Kusuma Dewa, S.Kom., M.C., yang senantiasa memberikan arahan dan bimbingan terhadap penulis.
9. Dosen pembimbing akademik, Ibu Elyza Gustri Wahyuni, S.T., M.Cs., yang senantiasa memberikan saran kepada penulis.
10. Dosen mata kuliah Manajemen Diri, Bapak Hanson Prihantoro Putro, S.T., M.T. dan Ibu Aridhanyati Arifin, S.T., M.CS., yang telah memberikan arahan ketika magang berlangsung, serta membantu dalam proses penyusunan Laporan Tengah ini.

11. Bapak H. Sukardi, pakde dari penulis yang telah memberikan dukungan dan semangat dalam pelaksanaan kegiatan magang.
12. Mas Adeva Oktoveri, kakak saudara dari penulis yang telah memberikan dukungan dalam pelaksanaan kegiatan magang
13. Bapak Ridwan dan bang Algie Sukma Pratama serta tim Infrastruktur di divisi Teknologi Informasi PT. Bio Farma yang telah membimbing dan memberikan arahan dalam kegiatan magang.
14. Para guru, dosen, asatidz, masyaikh, dan juga orang-orang yang telah mengajarkan penulis hal-hal yang baik dan bermanfaat walaupun hanya satu huruf.
15. Teman-teman seperjuangan yang telah memberikan dukungan, semangat, makanan dan bantuan lain dalam kegiatan magang.
16. Para mentor dan guru virtual dari manapun berada yang telah memberikan pemahaman dan juga kemudahan dikala datangnya kesulitan.

Semoga Allah SWT melimpahkan rahmat, taufik, inayah dan hidayahnya kepada kalian semua dan juga orang-orang yang tidak sempat dimasukkan pada daftar di atas. Semoga kita semua diberikan kemudahan dan keselamatan dalam kehidupan dunia dan akhirat, serta dijauhkan dari siksa api neraka dan Allah SWT meridhoi kita untuk dimasukkan ke surga-Nya. *Aamiin ya rabbal 'alamin.*

Wassalamu'alaikum Warahmatullahi Wabarakatuh.

Yogyakarta, 21 Juli 2023

(Ahmad Kusumo Haryo)

SARI

Setiap perusahaan memiliki produk digitalnya masing-masing, dengan pesatnya teknologi digital dan kebutuhan dari pengguna, mengharuskan perusahaan untuk dapat men-*deliver* layanan digitalnya agar selalu tersedia dan dapat diakses kapanpun. Demi menyajikan layanan digital yang ada setiap saat dan dalam jumlah yang banyak, para pengembang di perusahaan membuat aplikasi sesuai dengan banyaknya permintaan dan juga kebutuhan, hal ini menimbulkan masalah pada beberapa hal, misalnya semakin banyaknya aplikasi, semakin banyak pula server yang dibutuhkan, tentunya hal ini menambah biaya untuk pengembangan aplikasi, selain itu adanya dependensi antara aplikasi satu dengan yang lainnya membuat aplikasi tersebut mengalami masalah apabila pengembang mengeluarkan versi terbaru dari aplikasi. PT. Bio Farma merupakan salah satu perusahaan BUMN yang bergerak di bidang produksi vaksin, antisera, dan produk biologi lainnya. Dalam hal ini penulis berkesempatan untuk mengambil bagian pada magang pada divisi Teknologi Informasi, di tim Infrastruktur dan Operasional TI PT. Bio Farma, sebagai *Cloud Infrastructure Engineer*. Tim infrastruktur sendiri mengurus berbagai keperluan dalam lingkungan pengembangan aplikasi PT. Bio Farma, diantaranya merupakan bagian *IT Operations* yang menyiapkan lingkungan pengembangan aplikasi, lebih khususnya dalam tahap *deployment* aplikasi. *Google cloud platform* (GCP) merupakan salah satu platform *cloud computing* terkenal yang memiliki banyak layanan, seperti diantaranya Google Kubernetes Engine. Kubernetes yang merupakan teknologi orkestrasi *container* memungkinkan pengembang untuk memanajemen *container* yang berisikan aplikasi dalam jumlah yang besar, hal tersebut dapat menguntungkan aplikasi agar tidak mudah mengalami *downtime*, karena *container* sendiri memiliki sumber daya yang terpisah dari aplikasi lainnya. Dalam penelitian ini akan dilakukan uji coba untuk penerapan - *deployment* aplikasi wordpress dan wiki.js melalui kubernetes dan pengujian dalam *load testing* dalam variasi mulai dari 100, 500, 1000, 1500, dan 2000 *request*. Hasil uji coba dari penelitian ini adalah aplikasi wordpress dan wiki.js yang dapat diakses dengan cara *deployment* melalui teknologi cluster kubernetes dan *load testing* menyimpulkan hasil pengujian mendapatkan waktu yang variatif serta *availability* layanan Kubernetes masih dapat tersedia, sekalipun jumlah akses *user* dan *request* mencapai 1500 dan 2000 *request*.

Kata kunci: Kubernetes, *cloud computing*, GCP, *deployment*, wordpress, wikijs, *load testing*

GLOSARIUM

Apache bench	Perangkat pengujian web server *open source* milik Apache Organization.
CMS	Content Management System.
GCP	Google Cloud Platform.
Instance	Layanan sumber daya server yang disediakan oleh layanan *cloud*.
Supervisor	Pengawas atau mentor yang menjadi pembimbing dalam kegiatan magang.
Wikijs	Aplikasi CMS wiki *open-source* yang berjalanan dari bahasa pemrograman Javascript.
Wordpress	Aplikasi CMS *open-source* yang dapat dengan mudah untuk digunakan dan diintegrasikan, yang biasa digunakan untuk sebuah blog.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Profil Institusi Magang

Gambar 1.1 Logo PT. Bio Farma

PT. Bio Farma merupakan salah satu perusahaan BUMN yang bergerak di bidang produksi vaksin, antisera, dan produk biologi lainnya. PT. Bio Farma memiliki visi "Menjadi Perusahaan Life Science Kelas Dunia yang berdaya saing Global". Adapun dalam misinya PT. Bio Farma yaitu "Menyediakan dan mengembangkan produk life science berstandar internasional untuk meningkatkan kualitas hidup", yang mana hal ini sejalan dengan filosofi PT. Bio Farma yaitu mengabdi untuk meningkatkan kualitas hidup yang lebih baik.

Dahulu, PT. Bio Farma didirikan oleh pemerintah kolonial Hindia-Belanda pada tanggal 6 Agustus 1890 di Jakarta dengan nama *Parc-vaccinogène* yang artinya Lembaga Pengembangan Vaksin Negara. Lembaga tersebut menempati sebuah gedung di Batavia, yang kini menjadi Rumah Sakit Pusat Angkatan Darat (RSPAD) Gatot Soebroto, Jakarta. Pada awal berdirinya, lembaga ini berfokus pada penelitian untuk memberantas penyakit menular hingga pada akhirnya perusahaan tersebut menjalin kerjasama dengan Institut Pasteur dalam penelitian mikrobiologi dan perusahaan berganti nama menjadi *Parc-vaccinogène en Instituut Pasteur*.

Gambar 1.2 Gedung PT. Bio Farma zaman kolonial Belanda

Pada tahun 1923, *Parc-vaccinogène en Instituut Pasteur* berpindah tempat ke jalan Pasteur No.28, Bandung. Setelah kemerdekaan, tepatnya pada tahun 1955, perusahaan berubah nama lagi menjadi Perusahaan Negara Pasteur. Tahun 1961, berganti nama lagi menjadi Perusahaan Negara Bio Farma. Hingga pada akhirnya di tahun 1997 berdasarkan Peraturan Pemerintah RI Nomor 1 tahun 1997, perusahaan berubah menjadi Perusahaan Perseroan (Persero) yang sahamnya sepenuhnya menjadi milik Pemerintah Indonesia dengan nama PT. Bio Farma hingga kini, yang berkedudukan di Jalan Pasteur No. 28 Bandung, Jawa Barat (Biofarma, 2023).

Saat penempatan magang, penulis berkesempatan untuk mengambil bagian di divisi Teknologi Informasi, di tim Infrastruktur dan Operasional TI, sebagai *Cloud Infrastructure Engineer*. Tim infrastruktur sendiri mengurus berbagai keperluan dalam lingkungan pengembangan aplikasi PT. Bio Farma, diantaranya merupakan bagian *IT Operations* yang menyiapkan lingkungan pengembangan aplikasi, lebih khususnya dalam tahap *deployment* aplikasi. Suasana lingkungan kerja di PT. Bio Farma sangat menarik, setiap karyawan satu sama lainnya sangat menghormati dan tidak membeda-bedakan, sehingga semuanya dapat diterima dengan baik. Budaya kekeluargaan sangat kental dan juga membuat suasana semakin nyaman bagi para karyawannya, dan juga karena PT. Bio Farma merupakan perusahaan yang berada dibawah pemerintah atau BUMN, nilai-nilai AKHLAK (Amanah, Kompeten, Harmonis, Loyal, Adaptif, Kolaboratif) merupakan pedoman bagi sumber daya manusia yang ada di perusahaan ini.

1.2 Latar Belakang

Revolusi industri adalah perubahan cara hidup dan proses kerja manusia secara fundamental, yang mana teknologi informasi diintegrasikan dalam kehidupan dengan dunia digital (Hamdan, 2018). Sebagai upaya untuk beradaptasi dan bertransformasi ke ranah digital, PT. Bio Farma mengikuti alur revolusi industri yang semakin berkembang, terlebih di masa revolusi industri 4.0 saat ini. Bio Farma yang merupakan induk dari holding BUMN farmasi berusaha untuk mengembangkan layanan di bidang kesehatan secara *end-to-end*. *Digital Healthcare* (DHC) Bio Farma yang merupakan unit kerja transformasi digital, saat ini tengah mengembangkan beberapa inovasi baru. Kategori produk diantaranya yaitu kategori *Business to Business* yang terdiri dari Sistem Manajemen Distribusi Vaksin (SMDV) merupakan sistem informasi yang dapat memastikan pengelolaan distribusi vaksin dapat berjalan lancar, yang mana pada masa pandemi Covid – 19 sistem informasi ini sangat memiliki pengaruh besar sehingga pendistribusian 460 juta dosis vaksin Covid – 19 dapat terjamin kualitasnya. Selain itu, terdapat juga Q100+ yang merupakan unit digitalisasi dalam ranah manufaktur untuk menjaga integritas data yang meliputi ruang lingkup produksi, *Quality Control* (QC), dan *Quality Assurance* (QA) (Biofarma, 2022).

Demi menghasilkan produk – produk *Digital Healthcare* yang berkualitas, hal tersebut membutuhkan beberapa perangkat, SDM, dan pengelolaan yang berkualitas sehingga transformasi digital di Bio Farma dapat berjalan sesuai dan berkesinambungan. Salah satu penggunaan teknologi di PT. Bio Farma yaitu penggunaan infrastruktur *microservice* dari Kubernetes cluster pada Google Cloud Platform. Penggunaan Kubernetes di perusahaan Bio Farma sebagai infrastruktunya didasari karena banyaknya aplikasi yang dikembangkan, serta setiap aplikasi memiliki servis atau modul tersendiri yang membuat aplikasi semakin kompleks. Karena semakin banyaknya servis atau modul pada tiap aplikasi, diperlukan juga pendukung dalam men-*deliver* aplikasi secara cepat, berkesinambungan dan mudah dalam pengelolaan, maka digunakanlah layanan Kubernetes ini oleh PT. Bio Farma.

Saat mengikuti kegiatan magang di tim Infrastruktur dan Operasi TI sebagai *Infrastructure Cloud Engineer*, penulis berkesempatan untuk mencoba penggunaan layanan Kubernetes cluster pada Google Cloud Platform. Google Cloud Platform (GCP) sebagai penyedia layanan infrastruktur teknologi informasi memiliki banyak layanan di dalamnya, seperti halnya Google Kubernetes Engine (GKE) yang digunakan untuk pengelolaan aplikasi berbasis kontainer. Pada kegiatan magang yang diikuti, penulis mengerjakan beberapa pekerjaan, diantaranya penggunaan layanan Kubernetes cluster dalam *deployment* aplikasi

Wordpress dan Wiki.js. *Deployment* aplikasi tersebut dimulai dengan beberapa tahapan seperti konfigurasi pada cluster, konfigurasi database, konfigurasi jaringan, hingga pengujian aplikasi website Wordpress yang di-*deploy* menggunakan Kubernetes cluster.

Pada tahap *deployment* telah menghasilkan aplikasi website berbasis Wordpress dan Wiki.js yang dapat diakses. Kemudian berlanjut ke tahapan pengujian yang mana pengujian website Wordpress pada layanan Kubernetes cluster diuji dengan pengujian beban atau istilah lainnya yaitu *load testing*. Dari pengujian beban yang dilakukan, bertujuan untuk mengetahui waktu yang diperlukan untuk mengakses layanan dengan beberapa variasi *request* yang berbeda, diantaranya 100 *request,* 500 *request,* 1000 *request,* 1500 *request,* dan 2000 *request*. Dari pengujian yang dilakukan menghasilkan beberapa parameter yang akan menjadi tolak ukur pengujian diantaranya *time taken for test, request per second,* dan *time per request* serta parameter pada penggunaan CPU dan juga memori di layanan Kubernetes cluster GCP. Dari hasil tersebut pula mendapatkan kesimpulan bahwa saat pengujian tidak ditemukan kegagalan dalam pemenuhan permintaan yang menandakan servis atau layanan dari Wordpress yang ada pada Kubernetes cluster GCP dapat memenuhi permintaan dari 100, 500, 1000, 1500, dan 2000 *request* dengan waktu yang variatif namun dapat terpenuhi. Dalam penggunaan CPU sempat memenuhi ambang batas, namun saat mengakses website Wordpress pada layanan Kubernetes masih tetap bisa dilakukan. Hal tersebut menandakan bahwa *availability* layanan Kubernetes masih dapat tersedia, sekalipun jumlah akses *user* dan *request* mencapai 1500 dan 2000 *request*.

1.3 Ruang Lingkup

Selama mengikuti kegiatan magang sebagai *Infrastructure Cloud Engineer*, penulis mengerjakan beberapa proyek yang diberikan oleh mentor. Kebanyakan dari proyek tersebut yaitu hal-hal yang berkaitan dengan mempersiapkan pengembangan aplikasi dan juga lingkungan dari aplikasi tersebut yang berbasis *Cloud Computing* pada layanan Google Cloud Platform. Selain itu, penulis melakukan beberapa pekerjaan yang menggunakan layanan Google Kubernetes Engine (GKE), yang diantara peruntukannya yaitu:

a. Deployment aplikasi Wikijs melalui Kubernetes pada GCP
b. Deployment aplikasi Wordpress melalui Kubernetes pada GCP
c. Melakukan load testing menggunakan Apache bench pada aplikasi Wordpress

1.4 Tujuan

Tujuan yang diraih dalam ruang lingkup ini yaitu, mengimplementasikan penggunaan teknologi Kubernetes cluster yang ada pada layanan Google Cloud Platform untuk *deployment* aplikasi Wordpress dan Wiki.js. Kemudian dilakukan pengujian aplikasi website berbasis Wordpress yang ada pada Kubernetes cluster dengan cara melakukan simulasi *load testing* pada 100, 500, 1000, 1500, dan 2000 *request* untuk mengetahui performansi dari website Wordpress yang ada pada layanan Kubernetes cluster.

1.5 Manfaat

Manfaat yang dapat diperoleh dari ruang lingkup laporan tugas akhir ini yaitu:

a. Deployment aplikasi Wordpress dan Wiki.js dapat terimplementasikan pada layanan Kubernetes cluster yang ada pada Google Cloud Platform.
b. Dari pengujian yang dilakukan dapat mengetahui durasi dari aplikasi agar bisa menampilkan halaman.
c. Dari pengujian yang dilakukan dapat mengetahui ketersediaan aplikasi yang di-*deploy* pada layanan Kubernetes cluster GCP.

1.6 Sistematika Penulisan

Susunan sistematika penulisan dalam laporan ini sebagai berikut:

a. BAB I: Pendahuluan

   Bab ini menjelaskan tentang pembahasan secara umum yang meliputi latar belakang, ruang lingkup magang, tujuan dan manfaat serta sistematika penulisan.

b. BAB II: Landasan Teori dan Tinjauan Pusaka

   Bab ini berisi teori – teori pembahasan dari penelitian sebelumnya yang menjadi landasan dan juga pendukung atas penugasan yang dikerjakan selama mengikuti kegiatan magang.

c. BAB III: Pelaksanaan Magang

   Bab ini berisi uraian dari tugas dan aktivitas yang dilakukan ketika pelaksanaan magang.

d. BAB IV: Refleksi Pelaksanaan Magang

   Bab ini berisi penjelasan mengenai relevansi akademik dan pembelajaran yang diapatkan selama melaksanakan kegiatan magang.

e. BAB V: Penutup

Bab ini berisi kesimpulan dan saran dari pembahasan yang telah disampaikan pada bab-bab sebelumnya.

BAB II
LANDASAN TEORI DAN TINJAUAN PUSTAKA

2.1 Cloud Computing

Penggunaan Teknologi Informasi dari setiap tahunnya pasti akan berkembang, baik dari segi sistem informasi, bahasa pemrograman, dan bahkan infrastruktur yang merupakan salah satu pendukung bagi berjalanannya sebuah aplikasi. *Cloud Computing* atau komputasi awan merupakan suatu mekanisme yang mana terdapat sekumpulan teknologi informasi yang sumber dayanya saling terhubung dan nyaris tanpa batas. *Cloud Computing* sendiri merupakan bentuk komputasi terdistribusi, hal ini memungkinkan pengguna untuk mengaksesnya secara *remote* atau secara jauh asalkan perangkat yang digunakan terkoneksi dengan internet (Riana, 2020). Dikarenakan infrastruktur ataupun aplikasi yang dimiliki dan dikelola sepenuhnya oleh pihak ketiga, hal tersebut memungkinkan pengguna untuk menggunakan sumber daya tersebut tergantung dari kebutuhan pengguna. Berbeda dengan komputasi tradisional, pengguna diharuskan untuk memiliki perangkat keras dan perangkat lunaknya sendiri, hal ini tentu mengharuskan pengguna untuk mengeluarkan lebih banyak dana untuk pengelolaan dan pemeliharaan dari perangkat tersebut.

Menurut penelitian (Ahmad & Herri, 2011) terdapat 5 karakteristik yang ada pada *cloud computing,* yaitu:

1. *On-demand.* Hal ini memungkinkan pengguna untuk dapat mengonfigurasi layanan sesuai kebutuhan yang diinginkan.
2. *Broad network access.* Layanan ini merupakan akses jaringan yang luas yang memungkinkan untuk dapat tersedia dari mana saja dan kapan saja dengan perangkat yang dapat mengakses layanan (seperti, laptop, *smartphone*, dll.).
3. *Resource pooling.* Sumberdaya komputasi yang dimiliki penyedia layanan dapat digunakan secara bersamaan oleh beberapa pengguna.
4. *Rapid elasticity.* Layanan ini disediakan secara elastis dan cepat untuk memenuhi kebutuhan, dengan ketersediaan dan kapasitas yang tidak terbatas sehingga dapat disesuaikan dengan spesifikasi yang diinginkan setiap saat.
5. *Measured service.* Terdapat sistem yang secara otomatis dapat mengawasi dan memaksimalkan sumber daya dengan memaanfaatkan pengukuran untuk dapat mengukur penggunaan sumber daya komputasi yang telah terpakai.

2.2 Deployment

Deployment merupakan salah satu tahapan dalam proses pengembangan aplikasi atau perangkat lunak. Deployment yang dalam artian memindahkan aplikasi dari lingkungan pengembangan ke lingkungan yang dimana dapat dikunjungi oleh pengguna. Dalam hal aplikasi web, proses deployment melibatkan pemilihan host, penyiapan web server dan database, serta pemindahan semua file ke tempat yang tepat dan dengan akses izin yang tepat (Wisnu, Adam, & Hayati, 2013). Singkatnya deployment merupakan kegiatan dalam pengembangan aplikasi yang bertujuan untuk menyebarkan aplikasi yang telah dibuat oleh para *developer*.

2.3 Google Cloud Platform

Dengan seiring berjalannya waktu, komputasi awan yang sangat efektif saat ini banyak dibutuhkan oleh beberapa konsumen. Beberapa perusahaan yang bergerak di bidang IT banyak yang membuat dan menyewakan jasa layanan komputasi awan, seperti yang terkenal adalah *Amazon Web Service* (AWS), *Microsoft Azure*, dan juga perusahaan lain yang menyediakan jasa layanan komputasi awannya dengan keunggulan yang berbeda, tidak terkecuali Google. Sebagai perusahaan raksasa di dibang IT, Google membuat salah satu platform komputasi awan mereka sendiri yang dinamai *Google Cloud Platform* (GCP).

Google Cloud Platform merupakan layanan untuk memberikan kemudahan bagi setiap perusahaan untuk mengembangkan aplikasi dalam bisnis serta menawarkan berbagai produk layanan yang dapat digunakan untuk membangun infrastruktur server dengan tingkat standar yang tinggi. Setiap produk dan layanannya disediakan untuk memiliki serangkaian fitur dan manfaat yang berbeda sesuai dengan kebutuhan penggunanya (Dinda & Ria, 2022).

2.4 Kubernetes

Kubernetes adalah suatu layanan *open-source* yang mampu melakukan orkestrasi container dalam jumlah yang besar yang memiliki tugas untuk melakukan penjadwalan, penskalaan, recovery, dan monitoring container. Pada awalnya sistem ini dikembangkan oleh Google sebagai sistem internal mereka yang digunakan untuk membantu developer dan teknisi infrasutruktur IT untuk mengelola ribuan server yang ada di Google yang dinamai dengan *Borg* kemudian *Omeda* pada beberapa tahun setelahnya. Dengan fungsinya yang sebagai *clustering maganement container* membuat layanan Kubernetes dapat membuat sebuah aplikasi selalu tersedia dan dapat menerima banyak permintaan. Pada penggunaan

Kubernetes cluster dibutuhkan *service* sebagai jembatan yang dapat mengakses satu atau lebih dari satu *Pod*, dan *service* harus di ekspos ke luar jaringan agar aplikasi dari luar Kubernetes cluster dapat mengakses *Pod* yang berada diluar *service* tersebut (Wisnu & Fadhly, 2022).

Pada penelitian (Kezia, Agustinus, & Henry, 2019) implementasi container kubernetes untuk mendukung *scalability*, Kubernetes memiliki arsitekturnya tersendiri yang terdiri dari:

1. *Master Node.* Merupakan *master* yang berfungsi untuk mengontrol keseluruhan unit dari cluster, mengatur workload, dan komunikasi antar sistem. Berikut ini merupakan komponen yang ada pada *master node*:
    a. *Etcd.* Merupakan storage untuk menyimpan data *cluster* kubernetes.
    b. API *server.* Merupakan penyedia antarmuka internal ataupun eksternal pada Kubernetes.
    c. *Scheduler.* Merupakan komponen pengatur jadwal untuk *node* dan *pod*, berfungsi untuk memastikan beban kerja tidak melebihi sumber daya yang ada.
    d. *Controller manager.* Berfungsi untuk memonitor *cluster* agar sesuai dengan konfigurasi yang diinginkan.

