Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – IS184853

DESAIN DAN IMPLEMENTASI *TEMPLATE* CI/CD UNTUK OTOMATISASI PADA PROSES PENGUJIAN DAN *DEPLOYMENT*

DESIGN AND IMPLEMENTATION OF CI/CD TEMPLATE FOR AUTOMATED TESTING AND DEPLOYMENT AUTOMATION

RENDY ANANTA
NRP 05211840000021

Dosen Pembimbing
Nisfu Asrul Sani, S.Kom.,M.Sc.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2022

TUGAS AKHIR – IS184853

DESAIN DAN IMPLEMENTASI TEMPLATE CI/CD UNTUK OTOMATISASI PADA PROSES PENGUJIAN DAN DEPLOYMENT

RENDY ANANTA
NRP. 05211840000021

Dosen Pembimbing
Nisfu Asrul Sani, S.Kom.,M.Sc.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2022

UNDERGRADUATE THESIS – IS184853

DESIGN AND IMPLEMENTATION OF CI/CD TEMPLATE FOR AUTOMATED TESTING AND DEPLOYMENT AUTOMATION

RENDY ANANTA
NRP. 05211840000021

Supervisor
Nisfu Asrul Sani, S.Kom.,M.Sc.

INFORMATION SYSTEMS DEPARTMENT
Faculty of Intelligent Electrical and Informatics Technology
Sepuluh Nopember Institute of Technology
Surabaya 2022

LEMBAR PENGESAHAN

Desain dan Implementasi Template CI/CD untuk Otomatisasi Pada Proses Pengujian dan Deployment

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer (S.Kom)
pada
Departemen Sistem Informasi
Fakultas Teknologi Elektro dan Informatika Cerdas (ELECTICS)
Institut Teknologi Sepuluh Nopember

Oleh :

Rendy Ananta
NRP: 05211840000021

Surabaya, 10 Februari 2022

Kepala Departemen Sistem Informasi

LP/P/22/852

Dr. Mudjahidin, ST, MT
NIP. 197010102003121001

LEMBAR PERSETUJUAN

Desain dan Implementasi Template CI/CD untuk Otomatisasi Pada Proses Pengujian dan Deployment

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Oleh :

Rendy Ananta
NRP: 05211840000021

Disetujui Tim Penguji: Tanggal Ujian: 21 January 2022
Periode Wisudas: Maret 2022

Nisfu Asrul Sani, S.Kom, M.Sc (Pembimbing 1)

Dr. Eng. Febriliyan Samopa, S.Kom, M.Kom (Penguji 1)

Bekti Cahyo Hidayanto, S.Si, M.Kom (Penguji 2)

DESAIN DAN IMPLEMENTASI *TEMPLATE* CI/CD UNTUK OTOMATISASI PADA PROSES PENGUJIAN DAN *DEPLOYMENT*

Nama Mahasiswa : Rendy Ananta
NRP : 05211840000021
Departemen : Sistem Informasi FTEIC-ITS
Pembimbing 1 : Nisfu Asrul Sani, S.Kom, M.Sc.

ABSTRAK

Kepuasan pelanggan sekarang menjadi tujuan utama pengembangan perangkat lunak. Kepuasan pelanggan dapat dicapai menggunakan penerapan *agile*, yang bertujuan untuk memperpendek waktu perilisan perangkat lunak, dengan begitu pengembang produk dan layanan mendapatkan umpan balik dari pengguna dengan lebih cepat. Umpan balik tersebut berguna untuk mengembangkan produk atau layanan yang lebih baik, sehingga kepuasan pelanggan dapat meningkat. Namun, penggunaan *agile* saja dalam pengembangan perangkat lunak pada organisasi tidak cukup untuk mempercepat proses pengembangan dan kepuasan pelanggan. Beberapa permasalahan yang masih terjadi adalah masalah produktivitas dan kualitas perangkat lunak. Penelitian sebelumnya membuktikan bahwa penerapan *practice DevOps* terutama *Continuous Integration* dan pengujian terotomatisasi berbasiskan *test-driven development* (TDD) dapat meningkatkan produktivitas dan meningkatkan kualitas perangkat lunak. Pada tugas akhir ini, dilakukan perancangan *template* penerapan *DevOps* terkait *Continuous Integration* (CI), *Continuous Deployment* (CD), dengan memanfaatkan pengujian terotomatisasi untuk otomatisasi proses *deployment*. Perancangan *template* tersebut meliputi konfigurasi

infrastruktur CI/CD dan proses yang dijalankan pada *pipeline* CI/CD. Setelah itu, dilakukan pengembangan terkait *template* CI/CD, yang akan diujicobakan pada aplikasi berbasis web yang dibangun menggunakan kerangka kerja Laravel. Pengujian tersebut dilakukan untuk mencari kesamaan luaran antara proses *deployment* yang dilakukan secara manual dengan proses *deployment* yang dilakukan secara otomatis yang memanfaatkan teknologi CI/CD. Hasil dari pengujian menunjukkan proses *deployment* secara otomatis menggunakan CI/CD memiliki luaran yang sama terhadap proses *deployment* secara manual. Pemanfaatan CI/CD juga memberikan luaran yang lebih baik yaitu adanya analisis kode yang dapat menjadi tolok ukur kualitas perangkat lunak.

Kata kunci: ***Continuous Integration, Continuous Deployment, Automated Testing***

DESIGN AND IMPLEMENTATION OF CI/CD TEMPLATE FOR AUTOMATED TESTING AND DEPLOYMENT AUTOMATION

Student Name	: Rendy Ananta
NRP	: 05211840000021
Departement	: Sistem Informasi FTEIC-ITS
Supervisor 1	: Nisfu Asrul Sani, S.Kom, M.Sc.

ABSTRACT

Customer satisfaction has become significant in software engineering nowadays. An agile implementation could cut the time-to-market and gain feedback to achieve customer satisfaction. The organization could improve their products and services from the feedbacks they gain. However, the agile implementation is not enough to accelerate software engineering. Productivity and software quality issues are often happening in the software engineering process. The previous research proves that DevOps practice especially continuous integration and test-driven development (TDD) based automated testing lead to productivity and software quality improvement. In this research, a DevOps design and implementation template related to Continuous Integration (CI) and Continuous Deployment (CD) by utilizing automated testing to automate the deployment process. The template design is consisting the infrastructure and CI/CD configuration that runs over a CI/CD pipeline. The next step is to implement the CI/CD template then test it using Laravel based web application. Developed CI/CD template testing is necessary to know the similarity between manual and automated deployment processes. The testing results is automatic and manual deployment processes will stand for the same output. The

CI/CD implementation will give a better outcome with the static code analysis process as the quality standard of software.

Kata kunci : Continuous Integration, Continuous Deployment, Automated Testing

SURAT PERNYATAAN BEBAS PLAGIARISME

Saya yang bertandatangan di bawah ini:

Nama	: Rendy Ananta
NRP	: 05211840000021
Tempat/ Tanggal Lahir	: Surabaya, 15 Januari 2000
Fakultas/ Departemen	: FTEIC/ Sistem Informasi
Nomor Telp/ HP/e-mail	: 085106593870/ rendyananta66@gmail.com

Dengan ini menyatakan dengan sesungguhnya bahwa penelitian/makalah/tugas akhir saya yang berjudul:
DESAIN DAN IMPLEMENTASI *TEMPLATE* CI/CD UNTUK OTOMATISASI PADA PROSES PENGUJIAN DAN *DEPLOYMENT*

Bebas Dari Plagiarisme Dan Bukan Hasil Karya Orang Lain.

Apabila dikemudian hari ditemukan seluruh atau sebagian penelitian/makalah/tugas akhir tersebut terdapat indikasi plagiarisme, maka saya bersedia menerima sanksi sesuai peraturan dan ketentuan yang berlaku.

Demikian surat pernyataan ini saya buat dengan sesungguhnya dan untuk dipergunakan sebagaimana mestinya.

Surabaya, 8 Februari 2021

Rendy Ananta

NRP. 05211840000021

** Halaman ini sengaja dikosongkan **

KATA PENGANTAR

Puji syukur kehadirat Allah SWT atas rahmat dan karunia-Nya yang senatiasa memberikan kemudahan, kelancaran, dan kekuatan tiada henti hingga penulis dapat menyelesaikan buku tugas akhir dengan judul:

"DESAIN DAN IMPLEMENTASI *TEMPLATE* CI/CD UNTUK OTOMATISASI PADA PROSES PENGUJIAN DAN *DEPLOYMENT*"

Buku tugas akhir ini disusun dan diajukan untuk memenuhi syarat kelulusan guna mendapatkan gelar Sarjana Komputer (S.Kom) pada Institut Teknologi Sepuluh Nopember. Penyelesaian buku tugas akhir ini tidak akan terwujud tanpa bantuan dan dukungan dari berbagai pihak yang telah bersedia meluangkan waktu, pikiran, dan tenaganya bagi penulis. Pada kesempatan ini izinkan penulis mengucapkan syukur dan terima kasih banyak kepada pihak-pihak dibawah ini, yaitu:

- Tuhan Yang Maha Esa, karena atas kuasanya penulis diberikan kekuatan dan kemudahan dalam proses pengerjaan tugas akhir.
- Kedua orang tua penulis yang telah memberikan dukungan baik secara fisik maupun secara moral selama penulis memulai pendidikan tinggi hingga pengerjaan tugas akhir ini.
- Bapak Nisfu Asrul Sani, S.Kom, M.Sc. selaku dosen pembimbing yang telah meluangkan waktu, tenaga, dan pikirannya untuk berdiskusi, memberikan masukan, dan memberikan dukungan terhadap penulis dalam pengerjaan tugas akhir.
- Bapak Dr. Eng. Febriliyan Samopa, S.Kom, M.Kom dan Bapak Bekti Cahyo Hidayanto, S.Si, M.Kom selaku dosen penguij yang telah menguji, memberikan

masukan terhadap penulis untuk memperbaiki tugas akhir.

- Seluruh dosen dan karyawan Jurusan Sistem Informasi ITS yang senatiasa memberikan ilmu, inspirasi, pengalaman, dan bantuan sejak mahasiswa baru hingga saat ini.
- Teman-teman Valois, Sistem Informasi (SI) Angkatan 2018 yang merupakan tempat penulis belajar, berkembang dalam berbagai aspek kehidupan
- Seluruh pihak yang telah membantu penulis dalam pengerjaan tugas akhir ini yang tidak bisa disebutkan satu persatu

Penulis menyadari bahwa masih terdapat kekurangan dalam penyusunan buku tugas akhir ini dari segi materi maupun cara penyajiannya, oleh karena itu penulis mengharapkan kritik dan saran yang membangun untuk kebaikan di masa depan. Semoga buku tugas akhir ini dapat memberikan manfaat bagi para pembaca dan dorongan untuk penelitian selanjutnya.