2. *Worker Node.* Sesuai namanya, *worker node* merupakan sistem pekerja atau *minion* yang menjadi tempat mesin *container* (beban kerja) digunakan. Berikut adalah komponen yang ada pada *worker node*:
    a. *Kubelet.* Berfungsi untuk memastikan *pod* beroperasi pada *master node.*
    b. *Kube – proxy.* Berfungsi meneruskan koneksi dan mengatur jaringan serta membantu abstraksi layanan Kubernetes.
    c. Container runtime. Merupakan komponen yang bertanggung jawab untuk menampung aplikasi pada *container* serta menjalankan *container*. Docker merupakan salah satu *container orginizer* yang dapat digunakan dalam layanan Kubernetes.

Masih pada penelitian yang sama (Kezia, Agustinus, & Henry, 2019), Kubernetes memiliki beberapa objek dasar, diantaranya:

1. *Pod.* Merupakan unit yang terkecil dan sederhana yang ada pada Kubernetes. Berfungsi untuk menjalankan *docker images* pada *container*.

2. *Service.* Merupakan abstraksi layanan kubernetes yang berfungsi untuk mengarahkan *request* ke beberapa *pod* dengan *IP Address*.
3. *Volume.* Objek yang berfungsi sebagai media penyimpanan data suatu *container*.
4. *Namespace.* Merupakan objek yang berfungsi untuk memisahkan *resource* dari *cluster*.
5. *Node.* Objek ini merupakan mesin pekerja, bisa berupa *virtual machine* atau mesin fisik. Setiap *node* dapat berisi banyak *pod* serta *master*.

2.5 Wiki.js

Saat mendengar kata "wiki", kebanyakan orang akan mengingat atau menyebutkan Wikipedia, yang merupakan ensiklopedia online yang mana penggunanya dapat memberikan kontribusi pemikiran dan pengetahuan mereka ke setiap halaman Wikipedia. Padahal wiki adalah salah satu jenis aplikasi web yang merupakan halaman web dimana para pengguna halaman web tersebut dapat mengorganisir, mengedit, ataupun meninjau isi dari halaman web dengan mudah dan juga dapat berkolaborasi dengan pengguna lainnya dan masih ada banyak aplikasi wiki selain Wikipedia. Aplikasi wiki ini diciptakan oleh Ward Cunningham di tahun 1994. Sampai saat ini ada beberapa aplikasi wiki yang dijalankan dalam berbagai bahasa pemrograman yang termasuk diantaranya aplikasi wiki *open-source* yang berjalanan dari bahasa pemrograman Javascript seperti Wiki.js. Wiki.js merupakan salah satu perangkat lunak wiki berkemampuan modern yang didukung oleh Node.js, Markdown, dan Git, dan dikembangkan oleh Nicolas Giard. Dengan rilis awal di bulan September 2016, Wiki.js menjadi perangkat lunak termuda dari aplikasi Wiki (Jaroslav & Jakub, 2018).

2.6 Wordpress

Wordpress merupakan salah satu aplikasi web *content management system* atau CMS yang berbasis *open-source* yang dapat dengan mudah untuk digunakan dan diintegrasikan, serta memiliki kebebasan dalam pola desain *Model View Controller* (MVC) (Dian & dkk, 2020). Selain sangat mudah dan dapat digunakan untuk berbagai keperluan untuk membangun website ataupun blog dengan Wordpress, seseorang tidak perlu memiliki latar belakang keahlian programming.

2.7 Tinjauan Pustaka

Pada bagian ini akan menunjukkan penelitian – penelitian terkait yang telah dilakukan sebelumnya dari berbagai referensi yang berkaitan dengan penelitian ini, untuk menunjukkan persamaan dan perbedaan pada topik yang dipilih oleh penulis.

Tabel 2.1 Penelitian terkait

No	Judul	Penulis	Hasil
1	Implementasi Kubernetes Cluster Menggunakan Vagrant	Muhammad Ramadhan Muttakin, Muhammad Arif Fadhly Ridha	Dalam pengembangan aplikasi yang berbasis teknologi *container*, diperlukan manajemen terhadap banyaknya *container* yang berisikan aplikasi yang telah dibuat. Kubernetes sebagai salah satu *sofrware* untuk *container management & orchestration* dapat berjalan pada mesin virtual. Pada penelitian ini Kubernetes dicoba untuk dijalankan pada virtualisasi Vagrant dan KVM bertujuan untuk membandingkan antara 2 jenis virtualisasi tersebut dengan melihat tingkat keberhasilan dari *hight availability, scalability,* serta pengetesan performa. Hasil yang didapatkan yaitu, virtualisasi KVM lebih unggul dari pada virtualisasi Vagrant yang dilihat dari perbedaan *latency* dan pengujian performa (Muttakin & Fadhly, 2021).
2	Pengembangan Infrastruktur Analisis Data *Heart Rate* berbasis *Microservices* menggunakan Kubernetes	Abd. Jahiduddin, Eko Sakti Pramukantoro, Fariz Andri Bakhtiar	Dikarenakan dalam melakukan pemrosesan data membutuhkan sumber daya yang besar, proses instalasi yang kompleks, sulit dalam penggunaan alat analisis, serta proses pengembangan dan pemeliharaannya yang rumit. Maka, penelitian ini mencoba untuk memecahkan masalah dengan

			pengembangan infrastruktur analisis data *heart rate* yang menggunakan arsitektur *microservice,* kontainer Docker, serta Kubernetes. Hasil dari penelitian ini menunjukkan bahwa infrastruktur yang dibangun dengan Kubernetes mampu melakukan analisis hingga 100.000 data dan saat melakukan analisis 50.000 data terjadi peningkatan waktu analisis sebanyak 7 hingga 8 kali lipat. Lalu dihasilkan juga bahwa infrastruktur data *heart rate* dapat melakukan *self healing* dan juga penyimpanan data *persistent* (Jahiduddin, Eko, & Fariz, 2020).
3	Analisis Performansi Proses Scaling pada Kubernetes dan Docker Swarm Menggunakan Metode Horizontal Scaler	Bayu Arifat Firdaus, Vera Suryani, Siti Amatullah Karimah	Pada penelitian ini dilakukan sebuah perbandingan performa terhadap proses *Scaling* antara Kubernetes dan Docker Swarm yang menggunakan metode *Horizontal Scaler*, dengan parameternya adalah *Load Testing* pada skalabilitas, dan juga *Scaling* pada performansi. Hasil dari penelitian ini menunjukkan pada segi skalabilitas, Kubernetes menggunakan lebih banyak sumber daya pada *CPU Utilization* dengan 10.000 user yang dirata – rata memiliki nilai sebesar 94,20%. Sedangkan pada Docker Swarm memiliki nilai rata – rata yaitu 92,28%. Pada segi performa dalam *scaling up* Kubernetes lebih unggul dikarenakan penskalaannya yang automatis sedangkan Docker Swarm unggul dari segi *Load Balancing* dengan waktu rata – rata 55,8 detik, adapun

			Kubernetes di 61,2 detik. Lalu, pada *scaling down* Docker Swarm unggul saat penghapusan *container,* dengan penghapusannya yang manual memiliki waktu rata – rata yakni 11,4 detik. Berbeda dengan Kubernetes yang lebih lama dalam penghapusan *container* secara automatis dan dengan waktu rata – rata 4 menit 49 detik (Bayu, Vera, & Siti, 2020).
4	Perbandingan Kinerja Ingress Controller pada Kubernetes Menggunakan Traefik dan Nginx	Wisnu Ramadhani, Muhammad Arif Fadhly Ridha	Perbandingan kinerja antara Traefik ingress dan Nginx ingress sebagai *ingress controller* yang digunakan pada Kubernetes dalam penelitian ini bertujuan untuk melihat perbedaan performa saat cluster menerima *request*. Kesimpulan dari penelitian ini menunjukkan bahwa dalam pengujian yang menggunakan Apache Benchmark, server dari Kubernetes cluster ingress Traefik bisa menyelesaikan 100 hingga 500 *client request* lebih cepat dibandingkan server Kubernetes cluster Nginx. Meskipun server Kubernetes Traefik lebih unggul, namun penggunaan memori lebih besar dibandingkan dengan server Kubernetes Nginx. Server Kubernetes Nginx lebih tepat digunakan pada perangkat server yang menengah kebawah, dikarenakan penggunaan memori Nginx lebih kecil dari pada Traefik (Wisnu & Fadhly, 2022).
5	Implementasi Container	Kezia Yedutun, Agustinus	Dalam penelitian untuk mengimplementasikan Container

	Kubernetes untuk Mendukung Scalability	Noertjahyana, Henry Novianus Palit	Kubernetes untuk mendukung Scalability. Cakupannya adalah untuk mengetahui pengaruh implementasi *scalability* terhadap kinerja dari *container* Kubernetes, dengan melihat perbandingan dari *response time* serta *concurrent user*. Penelitian ini menyimpulkan bahwa dengan penerapan skalabilitas pada *container* akan terdapat penghematan pada *cpu usage pod*, lalu pada perbandingan *concurent user* antara *single* dan *multiple* server berkisar 1.998 : 1.998. Untuk perbandingan *resnponse time* antara *single* dan *multiple* server memakan waktu lebih lama pada *multiple* server dikarenakan adanya *delay* pembuatan *container* (Kezia, Agustinus, & Henry, 2019).
6	Analisis Perbandingan Performa Web Server Docker Swarm dengan Kubernetes Cluster	Stefanus Eko Prasetyo, Yulfan Salimin	Dalam melakukan penelitian ini berfokus untuk merancang dan menguji performa antara infrastruktur Kubernetes dan Docker Swarm untuk mengetahui perbandingan kinerja dari dua kontainer manajemen tersebut. Kesimpulan dari penelitian ini mengemukakan bahwa antara Kubernetes dan Docker Swarm memiliki keunggulannya masing – masing sebagai contoh pada *load testing*, Kubernetes memiliki *response time* 300 dan 2000 *thread* yang lebih cepat dibanding Docker Swarm. Namun, Docker Swarm memberikan kinerja yang tidak jauh dari Kubernetes dengan

			keunggulan dalam menggunakan sumber daya yang lebih efisien (Stefanus & Yulfan, 2021).
7	Cloud Storage dengan Teknologi Kubernetes untuk Platform Collaborative Research	Muhammad A. Nugroho, Rikie Kartadie	Penelitian ini mencoba untuk membuat sebuah sistem atau aplikasi berbasis *cloud* untuk penyimpanan data dari kolaborasi riset antar peneliti. Dengan menggunakan Kubernetes dan Nextcloud yang diintegrasikan dalam membangun *cloud storage,* diharapkan dapat memenuhi kebutuhan dalam *sharing* dan juga berkolaborasi dalam penelitian. Kesimpulan dari peneltian ini adalah penyimpanan berbasis *cloud* dapat dibangun dengan Kubernetes dan juga Nextcloud yang telah diintegrasikan, dan pada pengujian kinerja kecepatan dalam pengaksesannya mendapatkan nilai A. Adapun dalam pengujian kinerja *user request* dengan penggunaan dari 20 *user, cloud storage* ini mampu berjalan dengan *connect time* 0 – 5 ms. Kesimpulan lainnya yakni penurunan kinerja dapat diatasi juga dengan penggunaan model *scaling container* pada Kubernetes (Nugroho & Kartadie, 2021).
8	Provisioning Google Kubernetes Engine Cluster dengan Menggunakan Terraform dan Jenkins pada Dua	Odi Pramadika, Dian Widiyanto Chandra	Dalam penelitian ini berfokus pada uji coba *provisioning* GKE Cluster dengan Terraform sebagai *Infrastructure as Code* (IaC) yang mengelola server serta layanan cloud dan Jenkins sebagai pendukung dalam mempercepat alur kerja *Continuous Integration / Continuous Deployment*

	Environment		(CI/CD) pada dua lingkungan yang berbeda, yaitu *development* dan *production*. Penelitian ini dilakukan untuk memberikan kemudahan bagi para tim *DevOps* dalam menyiapkan server yang sesuai kebutuhan. Hasil dari penelitian ini membuktikan bahwa waktu yang dibutuhkan untuk menyiapkan satu cluster dengan tiga node di lingkungan *development* hanya memakan waktu 15 menit 18,412 detik, sedangkan untuk menyiapkannya pada lingkungan *production* dapat lebih cepat dengan torehan waktu 13 menit 26,407 detik, dari hasil ini dapat disimpulkan bahwa *provisioning* GKE cluster menggunakan Terraform dan Jenkins dapat memudahkan dalam menyiapkan server sesuai kebutuhan, cepat, dan minim kendala (Odi & Dian, 2023).
9	Analisis Cluster Container pada Kubernetes dengan Infrastruktur Google Cloud Platform	M. Agung Nugroho, Cuk Subiyantoro	Penelitian ini mencoba untuk melakukan komparasi kinerja pada layanan yang menggunakan *container* sebelum dan sesudah memakai Kubernetes dengan melihat tingkat *availibility service* pada layanan yang menggunakan *Google Cloud Platform*. Penelitian yang dilakukan menggunakan *locus.io* yang diuji coba dengan memberikan beban *request* hingga 25.600. Hasil penelitian ini menemukan bahwa *availability* dari layanan meningkat dan tidak ditemukannya *failure request* dari beban akses mulai dari 100 hingga

			25.600 *user,* serta dapat diselesaikan rata – rata 100% dalam kurun waktu 1 hingga 4 detik. Dengan data tersebut, menjelaskan bahwa Kubernetes dapat meningkatkan ketersediaan layanan dan juga mengurangi persentase dari *failure request* (Agung & Subiyantoro, 2018).
10	Pengamanan *Container Orchestration* Berbasis Kubernetes di Lembaga Penerbangan dan Antariksa Nasional (LAPAN)	Arief Indriarto Haris, Rd. Angga Ferianda, Budhi Riyanto, Fajar Iman Nugraha, Januar Abadi	*One Space* merupakan komitmen untuk pembangunan program Satu Data di lingkungan LAPAN. Peralihan dari teknologi *monolithic* menuju teknologi *container platform* yang menggunakan salah satu instrumennya yaitu orkestrasi kontainer yang berbasis Kubernetes dapat mendukung program Satu Data tersebut. Dengan maksud untuk melakukan tindakan pencegahan terhadap keamanan data, maka penelitian ini bertujuan untuk mengetahui tindakan apa saja yang harus dilakukan untuk mengamankan orkestrasi kontainer yang berbasis Kubernetes di LAPAN. Data demi data dikumpulkan dengan metode studi literatur, observasi, dan wawancara. Kemudian hasil pengumpulan data memperoleh 18 tindakan pengamanan yang harus dinilai. Tahap berikutnya yaitu penelitian ini melakukan prioritasi dengan metode *MoSCoW* dan juga *FGD* bersama pakar dan praktisi IT di LAPAN. Hasil penelitian ini memperoleh 7 tindakan dengan status *must have*, 10 tindakan *should have*, dan 1 tindakan *could have*

			untuk mengamankan orkestrasi kontainer berbasis Kubernetes yang ada di LAPAN (Arief & dkk, 2020).

Melalui hasil tinjauan pustaka dari beberapa penelitian yang telah dideskripsikan pada Tabel 2.1, terdapat beberapa persamaan, yaitu:

1. Penelitian yang terkait sama – sama menggunakan teknologi dari Kubernetes dan beberapa diantaranya menggunakan Kubernetes cluster yang ada pada layanan Google Cloud Platform.
2. Dari tinjauan pustaka yang didapat dari salah satu penelitian tersebut, dilakukan sebuah pengujian yang sama, yaitu dengan menguji beban pada layanan yang di-*deploy* menggunakan Kubernetes.

Adapun perbedaan dari beberapa hasil tinjauan pustaka yang didapatkan pada tabel 2.1 yaitu:

1. Melakukan implementasi teknologi Kubernetes yang ada pada Google Cloud Platform dengan cara men-*deploy* aplikasi pilihan berbasis docker image seperti Wordpress dan Wikijs ke dalam lingkungan Kubernetes cluster.
2. Melakukan uji beban atau *load testing* untuk mengetahui waktu permintaan layanan dan menguji ketersediaan layanan.

BAB III
PELAKSANAAN MAGANG

3.1 Aktivitas Magang

PT. Bio Farma sangat terbuka bagi para mahasiswa ataupun siswa dari sekolah menengah kejuruan untuk merasakan kegiatan magang di perusahaan, hal ini ditemukan saat penulis mengikuti program kegiatan magang. Program magang di PT. Bio Farma merupakan magang bersertifikat yang dapat diikuti oleh para mahasiswa dari berbagai jurusan ilmu pengetahuan dan nantinya akan ditempatkan pada bidang yang sesuai dengan latar belakang jurusan pemagang, seperti halnya di bidang *Human Capital*, *Marketing*, Produksi, Teknologi Informasi, Keuangan & Manajemen Risiko, Hubungan Kelembagaan, Pengembangan Usaha dan bagian lainnya yang ada di PT. Bio Farma. Program magang PT. Bio Farma diharapkan dapat menjadi sebuah proses kepada para pemagang agar dapat menerapkan ilmu dan kompetensi yang dimiliki ke dunia kerja secara langsung dan mengembangkannya, serta dapat memberikan peluang bagi para pemagang untuk menjadi salah satu bagian yang memiliki andil dalam kemajuan perusahaan. Selain itu, pemagang juga mendapatkan pengalaman, ilmu, dan relasi baru serta mengetahui dinamika yang ada pada dunia kerja secara langsung.

Adapun program magang yang diikuti penulis untuk memenuhi syarat penjaluran magang yaitu dilakukan di divisi Teknologi Informasi, lebih tepatnya pada tim Infrastruktur dan Operasi, yang diberikan tugas sebagai *Infrastructure Cloud Engineer*. Kegiatan magang tersebut dilaksanakan dalam kurung waktu kurang lebih 5 bulan, dan dilakukan secara *on site* dengan mematuhi protokol kesehatan yang berlaku. Berikut merupakan detail kegiatan yang dapat dilihat pada tabel 3.1.

Tabel 3.1 Aktivitas kegiatan magang

Aktivitas	Durasi (waktu)
Onboarding	1 hari
Mengikuti penjelasan tentang *Jobdesc* dan lingkungan kerja di perusahaan Bio Farma	1minggu
Mempelajari materi secara mandiri	2 minggu
Menyiapkan aplikasi yang dibutuhkan serta pembuatan sebuah aplikasi	3 minggu

login sederhana	
Mempelajari layanan yang ada di *Google Cloud Platform* serta mendeploy aplikasi melalui *virtual machine*	3 minggu
Mempelajari manajemen database *SQL instance* dengan layanan lain yang ada di *Google Cloud Platform*	3 minggu
Melakukan deploy aplikasi dengan bantuan layanan *virtual machine* dan *SQL instance*, serta pengaplikasian *docker*	4 minggu
Melakukan deploy aplikasi dengan menggunakan layanan Kubernetes Cluster serta menggunakan *SQL instance*	4 minggu

3.1.1 *Onboarding* dan Pengenalan Lingkungan PT. Bio Farma

Pada hari sebelum penempatan magang, penulis diharuskan untuk mengecek beberapa hal penting terkait kesehatan, dikarenakan kondisi magang yang masih dalam keadaan pandemi Covid-19. Penulis diharuskan untuk memeriksa agar mengetahui apakah terjangkit Covid-19 dengan swab antigen, lalu penulis diharuskan untuk menyerahkan bukti foto *rontgen* atau *x-ray* bagian thorax. Selanjutnya penulis melakukan sesi perkenalan dengan tim infrastruktur yang akan mendampingi penulis dalam pelaksanaan kegiatan magang. Pertemuan ini dilakukan secara *on site* dengan pemberian arahan serta pengertian terkait peraturan perusahaan, kultur budaya, serta *role* dan *job description* yang akan diberikan kepada penulis. Selanjutnya penulis diharuskan untuk melakukan beberapa instalasi yang diperlukan untuk membantu kegiatan magang di PT. Bio Farma berlangsung.

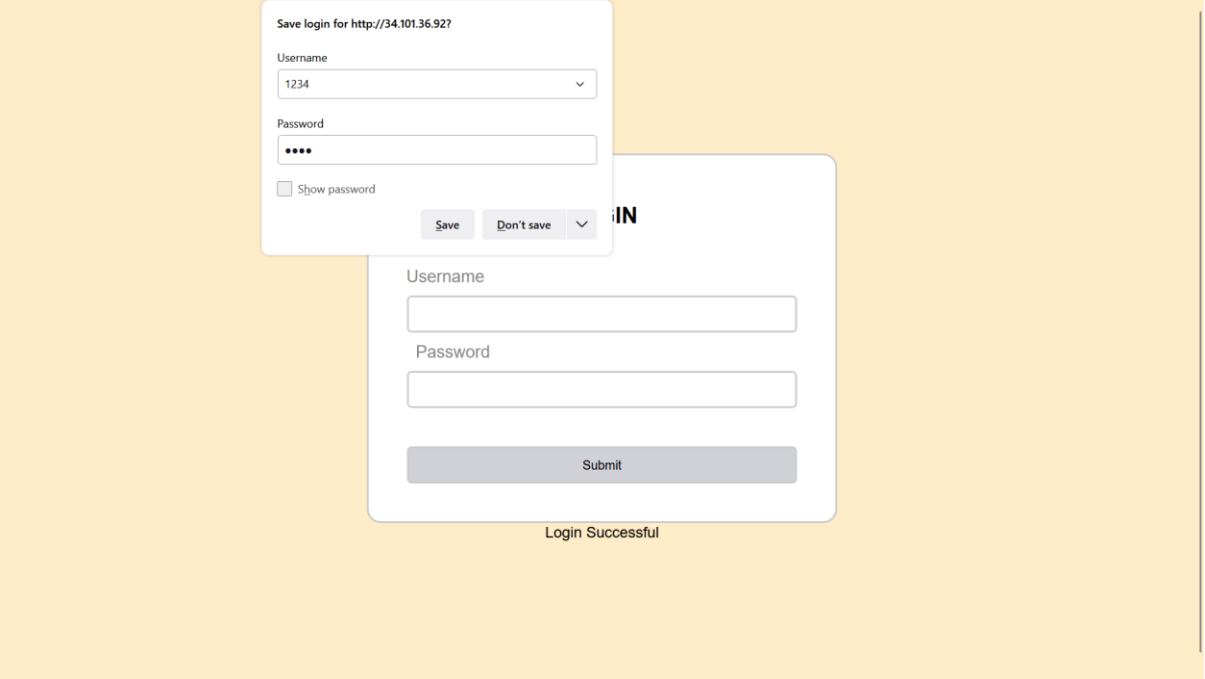
3.1.2 Pembelajaran Mandiri

Pada hari pertama kegiatan magang berlangsung, penulis diminta untuk membuat sebuah *virtual machine* dengan sistem operasi Ubuntu lengkap dengan penginstalan web server dan aplikasi pendukung lainnya. Hal ini memerlukan dasar pengoperasian sistem operasi Linux yang baik agar dapat memudahkan kegiatan magang ke depannya. Mentor memberikan tugas untuk lebih memperdalam cara kerja dan perintah, serta bagaimanan pengoperasiannya yang ada pada sistem operasi Linux dengan mengikuti *course* secara mandiri di situs TryHackMe. Dalam *course* tersebut dilengkapi dengan pendahuluan pada beberapa sistem operasi, dan juga menjelaskan secara detail hubungan dari setiap sistem operasi seperti antara hubungan sistem operasi Windows dan Linux, dasar-dasar jaringan, serta dasar dari keamanan jaringan.

3.1.3 Persiapan *Tools* Pendukung dan Membuat Aplikasi Login Sederhana

Sebelum melakukan aktivitas berupa mendeploy aplikasi, penulis diharuskan untuk membuat sebuah program atau aplikasi sederhana berupa aplikasi login yang berbasis web. Penulis menggunakan VSCode sebagai code editor untuk membuat aplikasi login tersebut, dengan PHP sebagai bahasa pemrogramannya. Setelah aplikasi tersebut berhasil dibuat dan telah dicoba terlebih dahulu pada *localhost*, maka aplikasi login tersebut siap digunakan dan selanjutnya akan di-*deploy* melalui VM yang ada di VirtualBox.

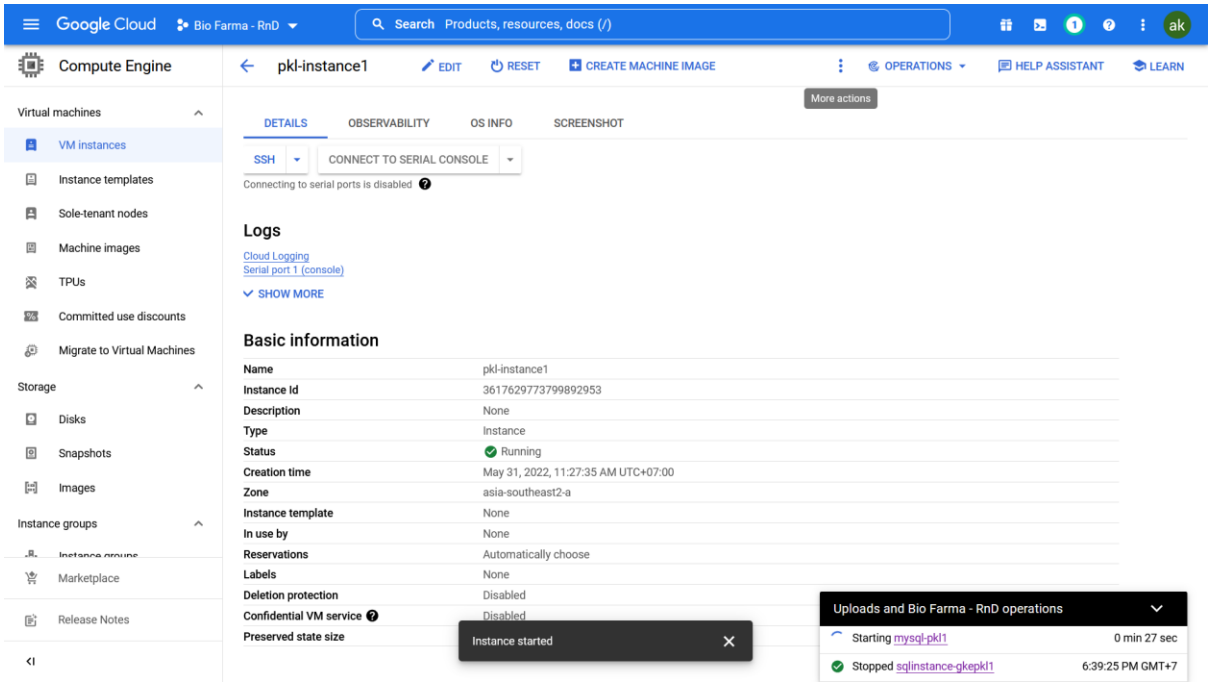
Pada VM yang ada di VBox, penulis menggunakan sistem operasi Ubuntu Linux dengan seri 20.04 *Focal Fosa* dan *Apache* sebagai web servernya, kemudian *MySQL* sebagai databasenya, lalu PHP sebagai *server side scripting,* atau yang biasa juga disebut dengan LAMP. Berbeda dengan yang ada di *localhost,* untuk men-*deploy* aplikasi login sederhana melalui VM yang ada di VBox cukup rumit dan memerlukan beberapa konfigurasi yang sesuai agar aplikasi tersebut dapat tampil dan berfungsi pada web server. Diantara kesulitan yang penulis hadapi adalah, bagaimana untuk mengkonfigurasi jaringan agar aplikasi yang ada pada VM atau yang disebut juga *client* dapat diakses pada web browser yang ada pada *host*.



Gambar 3.1 Contoh aplikasi login yang dibuat

3.1.4 Mempelajari Penggunaan Google Cloud Platform

Sebagai *Cloud Infrastructure Engineer* sangat diperlukan untuk mengetahui terlebih dahulu bagaimana cara kerja komputasi awan. Hal ini pernah penulis lakukan saat mendapatkan mata kuliah Sistem Jaringan dan Komputer, hanya saja perbedaannya terletak pada penggunaan *platform* komputasi awan. Pada mata kuliah SJK, *platform* yang digunakan merupakan Amazon Web Service, sedangkan saat kegiatan magang, *platform* yang digunakan adalah Google Cloud Platform. Penulis diharuskan untuk mengetahui caranya membuat, mengkonfigurasi, dan mengelola beberapa layanan yang ada seperti *virtual machine instance, SQL database instance* dan layanan lain yang disediakan oleh Google Cloud Platform. Selain itu, hal ini berguna untuk mempelajari layanan apa saja yang harus digunakan demi memenuhi kebutuhan dalam proyek di kegiatan magang penulis.



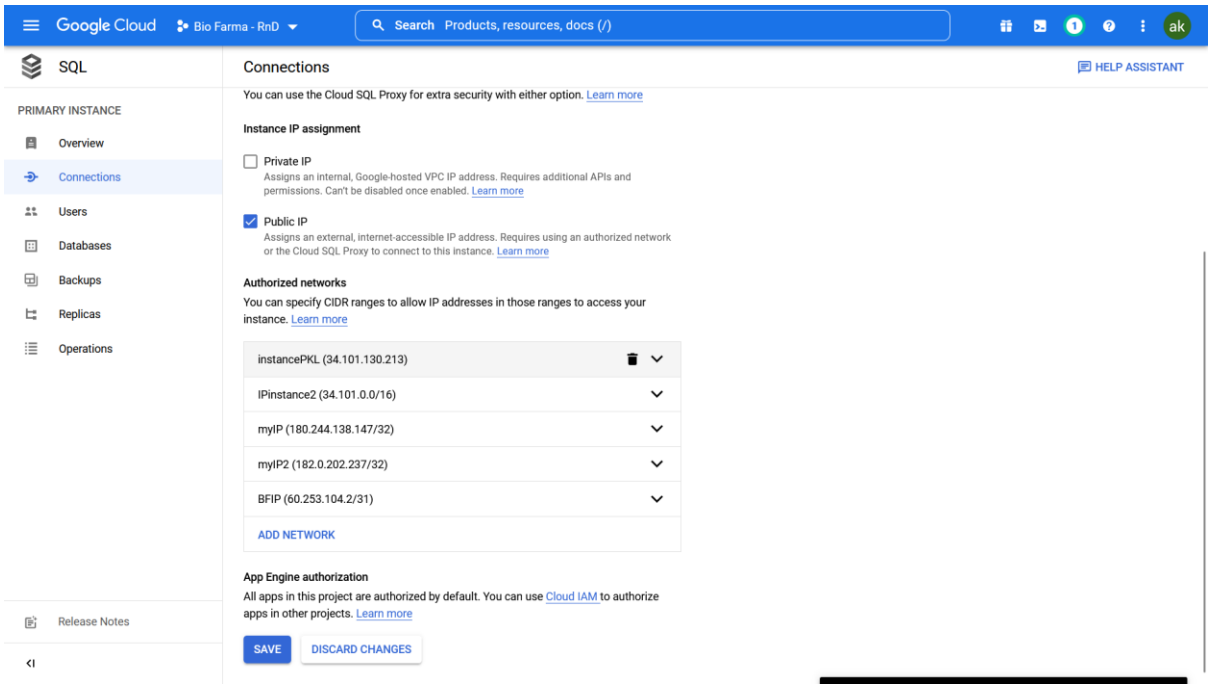
Gambar 3.2 Tampilan Google Cloud Platform

Pada gambar 3.2 merupakan salah satu layanan yang ada pada Google Cloud Platform. Layanan yang digunakan merupakan adalah instance untuk mesin virtual. Perbedaan yang ada dengan mesin virtual pada VBox adalah, mesin virtual yang digunakan pada Google Cloud Platform merupakan mesin virtual yang berbasis komputasi awan. Sedangkan pada VBox, mesin virtual hanya ada pada komputer lokal. Pada tugas kali ini, penulis diharuskan untuk men-*deploy* aplikasi yang telah penulis buat sebelumnya, perbedaannya hanyalah

terletak pada akses dari komputasi awan nya saja. Pertama penulis diharuskan untuk melakukan konfigurasi untuk mesin virtual dengan konfigurasi CPU 2 core dan Ram 4 GB.

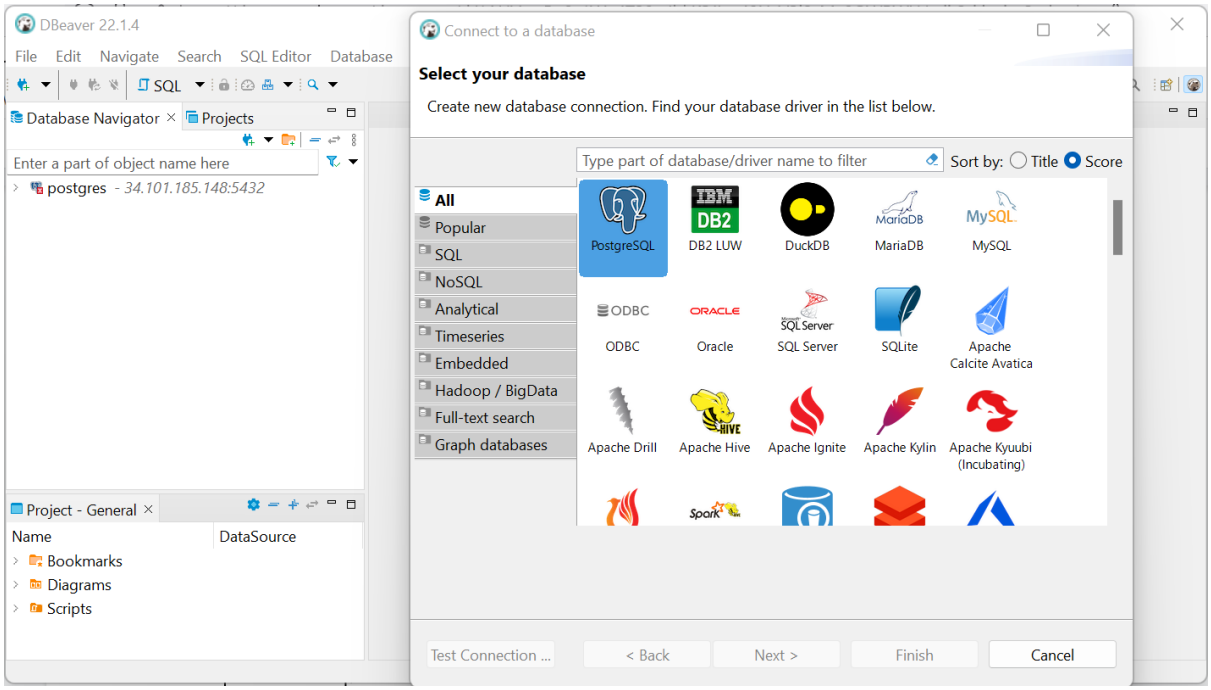
3.1.5 Melakukan Manajemen Database

SQL Instance dibuat agar aplikasi yang dimiliki bisa dijalankan. Perlu beberapa konfigurasi jaringan agar VM dapat mengakses data yang ada pada database SQL Instance tersebut, yaitu memasukkan IP dari VM ke konfigurasi jaringan yang ada di SQL Instance, seperti terlihat pada gambar 3.3. Hal ini agar aplikasi memiliki database dan juga agar aplikasi dapat diakses.



Gambar 3.3 Konfigurasi Jaringan Cloud SQL Instance

Dalam memanajemen database, kali ini penulis menggunakan aplikasi yang dinamakan DBeaver. Seperti halnya *phpMyadmin*, DBeaver digunakan agar dapat memudahkan dalam mengakses dan mengelola database SQL Instance pada Google Cloud Platform.



Gambar 3.4 Tampilan *tools* DBeaver

3.1.6 Menerapkan Penggunaan Docker dalam Deployment Aplikasi

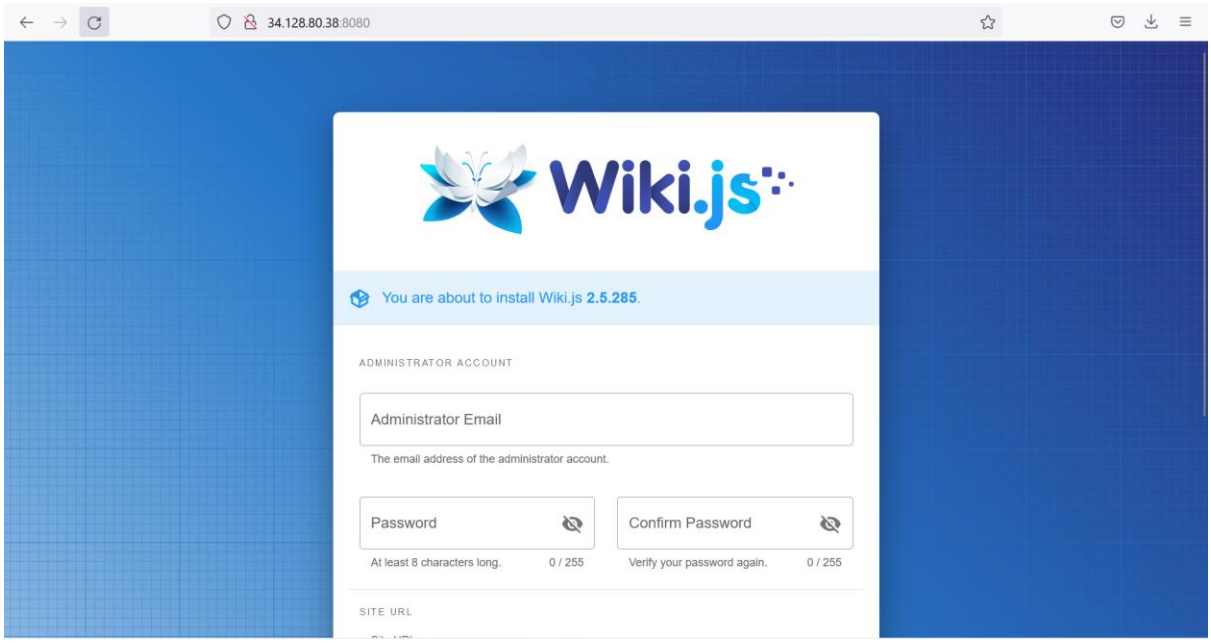
Setelah melakukan percobaan untuk mendeploy aplikasi dengan VM Instance, penulis diberikan tugas untuk men-deploy aplikasi dengan Docker. Hal yang dilakukan saat men-deploy aplikasi ini yaitu menggunakan VM yang sudah ada, lalu menginstall Docker pada VM tersebut. Adapun untuk aplikasi yang di-deploy kali ini merupakan aplikasi yang berbeda, aplikasi yang di-deploy merupakan aplikasi yang sudah jadi yang dinamakan Wikijs. Wikijs merupakan aplikasi open source yang memungkinkan banyak orang untuk menggunakannya. Wikijs ini termasuk kedalam CMS (Content Management System) seperti halnya Wordpress dan CMS lainnya.

Pada gambar 3.5 terlihat log dari aplikasi wikijs yang siap untuk *setup*. Untuk mengaksesnya dengan memasukkan alamat IP dengan port tujuan 8080:3000. Karena pada saat awal konfigurasi penulis menggunakan port 8080 dengan tujuan di port 3000, maka untuk mengaksesnya hanya dengan port 8080. Contoh: 127.0.0.1:8080.

```
dell18@pkl-instance1:/home/server12/docker/wikijs$ sudo docker-compose logs
-f

wikijs | 2022-08-25T15:02:18.639Z [MASTER] info: Starting setup wizard...
wikijs | 2022-08-25T15:02:19.061Z [MASTER] info: Starting HTTP server on
port 3000...
wikijs | 2022-08-25T15:02:19.062Z [MASTER] info: HTTP Server on port: [
3000 ]
wikijs | 2022-08-25T15:02:19.069Z [MASTER] info: HTTP Server: [ RUNNING ]
```

```
wikijs |          2022-08-25T15:02:19.069Z          [MASTER]          info:
 ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽
wikijs | 2022-08-25T15:02:19.069Z [MASTER] info:
wikijs |  2022-08-25T15:02:19.069Z  [MASTER]  info:  Browse  to  http://YOUR-
SERVER-IP:3000/ to complete setup!
wikijs | 2022-08-25T15:02:19.069Z [MASTER] info: wikijs  |          2022-08-
25T15:02:19.070Z [MASTER] info:  △ △ △ △ △ △ △ △ △ △ △ △ △ △ △
```

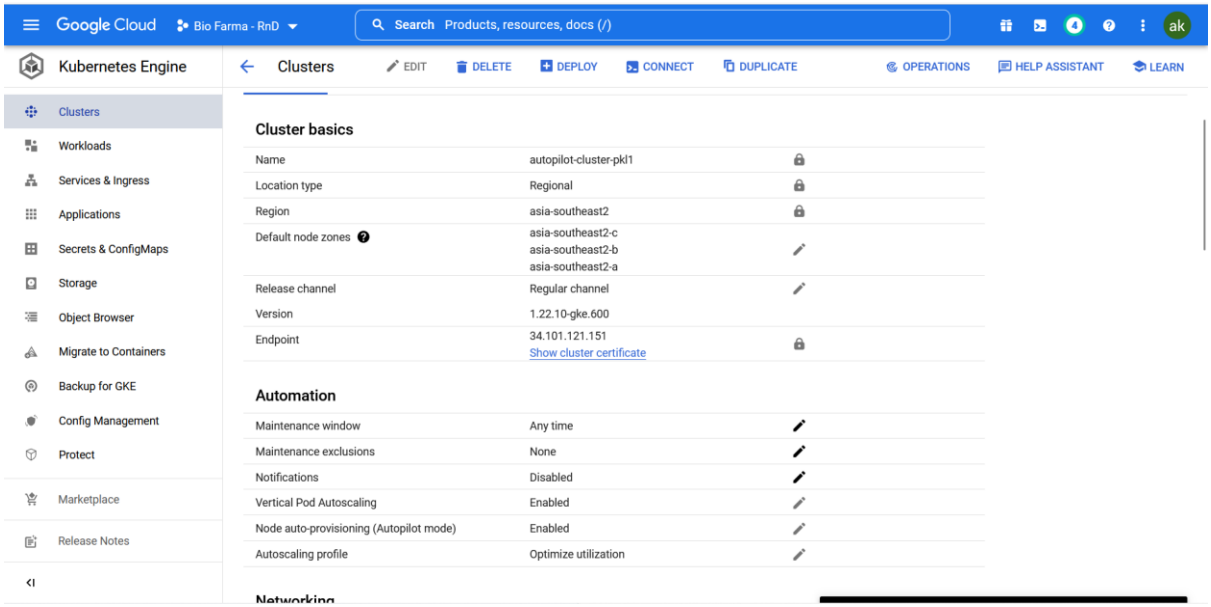
Gambar 3.5 Melihat log aplikasi



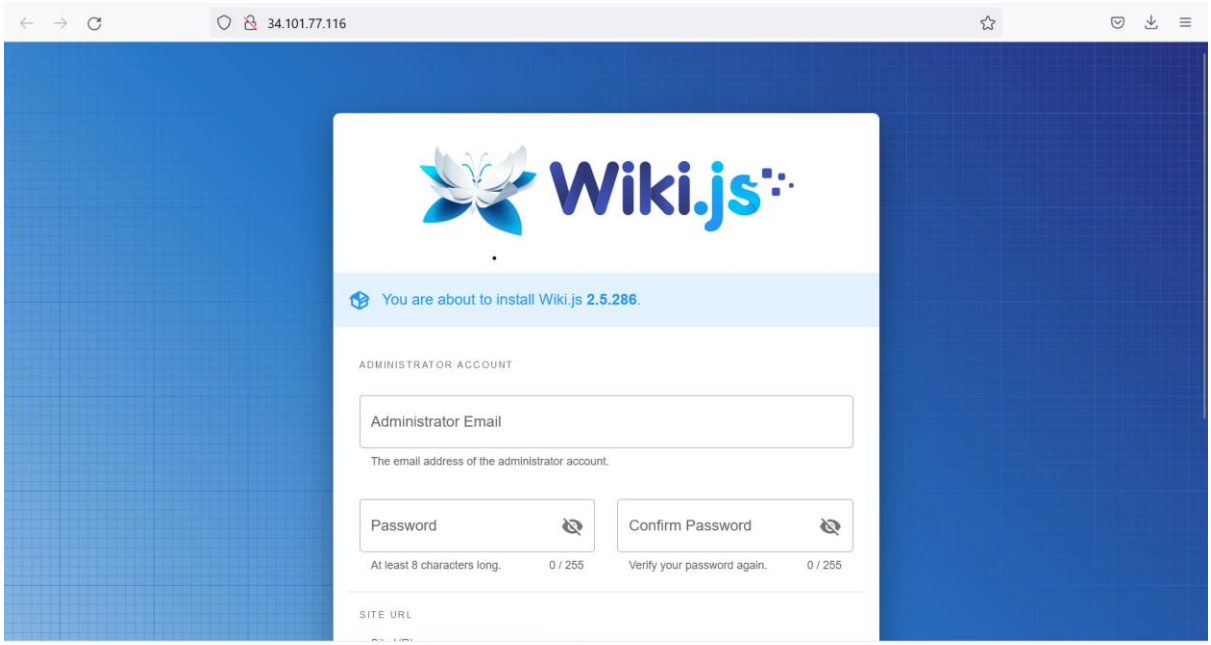
Gambar 3.6 Hasil deployment aplikasi dengan Docker

3.1.7 Menerapkan Penggunaan Kubernetes dalam Deployment Aplikasi

Pada kegiatan magang kali ini, penulis sebagai *Cloud Infrastructure Engineer* berkesempatan untuk mempelajari layanan dari Google Kubernetes Cluster. Penulis mempelajari hal-hal yang berkaitan dengan pembuatan file yaml, perkenalan dengan apa itu node, pod, dan istilah lain yang ada pada Kubernetes. Setelah mempelajari dasar-dasar dari Kubernetes, penulis ditugaskan untuk men-deploy aplikasi Wikijs melalui Kubernetes.