Surabaya, 15 Januari 2022

Penulis

** Halaman ini sengaja dikosongkan **

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

Bab I pada tugas akhir ini berisi tentang pendahuluan mengenai tugas akhir. Pendahuluan berisi mengenai latar belakang permasalahan yang diangkat. Selain itu, terdapat pengenalan singkat terhadap batasan permasalahan, metode yang digunakan, manfaat, relevansi penelitian, serta luaran tugas Akhir.

1.1 Latar Belakang

Merebaknya digitalisasi memiliki dampak yang cukup besar bagi organisasi profit (perusahaan) maupun non-profit. Implementasi teknologi-teknologi baru tersebut dilakukan dalam rangka meningkatkan kepuasan pelanggan dalam menggunakan layanan teknologinya. Dalam pengembangan layanan berbasis teknologi, pendekatan metodologi *agile* berguna agar proses pengembangan responsif dan adaptif dalam terhadap perubahan [1]. Berdasarkan laporan yang dipublikasikan oleh *State of Agile* banyak perusahaan menerapkan *agile* baik itu merupakan perusahaan teknologi maupun bukan perusahaan teknologi. Dari data yang dikumpulkan, sebanyak 27% dari responden adalah perusahaan teknologi yang merupakan proporsi terbesar penggunaan *agile* di industri [2]. Penerapan *agile* dipercaya dapat meningkatkan kepuasan pengguna secara keseluruhan karena tujuannya adalah melakukan iterasi pengembangan dan pengujian kepada pengguna sehingga organisasi dapat mengevaluasi layanannya secara langsung. Iterasi eksperimen yang dilakukan secara kontinu disertai perbaikan tersebut dapat menghasilkan kepuasan pelanggan yang lebih baik. Beberapa perusahaan

besar yang menerapkan eksperimen jangka pendek dan kontinu seperti Google, Facebook, Amazon, serta Spotify dapat mendapatkan respon yang cepat dari pelanggan sehingga mereka bisa melakukan *deployment* berkali-kali dalam waktu satu hari [3]. Agar dapat membantu efisiensi dalam mempercepat proses agile dalam pengembangan perangkat lunak, terdapat tradisi pengembangan perangkat lunak *DevOps* yang bermanfaat untuk meningkatkan efisiensi dengan otomatisasi pekerjaan operasional seperti *deployment* [4].

Terlalu menghabiskan waktu dalam proses perilisan, tren bisnis baru, teknologi yang sudah usang, kurangnya standar & otomatisasi, dan masalah perilisan lainnya menjadi alasan utama adopsi dari *DevOps* oleh organisasi [5], [6]. Beberapa permasalahan seperti tidak adanya standarisasi yang jelas menyebabkan sulitnya melakukan *governance* terhadap layanan yang dimiliki [6]. Selain itu, absennya otomatisasi pada pengujian dan *deployment* dapat memungkinkan adanya kesalahan jika terjadi penambahan kode maupun fitur baru. Pengujian manual tersebut juga menyebabkan waktu terkuras pada pengujian perangkat lunak. Alih-alih meningkatkan produktivitas pengujian, beberapa pengembang justru melewati beberapa pengujian pada fitur tertentu untuk memangkas waktu pengujian. Hal tersebut memungkinkan adanya kekutu (*bug*) dan kesalahan (*error*) pada sistem yang dikembangkan, karena mungkin ada beberapa fitur lama yang ikut terpengaruh saat pengembangan fitur yang baru.

Beberapa praktik yang lazim digunakan pada penerapan *DevOps* adalah perihal pengujian terotomatisasi seperti *unit testing*, *automated acceptance testing*, *Test-Driven Development* (TDD) dan *Behaviour Driven Development* (BDD) [7]. Pendekatan praktik *DevOps* pada pengujian terotomatisasi dalam pengembangan perangkat lunak dapat

meningkatkan kualitas, produktivitas dan efektivitas dari pengembangan perangkat lunak [8]–[11]. Penggunaan *tools DevOps* lain yang sangat umum adalah *Continuous Integration* (CI)/*Continuous Delivery* (CD) biasanya juga digunakan guna mempercepat proses *delivery* kepada pengguna [12], [13]. Dalam prosesnya, CI/CD dapat mengotomatisasi proses seperti pengujian dan perilisan aplikasi sehingga dapat meringankan pekerjaan pengembang system [14]. Oleh karena itu, dengan menggunakan pengujian terotomatisasi, CI/CD dapat mempercepat proses *agile* dan meningkatkan kualitas perangkat lunak.

Beberapa organisasi teknologi telah banyak yang menerapkan *agile* dalam proses pengembangan perangkat lunak. Penerapan CI/CD dan pengujian terotomatisasi secara umum dapat meningkatkan produktivitas pengembangan perangkat lunak. Meningkatnya produktivitas dan kualitas perangkat lunak juga memiliki pengaruh terhadap kepuasan pelanggann karena iterasi pengembangan dilakukan dengan lebih cepat, sehingga pelanggan dapat memberikan *feedback* juga dengan lebih cepat.

Berdasarkan kondisi industri saat ini, peneliti ingin mengetahui implementasi CI/CD dengan membuat *template* CI/CD dengan pengujian terotomatisasi yang dapat digunakan kembali pada proyek-proyek yang serupa. Adanya *template* tersebut dapat memangkas waktu inisiasi pengembangan perangkat lunak dan *deployment*, karena pengembang hanya perlu fokus untuk mengembangkan aplikasi dan menulis skenario pengujian.

1.2 Perumusan Masalah

Berdasarkan pemaparan latar belakang di atas, berikut adalah rumusan masalah yang akan diselesaikan pada penelitian ini.

1. Bagaimana rancangan *template* CI/CD untuk otomatisasi proses pengujian dan *deployment*?
2. Bagaimana implementasi CI/CD dari rancangan yang telah dibuat?

1.3 Batasan Tugas Akhir

Adapun batasan permasalahan pada tugas akhir ini adalah fokus kepada *template* infrastruktur CI/CD, *template deployment* aplikasi, serta *template pipeline* CI/CD untuk menjalankan proses pengujian dan *deployment* secara otomatis dengan kriteria aplikasi yang didefinisikan pada metodologi.

1.4 Metode Tugas Akhir

Pada tugas akhir ini, dilakukan perancangan *template* CI/CD baik mulai dari infrastruktur, *template deployment* aplikasi hingga *pipeline* CI/CD. *Pipeline* dirancang untuk menjalankan pengujian otomatis dan dilanjutkan dengan *deployment*. Setelah dilakukan perancangan, kemudian dilakukan pengujian pada paling tidak dua aplikasi *web* sebagai target pengujian termplat yang telah dikembangkan sebelumnya. Pengujian dilakukan untuk mengetahui kesamaan luaran dari proses *deployment* secara otomatis dengan menggunakan CI/CD dan proses *deployment* secara manual.

1.5 Tujuan Tugas Akhir

Berdasarkan rumusan masalah yang telah dipaparkan sebelumnya, tujuan dilakukannya tugas akhir ini adalah sebagai berikut.

1. Mendesain rancangan *template* implementasi CI/CD untuk menjalankan proses pengujian untuk *deployment* secara otomatis
2. Implementasi *template* CI/CD untuk menjalankan proses pengujian dan *deployment* secara otomatis.

1.6 Manfaat Tugas Akhir

Manfaat yang diharapkan dari tugas akhir ini meliputi manfaat teoritis dan manfaat praktis sebagai berikut.

1.6.1 Teoritis

1. Mengaplikasikan pengetahuan mengenai CI/CD dengan pengujian terotomatisasi untuk pengembangan perangkat lunak.
2. Menambah referensi untuk penelitian dengan metode sejenis.

1.6.2 Praktis

1. Perancangan *template* infrastruktur dan CI/CD untuk otomatisasi pengujian dan *deployment* dapat membantu organisasi yang hendak mengimplementasikan CI/CD.
2. Mempercepat proses *setup* lingkungan *deployment* CI/CD.
3. Hasil dari *template* yang dibuat, dapat diimplementasikan pada proyek organisasi yang telah sesuai dengan kriteria.

1.7 Relevansi Tugas Akhir

Tugas akhir yang dibuat sejalan dengan ranah penelitian dari laboratorium Infrastuktur Keamanan Teknologi Informasi (IKTI). Mata kuliah yang relevan terhadap penelitian ini adalah Pengembangan dan Operasi Sistem. Otomatisasi dilakukan agar

dapat lebih fokus dalam meningkatkan kualitas sistem dan bisnis yang sedang dikembangkan. Hal tersebut sejalan dengan inti riset dari laboratorium IKTI yang berfokus pada infrastuktur Teknologi Informasi (TI) dan *Software Architecture Design*.

1.8 Luaran Tugas Akhir

Luaran yang diharapkan dari tugas akhir ini adalah *template* CI/CD untuk otomatisasi proses pengujian dan *deployment* sekaligus dokumentasi sebagai media transfer pengetahuan antar pengembang yang mengunakan templat, sehingga memudahkan pengembang yang hendak menggunakan maupun yang melakukan kustomisasi.

** Halaman ini sengaja dikosongkan **

BAB II
TINJAUAN PUSTAKA DAN DASAR TEORI

Bab II pada tugas akhir ini berisi tentang tinjauan pustaka beserta dasar teori yang menjadi fondasi utama dari penelitian.

2.1 Tinjauan Pustaka

Penelitian terdahulu berjudul "*Effectiveness of Test-Driven Development and Continuous Integration*" yang dilakukan oleh Chintan Amrit et.al. yang membahas mengenai penggunaan TDD pada UMKM di Belanda. Penelitian tersebut menggunakan metode perhitungan kuantitatif statistika non parametrik untuk membuktikan efektivitas penggunaan CI dan TDD terhadap *defect level* pada saat pengujian internal dan eksternal. Perbandingan tersebut menggunakan perangkat lunak lama dengan perangkat lunak yang baru dengan karakteristik yang mirip. Keterkaitan penelitian pada tugas akhir adalah metode pemanfaatan pengujian terotomatisasi dengan *Continuous Integration.*

Penelitan berikutnya berjudul "*Why are many businesses instilling a DevOps culture into their organization?*" oleh Jessica D´ıaz et.al. yang membahas permasalahan-permasalahan organisasi yang dapat diselesaikan dengan implementasi *DevOps*. Implementasi kultur DevOps tersebut bertujuan untuk menyelesaikan permasalahan konflik antara pengembang dan pihak operasional. Selain itu, terdapat implementasi CI/CD untuk melakukan pekerjaan seperti menjalankan pengujian dan *deployment* secara otomatis yang dapat meningkatkan produktivitas, efektivitas, dan kualitas perangkat lunak. Keterkaitan penelitian pada tugas akhir ini adalah pada implementasi CI/CD yang dilakukan.

Penelitian berikutnya berjudul "*Combination of Test-Driven Development and Behavior-Driven Development for Improving Backend Testing Performance*" yang ditulis oleh Ida Bagus Kerthyayana Manuaba. Penelitian mengkombinasikan metode dari TDD dengan BDD untuk mengatasi permasalahan pada pengujian menggunakan *unit testing*. Pada penelitian tersebut, *tool* yang digunakan untuk otomatisasi proses pengujian adalah PHPUnit. Penggunaan PHPUnit tersebut digunakan untuk *API testing* untuk memastikan API berjalan normal. Kesamaan penelitian pada tugas akhir ini adalah *tool* yang digunakan yaitu penggunaan PHPUnit.

2.2 Dasar Teori

Berikut adalah konsep dan dasar teori yang menjadi pijakan dalam penelitian tugas akhir ini.

2.2.1 Agile

Agile adalah prinsip pengembangan perangkat lunak yang mengedepankan interaksi individu, perangkat lunak yang berfungsi dengan baik, hubungan serta kolaborasi dengan pengguna, dan tanggap terhadap perubahan dalam proses pengembangan perangkat lunak [15]. Prinsip *agile* tersebut mencerminkan bahwa tujuan utama yang dicapai adalah kepuasan konsumen dengan tim yang solid antar individu.

Pendekatan agile sendiri menekankan untuk mengambil keputusan berdasarkan realitas yang diamati dalam proyek *(empirical control method)* [16]. Pengambilan keputusan yang tepat, dapat berbuah kesuksesan dalam proyek. Karena tak jarang, proyek gagal karena mengambil keputusan yang salah.

Metode *Empirical control* memerlukan transparansi, inspeksi secara berkala, serta adaptasi dengan segera [16]. Realitas proyek dapat diamati dari eksperimen yang dilakukan secara berkala. Adanya pengamatan secara berkala dapat

menjadi data terkait metode yang bekerja dengan baik dan mana metode yang tidak bekerja maksimal. Data tersebut dijadikan evaluasi oleh para pemangku kepentingan secara terbuka dan transparan agar pengembang juga mengetahui keadaan sebenarnya seperti apa. Hasil dari evaluasi tersebut dapat menjadi bahan pertimbangan mana yang perlu dirubah dan dipertahankan.

Salah satu *practice* yang sering dilakukan dengan *agile* adalah implementasi *DevOps*. *Practice DevOps* umumnya diterapkan dengan tujuan untuk mempercepat proses iterasi karena bertujuan untuk meningkatkan efisiensi dan otomatisasi pekerjaan operasional [4]. Adanya percepatan iterasi dapat memudahkan pemangku kepentingan untuk melakukan perilisan aplikasi dan mendapatkan umpan balik dari pengguna semakin cepat. Secara tidak langsung, hal tersebut berpengaruh pada hasil akhir produk yang sesuai dengan kebutuhan pengguna.

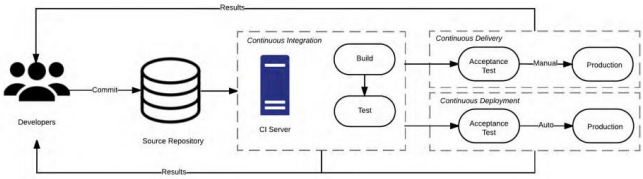
2.2.2 DevOps

DevOps adalah transformasi kultur kerja yang tujuannya adalah untuk mengatasi konflik antar pengembang dengan tim operasional dalam proses *delivery* perangkat lunak kepada pengguna [17]. Kebutuhan akan proses pengembangan perangkat lunak yang membutuhkan semua orang berkolaborasi dalam mempercepat waktu perilisan secara reliabel terdapat kultur bernama *DevOps* [18]. Karena *DevOps* sendiri merupakan kultur, maka tidak ada peraturan yang ketat terhadap penerapannya, sehingga setiap organisasi dapat menerapkan sesuai dengan keadaan dan situasinya secara spesifik.

a. CI/CD

Continuous Integration (CI) adalah praktik pengembangan perangkat lunak yang untuk mengintegrasikan pekerjaan dari para pengembang dengan tujuan untuk mengurangi konflik perubahan kode sumber *(source code)* dan mendeteksi kesalahan pada perangkat lunak sedini mungkin [14]. Penggunaan CI dalam pengembangan perangkat lunak adalah untuk memperpendek frekuensi untuk proses pengintegrasian pekerjaan dari beberapa pengembang sehingga jumlah konflik dapat dikurangi. Selain itu, CI juga berguna untuk meningkatkan kualitas dari perangkat lunak, hingga meningkatkan produktivitas tim [1]. Karena integrasi yang begitu sering, perlu adanya otomatisasi untuk mempermudah pengecekan kesalahan *(error)* pada perangkat lunak. Praktik yang juga digunakan dalam CI adalah pengujian terotomatisasi (*automated tests*) agar setiap perubahan pada *repository* VCS kemudian menjalankan CI yang menjalankan pengujian otomatis.

Sedangkan *Continuous Delivery* (CD) adalah disiplin pengembangan perangkat lunak sehingga perangkat lunak tersebut dapat dirilis ke *production* secara otomatis [18], [19]. untuk memastikan bahwa aplikasi layak rilis untuk dapat digunakan oleh pengguna. Proses CI dan CD terintegrasi, maka bisa diartikan bahwa CD memiliki ketergantungan oleh CI.



Gambar 2. 1 Perbedaan CI dan CD [20]–[22]

Berdasarkan ilustrasi Gambar 2. 1, dapat diketahui perbedaan berbagai macam praktik yang ada pada *DevOps*,

yaitu antara CI, *Continuous Delivery* dengan *Continuous Deployment*. Setelah dilakukannya pembaruan kode pada sumber repositori, maka CI Server akan menjalankan CI *pipeline* kemudian dapat diteruskan *Continuous Delivery* atau *Continuous Deployment* yang mana memiliki karakteristik yang hamper sama. Perbedaannya terletak pada cara melakukan *deployment* pada *production*, pada *Continuous Delivery* dilakukan secara manual dan pada *Continuous Deployment* dilakukan secara otomatis.

2.2.3 Pengujian Terotomatisasi

Pengujian terotomatisasi adalah paradigma pengembangan perangkat lunak dengan cara mengotomatisasi pengujian perangkat lunak. Berikut adalah pengaruh yang dapat terjadi pada penerapan pengujian perangkat lunak secara otomatis:

- Meningkatkan upaya untuk fokus pada pengujian terotomatisasi
- Kemampuan analisis penguji perangkat lunak manual tetap dibutuhkan sebagai landasan pembuatan pengujian terotomatisasi
- Pengujian perangkat lunak secara manual dan secara otomatis tidak dapat dipisahkan karena keduanya saling melengkapi satu sama lain.

Pada pengujian terotomatisasi, terdapat proses lain terkena implikasi yaitu proses pengembangan perangkat lunak. Proses pengembangan ini salah satu contohnya adalah *test-driven development* [23].

2.2.4 Template

Menurut KBBI, definisi templat (*template*) adalah format (ukuran, pola, dan sebagainya) yang diacu dalam pembuatan sesuatu [24]. Pada konteks tugas akhir ini, maka *template* adalah format yang diacu spesifik pada otomatisasi proses *deployment* menggunakan CI/CD.

2.2.5 Container

Container adalah unit standar perangkat lunak yang mengemas *source code* beserta *dependencies* dalam sebuah *image* agar aplikasi dapat dijalankan cepat dan reliabel pada berbagai macam kondisi lingkungan [25]. Hal tersebut dapat memudahkan pengembang perangkat lunak untuk menjalankan aplikasi yang telah dikembangkannya secara langsung tanpa perlu mengkhawatirkan perbedaan lingkungan dapat mempengaruhi aplikasi yang telah dikembangkan. Salah satu *container* yang sering digunakan pada industry adalah docker. Pada *Container*, terdapat komponen bernama containerd yang merupakan standar *format* dan *runtime* yang dibuat oleh Open Container Initiative (juga didirikan oleh docker) untuk menstandarisasi adopsi *container* yang kian banyak bermunculan [26].

2.2.6 Kubernetes

Google menginisiasi proyek *open-source* Kubernetes pada tahun 2014 karena mengalami permasalahan dalam mengatasi beban kerja pada skala tertentu. Kubernetes adalah sebuah platform *open-source* yang berguna untuk manajemen layanan berupa container secara deklaratif dan mudah diotomatisasi [27]. Hadirnya Kubernetes sendiri dapat memudahkan tim operasional dalam memanajemen skala dari sebuah proyek menggunakan *container* sesuai dengan beban kerja yang

dibutuhkan. Contoh *usecase* Kubernetes sendiri adalah proses *deployment* pada lingkungan *production*. Proses tersebut terdiri dari mematikan *container* yang telah lama ada, kemudian membuat *container* yang baru. Transisi *container* tersebut menyebabkan adanya *downtime* [27]. Kubernetes sendiri dapat bertugas untuk memanajemen proses tersebut agar lama *downtime* dari proses *deployment* bisa diminimalisir.

Beberapa *resource manifest* (*file*) dasar yang digunakan dalam tugas akhir ini adalah sebagai berikut.

- *Namespace* adalah *space* atau ruang untuk mengelompokkan berbagai macam *resource* yang ada pada Kubernetes.
- *Secret* adalah konfigurasi yang berisi *credentials* dan value yang *sensitif* dalam *deployment* aplikasi
- *Configmap* adalah konfigurasi aplikasi yang dapat digunakan oleh *container*.
- *PersistenceVolumeClajm* adalah resource untuk mengklaim penyimpanan
- *Service Account* adalah akun yang digunakan untuk mengidentifikasi *user* dari Kubernetes *cluster*.
- *Pod* adalah kumpulan dari beberapa container yang berada pada satu network yang sama, yaitu localhost.
- *Deployment* adalah manajemen pod yang memastikan bahwa *pod* ini dalam keadaan sehat dan sesuai dengan jumlah replika yang diinginkan
- Job adalah sebuah tugas yang dijalankan sekali.
- CronJob adalah sebuah tugas yang dijalankan secara terus menerus, tergantung pada kofigurasi interval yang digunakan
- *Service* adalah resource yang berguna untuk mengekspos deployment agar dapat diakses oleh *service* lain.

- Ingress adalah *resource* yang berguna untuk meneruskan *service* agar dapat diakses di internet secara publik.

2.2.7 Jenkins

Jenkins adalah *server* otomatisasi *open-source* yang dapat membantu proses otomatisasi pada pengembangan perangkat lunak. Proses tersebut meliputi *building*, pengujian, dan *deployment*. Jenkins sendiri saat ini adalah salah satu *tools* yang paling sering digunakan untuk CI/CD karena penggunaannya yang mudah dan tersedianya berbagai macam *plugins* untuk keperluan integrasi pada berbagai macam *tools* yang lain [28]. Penggunaan *tools* Jenkins sendiri cocok untuk diimplementasikan mandiri karena sifatnya yang *open-sourced*, sehingga organisasi dapat mencobanya secara gratis pada *platform (server)* yang dimiliki.