Gambar 3.7 Tampilan Kubernetes Cluster Google Cloud Platform



Gambar 3.8 Hasil *deployment* aplikasi menggunakan Kubernetes

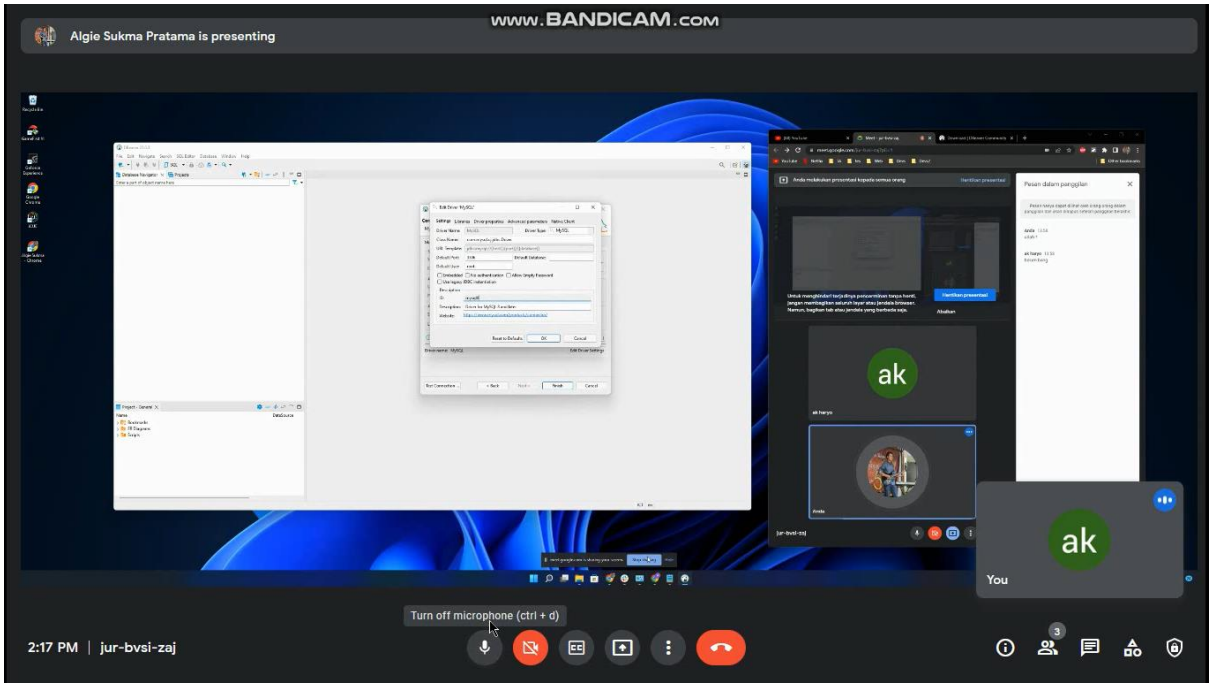
Terlihat bahwa penginstallan aplikasi wikijs dengan Kubernetes Cluster Google Cloud Platform dapat dilakukan dengan baik.

3.2 Manajemen Proyek

Dalam manajemen proyek implementasi *deployment* aplikasi menggunakan Kubernetes ini. Diawali dengan pengarahan yang diberikan oleh *supervisor* magang dengan memberikan tugas secara langsung untuk mengerjakan proyek yang berisi pendahuluan proyek, deskripsi tentang proyek, apa yang harus dilakukan, dan apa saja yang diperlukan untuk mengerjakan proyek tersebut diberi tahu secara langsung oleh *supervisor* dan penulis diharuskan untuk melapor perkembangan pada tiap 3 hari saat menjalankan proyek. Setiap dalam progres pengerjaan proyeknya terkadang *supervisor* menanyakan secara langsung terkait perkembangan proyek yang dikerjakan seperti, sejauh mana progres proyek yang telah dikerjakan, kesulitan apa yang dihadapi, dan menjelaskan secara detail tentang proses atau hal – hal yang harus dikerjakan dengan benar.

Gambar 3.9 Whatsapp sebagai salah satu tools untuk koordinasi

Selain penggunaan aplikasi Whatsapp dalam mendukung koordinasi, selama magang berlangsung apabila pemagang ataupun supervisor memiliki keadaan mendesak yang tidak memungkinkan untuk hadir dalam kegiatan magang di kantor maka pertemuan akan dialihkan menggunakan Google Meet sebagai aplikasi yang dapat membantu dalam koordinasi. Saat penggunaan Google Meet sangat memudahkan supervisor dan juga pemagang untuk dapat melihat dan bertukar *screen* agar dapat mengetahui dimana letak kesalahan pada saat pengerjaan proyek magang. Pada gambar 3.10 merupakan tampilan dari aplikasi Google Meet.



Gambar 3.10 Google Meet sebagai salah satu tools untuk koordinasi

Dalam kegiatannya sebagai *Infrastructure Cloud Engineer* di tim infrastruktur dan operasi IT Bio Farma, ada beberapa tugas atau *job description* yang dapat dilihat pada tabel 3.2.

Tabel 3.2 *Job description* sebagai *Infrastructure Cloud Engineer*

Aktivitas	Keterangan
Networking	Melakukan konfigurasi jaringan pada sumber daya yang akan digunakan untuk *delpoyment* aplikasi.
Database Admin	Menyiapkan dan mengonfigurasi database untuk aplikasi yang akan di-*deploy*.
Google Cloud Platform Admin	Menyiapkan dan mengonfigurasi sumber daya *cloud* yang akan digunakan untuk melakukan *deployment* aplikasi.

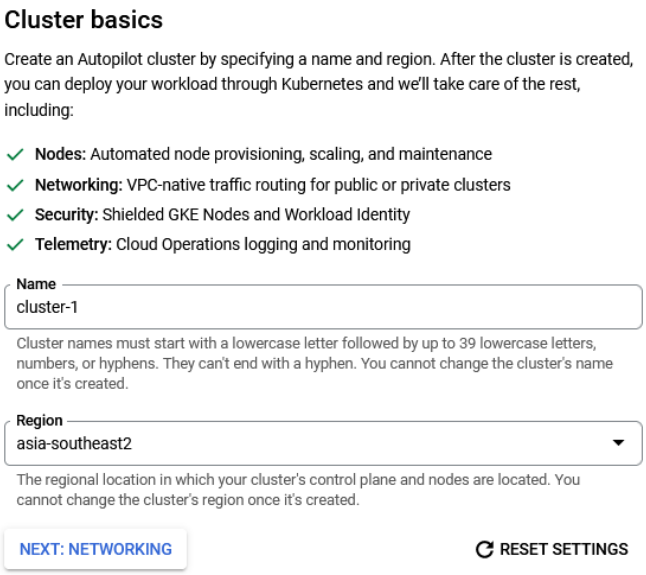
Selain *job description* yang ada pada tabel 3.2, diharuskan juga untuk mampu melakukan pengujian, diantaranya pengujian load testing, stress testing. Hal ini tentunya menuntut agar mempelajari penggunaan Kubernetes untuk *deployment* aplikasi hingga ke tahap pengujian.

3.3 Uji Coba Deployment dan Pengujian Aplikasi

Pada saat pelaksanaan kegiatan magang, penulis diberikan tugas untuk melakukan *deployment* aplikasi seperti pada layanan Kubernetes Cluster di Google Cloud Platform. Untuk deployment aplikasi pada Kubernetes cluster perlu menyiapkan beberapa hal seperti keperluan untuk menggunakan beberapa *tools* yaitu menyiapkan *instance* cloud SQL dan Kubernetes cluster. Untuk aplikasi yang akan di – *deploy* sendiri yaitu merupakan aplikasi *Content Management System* (CMS) seperti Wikijs dan Wordpress.

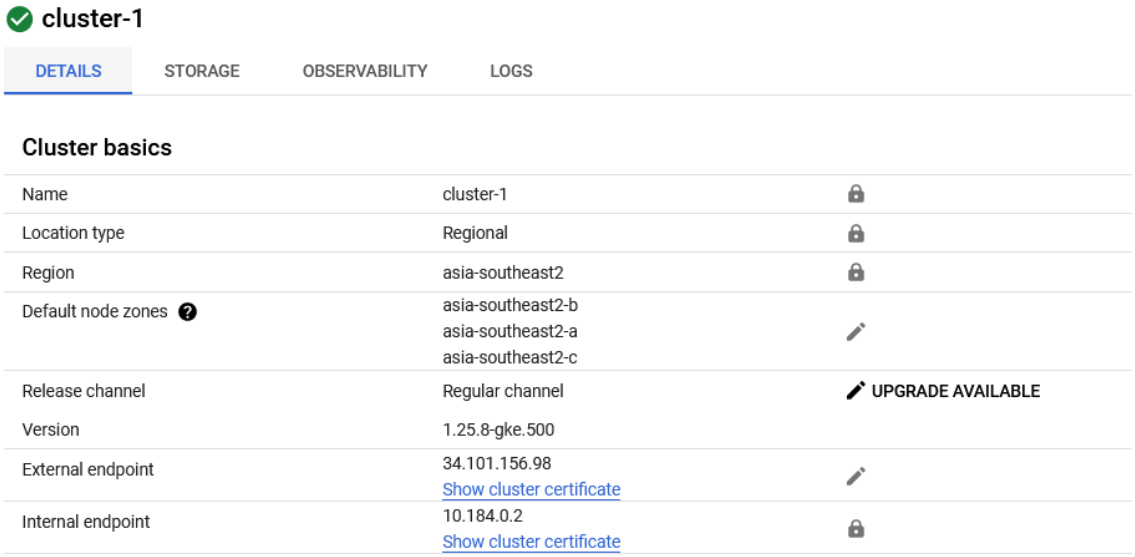
3.3.1 Tahap Persiapan dan Konfigurasi Sistem

Kubernetes merupakan *software* pendukung yang dibuat sebagai penunjang dalam memanajemen aplikasi yang berbasis *container*. Pada kegiatan magang kali ini, penulis sebagai *Cloud Infrastructure Engineer* berkesempatan untuk mempelajari layanan dari Google Kubernetes Cluster. Sebagai langkah awal untuk men-*deploy* aplikasi melalui kubernetes cluster GCP, konfigurasi sistem diperlukan untuk mempersiapkan lingkungan pengembangan dan penyebaran aplikasi. Pembuatan cluster kubernetes GCP dilakukan dengan menggunakan menu yang ada pada *interface* GCP, lalu dinamai dengan nama yang diinginkan dan memilih zona yang sesuai dengan lokasi terdekat dengan penggunanya, pada pembuatan cluster ini penulis memilih zona *asia-southeast2*. Seperti yang dapat dilihat pada gambar 3.11 merupakan langkah pembuatan cluster kubernetes pada GCP.



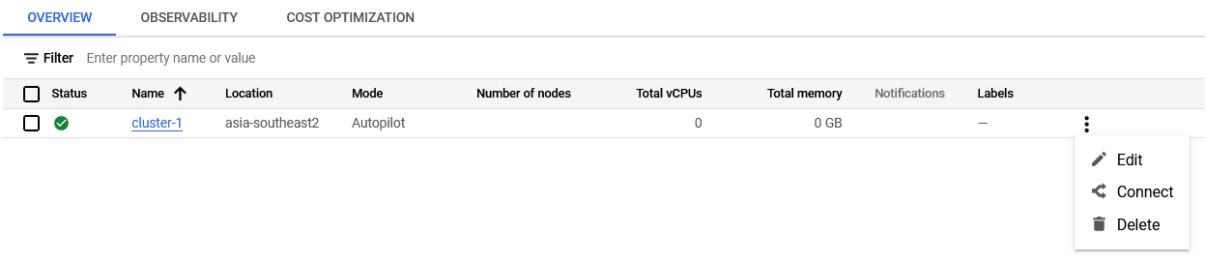
Gambar 3.11 Konfigurasi Pembuatan Google Kubernetes Engine

Selanjutnya dapat dilihat pada gambar 3.12 merupakan detail dari Kubernetes cluster yang telah dibuat.



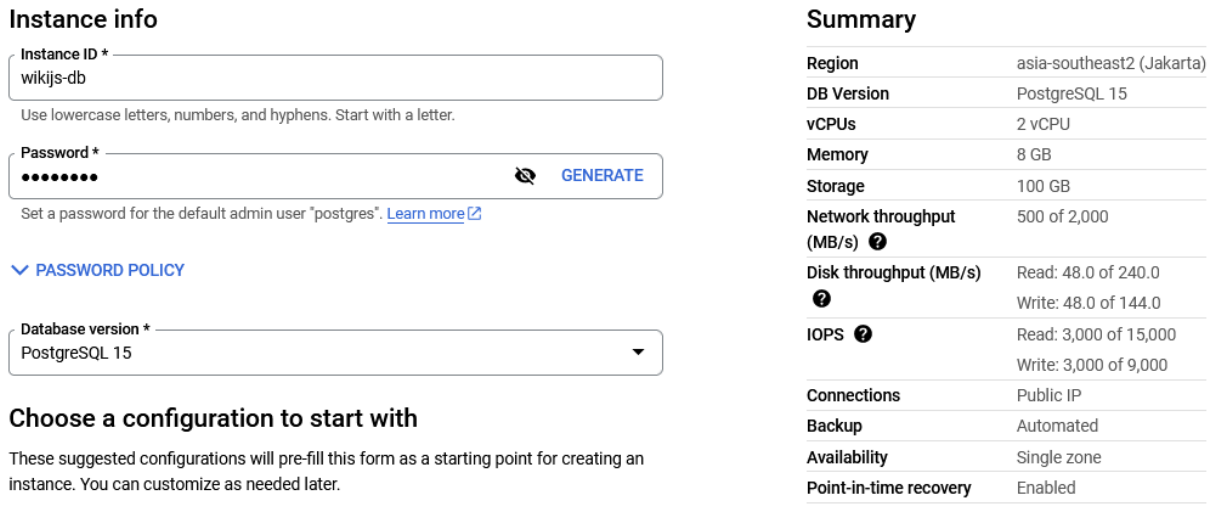
Gambar 3.12 Detail Cluster Kubernetes

Pada gambar 3.13 merupakan hasil dari Kubernetes Cluster yang telah dibuat.



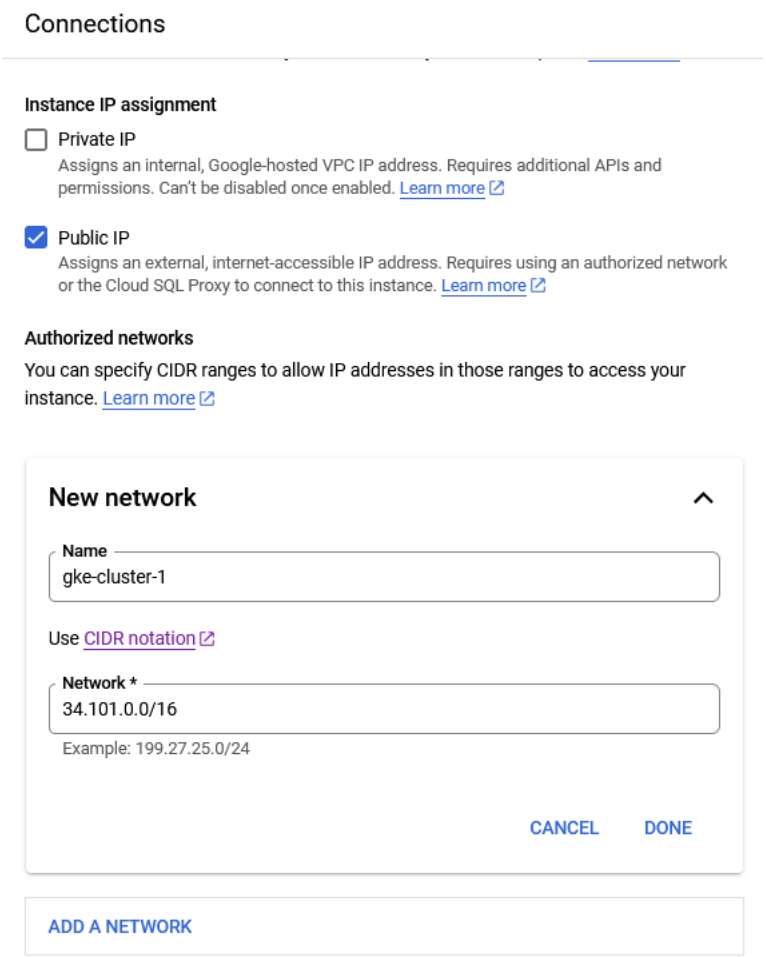
Gambar 3.13 Detail cluster Kubernetes

Hal berikutnya setelah membuat cluster pada Google Kubernetes Engine yaitu membuat dua *cloud SQL instance* untuk aplikasi Wikijs dan Wordpress yang akan di-*deploy*. Yang pertama merupakan *cloud SQL instance* untuk aplikasi Wikijs yang dinamai dengan *wikijs-db* dan menggunakan database berjenis *Postgre* dengan versi 15. *Instance* yang dimiliki memiliki spesifikasi singkat berupa CPU 2 core, 8 Gb memori, penyimpanan 100 Gb, dan berlokasi di *asia-southeast2* (Jakarta). Untuk spesifikasi lengkapnya dapat dilihat pada gambar 3.14.



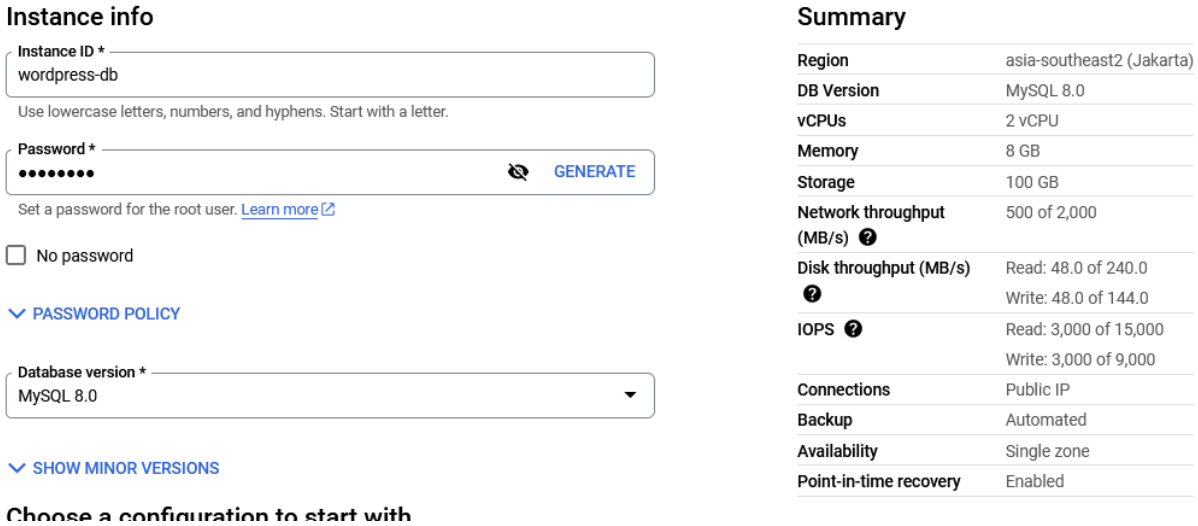
Gambar 3.14 Konfigurasi database aplikasi Wikijs

Setelah pembuatan *cloud SQL instance,* dilakukan sebuah konfigurasi jaringan pada *SQL instance* tersebut. Hal ini bertujuan agar Kubernetes cluster yang dimilki dapat memiliki koneksi dan mengakses *SQL instance* untuk penyimpanan data dari aplikasi Wikijs yang akan di-*deploy*. Konfigurasi dilakukan dengan cara memasukkan sub alamat IP dari Kubernetes cluster yang dimiliki, yaitu 34.101.0.0/16. Selengkapnya dapat dilihat pada gambar 3.15.



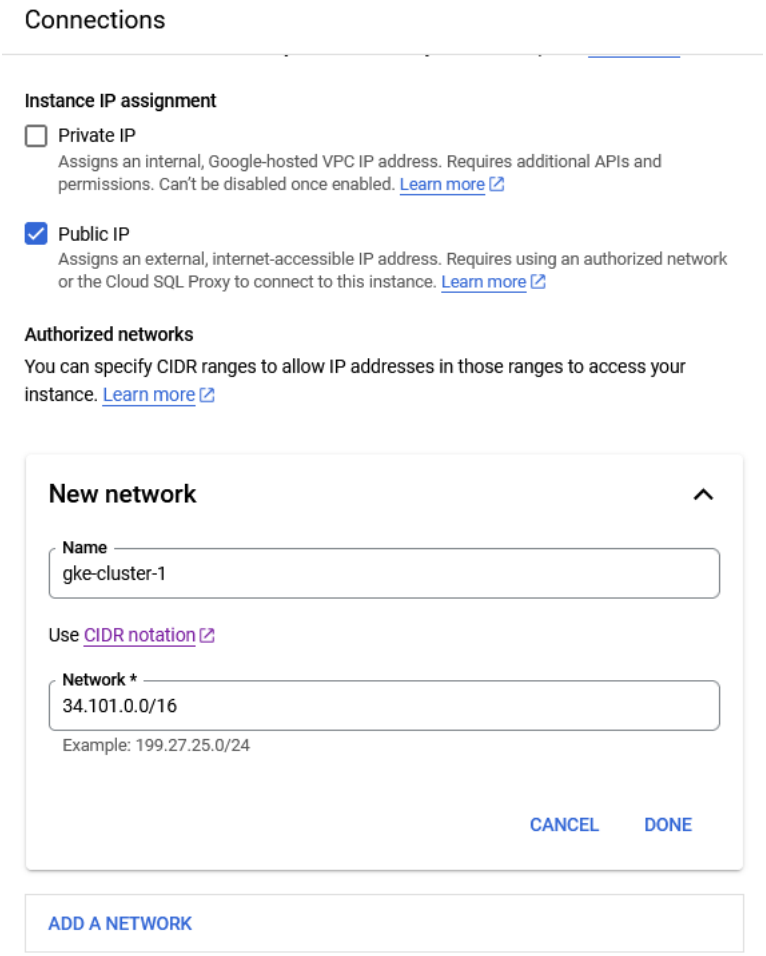
Gambar 3.15 Konfigurasi jaringan database wiki.js

Kemudian dibuat sebuah *cloud SQL instance* yang kedua untuk aplikasi Wordpress yang akan di-*deploy* dinamai dengan *wordpress-db* dan menggunakan database berjenis *MySQL* dengan versi 8.0. *Instance* yang memiliki spesifikasi singkat berupa CPU 2 core, 8 Gb memori, penyimpanan 100 Gb, dan berlokasi di *asia-southeast2* (Jakarta). Untuk spesifikasi lengkapnya dapat terlihat pada gambar 3.16.



Gambar 3.16 Konfigurasi database aplikasi Wordpress

Sama seperti pada langkah sebelumnya saat membuat *cloud SQL instance* untuk aplikasi Wikijs. Koneksi jaringan pada *cloud SQL instance* untuk aplikasi Wordpress dilakukan dengan cara memasukkan sub alamat IP dari Kubernetes cluster yang sama, yaitu 34.101.0.0/16. Selengkapnya dapat dilihat pada gambar 3.17.