Pada *deployment* Jenkins pada Kubernetes, ada dua komponen yaitu Jenkins *master* yang berfungsi sebagai *controller* dan *spawn* Jenkins *agent* untuk menjalankan sebuah Jenkins job. Jenkins *job* sendiri adalah tugas untuk menjalankan sebuah *pipeline* CI/CD.

2.2.8 SonarQube

SonarQube adalah sebuah proyek yang diinisiasi oleh beberapa pengembang di Switzerland untuk analisis kode yang berfokus pada kualitas dan keamanan kode. SonarQube dapat diintegrasikan pada berbagai macam tools CI/CD, salah satunya adalah Jenkins. Selain itu, SonarQube juga mendukung berbagai macam bahasa pemgrograman untuk analisis kode yang dilakukan [29]. Penggunaan SonarQube pada salah satu *step* integrasi pada CI/CD dapat memberikan umpan balik bagian dari kode mana yang memiliki isu terhadap kualitas dan

keamanan. Akibatnya, pengembang memiliki referensi bagian mana kode yang memiliki masalah, dan akan memperbaikinya secara kontinu. Perbaikan tersebut dapat menghasilkan kualitas perangkat lunak yang baik secara keseluruhan jika konsisten dilakukan.

2.2.9 PHPUnit

PHPUnit adalah sebuah kerangka kerja untuk melakukan *unit testing* pada bahasa pemrograman PHP [30]. Unit testing di sini adalah *practice* dari pengembangan perangkat lunak dengan melakukan pengujian terotomatisasi untuk menguji komponen dari sebuah perangkat lunak meliputi kontrol data, kontrol *procedure*, maupun penggunaan *procedure* [31]. PHPUnit sendiri sering dapat digunakan pada pengembangan perangkat lunak dengan pengujian otomatis menggunakan metode TDD maupun T-BDD [11]. Hal tersebut menujukkan bahwa penggunaan PHPUnit sangat luas pada proses pengembangan perangkat lunak.

2.2.10 Laravel

Laravel adalah kerangka kerja *full-stack* untuk membangun website dengan menggunakan bahasa pemrograman PHP. *Full-stack* yang dimaksud adalah Laravel menyediakan abstraksi untuk berbagai macam fungsi standar baik itu *routing*, *testing* cara berinteraksi pada basis data, hingga rendering aplikasi *front-end* [32]. Hal tersebut yang menjadikan Laravel sebagai kerangka kerja pembuatan *website* paling popular saat ini. Kerangka kerja yang kaya fitur dan adanya *scaffolding* pada beberapa fungsi standar yang sering dipakai sangat memudahkan pengembangan perangkat lunak, seperti *queue*, *scheduler* dan *migrations*. Bahkan, laravel juga menyediakan lingkungan pengembangan perangkat lunak dengan

berbasiskan menggunakan teknologi *container* bernama Laravel Sail, sehingga pengembangan perangkat lunak menggunakan Laravel dapat menghemat waktu dan tenaga.

2.2.11 GnuPG

GnuPG adalah GNU Privacy Guard, yaitu salah satu implementasi openPGP. OpenPGP sendiri adalah perangkat lunak keamanan yang menggunakan kriptografi *Pretty Good Privacy* (PGP) *software family* sebagai basisnya [33]. Penggunaan GnuPG pada tugas akhir ini adalah untuk menghasilkan *private key* yang digunakan sebagai basis enkripsi dan dekripsi konten sensitif.

2.2.12 Mozilla SOPS

Mozilla *Secret OPerationS* atau SOPS adalah perangkat lunak yang dikembangkan oleh Mozilla untuk melakukan enkripsi konten dari sebuah dokumen seperti json, yaml, env, ini, dan binary dengan berbagai macam dukungan terhadap *key store* seperti *AWS Key Management Service* (KMS), GCP KMS, *Azure Key Vault*, dan PGP [34]. Penggunaan SOPS pada tugas akhir ini adalah untuk mengenkripsi konten pada Kubernetes *manifest* yang sifatnya sensitif seperti *manifest Secret*.

2.2.13 Redis

Redis adalah *in-memory datastore* yang dapat digunakan sebagai *cache* maupun *message broker* [35]. Pada tugas akhir ini redis digunakan sebagai *session* dan *queue*, yang dapat diaktifkan pada kerangka kerja Laravel dengan mengganti nilai dari *session driver* dan *queue driver*.

2.2.14 Nginx

Nginx adalah sebuah aplikasi yang umum digunakan untuk HTTP (*Hypertext Transfer Protocol*) *server*, *reverse proxy server*, *mail proxy server*, dan *generic* TCP (*Transmission Control Protocol*)/UDP (*User Datagram Protocol*) *proxy server* [36]. Pada tugas akhir ini, nginx digunakan sebagai HTTP server untuk meneruskan *request* ke aplikasi.

2.2.15 Supervisor

Supervisor adalah process control system yang memanajemen dan memonitor proses pada *unix-like operating systems* [37]. Supervisor digunakan pada tugas akhir ini untuk mengontrol proses *queue worker* pada aplikasi.

2.2.16 Cron

Cron adalah program yang digunakan untuk eksekusi sebuah task secara berkala dalam periode waktu tertentu yang telah didefinisikan pada crontab (*cron table*) [38]. Penggunaan cron dalam tugas akhir ini adalah untuk menjalankan *scheduler* pada aplikasi.

2.2.17 Certbot

Certbot adalah program yang dikembangkan untuk otomatisasi akuisisi *certificate* dari *Let's Encrypt* agar sebuah website dapat diakses menggunakan protokol HTTPS (*Hypertext Transfer Protocol Secure*) [39]. Penggunaan certbot dapat memudahkan proses konfigurasi *server* untuk mengaktifkan HTTPS.

2.2.18 Amazon S3

Amazon S3 atau yang juga dikenal dengan *Amazon Web Service Simple Storage Service* (AWS S3) adalah layanan penyimpanan objek yang ada pada internet secara mudah [40]. Penggunaan AWS S3 tersebut dapat memudahkan pengembang perangkat lunak dalam hal penyimpanan media tanpa perlu memikirkan ketahanan, skalabilitas, kinerja, dan ketersediaannya.

** Halaman ini sengaja dikosongkan **

BAB III
METODOLOGI

Bab III pada tugas akhir ini berisi tentang uraian metodologi yang digunakan.

3.1 Uraian Metodologi

Metodologi pada sub bab ini berisi tentang penjelasan studi kasus yang digunakan dalam tugas akhir beserta diagram alir pengerjaan tugas akhir.

3.1.1 Studi Kasus

Pada tugas akhir ini, diperlukan dua atau lebih aplikasi yang digunakan sebagai uji validitas dari rancangan infrastruktur dan *template* CI/CD yang dikembangkan. Berikut adalah kriteria aplikasi yang dapat digunakan sebagai pengujian *template*.

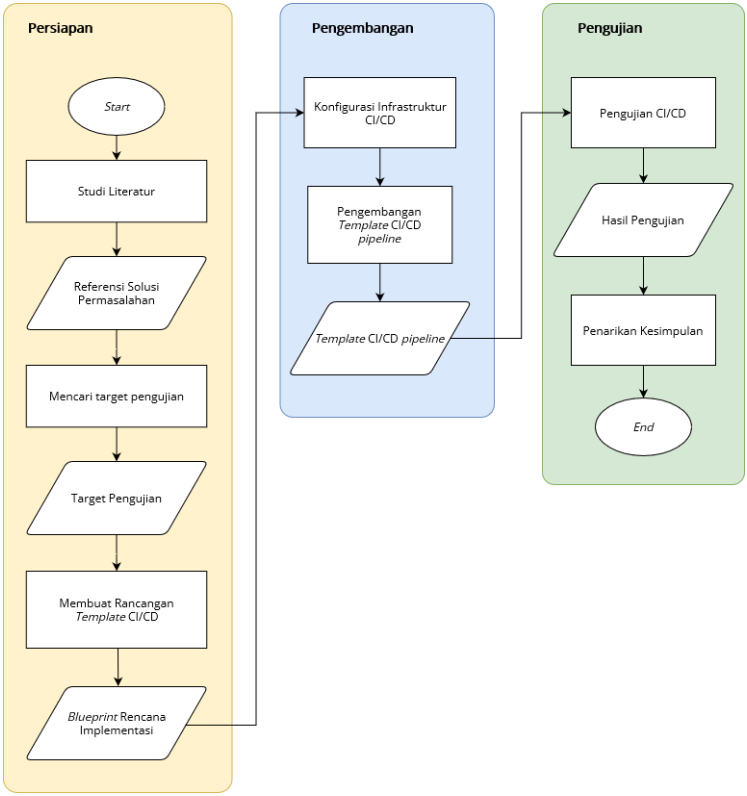
1. Aplikasi menggunakan repositori git sebagai VCS.
2. Aplikasi berbasiskan *web* yang dibangun menggunakan bahasa pemrograman PHP minimal versi 7.4 hingga 8.1 dan memanfaatkan kerangka kerja Laravel.
3. Aplikasi menggunakan arsitektur monolitik dan bukan *microservice* (menggunakan teknologi *websocket*).
4. Jumlah endpoint atau halaman aplikasi tidak lebih dari 100 (skala kecil hingga menengah).
5. Aplikasi tidak melakukan pemrosesan data berjumlah besar seperti *machine learning*, maupun *artificial intelligence*.
6. Aplikasi telah mengimplementasikan *unit testing* menggunakan PHPUnit
7. Aplikasi harus *stateful*, yaitu memanfaatkan basis data. Jika di dalam aplikasi terdapat proses *create, read,*

update, dan *delete* (CRUD) atau proses transaksi, maka dapat dipastikan bahwa aplikasi tersebut adalah aplikasi yang *stateful*.

8. Basis data yang digunakan aplikasi adalah MySQL, dan PostgreSQL.
9. Jika terdapat integrasi layanan dari luar aplikasi seperti penggunaan *payment gateway*, Google Maps API, atau layanan lainnya, maka harus tersedia respon palsu *(mock)* untuk layanan tersebut.
10. Aplikasi perlu menggunakan Laravel *storage* API untuk berpindah mode penyimpanan tanpa adanya perubahan kode, jika aplikasi memiliki proses terkait penyimpanan.

3.1.2 Diagram Alir

Alur pengerjaan tugas akhir dapat ditunjukkan dengan diagram alir berikut.