Gambar 3.17 Konfigurasi jaringan database Wordpress

3.3.2 Mendeploy Aplikasi Wiki.js dengan Google Kubernetes

Men-*deploy* aplikasi Wikijs merupakan hal pertama yang dilakukan dalam kegiatan magang. Setelah mempelajari dasar-dasar dari penggunaan Kubernetes, penulis ditugaskan untuk men-*deploy* aplikasi Wikijs melalui Kubernetes cluster GCP. Hal pertama yang disiapkan yaitu, penulis menyiapkan beberapa file dalam bentuk *yaml* yang nantinya akan di *apply* menggunakan perintah *kubectl*. Selanjutnya, dibuat sebuah *namespace* yang dinamai dengan *wikijs* melalui terminal *cloud shell* dengan perintah seperti pada gambar 3.18. Pembuatan dari *namespace* ini sendiri nantinya bertujuan untuk memisahkan sumberdaya cluster dengan penyebutan nama dari aplikasi yang akan dibuat.

```
$ kubectl create namespace wikijs

$ kubectl get namespace
NAME                STATUS   AGE
default             Active   42d
kube-node-lease     Active   42d
kube-public         Active   42d
kube-system         Active   42d
wikijs              Active   10d
wordpress           Active   5d
```

Gambar 3.18 Membuat dan mengecek *namespace* Wikijs

Hasil dari perintah pada gambar 3.18, memperlihatkan daftar dari *namespace* apa saja yang telah dibuat, dan dapat dilihat bahwa *namespace* Wikijs telah terbuat dengan statusnya yang aktif terhitung sejak 10 hari dari waktu pembuatannya.

Kemudian hal selanjutnya yaitu membuat sebuah file konfigurasi *configMap* dengan ekstensi file *yaml* yang diberi nama *wikijs-config.yaml.* File ini nantinya akan mengonfigurasikan koneksi antara aplikasi yang akan di-*deploy* dengan database cloud SQL yang telah disiapkan pada layanan Google Cloud Platform.

```
$ nano wikijs-config.yaml
apiVersion: v1
data:
  DB_HOST: "34.101.206.105"
  DB_PORT: "5432"
  DB_USER: "postgres"
  DB_TYPE: "postgres"
  DB_NAME: "postgres"
kind: ConfigMap
metadata:
  name: postgres-config
  namespace: wikijs

$ kubectl apply -f wikijs-config.yaml

$ kubectl get config -n wikijs
NAME              DATA   AGE
```

```
postgres-config   5      19s
```

Gambar 3.19 Membuat konfigurasi *configmaps* Wikijs

Configmaps ini dimaksudkan agar aplikasi yang kita buat dapat memiliki sebuah database, adapun database yang dibuat pada aplikasi ini merupakan database dari *SQL instance* dengan tipe *postgres*. Pada gambar 3.19 Terlihat beberapa variabel seperti variabel jenis atay *kind*, yang mendefinisikan untuk apa konfigurasi tersebut dibuat. Pada variabel *DB_HOST* diisikan dengan alamat IP dari *cloud SQL instance* yang telah dibuat, yaitu '34.101.206.105'. Selanjutnya pada variabel *DB_PORT* diisi dengan port bawaan dari *PostgreSQL* yaitu '5432'. Pada variabel *DB_USER* diisi dengan nama dari jenis SQL yang dipakai yaitu 'postgres', begitu pula pada variabel *DB_TYPE* diisi dengan nilai yang sama. Terakhir mengisi nilai pada variabel *DB_NAME* dengan nama database yang ada pada *cloud SQL instance* yang telah dibuat, yaitu dengan nama 'postgres'. Setelah pengaturan untuk konfigurasi selesai, dilakukan *apply* konfigurasi untuk *configMaps*, lalu masukkan perintah untuk melihat *configMaps* yang telah dibuat.

Selanjutnya yaitu membuat file konfigurasi *secret* yang diberi nama dengan *wikijs-secret.yaml. Secret* digunakan untuk menyimpan dan mengatur informasi kredensial dari data pada *pod* atau *image* yang tidak untuk diekspos, seperti halnya kata sandi atau bahkan alamat IP dari database SQL yang dimiliki. Pada gambar 3.20 merupakan perintah untuk meng-*apply* serta melihat konfigurasi file *secret* yang telah dibuat dan berjalan pada sistem.

```
$ nano wikijs-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: postgres
  namespace: wikijs
type: Opaque
stringData:
  password: keamanan

$ kubectl apply -f wikijs-secret.yaml

$ kubectl get secret -n wikijs
NAME       TYPE     DATA   AGE
postgres   Opaque   1      7d2h
```

Gambar 3.20 Membuat konfigurasi *secret* Wikijs

Setelah itu aktivitas selanjutnya merupakan pembuatan file konfigurasi *service.* Penggunaan dari *service* merupakan objek pada Kubernetes yang digunakan untuk mengarahkan *request* atau *traffic* ke beberapa pod menggunakan alamat IP, dengan kata lain dengan *service* ini aplikasi dapat terekspos dan dapat diakses. *Service* yang digunakan merupakan tipe *LoadBalancer* dengan protokol TCP, dan pada port 80, serta target port 3000. Kemudian konfigurasi *service* diterapkan atau di-*apply* dengan menggunakan perintah pada gambar 3.21, dan dapat dilihat juga hasil dari *service* yang telah dibuat.

```
$ nano wikijs-service.yaml
apiVersion: v1
kind: Service
metadata:
 name: wikijs
 namespace: wikijs
spec:
 type: LoadBalancer
 selector:
   app: wikijs
 ports:
 - protocol: TCP
   port: 80
   targetPort: 3000

$ kubectl apply -f wikijs-secret.yaml

$ kubectl get svc -n wikijs
NAME     TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)         AGE
wikijs   LoadBalancer   10.103.0.86   34.128.80.181   80:30567/TCP    6d23h
```

Gambar 3.21 Membuat konfigurasi *service* Wikijs

Pada tahapan selanjutnya yaitu membuat file konfigurasi dengan jenis *deployment.* Dalam file *deployment* ini akan berisikan dengan jelas aplikasi apa yang akan di-*deploy*, serta deskripsi dari beberapa *state*. Pada gambar 3.22 dituliskan berdasarkan templat yang ada pada dokumentasi kubernetes.

```
$ nano wikijs-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: wikijs
  name: wikijs
  namespace: wikijs
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wikijs
  template:
    metadata:
      labels:
```

```
        app: wikijs
    spec:
      containers:
        - name: wikijs
          image: lscr.io/linuxserver/wikijs:latest
          env:
          - name: DB_HOST
            valueFrom:
              configMapKeyRef:
                key: DB_HOST
                name: postgres-config
          - name: DB_PORT
            valueFrom:
              configMapKeyRef:
                key: DB_PORT
                name: postgres-config
          - name: DB_USER
            valueFrom:
              configMapKeyRef:
                key: DB_USER
                name: postgres-config
          - name: DB_TYPE
            valueFrom:
              configMapKeyRef:
                key: DB_TYPE
                name: postgres-config
          - name: DB_NAME
            valueFrom:
              configMapKeyRef:
                key: DB_NAME
                name: postgres-config
          - name: DB_PASS
            valueFrom:
              secretKeyRef:
                name: postgres
                key: password
          ports:
          - containerPort: 3000
            protocol: TCP

$ kubectl apply -f wikijs-deployment.yaml

$ kubectl get deploy -n wikijs
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
wikijs   1/1     1            1           6d23h

$ kubectl get pods -n wikijs
NAME                     READY   STATUS    RESTARTS   AGE
wikijs-9b967f595-ghr8p   1/1     Running   0          4d22h
```

Gambar 3.22 Membuat konfigurasi *deployment* Wikijs

Pada kolom *container* merupakan kolom nama yang diisi nilai ‘wikijs’, lalu pada kolom *image* diisikan value dari *image* aplikasi Wikijs yaitu ‘lscr.io/linuxserver/wikijs:latest’. Selanjutnya pada kolom *environment* terdapat beberapa kolom dibawahnya yang berisi beberapa variabel. Pada saat pembuatan file konfigurasi *configMap*, telah dimasukkan beberapa data seperti DB_HOST, DB_PORT, DB_USER, DB_TYPE, dan DB_NAME.

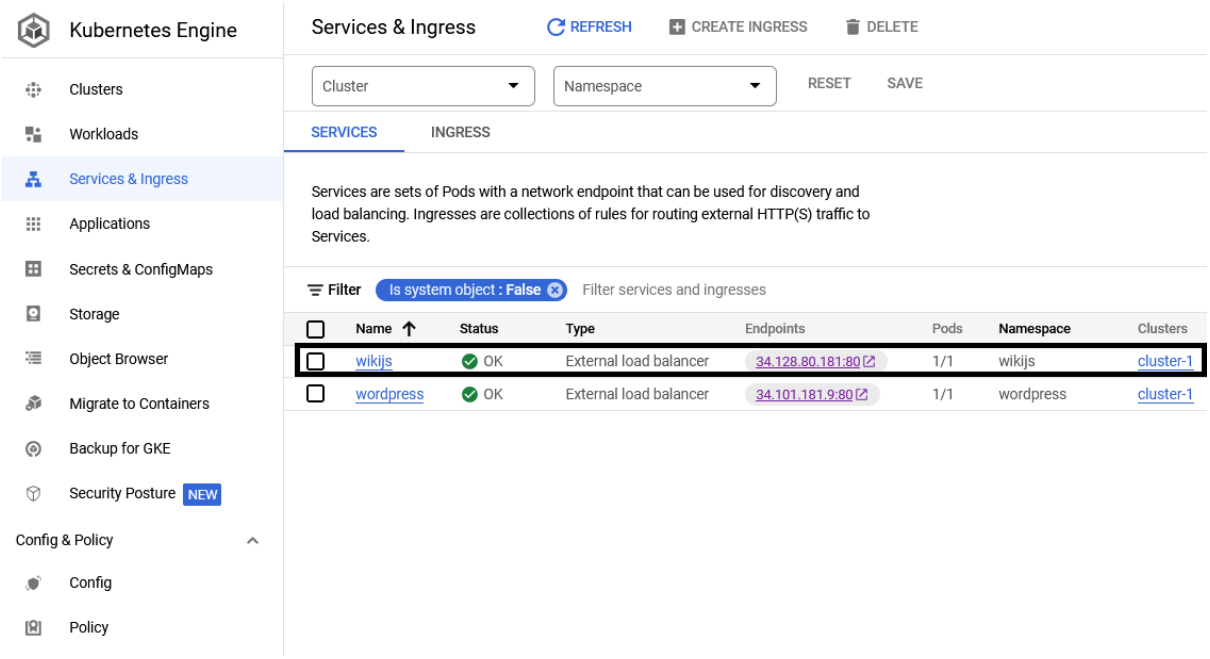
Konfigurasi tersebut ditulis dengan maksud agar *deployment* dapat mengambil nilai dari (`valueFrom`) konfigurasi *configMap* yang telah dibuat sebelumnya. Dengan mengambil referensi dari *ConfigMap* salah satu contohnya yaitu, kata kunci (`key`) DB_HOST akan mengambil data pada variabel DB_HOST di *ConfigMap*, yang dinamai dengan nama 'postgres-config'.

Selanjutnya akan dilihat log dari aplikasi pada *pod* dan dengan *namespace* spesifik yang telah dibuat menggunakan perintah seperti pada gambar 3.23 Apabila aplikasi tersebut berjalan maka akan didapatkan log seperti pada gambar 3.23.

```
$ kubectl logs -f wikijs-9b967f595-4vds8 -n wikijs
[custom-init] No custom files found, skipping...
Loading configuration from /app/wiki/config.yml... OK
2023-06-29T14:02:48.706Z[MASTER]info:
=====================================
2023-06-29T14:02:48.709Z[MASTER]info: = Wiki.js 2.5.299
=====================
2023-06-29T14:02:48.710Z[MASTER]info:
=====================================
2023-06-29T14:02:48.710Z[MASTER]info: Initializing...
2023-06-29T14:02:51.545Z[MASTER]info: Using database driver pg for postgres
[ OK ]
2023-06-29T14:02:51.596Z[MASTER]info: Connecting to database...
2023-06-29T14:02:51.729Z[MASTER]info: Database Connection Successful [ OK ]
2023-06-29T14:02:51.825Z[MASTER]warn:   DB   Configuration   is   empty   or
incomplete. Switching to Setup mode...
2023-06-29T14:02:51.826Z[MASTER]info: Starting setup wizard...
2023-06-29T14:02:52.435Z[MASTER]info: Starting HTTP server on port 3000...
2023-06-29T14:02:52.435Z[MASTER]info: HTTP Server on port: [ 3000 ]
2023-06-29T14:02:52.440Z[MASTER]info: HTTP Server: [ RUNNING ]
2023-06-29T14:02:52.440Z[MASTER]                                        info:
 ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽
2023-06-29T14:02:52.440Z[MASTER]info:
2023-06-29T14:02:52.441Z[MASTER]info: Browse to http://YOUR-SERVER-IP:3000/
to complete setup!
2023-06-29T14:02:52.441Z [MASTER] info:
2023-06-29T14:02:52.441Z[MASTER]info:
 △ △ △ △ △ △ △ △ △ △ △ △ △ △ △ △ △ △ △ △ △ △ △
[ls.io-init] done.
```

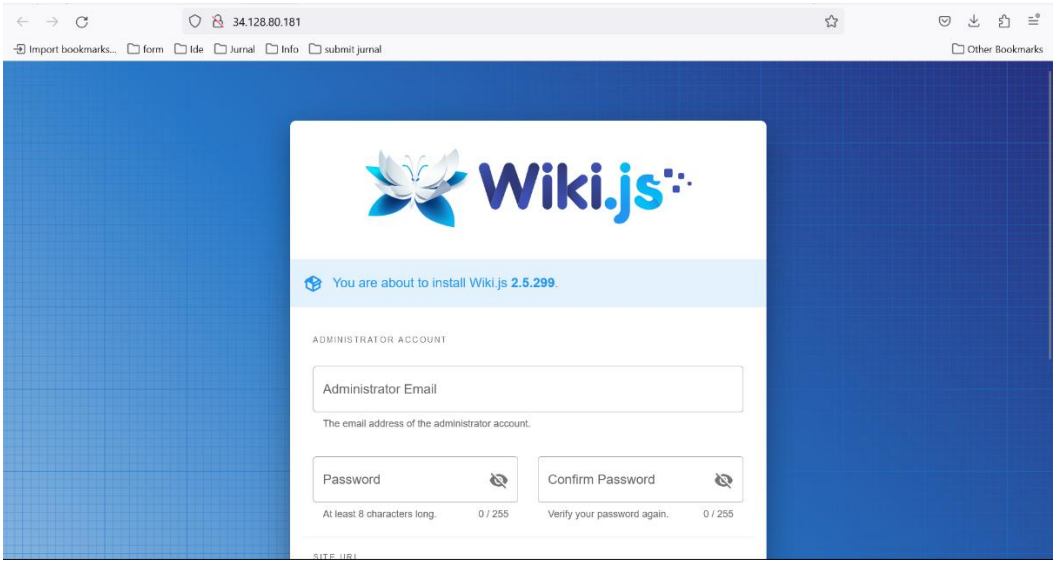
Gambar 3.23 Melihat log aplikasi Wikijs yang ter-*deploy*

Setelah semua aktivitas konfigurasi sistem telah dilakukan, dapat dilihat pada menu *service & ingress* pada *console* GCP Kubernetes Engine bahwa aplikasi Wikijs yang telah di-*deploy* sudah terekspos dengan tipe *LoadBalancer* dan juga alamat IP sudah terbuat yang dapat dilihat pada tabel *endpoints* seperti pada gambar 3.24.



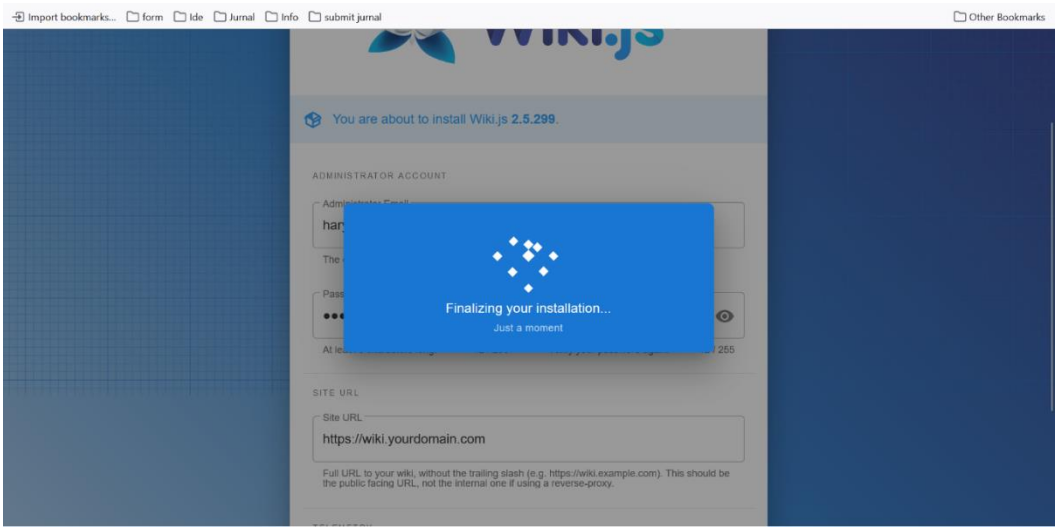
Gambar 3.24 Melihat *service* Wikijs yang telah berjalan pada Kubernetes cluster

Dengan membuka atau mengklik alamat IP pada tabel *endpoints,* dapat dibuka pada browser yang ada pada gambar 3.24, akan menampilkan halaman awal yang berisikan form untuk instalasi Wikijs. Hal tersebut bertujuan untuk mendaftar sebagai admin web yang mengelola konten yang ada pada website berbasis Wikijs.



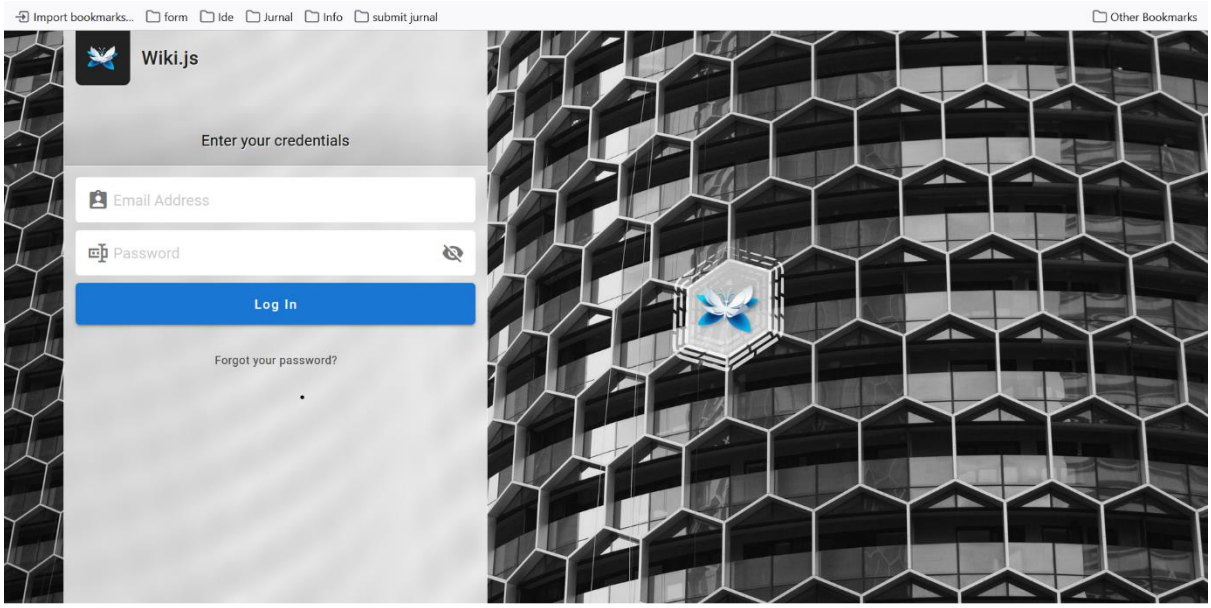
Gambar 3.25 Halaman awal untuk *setting up* aplikasi Wikijs

Setelah melakukan pendaftaran sebagai admin web, akan terlihat instalasi seperti pada gambar 3.26. Hal yang perlu diperhatikan dalam penginstalan aplikasi Wikijs ini adalah dengan menghubungkan Kubernetes Cluster dengan SQL Instance, agar aplikasi yang dibuat dapat memiliki database. Hal tersebut sudah dilakukan pada tahap penyiapan dan konfigurasi sistem sebelumnya.



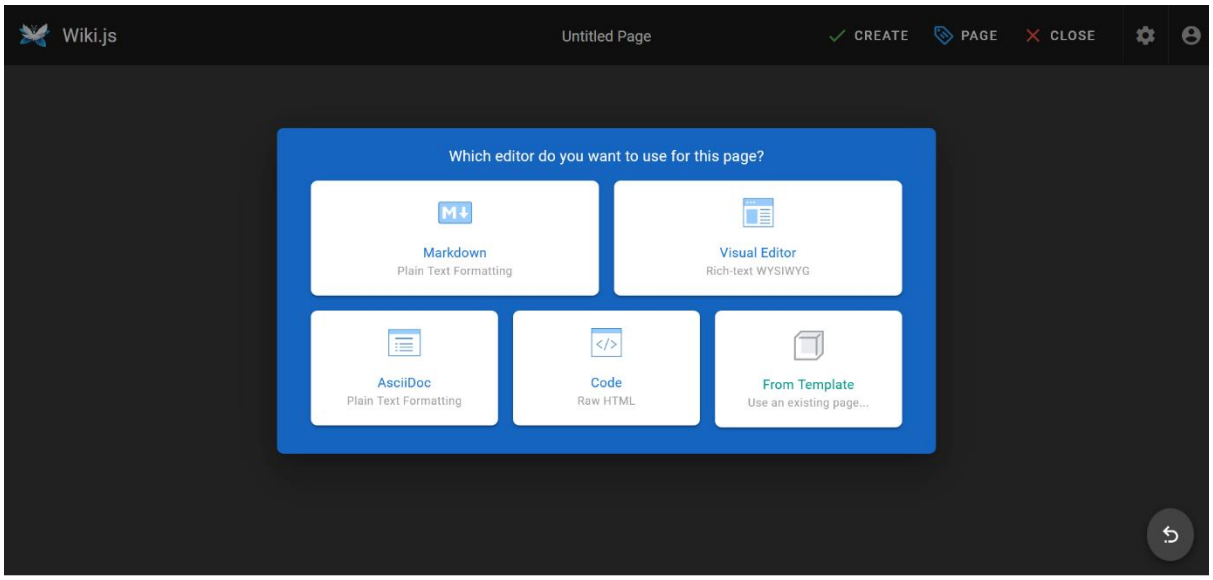
Gambar 3.26 Proses instalasi aplikasi Wikijs

Setelah proses instalasi Wikijs selesai, akan diarahkan ke halaman login aplikasi Wikijs untuk masuk ke akun yang telah dibuat sebelumnya seperti pada gambar 3.27.



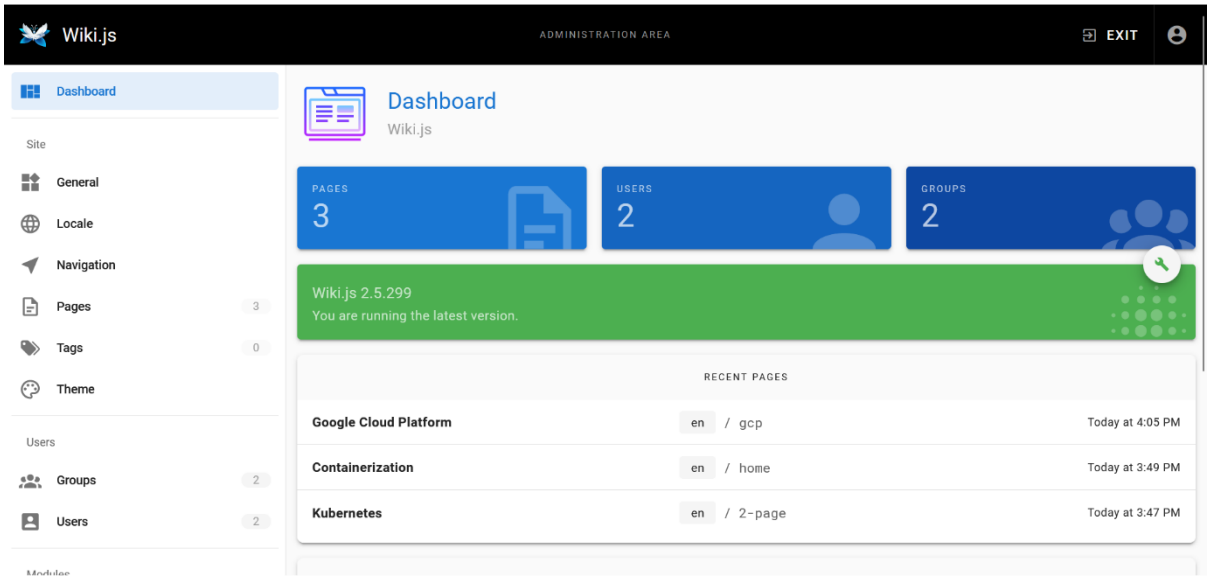
Gambar 3.27 Halaman masuk ke admin website Wikijs

Pada gambar 3.28, terdapat beberapa pilihan menu untuk mengostumisasi web Wikijs, diantaranya seperti *markdown, visual editor, ascii doc, code,* dan *form template*. Pada percobaan implementasi aplikasi ini, penulis memilih kustomisasi konten pada web dengan menggunakan *visual editor* dikarenakan dapat mengedit secara mudah dan fleksibel.



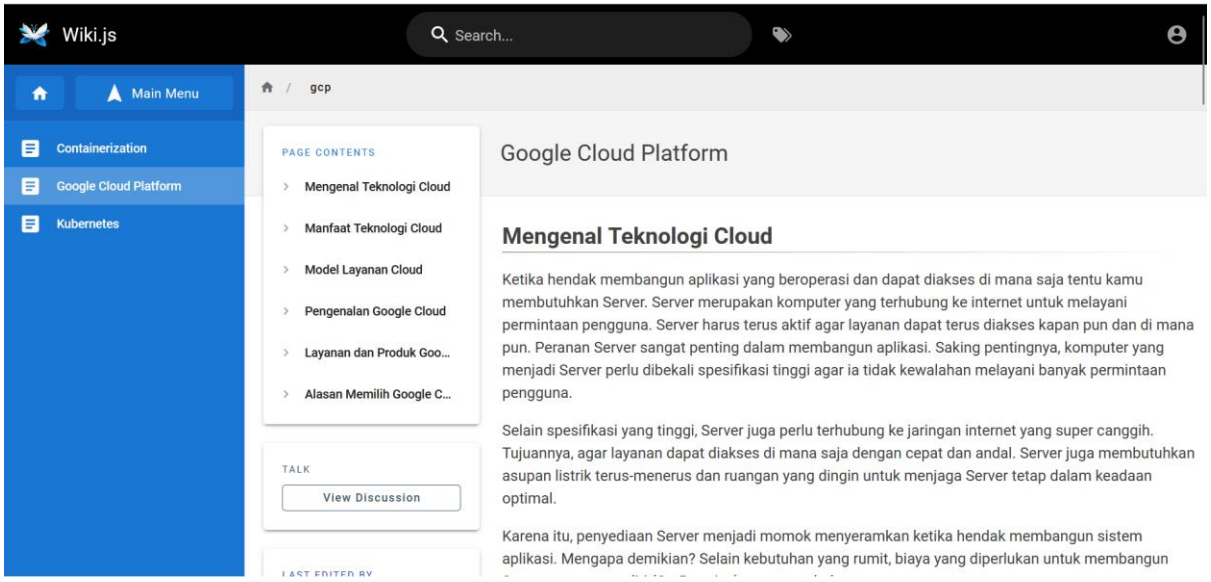
Gambar 3.28 Halaman kustomisasi website berbasis Wikijs

Pada gambar 3.29, merupakan tampilan pada dashboard untuk manajemen konten dan halaman yang ada pada web yang berbasis Wikijs.



Gambar 3.29 Halaman dashboard website Wikijs

Lalu pada gambar 3.30, merupakan hasil dari web yang telah dikustomisasi sesuai dengan keinginan penulis. Dengan website berbasis Wikijs ini dapat membuat penggunanya untuk berkolaborasi dalam menyunting dokumen, atau seperti membuat suatu blog berisi rangkuman informasi terhadap suatu topik tertentu.



Gambar 3.30 Hasil kustomisasi konten pada website

3.3.3 Mendeploy Aplikasi Wordpress dengan Google Kubernetes

Setelah mencoba dan berhasil untuk men-*deploy* aplikasi Wikijs hingga dapat mengaksesnya, penulis melakukan percobaan lain dengan men-*deploy* aplikasi Wordpress. Hal ini penulis lakukan untuk mencari tahu apakah aplikasi Wordpress dapat berjalan dengan file konfigurasi yang berbeda dari dokumentasi atau langkah – langkah yang dimiliki oleh percobaan lain, dengan menggunakan file konfigurasi yang telah dimodifikasi oleh penulis. Hal pertama yang dilakukan yaitu, membuat *namespace* yang dinamai dengan *wordpress*.

```
$ kubectl create namespace wordpress

$ kubectl get namespace
NAME                STATUS   AGE
default             Active   42d
kube-node-lease     Active   42d
kube-public         Active   42d
kube-system         Active   42d
wikijs              Active   10d
wordpress           Active   5d
```

Gambar 3.31 Membuat dan mengecek *namespace* Wordpress

Hasil dari perintah pada gambar 3.31, memperlihatkan daftar dari *namespace* apa saja yang telah dibuat, dan dapat dilihat bahwa *namespace* Wordpress telah terbuat dengan statusnya yang aktif terhitung sejak 5 hari dari waktu pembuatannya.

Hal selanjutnya yaitu membuat konfigurasi file untuk *persistent volume claim* yang berfungsi untuk *claiming* sumber daya penyimpanan aplikasi Wordpress yang akan dibuat. Hal ini dilakukan karena Wordpress membutuhkan *persistent volume* untuk menyimpan data di luar kontainer dikarenakan untuk menghindari tidak tersedianya node yang terhapus atau gagal (Google Cloud, 2023), apabila hal tersebut terjadi maka akan menghapus semua data hingga file sistem Wordpress yang telah dibuat. Konfigurasi dan hasil perintah terlihat pada gambar 3.32.

```
$ nano wp-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  namespace: wordpress
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
```

```
$ kubectl apply -f wp-pvc.yaml

$ kubectl get pvc -n wordpress
NAME        STATUS  VOLUME                                   CAPACITY  ACCESS MODES  STORAGECLASS  AGE
wp-pv-claim  Bound   pvc-d66e8a81-e3a0-4a1d-8562-fcba5edf5d93  8Gi       RWO           standard-rwo  2d5h
```

Gambar 3.32 Membuat konfigurasi *pvc* untuk aplikasi Wordpress

Selanjutnya dilakukan pembuatan file konfigurasi *configMap* yang diberi nama *wp-config.yaml.* File ini akan mengonfigurasikan koneksi antara aplikasi Wordpress yang akan di-*deploy* dengan database *cloud SQL instance* yang telah disiapkan pada layanan Google Cloud Platform.

```
$ nano wp-config.yaml
apiVersion: v1
data:
  WORDPRESS_DB_HOST: "34.101.191.233"
  WORDPRESS_DB_PORT: "3306"
  WORDPRESS_DB_USER: "wordpress"
  WORDPRESS_DB_TYPE: "mysql"
  WORDPRESS_DB_NAME: "wordpress"
kind: ConfigMap
metadata:
  name: mysql-config
  namespace: wordpress

$ kubectl apply -f wp-config.yaml
$ kubectl get configmaps -n wordpress
NAME                DATA   AGE
kube-root-ca.crt    1      2d2h
mysql-config        5      31h
```

Gambar 3.33 Membuat konfigurasi *configmaps* aplikasi Wordpress

File c*onfigMaps* untuk aplikasi Wordpress ini dinamai dengan *wp-config.yaml.* Database yang digunakan pada aplikasi ini merupakan database *SQL instance* berjenis *MySQL* versi 8.0. Pada gambar 3.33 Variabel *DB_HOST* diisikan dengan alamat IP dari *cloud SQL instance MySQL* yang telah dibuat, yaitu ’34.101.191.233’. Lalu variabel *DB_PORT* diisi dengan port bawaan dari *MySQL* yaitu ‘3306’. Pada variabel *DB_USER* diisi dengan nama user yang ada di *MySQL* yaitu ‘wordpress’. Pada variabel *DB_TYPE* diisi dengan tipe atau jenis dari SQL nya, yaitu ‘mysql’. Terakhir mengisi nilai pada variabel *DB_NAME* dengan nama database yang ada pada *cloud SQL instance* yang telah dibuat, yaitu dengan nama ‘wordpress’. Setelah pengaturan untuk konfigurasi selesai, dilakukan *apply* konfigurasi *configMaps*, lalu masukkan perintah untuk melihat *configMaps* yang telah dibuat.

Selanjutnya yaitu membuat file konfigurasi *secret* yang diberi nama dengan *wp-secret.yaml*. *Secret* digunakan untuk menyimpan dan mengatur informasi yang tidak untuk diketahui oleh banyak orang, seperti halnya kata sandi atau bahkan alamat IP dari database SQL yang dimiliki. Pada gambar 3.34 merupakan isi konfigurasi *secret* dan perintah untuk meng-*apply* serta melihat konfigurasi file *secret* yang telah dibuat.

```
$ nano wikijs-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: wordpress
  namespace: wordpress
type: Opaque
stringData:
  password: keamanan

$ kubectl apply -f wp-secret.yaml

$ kubectl get secret -n wordpress
NAME        TYPE     DATA   AGE
wordpress   Opaque   1      32h
```

Gambar 3.34 Membuat konfigurasi *secret* Wordpress

Selanjutnya membuat file konfigurasi *service* yang dinamai dengan *wp-svc.yaml*, gunanya konfigurasi ini agar aplikasi yang akan di-*deploy* dapat terekspos dan bisa diakses. *Service* yang digunakan yaitu tipe *LoadBalancer* dengan protokol TCP, dan pada port 80, serta target port 80. Kemudian konfigurasi *service* diterapkan atau di-*apply* dengan menggunakan perintah pada gambar 3.35, dan dapat dilihat juga hasil dari *service* yang telah dibuat.

```
$ nano wp-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  namespace: wordpress
spec:
  type: LoadBalancer
  selector:
    app: wordpress
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80

$ kubectl apply -f wp-svc.yaml

$ kubectl get svc -n wordpress
NAME        TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
```

```
wordpress   LoadBalancer   10.103.1.154   34.101.181.9   80:30321/TCP   32h
```

Gambar 3.35 Membuat konfigurasi *service* Wordpress

Lalu selanjutnya yaitu membuat file konfigurasi dengan jenis *deployment* yang dinamai *wp-deploy.yaml.* Dalam file *deployment* ini akan berisikan dengan jelas aplikasi apa yang akan di-*deploy*, serta beberapa deskripsi. Pada gambar 3.36.

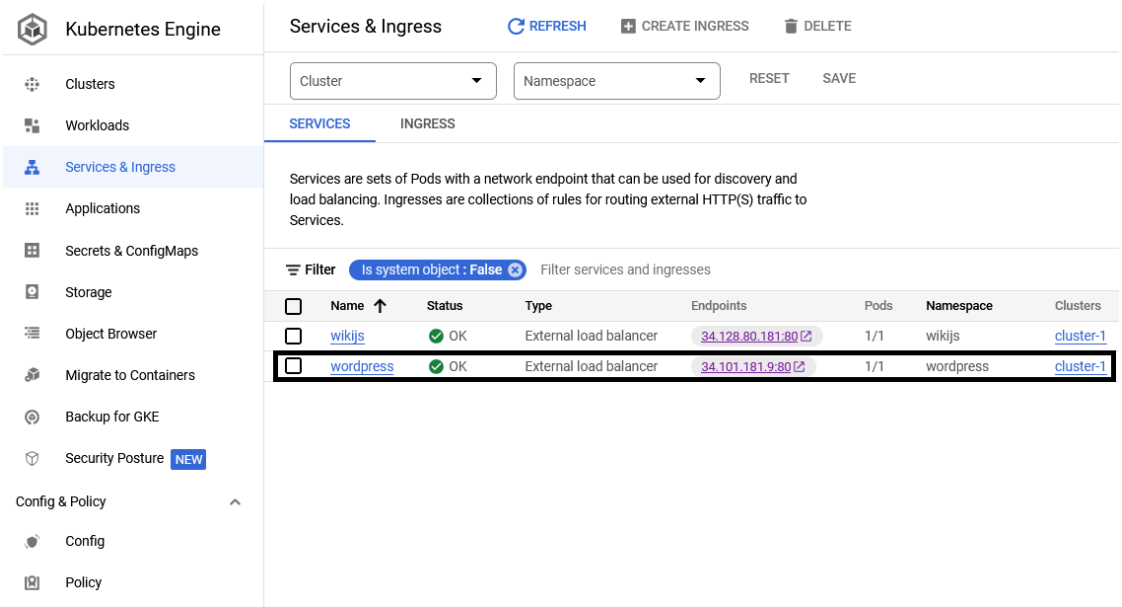
```
& nano wp-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: wordpress
  name: wordpress
  namespace: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
        - name: wordpress
          image: wordpress:6.2.1-apache
          env:
          - name: WORDPRESS_DB_HOST
            valueFrom:
              configMapKeyRef:
                key: WORDPRESS_DB_HOST
                name: mysql-config
          - name: WORDPRESS_DB_PORT
            valueFrom:
              configMapKeyRef:
                key: WORDPRESS_DB_PORT
                name: mysql-config
          - name: WORDPRESS_DB_USER
            valueFrom:
              configMapKeyRef:
                key: WORDPRESS_DB_USER
                name: mysql-config
          - name: WORDPRESS_DB_TYPE
            valueFrom:
              configMapKeyRef:
                key: WORDPRESS_DB_TYPE
                name: mysql-config
          - name: WORDPRESS_DB_NAME
            valueFrom:
              configMapKeyRef:
                key: WORDPRESS_DB_NAME
                name: mysql-config
          - name: WORDPRESS_DB_PASSWORD
            valueFrom:
              secretKeyRef:
```

```
                name: wordpress
                key: password
          ports:
          - containerPort: 80
            name: wordpress
          volumeMounts:
          - name: wordpress-persistent-storage
            mountPath: /var/www/html
      volumes:
      - name: wordpress-persistent-storage
        persistentVolumeClaim:
          claimName: wp-pv-claim

$ kubectl apply -f wp-deploy.yaml

$ kubectl get deploy -n wordpress
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
wordpress   1/1     1            1           28h

$ kubectl get pod -n wordpress
NAME                         READY   STATUS    RESTARTS   AGE
wordpress-57dddb7788-cww79   1/1     Running   0          28h
```

Gambar 3.36 Membuat konfigurasi *deployment* Wordpress

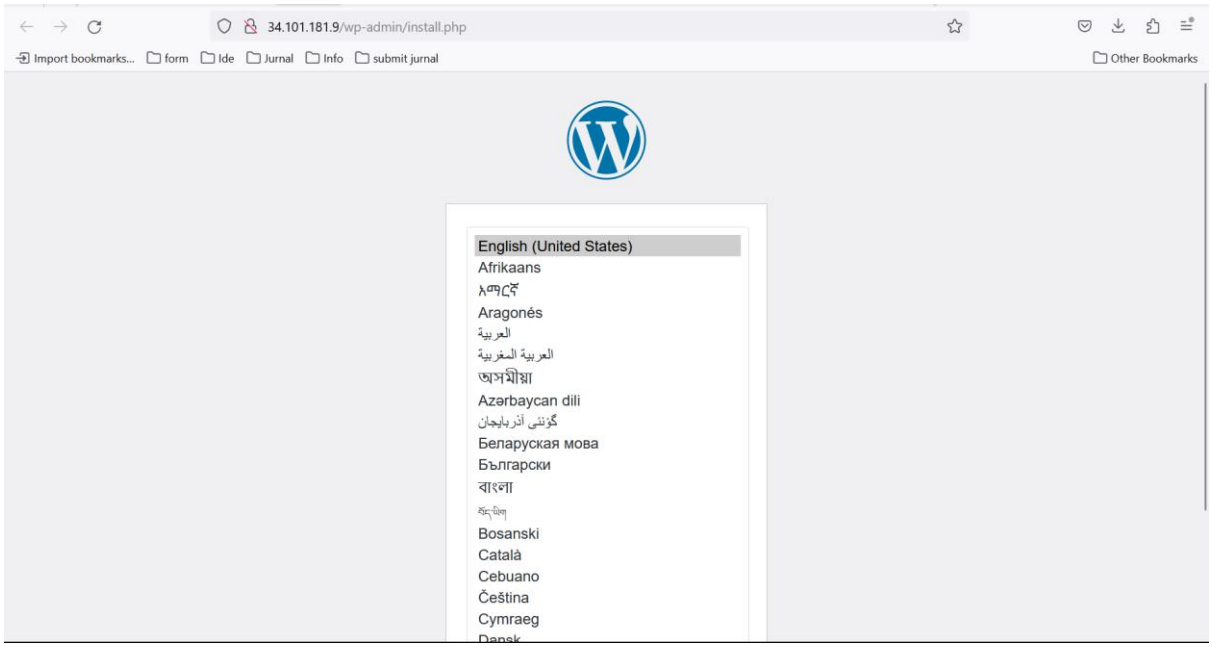
Pada kolom *container* merupakan kolom nama yang diisi nilai ‘wordpress’, lalu pada kolom *image* diisikan *value* dari *image* aplikasi Wordpress yang akan kita ambil dari web resmi Wordpress yaitu ‘wordpress:6.2.1-apache’.

Setelah tahapan konfigurasi untuk *deployment* aplikasi Wordpress selesai, dapat kita buka dan lihat pada menu *service & ingress* di console GCP, terdapat aplikasi Wordpress dengan tipe *service LoadBalancer* dan dibarengi dengan adanya alamat IP untuk mengakses aplikasi yang telah di-*deploy*. Secara jelas dapat dilihat pada gambar 3.37.



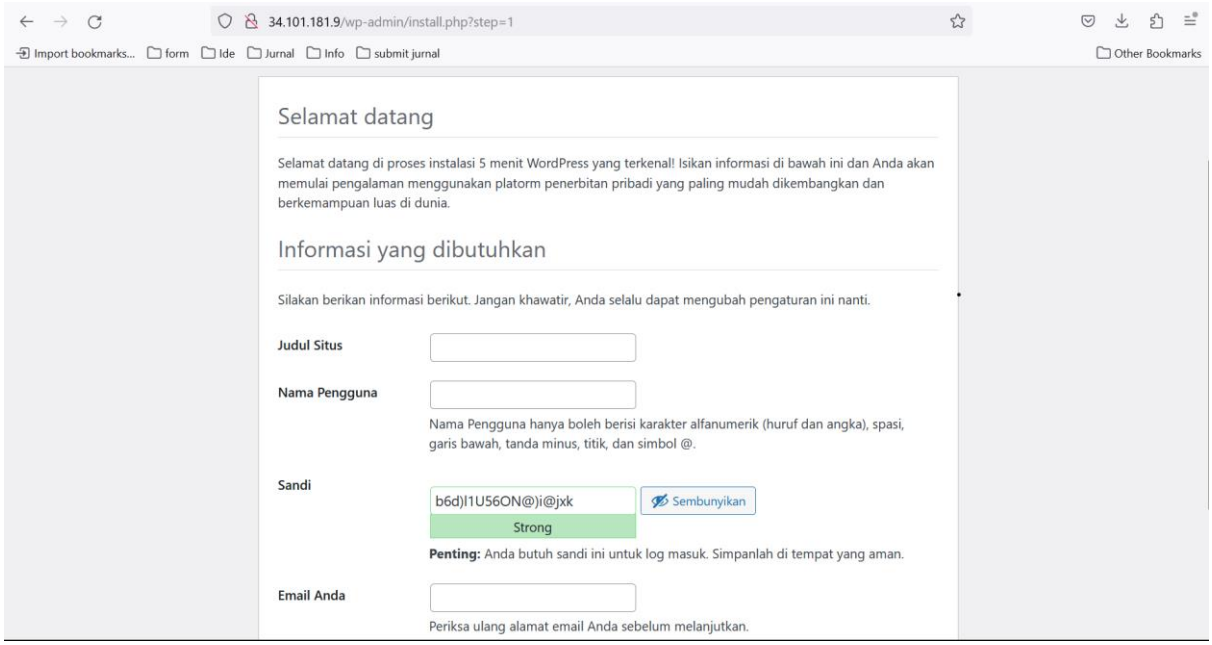
Gambar 3.37 Melihat *service* Wordpress yang telah berjalan pada Kubernetes cluster

Berikut merupakan aplikasi Wordpress yang telah berhasil di-*deploy*. Saat membuka alamat IP yang ada pada menu *service & ingress,* akan langsung diarahkan ke halaman awal aplikasi Wordpress seperti pada gambar 3.38.