Gambar 3. 1 Diagram alir pengerjaan tugas akhir

Berdasarkan Gambar 3. 1, tahapan pengerjaan tugas akhir dibagi menjadi 3 tahapan sebagai berikut.

a. Persiapan

Pada tahap ini, terdapat 3 proses yaitu sebagai berikut

i. Studi Literatur

Pertama kali yang dilakukan pada tahap persiapan adalah studi literatur untuk menambah referensi dan pemahaman dari permasalahan serupa yang dihadapi. Luaran yang diharapkan dari proses ini adalah

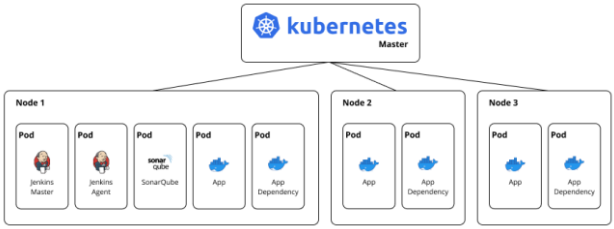
pengetahuan yang berasal dari rangkuman studi yang telah disusun.

ii. Mencari target uji coba

Pencarian target dilakukan pada repositori git publik atau pada aplikasi privat yang telah mendapatkan izin oleh pihak terkait. Aplikasi tersebut akan dijadikan objek uji coba pada *template* CI/CD yang dikembangkan.

iii. Membangun Rancangan *Template*

Referensi yang didapatkan dari proses pertama digunakan sebagai bahan pertimbangan dalam penyusunan *template* yang akan diimplementasikan. Sebelum dilakukan pembuatan templat, terlebih daulu dilakukan konfigurasi infrastruktur yang digunakan sebagai sarana implementasi CI/CD. Berikut adalah gambaran arsitektur infrastruktur yang digunakan.

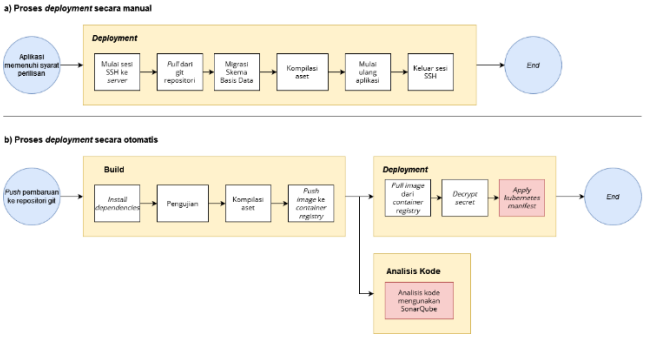


Gambar 3. 2 Rancangan Infrastuktur CI/CD

Penggunaan Kubernetes pada Gambar 3. 2 dimaksudkan untuk memanajemen *container* yang akan dilakukan *deployment* seperti Jenkins, SonarQube, dan aplikasi yang akan diujicobakan. Aplikasi yang diujicobakan juga memiliki ketergantungan (*dependency*) terhadap aplikasi lain

seperti basis data, agar aplikasi tersebut dapat berjalan dengan baik. Pada *deployment* secara manual, infrastruktur yang digunakan hanya *virtual machine* (VM) atau *baremetal machine* yang dipasang aplikasi dan *dependency* yang dibutuhkan. Perbedaan mendasar pada infrastruktur yang digunakan untuk *deployment* secara otomatis yang dibahas pada tugas akhir ini ialah pada pemanfaatan Kubernetes untuk populasi dan manajemen *container* sejumlah dengan apa yang dibutuhkan.

Pada proses *deployment* secara manual, tugas-tugas yang dilakukan cenderung pendek daripada apa yang dilakukan pada *deployment* secara otomatis. Diawali dengan melakukan SSH pada VM atau *baremetal machine* kemudian melakukan pull dari repositori git. Selanjutnya dapat dilakukan populasi data *dummy* (diperlukan bila deployment ditujukan pada lingkungan *staging*), kemudian kompilasi asset hingga keluar sesi SSH. Proses tersebut dapat dilihat pada Gambar 3. 3 Deployment secara manual dan otomatis bagian a berikut.



Gambar 3. 3 Deployment secara manual dan otomatis

Pada Gambar 3. 3 bagian b, dapat diketahui bahwa CI/CD dijalankan setelah adanya *triggering event* yaitu *push* pembaruan kode ke repositori git. Setelah itu, terdapat 3 *stage* yaitu pengujian, *deployment* dan analisis kode. Stage *deployment* dan analisis kode akan dijalankan secara paralel ketika *job* pada *stage* pengujian telah sukses dilakukan. Rangkaian tugas tersebut dilakukan secara otomatis sehingga dapat mempermudah proses *deployment* secara umum.

b. Pengembangan

Pada tahap ini, dilakukan implementasi seperti yang telah dipaparkan pada tahap sebelumnya. Implementasi yang dilakukan yang pertama adalah konfigurasi infrastruktur kemudian dilanjutkan dengan pengembangan *template pipeline* CI/CD.

c. Pengujian

Setelah dilakukan pengembangan *template* CI/CD, maka selanjutnya dapat dilakukan pengujian terhadap target aplikasi yang telah didapatkan pada tahap persiapan sebelumnya. Pengujian dilakukan dua kali pada masing-masing aplikasi untuk mengetahui kesamaan luaran pada proses *deployment* yang dilakukan secara manual dengan proses *deployment* secara otomatis.

** Halaman ini sengaja dikosongkan **

BAB IV
PERANCANGAN

Pada bab ini, dijelaskan mengenai perancangan terkait tugas akhir seperti perancangan infrastruktur penunjang CI/CD dan perancangan *template* CI/CD pada aplikasi.

4.1 Perancangan *Template* Infrastruktur CI/CD

Sebelum dilakukan percobaan terhadap CI/CD pada aplikasi, terlebih dahulu dipersiapkan infrastruktur. Infrastruktur di sini berguna sebagai tempat menjalankan CI/CD aplikasi sehingga otomasi dapat dilakukan. Pada perancangan *template* infrastruktur, hal ini dibagi menjadi 3 kategori seperti berikut.

4.1.1 Kubernetes Engine

Pada proses deployment, diperlukan sebuah Kubernetes *engine* sebagai target *deployment* baik aplikasi yang berguna sebagai infrastruktur pendukung CI/CD maupun aplikasi yang oleh pengguna akhir. Namun, pada praktiknya terdapat banyak sekali pilihan untuk memilih Kubernetes *engine* yang dapat dipakai. Beberapa Kubernetes *engine* yang disediakan oleh *cloud platform* sudah sangat luas dan bervariasi.

Platform Kubernetes yang umum digunakan pada *enterprise* adalah Elastic Kubernetes Service (EKS) milik Amazon Web Service (AWS), dan Google Kubernetes Engine milik Google Cloud Platform (GCP). Selain itu, terdapat juga beberapa perusahaan yang lebih memilih melakukan inisiasi Kubernetes secara *on premise* menggunakan K3s, ataupun Rancher Kubernetes Engine (RKE) milik rancher. Pilihan layanan tersebut memiliki keunggulan dan kelemahannya

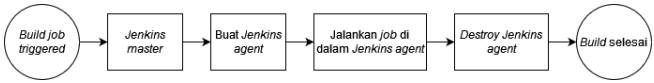
masing-masing, sehingga penggunaan layanan kembali lagi kepada keperluan dan preferensi pengguna. Namun, pada tugas akhir ini digunakan distribusi GCP, yaitu GKE karena tidak perlu memikirkan hal-hal terkait *setup cluster* secara manual.

4.1.2 Operasional CI/CD

Tempat di mana CI/CD berlangsung, dikelompokkan menjadi satu, yaitu *ops*. Beberapa aplikasi yang termasuk di sini adalah Jenkins dan Sonarqube. Hal tersebut dikarenakan kedua aplikasi tersebut yang berperan banyak di bidang *ops*. Kedua aplikasi tersebut nantinya akan dipasang pada namespace *ops*, untuk memisahkan mana *tools* pendukung dan aplikasi utama yang di*deploy*.

A. Jenkins

Implementasi Jenkins pada Kubernetes, merupakan hal yang cukup baru. Fitur dari Kubernetes sendiri sudah dapat digunakan pada Jenkins. Fitur tersebut adalah ketika sebuah *job* berjalan, maka Kubernetes akan melakukan *scheduling* untuk membuat dan menjalankan *container* untuk menjalankan job tersebut. Kemudian, setelah job tersebut selesai, maka *container* yang tadi telah dijalankan akan dihentikan kembali seperti yang tampak pada Gambar 4. 1 berikut.



Gambar 4. 1 Alur Jenkins ketika menjalankan build job

Pemanfaatan Kubernetes untuk memanajemen *container* memungkinkan untuk menghemat *resource* dari sistem.

Berkat adanya fitur *configuration-as-a-code* dari Jenkins, dapat memungkinkan penulis untuk melakukan konfigurasi awal Jenkins *master* dan *container* pada Jenkins *agent* yang menyesuaikan pada *usecase* pada tugas akhir ini. Selain konfigurasi Jenkins, konfigurasi lainnya juga meliputi besarnya penyimpanan yang dibutuhkan, resource maksimal yang dapat digunakan, *domain* dari Jenkins jika diakses dari luar, serta pembaruan SSL secara otomatis pada domain tersebut.

B. SonarQube

Ada beberapa macam komponen dan *dependency* yang perlu dijalankan terlebih dahulu untuk dapat menjalankan SonarQube. Agar dapat memudahkan instalasi, penulis melakukan konfigurasi sehingga infrastruktur dapat dengan mudah dipasang. Secara alur, tidak ada yang berbeda dari SonarQube yang dipasang pada mesin *baremetal*, *virtual machine*, maupun menggunakan Kubernetes.

Konfigurasi pada SonarQube ini meliputi koneksi *database* yang digunakan, besaran penyimpanan, domain akses SonarQube jika diakses dari luar, dan juga pembaruan SSL secara otomatis pada domain tersebut.

4.1.3 *Dependency* Aplikasi

Agar aplikasi dapat berjalan dengan baik, maka diperlukan beberapa aplikasi lain yang *dependant*. Pada *template* ini, beberapa *dependency* yang digunakan secara garis besar sama terhadap aplikasi yang dibutuhkan untuk menjalankan Laravel. Berikut adalah *dependency* yang dapat digunakan pada *template*.

A. Database

Pada aplikasi *stateful*, pasti terdapat dependency terhadap *database*. Aplikasi yang dikembangkan dengan Laravel, basis data yang didukung adalah MySQL, PostgreSQL, dan Microsoft SQL Server. Pada *template* ini, tidak dilakukan percobaan pada Microsoft SQL Server.

Support terhadap MySQL dan PostgreSQL pada Laravel juga meliputi model *master-slave replication* yang menggunakan *master* sebagai terget *write* pada database dan *slave* sebagai *readonly* yang sudah didukung oleh Laravel. Hal tersebut serupa dengan dukungan *multi database connections* yang juga telah didukung oleh Laravel, sehingga kedua fitur tersebut juga dapat digunakan pada *template*.

B. Web Server

Pada *template* yang dikembangkan, *web server* yang digunakan adalah nginx. Pada *template* ini, *web server* tidak dapat digantikan karena kebutuhannya atas *static file asset* dapat terpenuhi. Selain itu dukungan komunitas *open-source* terhadap nginx juga cukup luas sehingga dapat penggunaan web server nginx dirasa sudah cukup.

C. Media Penyimpanan

Jika *load* aplikasi lebih besar pada pembacaan *static file*, maka penggunaan *web server* nginx beserta PHP-FPM pada satu buah *container* dapat terjadi kemungkinan *bottleneck* pada skalabilitas aplikasi. Agar dapat mengatasi masalah tersebut, disarankan untuk menggunakan media penyimpanan luar seperti AWS S3.

Penggunaan media penyimpanan luar dapat membagi load aplikasi secara lebih merata. Selain itu, pada aplikasi yang menganut *deployment* secara

distributed, penyimpanan media secara lokal juga dapat menyebabkan aplikasi mengalami *error*.

Error tersebut dikarenakan *state* suatu *file* pada *container* pertama dan *container* replika tidak sama. Sehingga jika suatu *request* untuk menyimpan *file* pada pada *container* pertama, maka file yang disimpan tersebut hanya terjadi pada media penyimpanan *container* pertama dan file tersebut tidak ada pada media penyimpanan *container* kedua. Hal tersebut menjadi dasar penggunaan media penyimpanan eksternal adalah solusi yang paling optimal.

D. Cache dan *session*

Pada *cache*, Laravel memiliki berbagai macam *cache driver* yang dapat diganti dengan mudah melalui konfigurasi. *Cache driver* yang didukung Laravel adalah *file*, *redis*, dan *memcached*. Seperti pada media penyimpanan, pada lingkungan *deployment* yang *distributed*, penggunaan media penyimpanan *file* dapat menyebabkan terjadinya *error* pada aplikasi.

Pada lingkungan *deployment* secara *distributed*, terdapat permasalahan serupa yaitu mengenai *session*. Namun hal tersebut dapat diatasi dengan menggunakan media eksternal untuk menyimpan *session*. Namun, pada *session* Laravel tidak mendukung penggunaan *memcached* dan hanya mendukung redis. Pertimbangan tersebut menjadi dasar digunakannya redis pada *template* sebagai media penyimpanan *cache* dan *session.*

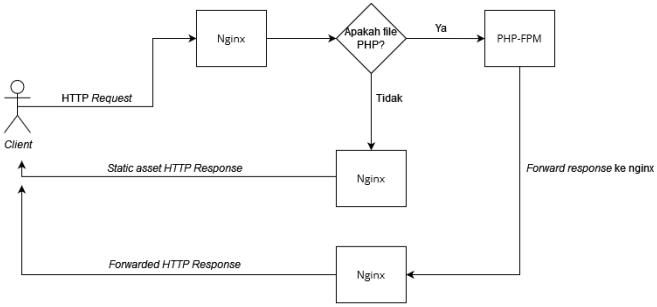
4.2 Perancangan *Template* CI/CD Aplikasi

Pada tahap perancangan *template* aplikasi, perlu diketahui bahwa aplikasi memerlukan beberapa komponen yang terpisah.

Berikut adalah komponen *template* yang akan dibuat pada aplikasi.

4.2.1 Container

Container berfungsi sebagai layer di mana aplikasi berjalan. Pada kasus kali ini, diperlukan sebuah *container* yang di dalamnya terdapat nginx dan PHP-FPM. Nginx berfungsi sebagai *gateway HTTP request* dan mengarahkan ke *static file asset* ketika yang diminta adalah *static asset*. Kemudian, jika yang diminta adalah file berbentuk PHP, maka akan diteruskan ke PHP-FPM. Skema tersebut dapat dilihat pada Gambar 4. 2 sebagai berikut.

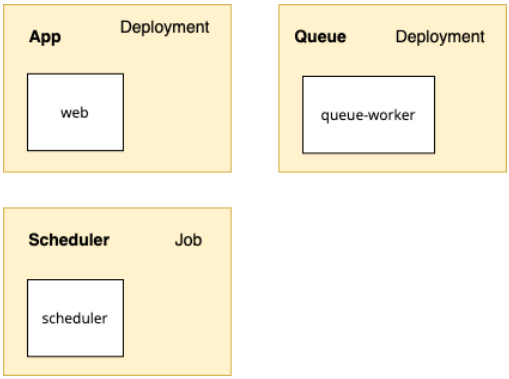


Gambar 4. 2 Alur HTTP request pada container

Berdasarkan skema tersebut, maka diperlukan pembuatan sebuah *dockerfile* (untuk menghasilkan *container image*). Penggabungan dua buah *service* (nginx dan PHP-FPM) dalam satu *container* ini dilandasi dengan ketergantungan Laravel dalam melakukan *rendering* halaman yang membutuhkan *static asset*, sehingga diperlukan untuk kedua buah service tersebut dapat mengakses *file* yang sama dengan cara menjalankan dua buah service pada sebuah *container*.

4.2.2 Kubernetes Resource

Agar aplikasi dapat dideploy pada Kubernetes *cluster*, pasti diperlukan adanya Kubernetes *resource* yang akan dibuat. Berikut adalah gambaran skema Kubernetes *deployment resource* yang akan dibuat.

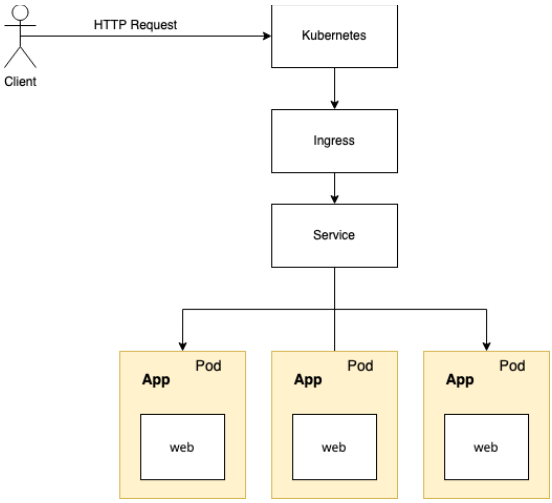


Gambar 4. 3 Kubernetes deployment yang digunakan

Berdasarkan Gambar 4. 3, dapat diketahui bahwa ada 3 komponen utama *deployment*, ketiganya memiliki *image* yang hanya memiliki perbedaan nama. Terdapat dua jenis *deployment* yang digunakan, yaitu Deployment dan Job. Kedua jenis Kubernetes tersebut memiliki tujuan yang berbeda, mengingat *behaviour deployment* lebih ke arah menggunakan *command* untuk menjalankan *long running process*. Sedangkan, untuk jenis *deployment* Job, maka *command* akan dijalankan pada interval tertentu saja.

Selain deployment, terdapat *resource* lain yang dibutuhkan yaitu ConfigMap dan Secret untuk injeksi konfigurasi aplikasi di dalam *container*. Selain itu, aplikasi juga memerlukan

adanya penyimpanan sebagai tempat penyimpanan *cache* dari *framework*.



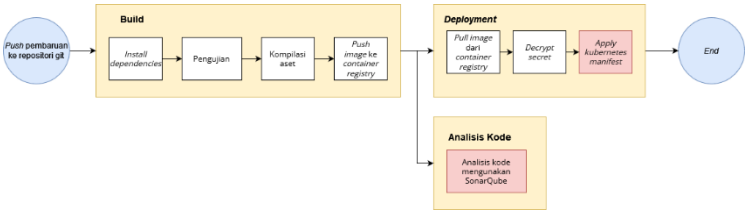
Gambar 4. 4 Penggunaan Kubernetes pada deployment

Dapat dilihat pada Gambar 4. 4, sebuah HTTP request berasal dari Kubernetes cluster, kemudian mengarahkannya ke *Ingress controller*, kemudian ke *service*. Pada *service* inilah, terdapat peran *kube-proxy* untuk *load-balancing request* tersebut kepada pada pod yang dituju.

Saat ini, penggunaan HTTPS menjadi sangat penting untuk mengamankan *request* antara *client* dengan *server* melalui enkripsi. Adanya Kubernetes *scheduler* dimanfaatkan untuk memanajemen TLS *certificate* secara otomatis dan memperpanjangnya sebelum masa dari *certificate* tersebut kadaluarsa. Pada tugas akhir ini, *issuer* yang digunakan pada *template* adalah dari Let's Encrypt.

4.2.3 Pipeline CI/CD

Konfigurasi *Pipeline* yang digunakan sebagai *template* adalah sebuah *Jenkinsfile* pada *root directory* sebuah proyek. Alur *pipeline* yang akan diimplementasikan adalah sebagai berikut.



Gambar 4. 5 Alur pipeline CI/CD

Berdasarkan gambar Gambar 4. 5, *pipeline* CI/CD dimulai dengan proses *build* yang terdiri atas instalasi *dependencies* aplikasi, kemudian menjalankan pengujian otomatis. Selanjutnya dilanjutkan dengan kompilasi *asset*.

Pada *stage* selanjutnya, dilakukan dua buah task secara paralel yaitu *stage* analisis kode dan juga *deployment*. Hal ini dilakukan untuk mencegah *blocking* yang terjadi pada proses *deployment*, karena yang paling penting di sini adalah aplikasi telah lolos pada tahap pengujian, sehingga dapat dilakukan proses *deployment*. Sehingga analisis kode dapat dijadikan sebagai pengetahuan tambahan terkait *source code* bagi pemangku kepentingan terkait.

BAB V
IMPLEMENTASI

Pada bab ini berisikan penjelasan mengenai implementasi dari perancangan penelitian yang telah dijelaskan pada bab sebelumnya.

5.1 Lingkungan Percobaan

Pada tugas akhir ini, lingkungan percobaan yang digunakan ada dua, yaitu Kubernetes *cluster* yang digunakan sebagai *deployment* secara otomatis dan sebuah *instance virtual machine* yang digunakan sebagai percobaan *deployment* secara manual. Spesifikasi *virtual machine* yang digunakan adalah sebagai berikut.

- Total *virtual* CPU 2 *core*,
- Total memori 8 GB,
- Jumlah 1 *instance*,
- Penyimpanan sebesar 20GB, dan
- Sistem Operasi Ubuntu 20.04 LTS

Sedangkan untuk Kubernetes cluster yang digunakan adalah layanan dari Google Cloud Platform yaitu Google Kubernetes Engine dengan menggunakan total *Compute Instance* sebagai berikut.