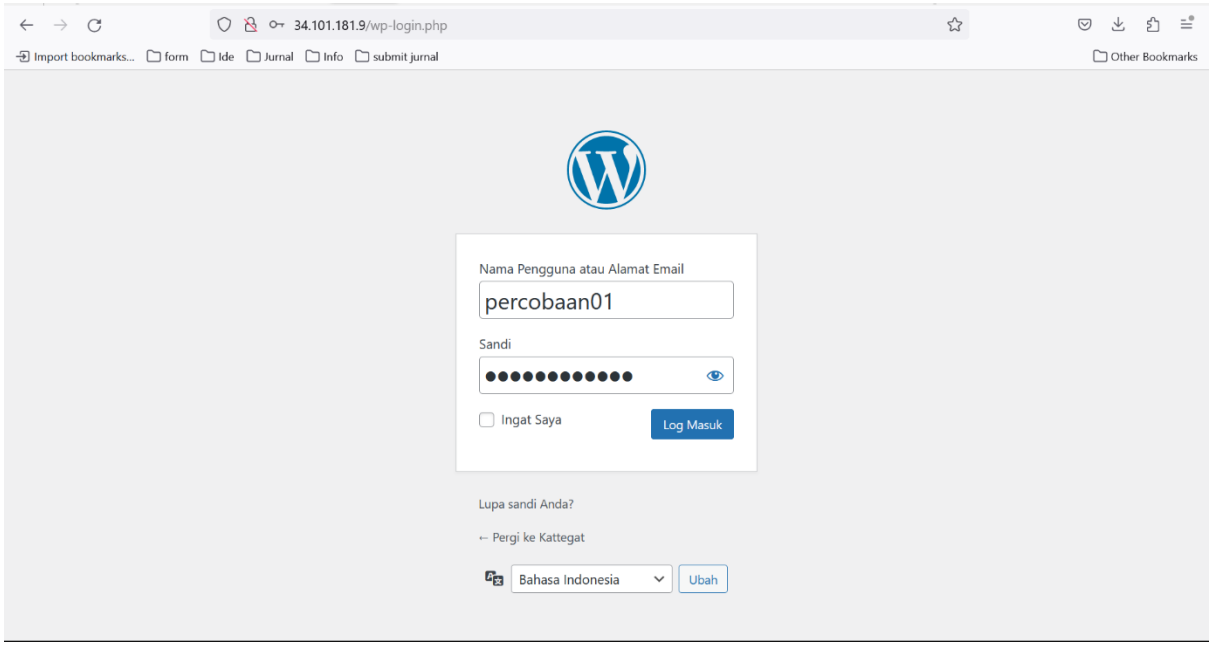
Gambar 3.38 Halaman awal untuk *setting up* aplikasi Wordpress

Pada gambar 3.39 merupakan halaman berisikan form untuk pendaftaran website berbasis Wordpress.



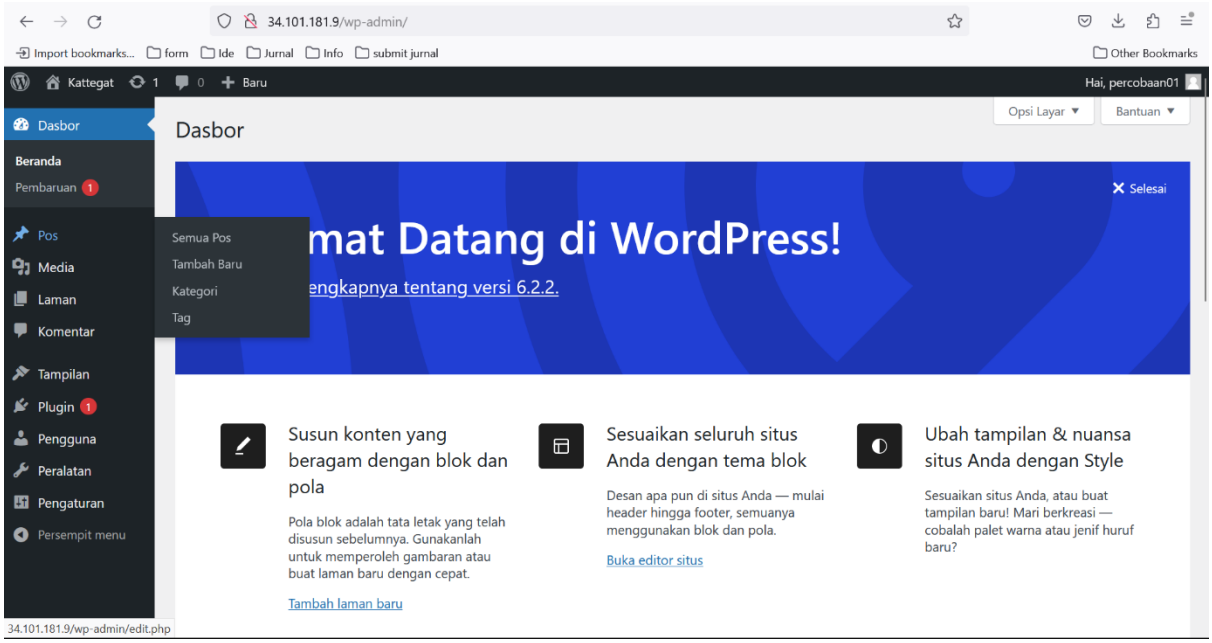
Gambar 3.39 Form pendaftaran akun admin Wordpress

Setelah pendaftaran admin selesai akan langsung diarahkan ke halaman login website Wordpress seperti pada gambar 3.40.



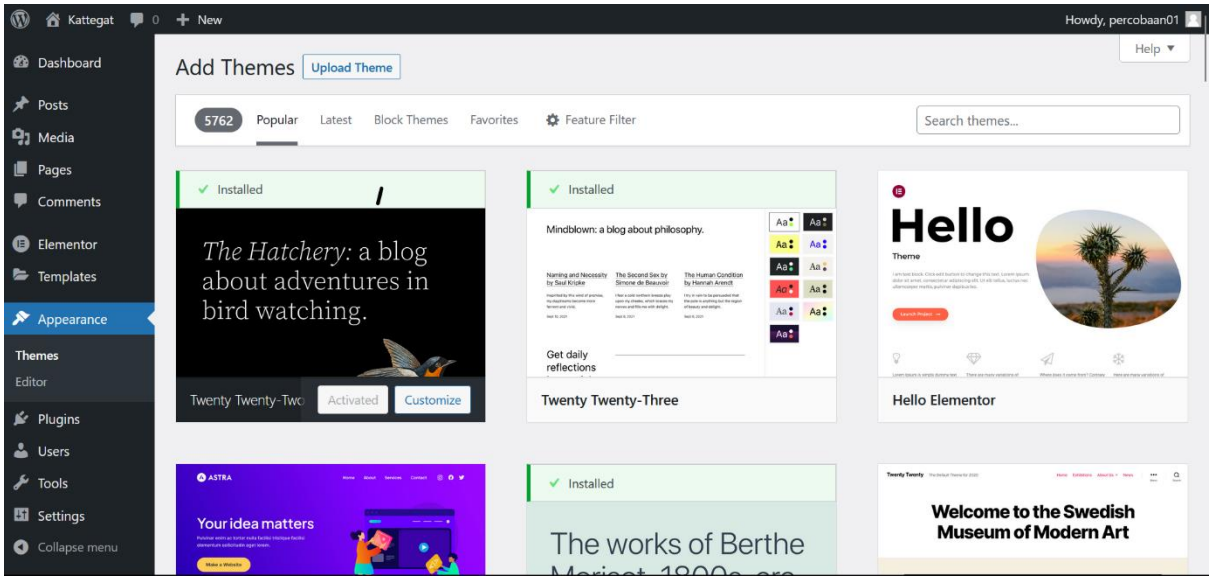
Gambar 3.40 Halaman masuk ke admin website Wordpress

Pada gambar 3.41 terlihat awal halaman setelah masuk ke akun admin Wordpress, akan ditampilkan *dashboard* dan juga beberapa menu lain untuk kustomisasi website berbasis Wordpress.



Gambar 3.41 Halaman dashboard website Wordpress

Pada website Wordpress yang telah di-*deploy* ini, penulis mencoba untuk membuat website yang berisikan beberapa konten informatif. Seperti pada gambar 3.42 penulis mengostumisasi website dengan pilihan tema yang diinginkan yang ada dari Wordpress.

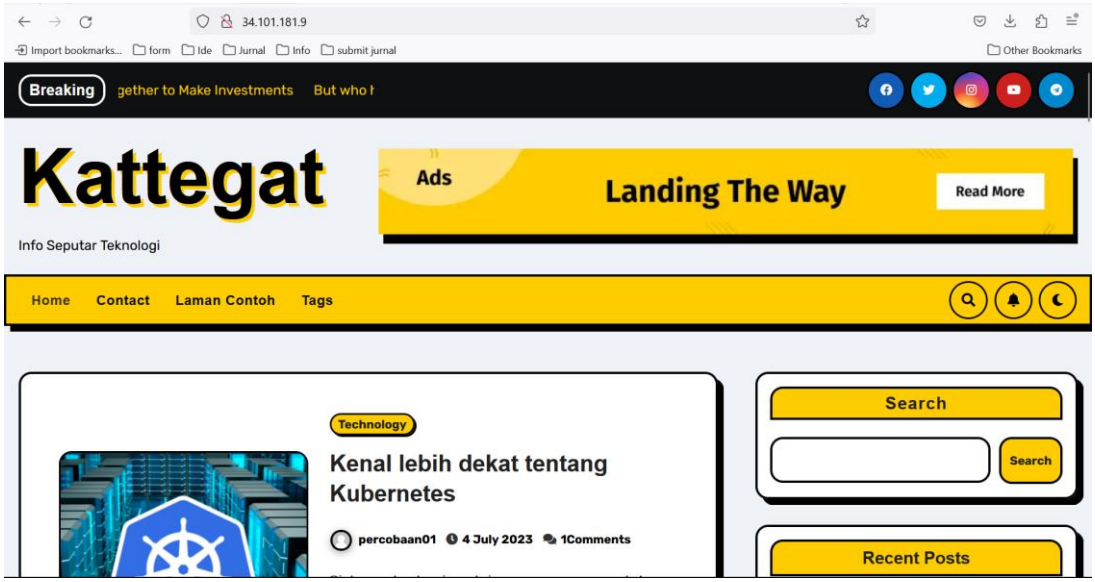


Gambar 3.42 Halaman kustomisasi website berbasis Wordpress

Pada gambar 3.43 merupakan contoh pengisian konten yang akan ditampilkan pada website.

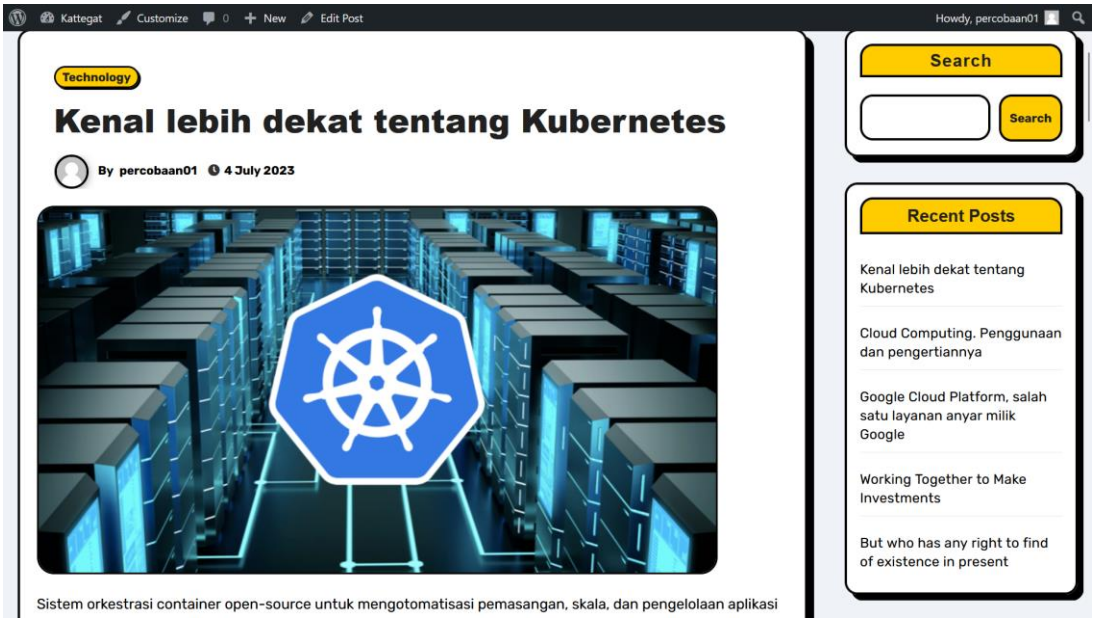
Gambar 3.43 Proses mengostumisasi konten yang ada pada website Wordpress

Hasil dari kustomisasi pengisian konten pada website Wordpress dapat dilihat pada gambar 3.44 yang merupakan halaman home dari website.



Gambar 3.44 Hasil kostumisasi konten pada halaman *home*

Pada gambar 3.45 merupakan tampilan dari isi konten yang dipilih pada halaman *home*. *Content management system* (CMS) berbasis Wordpress ini merupakan CMS yang familiar dan banyak keunggulan dari CMS lainnya. Website berbasis wordpress digunakan dalam berbagai hal, seperti situs untuk portofolio, perusahaan, dll. Bahkan Universitas Islam Indonesia menggunakan Wordpress sebagai basis CMS pada websitenya.



Gambar 3 45 Hasil kostumisasi konten pada sebuah halaman

3.3.4 *Load Testing* Aplikasi Wordpress pada layanan Kubernetes

Setelah serangkaian aktivitas *deployment* aplikasi selesai, selanjutnya dilakukan sebuah pengujian. Pengujian yang akan dilakukan pada aplikasi Wordpress yang telah di-*deploy* merupakan uji beban atau *load testing*. Pengujian beban atau *load testing* merupakan salah satu pengujian yang termasuk dalam uji performa. *Load testing* dilakukan dengan cara mengestimasi trafik pada sebuah website yang tersedia. Dengan mendefinisikan waktu maksimum untuk mengakses website tersebut, pengujian dapat menghasilkan seberapa maksimum atau seberapa lama waktu yang dibutuhkan untuk membuka halaman website pada sebuah web server (Molavi, 2016). Selain untuk mengetahui waktu maksimum dalam mengakses website Wordpress yang telah dibuat, pengujian ini juga diharapkan dapat membuktikan *availability* yang tinggi dari layanan meskipun terdapat banyak jumlah akses dan request yang telah ditentukan.

3.3.4.1 Pengujian

Pada pengujian ini akan menggunakan sebuah *tool* sumber terbuka yang bernama *apache bench* yang dibuat oleh Apache Organization untuk mengukur performa dari *Hypertext Transfer Protocol* (HTTP) web server. *Tool* ini dapat menghitung banyaknya *request per second* yang bisa dilayani oleh web server. Selain itu, fitur lainnya dapat digunakan untuk *load and performance test* yang biasa digunakan dalam menguji performa web server dengan batas pengujiannya seperti *transfer rate* dan *request per second* (Kadek & Ketut, 2020). Dalam pengujian ini, pengukuran kinerja dilakukan dengan melihat beberapa paramater, diantaranya seperti *time taken for test, request per second, time per second,* penggunaan CPU, dan penggunaan memori. Contoh perintah pengujian dapat dilihat pada gambar 3.46.

```
$ ab -n 100 -c 10 http://34.101.107.205/
```

Gambar 3.46 Perintah penggunaan Apache Benchmarking tool

Pada gambar 3.46 terdapat beberapa parameter. Parameter “-n” merupakan banyaknya jumlah koneksi yang nantinya akan dibuat ke server tujuan, sebagai contoh diatas akan dibuat sebanyak 100 koneksi ke server tujuan. Selanjutnya parameter “-c”, merupakan jumlah *concurrent request* atau permintaan bersamaan yang akan dibuat, dengan analogi server tersebut akan diakses bersamaan sebanyak 10 user yang akan mengaksesnya dalam satu waktu. Kemudian parameter terakhir yaitu alamat IP atau halaman yang akan dilakukan

proses *benchmarking* (Kadek & Ketut, 2020). Pada pengujian kali ini akan dilakukan variasi jumlah *request* yang dimulai dari 100, 500, 1000, 1500, dan 2000.

a. 100 *request*

Perintah yang digunakan untuk mengeksekusi 100 *request* dapat dilihat pada gambar 3.47.

```
$ ab -n 100 -c 10 http://34.101.107.205/
```

Gambar 3.47 Perintah simulasi beban akses 100 *request*

```
Benchmarking 34.101.107.205 (be patient).....done

Server Software:        Apache/2.4.56
Server Hostname:        34.101.107.205
Server Port:            80

Document Path:          /
Document Length:        64273 bytes

Concurrency Level:      10
Time taken for tests:   13.557 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      6453500 bytes
HTML transferred:       6427300 bytes
Requests per second:    7.38 [#/sec] (mean)
Time per request:       1355.691 [ms] (mean)
Time per request:       135.569 [ms] (mean, across all concurrent
requests)
Transfer rate:          464.87 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       13   14   1.0     14      17
Processing:   429 1316 244.8   1347    1885
Waiting:       62  678 156.9    687    1089
Total:        446 1330 244.8   1361    1899

Percentage of the requests served within a certain time (ms)
 50%   1361
 66%   1413
 75%   1494
 80%   1503
 90%   1611
 95%   1712
 98%   1891
 99%   1899
 100%   1899 (longest request)
```

Gambar 3.48 Hasil pengujian Wordpress pada layanan Kubernetes di 100 *request*

Pada pengujian dengan jumlah 100 *request* dan 10 *concurrent* hasilnya dapat dilihat pada gambar 3.48. Dari hasil pengujian tersebut, pada parameter *time taken for test* atau waktu yang dibutuhkan untuk dapat menampilkan website dengan 100 *request* dan 10 permintaan yang bersamaan yaitu 13.557 detik. Kemudian pada parameter

request per second didapati dengan rata – rata atau *mean* 7.38 detik. Selanjutnya pada parameter *time per request* didapati rata – rata nya yaitu 1355.691 milisekon.

b. 500 *request*

Selanjutnya berikut merupakan perintah yang digunakan dalam mengeksekusi 500 *request* dapat dilihat pada gambar 3.49.

```
$ ab -n 500 -c 50 http://34.101.107.205/
```

Gambar 3.49 Perintah simulasi beban akses 500 *request*

```
Benchmarking 34.101.107.205 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests

Server Software:        Apache/2.4.56
Server Hostname:        34.101.107.205
Server Port:            80

Document Path:          /
Document Length:        64273 bytes

Concurrency Level:      50
Time taken for tests:   71.963 seconds
Complete requests:      500
Failed requests:        0
Total transferred:      32267500 bytes
HTML transferred:       32136500 bytes
Requests per second:    6.95 [#/sec] (mean)
Time per request:       7196.341 [ms] (mean)
Time per request:       143.927 [ms] (mean, across all concurrent
requests)
Transfer rate:          437.88 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       13   14   0.9     14      17
Processing:   534 7069 2141.9   6893   15380
Waiting:       73 3692 1602.5   3305   10778
Total:        548 7083 2142.0   6907   15393

Percentage of the requests served within a certain time (ms)
  50%   6907
  66%   7496
  75%   7985
  80%   8196
  90%   9616
  95%  10892
  98%  13808
  99%  14206
 100%  15393 (longest request)
```

Gambar 3.50 Hasil pengujian Wordpress pada layanan Kubernetes di 500 *request*

Pada pengujian dengan jumlah 500 *request* dan 50 *concurrent* hasilnya dapat dilihat pada gambar 3.50. Dari hasil pengujian tersebut, pada parameter *time taken for test*

atau waktu yang dibutuhkan untuk dapat menampilkan website dengan 500 *request* dan 50 permintaan yang bersamaan yaitu 71.963 detik. Kemudian pada parameter *request per second* didapati dengan rata – rata atau *mean* 6.95 detik. Selanjutnya pada parameter *time per request* didapati rata – rata nya yaitu 7196.341 milisekon.

c. 1000 *request*

Perintah yang digunakan untuk mengeksekusi 1000 *request* dapat dilihat pada gambar 3.51.

```
$ ab -n 1000 -c 100 http://34.101.107.205/
```

Gambar 3.51 Perintah simulasi beban akses 1000 *request*

```
Benchmarking 34.101.107.205 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests

Server Software:        Apache/2.4.56
Server Hostname:        34.101.107.205
Server Port:            80

Document Path:          /
Document Length:        64273 bytes

Concurrency Level:      100
Time taken for tests:   146.321 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      64535000 bytes
HTML transferred:       64273000 bytes
Requests per second:    6.83 [#/sec] (mean)
Time per request:       14632.110 [ms] (mean)
Time per request:       146.321 [ms] (mean, across all concurrent
requests)
Transfer rate:          430.71 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       13   14   1.0     14      18
Processing:   681 14401 4124.3  14391   27580
Waiting:       73 7712 3056.3   7078   17982
Total:        698 14415 4124.5  14405   27594

Percentage of the requests served within a certain time (ms)
  50%  14405
  66%  15598
  75%  16383
  80%  17085
  90%  19399
  95%  21596
  98%  24380
  99%  25790
 100%  27594 (longest request)
```

Gambar 3.52 Hasil pengujian Wordpress pada layanan Kubernetes di 1000 *request*

Pada pengujian dengan jumlah 1000 *request* dan 100 *concurrent* hasilnya dapat dilihat pada gambar 3.52. Dari hasil pengujian tersebut, pada parameter *time taken for test* atau waktu yang dibutuhkan untuk dapat menampilkan website dengan 1000 *request* dan 100 permintaan yang bersamaan yaitu 146.321 detik. Kemudian pada parameter *request per second* didapati dengan rata – rata atau *mean* 6.83 detik. Selanjutnya pada parameter *time per request* didapati rata – rata nya yaitu 14632.110 milisekon.

d. 1500 *request*

Pada gambar 3.53 merupakan perintah yang digunakan untuk mengeksekusi.

```
$ ab -n 1500 -c 150 http://34.101.107.205/
```

Gambar 3.53 Perintah simulasi beban akses 1500 *request*

```
Benchmarking 34.101.107.205 (be patient)
Completed 150 requests
Completed 300 requests
Completed 450 requests
Completed 600 requests
Completed 750 requests
Completed 900 requests
Completed 1050 requests
Completed 1200 requests
Completed 1350 requests
Completed 1500 requests
Finished 1500 requests

Server Software:        Apache/2.4.56
Server Hostname:        34.101.107.205
Server Port:            80

Document Path:          /
Document Length:        64273 bytes

Concurrency Level:      150
Time taken for tests:   219.287 seconds
Complete requests:      1500
Failed requests:        0
Total transferred:      96802500 bytes
HTML transferred:       96409500 bytes
Requests per second:    6.84 [#/sec] (mean)
Time per request:       21928.715 [ms] (mean)
Time per request:       146.191 [ms] (mean, across all concurrent
requests)
Transfer rate:          431.10 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       12   14   1.0     14      18
Processing:   557 21507 5581.4  22293   35593
Waiting:       71 12439 4122.4  11997   26874
Total:        572 21521 5581.5  22307   35607

Percentage of the requests served within a certain time (ms)
  50%  22307
  66%  24191
  75%  25108
  80%  25604
  90%  27395
  95%  29778
  98%  32009
  99%  33579
 100%  35607 (longest request)
```

Gambar 3.54 Hasil pengujian Wordpress pada layanan Kubernetes di 1500 *request*

Pada pengujian dengan jumlah 1500 *request* dan 150 *concurrent* hasilnya dapat dilihat pada gambar 3.54. Dari hasil pengujian tersebut, pada parameter *time taken for test* atau waktu yang dibutuhkan untuk dapat menampilkan website dengan 1500 *request* dan 150 permintaan yang bersamaan yaitu 219.287 detik. Kemudian pada parameter *request per second* didapati dengan rata – rata atau *mean* 6.84 detik. Selanjutnya pada parameter *time per request* didapati rata – rata nya yaitu 21928.715 milisekon.

e. 2000 *request*

Pada gambar 3.55 merupakan perintah pada 2000 *request.*

```
$ ab -n 2000 -c 200 http://34.101.107.205/
```

Gambar 3.55 Perintah simulasi beban akses 2000 *request*

```
Benchmarking 34.101.107.205 (be patient)
Completed 200 requests
Completed 400 requests
Completed 600 requests
Completed 800 requests
Completed 1000 requests
Completed 1200 requests
Completed 1400 requests
Completed 1600 requests
Completed 1800 requests
Completed 2000 requests
Finished 2000 requests

Server Software:        Apache/2.4.56
Server Hostname:        34.101.107.205
Server Port:            80

Document Path:          /
Document Length:        64273 bytes

Concurrency Level:      200
Time taken for tests:   282.529 seconds
Complete requests:      2000
Failed requests:        0
Total transferred:      129070000 bytes
HTML transferred:       128546000 bytes
Requests per second:    7.08 [#/sec] (mean)
Time per request:       28252.949 [ms] (mean)
Time per request:       141.265 [ms] (mean, across all concurrent
requests)
Transfer rate:          446.13 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       13   14   1.0     14      18
Processing:  5030 27610 4928.1  27898   48013
Waiting:       76 18156 3866.5  17875   36704
Total:       5043 27624 4928.2  27912   48029

Percentage of the requests served within a certain time (ms)
  50%  27912
  66%  29193
  75%  30082
  80%  30709
  90%  32296
  95%  34109
  98%  37808
  99%  39720
 100%  48029 (longest request)
```

Gambar 3.56 Hasil pengujian Wordpress pada layanan Kubernetes di 2000 *request*

Pada pengujian dengan jumlah 2000 *request* dan 200 *concurrent* hasilnya dapat dilihat pada gambar 3.56. Dari hasil pengujian tersebut, pada parameter *time taken for test* atau waktu yang dibutuhkan untuk dapat menampilkan website dengan 2000 *request* dan 200 permintaan yang bersamaan yaitu 282.259 detik. Kemudian pada parameter *request per second* didapati dengan rata – rata atau *mean* 7.08 detik. Selanjutnya pada parameter *time per request* didapati rata – rata nya yaitu 28252.949 milisekon. Terlihat juga bahwa pengujian yang dilakukan dari 100 hingga 2000 *request* tidak terdapat satupun *request* yang gagal.