- Total *virtual* CPU 6 *core*,
- Total memori 24GB,
- Jumlah 3 *node*,
- Lokasi *cluster* di *asia-southeast-1a*, dan
- Versi Kubernetes v1.21.5-gke.1802

5.2 Objek Percobaan

Pada pembuatan templat, digunakan dua buah aplikasi sebagai indikator apakah *template* dapat menjalankan CI/CD dan *deployment* aplikasi secara normal. Objek yang digunakan berasal dari proyek menggunakan kerangka kerja Laravel dan tersedi di repositori publik Github, yaitu UrlHub dan Crater. Kedua objek ini telah memenuhi prasyarat penggunaan *template* seperti yang telah disampaikan pada bab metodologi.

5.2.1 UrlHub

UrlHub adalah sebuah aplikasi *open-source link shortener* yang dibangun di atas kerangka kerja Laravel [41]. UrlHub memiliki beberapa fitur untuk memperpendek tautan, penyimpanan tautan pada pengguna, serta scan kode QR untuk membuka tautan.

5.2.2 Crater

Crater adalah sebuah aplikasi *invoicing* yang berguna untuk manajemen *invoice*, pelacakan pengeluaran, dan pelacakan pembayaran yang dibangun di atas kerangka kerja Laravel [42]. Fitur utama yang ditawarkan adalah adanya *dashboard*, manajemen hingga tersedianya *template invoice* professional yang dapat dikirimkan kepada pelanggan yang ditagihkan melalui surel.

5.3 Pengambilan Data Percobaan

Seperti yang telah dijelaskan pada metodologi, proses percobaan dilakukan dua kali yaitu proses *deployment* secara manual dan proses *deployment* secara otomatis pada masing-masing objek penelitian. Pada prosesnya, terdapat beberapa hal

yang perlu dipertimbangkan dalam perbandingan kedua proses deployment. Berikut adalah variabel yang akan diamati pada proses percobaan.

5.3.3 Pengujian *Template*

Pengujian *template* di sini bertujuan untuk mengetahui *acceptance* yang meliputi, keberhasilan CI/CD untuk *build* serta *deploy* aplikasi. Selain itu, perlu juga diuji apakah aplikasi dapat berjalan dengan normal dan tidak terkendala ketika menggunakan *template* yang telah dibuat. Percobaan tersebut dilakukan dengan cara mencoba fitur yang tersedia pada objek penelitian.

5.3.4 Perbandingan Proses

Selanjutnya dilakukan perbandingan antara proses *deployment* secara manual dan secara otomatis. Proses yang dibandingkan ada dua, yaitu:

5.3.4.1 Prasyarat *Deployment*

Pada proses *deployment* secara manual, *instance* yang digunakan memiliki spesifikasi yang lebih rendah daripada *instance* yang digunakan sebagai *node* dalam Kubernetes *cluster*. Oleh karena itu, diperlukan dua buah proses konfigurasi yaitu proses konfigurasi pada *environment deployment* secara manual (non-CI/CD) dan konfigurasi *environment deployment* secara otomatis (CI/CD).

5.3.4.1 Waktu *Deployment*

Selain prasyarat deployment, hal yang perlu diperhatikan di sini adalah waktu yang dialokasikan untuk *deployment*. Di sini dilakukan dua kali

percobaan, yang pertama adalah percobaan *deployment* secara manual dan *deployment* secara otomatis menggunakan *pipeline* CI/CD. Pada percobaan tersebut dilakukan iterasi dengan jumlah iterasi pertama berjumlah lima. Iterasi tersebut berlaku kelipatan lima, jika angka yang didapatkan sudah stabil, maka iterasi dihentikan.

5.4 Konfigurasi Infrastruktur non-CI/CD

Pada proses *deployment* secara manual, diperlukan infrastruktur yang tidak terlalu tinggi spesifikasinya. Oleh karena itu, digunakan infrastruktur berupa satu buah *compute instance* yang telah dijelaskan pada bagian lingkungan percobaan. Satu buah *instance* tersebut digunakan secara bersamaan dengan dua buah aplikasi dengan versi PHP yang berbeda. Hal tersebut memungkinkan untuk dilakukan karena *service* yang berjalan pada Linux berjalan pada dua *socket file* yang berbeda.

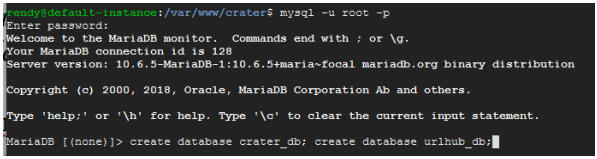
5.4.5 Konfigurasi MariaDB

Pada objek penelitian yang ditentukan, keduanya dapat mendukung *database* MySQL maupun PostgreSQL, namun pada proses *deployment* secara manual dipilihlah *database* MariaDB yang merupakan *database* yang kompatibel MySQL. Untuk pemasangan, terlebih dahulu dilakukan penambahan repositori baru dengan perintah pada Gambar 5. 1 berikut.

```
sudo apt-key adv --fetch-keys 'https://mariadb.org/mariadb_release_signing_key.asc'
sudo add-apt-repository 'deb [arch=amd64,arm64,ppc64el,s390x] https://mirror.telkomuniversity.ac.id/mariadb/repo/10.6/ubuntu focal main'
sudo apt install mariadb-server
sudo mysql_secure_installation
```

Gambar 5. 1 Perintah instalasi MariaDB

Setelah mariadb terpasang, maka selanjutnya yang dilakukan adalah dengan inisiasi *database* untuk kedua objek penelitian tersebut. *Database* yang digunakan kali ini adalah *crater_db*, dan *urlhub_db*.

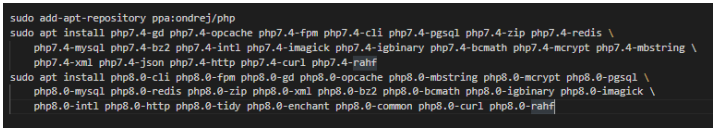


Gambar 5. 2 Inisialisasi database kosong

Inisiasi *database* seperti yang tampil pada Gambar 5. 2 bertujuan agar aplikasi dapat menggunakan *database* yang telah dibuat.

5.4.6 Konfigurasi PHP

Selanjutnya adalah konfigurasi PHP. Kedua objek tersebut memerlukan dua buah versi PHP yang berbeda yaitu masing-masing 7.4 dan 8.0. Oleh karena itu, diperlukan instalasi dua versi PHP. Instalasi tersebut dapat dilakukan dengan eksekusi perintah seperti pada Gambar 5. 3 berikut.



Gambar 5. 3 Perintah instalasi PHP

Setelah dilakukan instalasi PHP, diperlukan instalasi composer yang diperlukan aplikasi dengan kerangka kerja Laravel untuk berjalan dan manajemen *dependency*.

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php composer-setup.php
sudo mv composer.phar /usr/bin/composer
php -r "unlink('composer-setup.php');"
```

Gambar 5. 4 Instalasi composer

Setelah melakukan instalasi composer, selanjutnya dapat melakukan cek dengan menjalankan *composer -v* untuk memastikan composer telah terpasang dengan benar. Jika tidak keluar pesan "*composer: command not found*", berarti composer terpasang dengan benar.

5.4.7 Instalasi NodeJS

Pada proses *build static asset*, tentu diperlukan NodeJS. Untuk proses instalasi, dapat menggunakan perintah berikut.

```
curl -sL https://deb.nodesource.com/setup_16.x -o nodesource_setup.sh
sudo bash nodesource_setup.sh
curl -sL https://dl.yarnpkg.com/debian/pubkey.gpg | gpg --dearmor | sudo tee /usr/share/keyrings/yarnkey.gpg > /dev/null
echo "deb [signed-by=/usr/share/keyrings/yarnkey.gpg] https://dl.yarnpkg.com/debian stable main" | sudo tee /etc/apt/sources.list.d/yarn.list
sudo apt update -y && sudo apt install yarn -y
```

Gambar 5. 5 Instalasi NodeJS

Instalasi seperti yang tampak Gambar 5. 5 dapat dikonfirmasi dengan mencoba menjalankan perintah *node -v* dan *yarn -v*. Jika tidak ada pesan kesalahan maka instalasi sukses dilakukan.

5.4.8 Konfigurasi Nginx

Untuk ekspos aplikasi agar dapat diakses secara publik, diperlukan nginx web server. Untuk instalasi nginx dapat dilakukan dengan menjalankan perintah berikut.

sudo apt install nginx

sudo systemctl enable --now nginx

Setelah perintah tersebut dijalankan, maka untuk konfirmasi dapat dilakukan dengan eksusi perintah *nginx -v*. Jika tidak terdapat pesan kesalahan, maka nginx sudah berhasil terpasang.

5.4.9 Konfigurasi Supervisor

Supervisor diperlukan untuk memanajemen aplikasi agar tetap berjalan di latar belakang. Untuk instalasi supervisor, cukup menggunakan perintah sebagai berikut.

sudo apt install supervisor

Aplikasi supervisor dapat dipastikan bahwa telah terpasang menggunakan perintah *supervisorctl –help* untuk menampilkan menu bantuan.

5.4.10 Konfigurasi Certbot

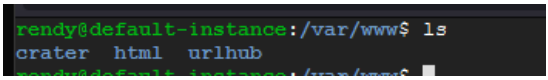
Certbot digunakan untuk otomatisasi dalam proses *request* TLS dari Let’s Encrypt. Instalasi certbot dapat dilakukan dengan mengeksekusi perintah sebagai berikut.

sudo snap install --classic certbot

Setelah selesai, dapat dikonfirmasia dengan menggunakan perintah *which certbot* untuk melihat apakah *executable binary* dari certbot telah ada dan bisa dieksekusi. Hal ini dikarenakan certbot tidak memiliki perintah untuk menunjukkan versi certbot.

5.5 Konfigurasi Aplikasi non-CI/CD

Setelah semua *dependency* terpasang, maka berikutnya yang dilakukan adalah dengan inisiasi *git clone* ke direktori direktori /var/www seperti yang ditunjukkan pada Gambar 5. 6 berikut.



Gambar 5. 6 Direktori deployment aplikasi

Selanjutnya adalah konfigurasi nginx *virtual host* yang disimpan pada direktori /etc/nginx/sites-enabled. Berikut adalah isi dari *virtual host* tersebut.