3.3.4.2 Analisis Hasil Pengujian

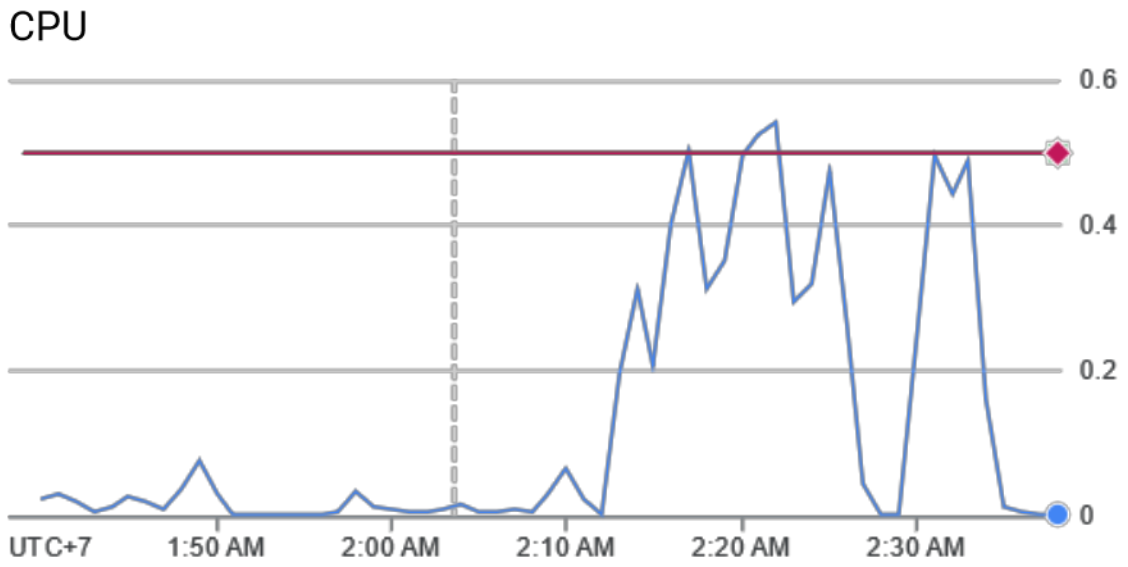
Dari pengujian yang telah dilakukan terdapat hasil yang akan dianalisa dan ditarik kesimpulan, hal ini bertujuan untuk mempermudah dalam memahami luaran dari pengujian. Pengukuran kinerja aplikasi Wordpress yang di-*deploy* menggunakan layanan Kubernetes dilihat dari parameter *time taken for test, request per second,* dan *time per request* serta parameter pada penggunaan CPU dan memori di layanan Kubernetes cluster GCP. Pada tabel 3.3 merupakan ringkasan hasil pengujian.

Tabel 3.3 Hasil pengujian *Load Testing* pada beberapa *request*

Pengujian	*Time taken for test*	*Request per second*	*Time per request*
100 *request* 10 *concurrent*	13.557 detik	7.38 [sec] (mean)	1355.691 [ms] (mean)
500 *request* 50 *concurrent*	71.963 detik	6.95 [sec] (mean)	7196.341 [ms] (mean)
1000 *request* 100 *concurrent*	146.321 detik	6.83 [sec] (mean)	14632.110 [ms] (mean)
1500 *request* 150 *concurrent*	219.287 detik	6.84 [sec] (mean)	21928.715 [ms] (mean)
2000 *request* 200 *concurrent*	282.259 detik	7.08 [sec] (mean)	28252.949 [ms] (mean)

Pada tabel 3.3 terlihat bahwa *time taken for test* yang dimulai pada 100, 500, 1000, 1500, hingga 2000 *request,* terdapat peningkatan waktu dalam menyelesaikan *request*.

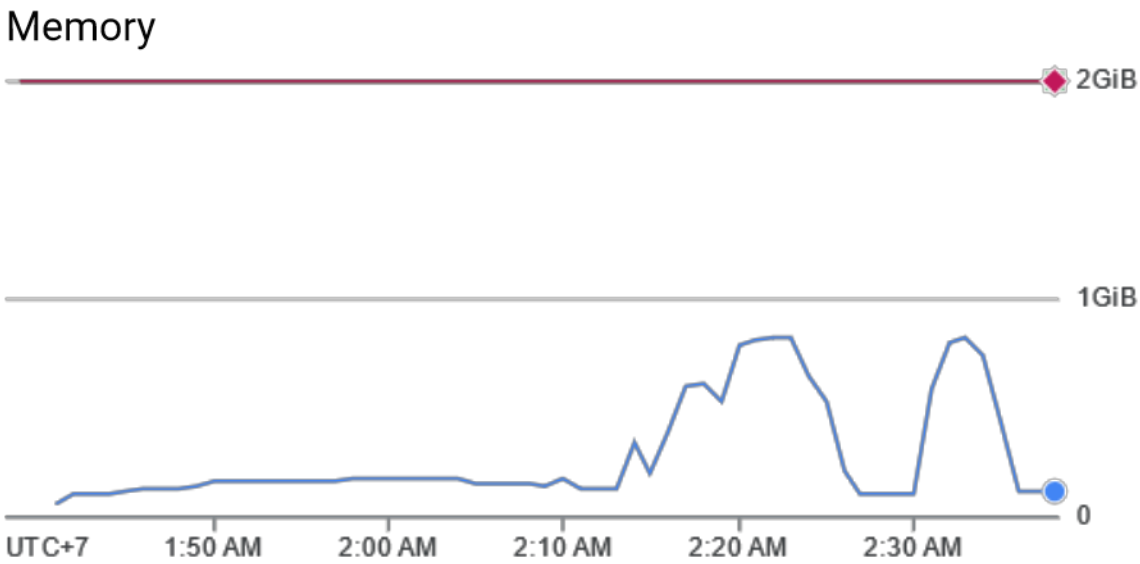
Kemudian pada *request per second* setiap variasi pengujian *request* nya berkisar rata – rata 7.38 detik pada 100 *request*, kemudian pada 500 *request* didapati rata – rata 6.95 detik, lalu pada 1000 *request* didapati rata – rata 6.83 detik, pada 1500 *request* rata – rata nya 6.84 detik, dan terakhir pada 2000 *request* rata – rata nya 7.08 detik. Hasil pengujian selanjutnya yaitu dari *time per request* yang didapati rata – ratanya yaitu 1355.691 milisekon pada 100 *request*, kemudian pada 500 *request* rata – ratanya 7196.341 milisekon, 14632.110 milisekon pada 1000 *request*, 21928.715 milisekon pada 1500 *request*, lalu 28252.949 milisekon pada 2000 *request*. Saat pengujian tersebut tidak ditemukan kegagalan dalam pemenuhan permintaan yang menandakan servis atau layanan dari Wordpress yang ada pada Kubernetes cluster GCP dapat memenuhi permintaan dari 100, 500, 1000, 1500, dan 2000 *request* disertai juga dengan permintaan yang bersamaan.



Gambar 3.57 Penggunaan CPU saat pengujian Wordpress pada Kubernetes cluster GCP

Parameter hasil pengujian selanjutnya dapat dilihat dari CPU yang digunakan pada layanan Kubernetes cluster untuk aplikasi Wordpress. Dapat dilihat pada gambar 3.57 merupakan grafik yang terlihat meningkat dalam penggunaan CPU saat pengujian. Konfigurasi Kubernetes cluster untuk sumber daya CPU yang dipilih merupakan spesifikasi yang diautomisasi oleh penyedia layanan, hal tersebut akan membuat penggunaan CPU menyesuaikan dari penggunaan, apabila penggunaan CPU diharuskan melebihi ambang batas untuk tetap dapat menyediakan layanan, maka ambang batas penggunaan CPU akan automatis meningkat. Seperti contoh pada gambar 3.57, dalam pengujian 100, 500, dan 1000

request penggunaan CPU masih pada ambang batas bawaan dan tidak melebihi kapasitas, namun pada saat dilakukan pengujian di 1500 dan 2000 *request,* penggunaan CPU melebihi ambang batas yang ditentukan, namun saat mengakses website Wordpress pada layanan Kubernetes masih tetap bisa dilakukan. Hal tersebut menandakan bahwa *availability* layanan Kubernetes masih dapat tersedia, sekalipun jumlah akses *user* dan *request* mencapai 1500 dan 2000 *request*.



Gambar 3.58 Penggunaan memori saat pengujian Wordpress pada Kubernetes cluster GCP

Parameter pengujian selanjutnya yaitu memori yang digunakan untuk aplikasi Wordpress pada layanan Kubernetes cluster GCP. Penggunaan sumber daya memori pada saat pengujian terlihat pada gambar 3.58 yang hasilnya terdapat adanya peningkatan dalam penggunaan memori, namun tidak melebihi ambang batas yang telah ditentukan oleh penyedia layanan, hal ini menandakan dalam pengujian dengan *request* yang variatif, khususnya dalam pengujian di penelitian ini, penggunaan memori tidak terdapat peningkatan yang signifikan.

BAB IV
REFLEKSI PELAKSANAAN MAGANG

4.1 Relevansi Akademik

Kegiatan magang yang penulis lakukan merupakan pengalaman yang sangat berkesan. Manfaat yang dirasakan meliputi banyak hal seperti teknis maupun non teknis, hal ini tentunya akan semakin membuka wawasan yang lebih luas bagi mahasiswa yang haus akan ilmu. Pada pelaksanaan kegiatan magang berlangsung, penulis mendapatkan posisi sebagai *Cloud Infrastructure Engineer*. Semakin berkembangnya teknologi dalam pengelolaan aplikasi, *Cloud Computing* merupakan salah satu *role* yang banyak dibutuhkan pada beberapa perusahaan.

Dikembangkannya teknologi Kubernetes merupakan sebuah cara untuk memanajemen dan mengorkestrasi banyaknya aplikasi yang dikemas dalam sebuah *container*. Selain itu, penggunaan layanan dari Google Cloud Platform sendiri bukanlah salah satu platform dalam penggunaan *cloud computing*, selain itu terdapat *Amazon Web Service* (AWS) ataupun *Microsoft Azure* yang juga merupakan platform layanan untuk *cloud computing*. Namun setiap perusahaan memiliki perbedaan kebutuhan dan hal tersebut merupakan sebuah pilihan.

Pada praktiknya, tujuan dari penggunaan teknologi Kubernetes di perusahaan Bio Farma yang dilakukan oleh para *developer* di perusahaan tersebut yaitu, digunakan sebagai *tools* yang dapat memudahkan dalam hal *maintenance* aplikasi. Dikarenakan di perusahaan terdapat banyak aplikasi yang telah dikembangkan dan tiap aplikasinya memiliki banyak servis atau modul, maka dengan adanya penerapan Kubernetes dalam *deployment* aplikasi akan mengisolasi tiap servis pada aplikasi, hal ini tentunya tidak akan mempengaruhi servis yang lain dan juga para *developer* dapat mengetahui dari mana letak kesalahan atau error yang muncul.

4.2 Pembelajaran Magang

4.2.1 Teknis

Dalam kesempatan magang tersebut, penulis menjadi lebih mengetahui tentang hal-hal apa saja yang diperlukan untuk menjadi seorang *Cloud Infrastructure Engineer*, seperti halnya kemampuan dalam pengoperasian sistem operasi Linux, memanfaatkan beberapa layanan *instance* yang ada di Google Cloud Platform, dan juga mengelola jaringan. Tentunya pengalaman tersebut sangat berguna dan membuka wawasan yang luas bagi penulis, terlebih

dalam ilmu *Cloud Computing*. Hal yang penulis rasakan manfaat dalam magang ini adalah saat menggunakan layanan Kubernetes dalam pengembangan aplikasi, hal tersebut sangat bermanfaat, apalagi hal itu merupakan hal yang sangat baru dan belum diajarkan di bangku perkuliahan, terutama dalam penggunaan teknologi Kubernetes.

Beberapa hal identik yang pernah penulis lakukan yaitu saat mendapatkan mata kuliah Sistem Jaringan dan Komputer (SJK), hanya saja perbedaannya terletak pada penggunaan *platform* komputasi awan. Pada mata kuliah SJK, *platform* yang digunakan merupakan Amazon Web Service, sedangkan saat kegiatan magang, *platform* yang digunakan adalah Google Cloud Platform. Penulis diharuskan untuk mengetahui caranya membuat, mengkonfigurasi, dan mengelola beberapa layanan yang ada seperti *virtual machine instance, SQL database instance* dan layanan lain yang disediakan oleh Google Cloud Platform. Selain itu, hal ini berguna untuk mempelajari layanan apa saja yang harus digunakan demi memenuhi kebutuhan dalam proyek di kegiatan magang penulis. Selain itu, untuk bisa mengakses layanan yang ada di Google Cloud Platform mengharuskan penggunanya untuk membayar layanan yang digunakan.

4.2.2 Non Teknis

Dalam manfaat non teknis yaitu bertemu dengan *expertise* di bidangnya yang bukan hanya menambah ilmu saja tetapi juga menambah relasi serta mempererat hubungan dengan mentor. Manajemen diri juga merupakan salah satu hal yang sangat penting dalam berlangsungnya kegiatan magang. Demi menjaga keberlangsungan agar aktivitas magang dan aktivitas lainnya tidak bertabrakan dibutuhkan manajemen waktu yang handal. Terlebih saat kegiatan magang berlangsung hal itu berbarengan dengan kegiatan perkuliahan, meskipun pada saat itu kegiatan perkuliahan berlangsung secara daring, namun kegiatan magang juga dapat berjalan lancar. Kemudian selama menjalankan kegiatan magang penulis dapat belajar bersosialisasi dengan orang baru serta menjalin komunikasi yang baik dengan orang lain.

4.2.3 Hambatan dan Tantangan

Dalam beberapa hal pastinya akan ada kendala dan hambatan, tak terkecuali dalam kegiatan magang yang penulis alami selama magang di PT. Bio Farma. Kendala dan hambatan yang dihadapi penulis ketika melakukan kegiatan magang seperti halnya, ada beberapa kesalahan seperti miskonfigurasi pada *virtual machine* dan kode program yang mana hal ini menyebabkan aplikasi yang dikembangkan tidak bisa diakses, namun dalam hal

ini mentor bersedia memberikan arahan dan juga pengalamannya dalam memecahkan masalah tersebut. Selain itu penulis juga mendapatkan musibah kecelakaan motor sehabis pulang kantor yang menyebabkan laptop yang penulis gunakan untuk melakukan kegiatan magang tidak dapat digunakan sebagaimana mestinya, namun penulis bisa melakukan kegiatan magang dengan menggunakan laptop lama yang penulis miliki, dan untungnya pekerjaan yang penulis kerjakan bisa dilakukan secara *remote* karena layanan Google Cloud Computing bisa diakses dimanapun dan kapanpun.

Selain itu, tantangan yang penulis alami saat kegiatan magang berlangsung merupakan hal-hal yang menyangkut teknis lain, seperti halnya penulis diharuskan untuk menggunakan *Google Cloud Platform*, Docker, Kubernetes, DBeaver, dan tools lain yang penulis gunakan yang mana aplikasi dan layanan tersebut terdengar baru ditelinga penulis, namun dengan dasar semangat dan haus akan ilmu, penulis bisa mempelajari hal tersebut dengan baik.

BAB V
PENUTUP

5.1 Kesimpulan

Pada setiap tahapan yang dilakukan dimulai dari persiapan beberapa sumber daya, konfigurasi perangkat, kemudian pada pembuatan konfigurasi file untuk *deployment* aplikasi Wordpress dan Wikijs, hingga pengujian telah menemukan hasil. Hal ini memperlihatkan bahwa dengan tahapan – tahapan tersebut, pengimplementasian *deployment* aplikasi menggunakan Kubernetes cluster pada layanan Google Cloud Platform telah dapat diimplementasikan sehingga aplikasi Wordpress dan Wikijs yang telah di-*deploy* dapat dijalankan dan diakses. Dari pengujian beban yang dilakukan, terdapat kesimpulan bahwa saat pengujian tidak ditemukan kegagalan dalam pemenuhan permintaan yang menandakan servis atau layanan dari Wordpress yang ada pada Kubernetes cluster GCP dapat memenuhi permintaan dari 100, 500, 1000, 1500, dan 2000 *request* dengan waktu yang variatif namun dapat terpenuhi. Dalam penggunaan CPU sempat memenuhi ambang batas, namun masih tetap dapat diakses. Hal tersebut menandakan bahwa *availability* layanan Kubernetes masih dapat tersedia, sekalipun jumlah akses *user* dan *request* mencapai 1500 dan 2000 *request*.

5.2 Saran

Terdapat saran yang ingin disampaikan dalam penelitian ini. Untuk penelitian selanjutnya yaitu dilakukan sebuah hosting menggunakan nama domain agar aplikasi dapat diakses menggunakan nama domain yang diinginkan serta menggunakan sertifikat SSL agar website yang dimiliki lebih aman. Selain itu dilakukan juga uji keamanan terhadap aplikasi yang di-*deploy*. Kemudian hal yang tidak boleh luput dalam pengerjaan untuk *deployment* aplikasi menggunakan Kubernetes adalah, diharuskan untuk memiliki ketelitian yang sangat tinggi, sebab konfigurasi file satu dan yang lainnya sangat terkait, selain itu juga penamaan dari database, password dari database, dan hal lainnya yang menyangkut dengan database diharuskan untuk memiliki perhatian agar dapat mengakses database yang telah dibuat.

DAFTAR PUSTAKA

Agung, & Subiyantoro. (2018). Analisis Cluster Container pada Kubernetes dengan Infrastruktur Google Cloud Platform. *JIPI*, 84-93.

Ahmad, & Herri. (2011). Cloud Computing : Solusi ICT ? *Jurnal Sistem Informasi*, 336-345.

Arief, & dkk. (2020). Pengamanan Container Orchestration Berbasis Kubernetes di Lembaga Penerbangan dan Antariksa Nasional (LAPAN). *Jurnal TEKNOINFO*, 1-8.

Bayu, Vera, & Siti. (2020). Analisis Performansi Proses Scaling pada Kubernetes dan Docker Swarm Menggunakan Metode Horizontal Scaler. *e-Proceeding of Engineering* (p. 7793). Bandung: Telkom University.

Biofarma. (2022, Agustus 26). *Bio Farma Raih Sertifikat INDI 4.0 Level 3 dari Kemenperin dan Hadirkan Digitalisasi Farmasi dalam Expo Indonesia 4.0 2022*. Retrieved from www.biofarma.co.id: https://www.biofarma.co.id/id/announcement/detail/bio-farma-raih-sertifikat-indi-40-level-3-dari-kemenperin-dan-hadirkan-digitalisasi-farmasi-dalam-expo-indonesia-40-2022

Google Cloud. (2023, 7 5). *Deploy WordPress on GKE with Persistent Disk and Cloud SQL*. Retrieved from Google Cloud: https://cloud.google.com/kubernetes-engine/docs/tutorials/persistent-disk

Dian, & dkk. (2020). Pengelolaan Konten Web Menggunakan Wordpress, Canva dan Photoshop untuk Guru-Guru Wilayah Jakarta. *ABDIHAZ: Jurnal Ilmiah Pengabdian pada Masyarakat*, 11-15.

Dinda, & Ria. (2022). Penerapan Cloud Computing Dalam Aplikasi Panggil Teknisi Berbasis Android Menggunakan Google Cloud Platform. *Jurnal Sains Komputer & Informatika*, 1292-1300.

Hamdan. (2018). Industri 4.0: Pengaruh Revolusi Industri pada Kewirausahaan Demi Kemandirian Ekonomi. *JURNAL NUSAMBA*, 1-8.

Jahiduddin, Eko, & Fariz. (2020). Pengembangan Infrastruktur Analisis Data Heart Rate berbasis Microservices menggunakan Kubernetes. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, 100-108.

Jaroslav, & Jakub. (2018). *Git-based Wiki System.* Prague: Czech Technical University.

Kezia, Agustinus, & Henry. (2019). Implementasi Container Kubernetes untuk Mendukung. *Jurnal Infra*, 1-5.

Muttakin, & Fadhly. (2021). Implementasi Kubernetes Cluster Menggunakan Vagrant. *9th Applied Business and Engineering* (pp. 218-227). Pekanbaru: Applied Business and Engineering.

Nugroho, & Kartadie. (2021). Cloud Storage dengan Teknologi Kubernetes untuk Platform Collaborative Research. *JIPI*, 74-81.

Odi, & Dian. (2023). Provisioning Google Kubernetes Engine Cluster dengan Menggunakan Terraform dan Jenkins pada Dua Environment. *JIPI*, 597-606.

Riana. (2020). Implementasi Cloud Computing Technology dan Dampaknya Terhadap Kelangsungan Bisnis Perusahaan Dengan Menggunakan Metode Agile dan Studi Literatur. *JURIKOM*, 439-449.

Stefanus, & Yulfan. (2021). Analisis Perbandingan Performa Web Server Docker Swarm dengan Kubernetes Cluster. *Conference on Management, Business, Innovation, Education and Social Science* (pp. 825-833). Batam: Universitas Internasional Batam.

Wisnu, & Fadhly. (2022). Perbandingan Kinerja Ingress Controller pada Kubernetes Menggunakan Traefik dan Nginx. *Jurnal Politeknik Caltex Riau*, 289-295.

Wisnu, Adam, & Hayati. (2013). Deployment Aplikasi untuk Multi Server dengan Menggunakan Capistrano. *JURNAL ISTEK*, 47-59.

LAMPIRAN