```
server {
    server_name crater.vm.rendyananta.my.id 80;
    root /var/www/crater/public;
    index index.php;

    error_log /var/log/nginx/crater.error_log;
    access_log /var/log/nginx/crater.access_log;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ .php$ {
        try_files  $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass unix://var/run/php/php7.4-fpm.sock;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param  PATH_INFO $fastcgi_script_name;
        fastcgi_intercept_errors off;
        fastcgi_buffer_size 16k;
        fastcgi_buffers 4 16k;
        fastcgi_connect_timeout 300;
        fastcgi_send_timeout 300;
        fastcgi_read_timeout 300;
    }
}
```

Gambar 5. 7 Konfigurasi virtualhost crater

```
server {
    server_name url.vm.rendyananta.my.id;
    root /var/www/urlhub/public;
    index index.php;

    error_log /var/log/nginx/urlhub.error_log;
    access_log /var/log/nginx/urlhub.access_log;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ .php$ {
        try_files  $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass unix:/var/run/php/php8.0-fpm.sock;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param  PATH_INFO $fastcgi_script_name;
        fastcgi_intercept_errors off;
        fastcgi_buffer_size 16k;
        fastcgi_buffers 4 16k;
        fastcgi_connect_timeout 300;
        fastcgi_send_timeout 300;
        fastcgi_read_timeout 300;
    }
}
```

Gambar 5. 8 Konfigurasi virtualhost urlhub

Berdasarkan Gambar 5. 7 dan Gambar 5. 8, tiap request yang memiliki format PHP, akan diarahkan pada PHP FPM yang masing-masing memiliki versi 7.4 dan 8.0. Selain itu, *directive* server_name juga menyesuaikan dengan domain. Kemudian langkah selanjutnya adalah dengan *restart service nginx* dengan perintah berikut.

sudo systemctl restart nginx.service

Selanjutnya, dapat dilanjutkan dengan konfigurasi supervisor yang terletak pada direktori /etc/supervisor/conf.d. Berikut adalah konfigurasi program pada supervisor.

```
[program:crater-worker]
process_name=%(program_name)s_%(process_num)02d
command=php /var/www/crater/artisan queue:work --sleep=3 --tries=3 --max-time=3600
autostart=true
autorestart=true
stopasgroup=true
killasgroup=true
user=www-data
numprocs=2
redirect_stderr=true
stdout_logfile=/var/crater/worker.log
stopwaitsecs=3600
```

Gambar 5. 9 Konfigurasi crater-worker supervisor

```
[program:urlhub-worker]
process_name=%(program_name)s_%(process_num)02d
command=php /var/www/urlhub/artisan queue:work --sleep=3 --tries=3 --max-time=3600
autostart=true
autorestart=true
stopasgroup=true
killasgroup=true
user=www-data
numprocs=2
redirect_stderr=true
stdout_logfile=/var/urlhub/worker.log
stopwaitsecs=3600
```

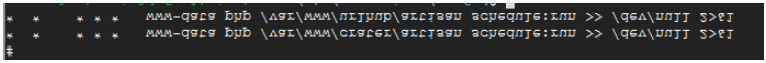
Gambar 5. 10 Konfigurasi urlhub-worker supervisor

Setelah melakukan konfigurasi seperti yang terdapat pada Gambar 5. 9 dan Gambar 5. 10, program tersebut belum berjalan. Untuk menjalankan program, perintah yang digunakan adalah sebagaimana Gambar 5. 11 berikut.

```
sudo supervisorctl reread
sudo supervisorctl start crater-worker:*
sudo supervisorctl start urlhub-worker:*
```

Gambar 5. 11 Perintah menjalankan supervisor

Selain *queue*, terdapat juga konfigurasi untuk menambahkan baris pada crontab seperti berikut.



Gambar 5. 12 Konfigurasi scheduler pada crontab

Konfigurasi pada Gambar 5. 12 akan langsung berjalan dan dieksekusi tanpa perlu menambahkan perintah.

Terakhir, yaitu konfigurasi TLS agar laman web dari kedua objek penelitian dapat diakses menggunakan protokol HTTPS. Untuk mengkonfigurasi TLS, dapat menggunakan perintah sebagai berikut.

sudo certbot --nginx

Perintah di atas akan menampilkan domain mana yang ingin dilakukan *request* TLS. Isi beberapa input seperti nama dan email, kemudian biarkan input yang lainnya bernilai default.

Aplikasi dapat dibuka seperti biasa pada domain yang telah didefinisikan pada *virtual host* nginx, tepatnya pada *directive 'server_name*', yaitu https://crater.vm.rendyananta.my.id untuk mengakses crater serta https://url.vm.rendyananta.my.id untuk mengakses urlhub.

5.6 Konfigurasi Infrastruktur CI/CD

Pada proses deployment secara otomatis menggunakan CI/CD, diperlukan beberapa konfigurasi infrastruktur seperti Jenkins, Sonarqube, basis data, redis, dan minio. Konfigurasi yang dilakukan memiliki 2 jenis, yaitu *Configuration as a Code*, dan konfigurasi yang dilakukan melalui antarmuka aplikasi. Sebelumnya perlu dilakukan definisi *credentials* untuk *image*

registry untuk *push* dan *pull image* dengan terlebih dahulu dengan perintah berikut.

kubectl create secret docker-registry -n ops docker-credentials

--docker-server=<registry-server> \

--docker-username=<username> \

--docker-password=<password> \

--docker-email=<email>

Perintah tersebut berguna untuk membuat *secret* untuk menyimpan autentikasi pada *container registry*, baik ketika melakukan *pull* dan *push image*. Selanjutnya dapat dilanjutkan untuk konfigurasi infrastruktur CI/CD yang akan dijelaskan berikut.

5.6.11 Konfigurasi Jenkins

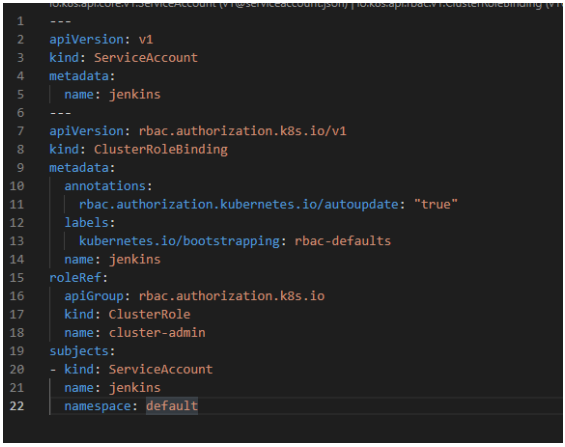
Pada tahap konfigurasi, digunakan *helm chart* untuk melakukan *generation* terhadap Kubernetes *resource* yang digunakan dalam *deployment* Jenkins. Pada *template* ini, dilakukan beberapa perubahan dari *default value* pada *helm chart* untuk menyesuaikan *usecase*. Berikut adalah konfigurasi yang digunakan.

5.6.11.1 Jenkins Master

Konfigurasi yang ada pada Jenkins *master* adalah konfigurasi yang banyak berperan penting, karena beberapa *plugin* dan sistem *runner* akan berjalan dengan perintah dari Jenkins *master*.

Seperti yang telah dijelaskan sebelumnya, konfigurasi yang digunakan untuk inisiasi Jenkins *master* berasal dari *default value* helm yang secara *open-source* ada pada Github yang tertera pada dokumentasi resminya[43].

Pada konfigurasi ini, terdapat juga Kubernetes resource yang didefinisikan secara manual seperti *service account* dan TLS *issuer*. Berikut adalah potongan kode dari konfigurasi *service account*.



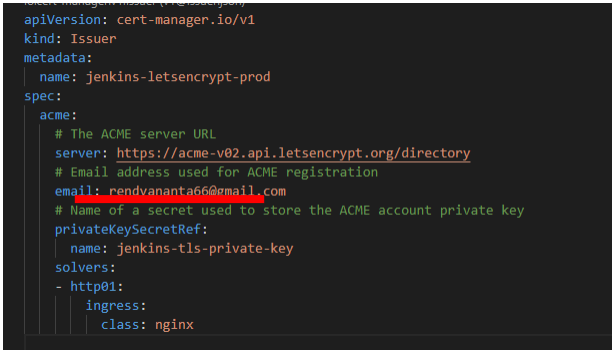
```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: jenkins
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: jenkins
  namespace: default
```

Gambar 5. 13 Konfigurasi Service Account Jenkins

Konfigurasi *service account* yang dapat dilihat pada Gambar 5. 13 tersebut ada dua buah, yaitu definisi ServiceAccount dan ClusterRoleBinding. Keduanya berfungsi untuk memberikan akses role menjadi cluster-admin terhadap Jenkins terhadap Kubernetes API. Hal ini diperlukan karena pada pod Jenkins agent yang akan menjalankan perintah kubectl yang secara langsung mengakses Kubernetes API dari dalam *container*. Perintah kubectl tersebut akan diperlukan

ketika dilakukan *deployment* aplikasi menggunakan Jenkins.

Selanjutnya adalah konfigurasi TLS *issuer*. Konfigurasi ini digunakan untuk menyimpan beberapa informasi yang dimanfaatkan oleh cert-manager untuk memperbarui TLS *certificate* yang didapatkan dari Let's Encrypt menggunakan ACME *protocol*.



Gambar 5. 14 Konfigurasi TLS Jenkins

Seperti yang dapat dilihat pada Gambar 5. 14, spec yang digunakan adalah acme, yaitu menggunakan protokol dari ACME untuk mendapatkan *certificate*. *Certificate* tersebut berasal dari Let's Encrypt yang kemudian disimpan ke dalam Secret bernama Jenkins-tls-private-key.

5.6.11.2 Jenkins Agent

Jenkins *agent* adalah *instance* di mana proses *deployment* dilakukan. Secara umum, di dalam *official image* pada Jenkins *agent* tidak terdapat beberapa perintah yang diperlukan. Oleh karena itu, diperlukan

penambahan ekstensi terhadap *image* tersebut. Perintah yang diperlulan di sini adalah.

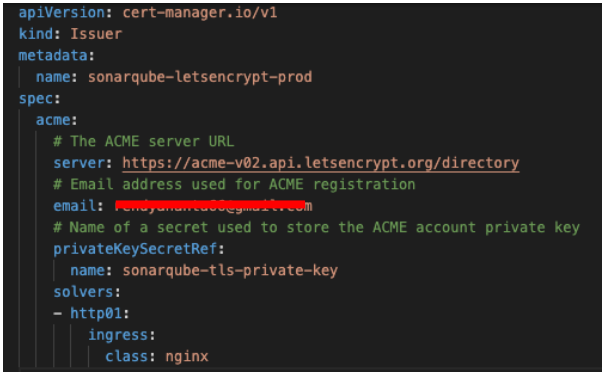
- Kubectl, untuk mengakses Kubernetes API
- Sops, untuk melakukan proses *decryption* terhadap Secret terlebih dahulu ada di dalam git repositori. Sops digunakan untuk menghindari melakukan *push credentials* penting ke dalam git repositori.
- Gettext, yaitu sebuah *package* yang didalamnya terdapat perintah envsubts. Perintah tersebut berguna untuk melakukan substitusi *environment variable* pada Kubernetes resource yang didefinisikan di dalam git repositori.

Gambar 5. 15 Ekstensi pada image jenkins agent

Seperti yang dapat dilihat pada Gambar 5. 15, penulis menambahkan perintah pada baris ke-5 untuk memasang *package gettext* melalui *package manager*. Kemudian dilanjutkan pada baris 8 menjalankan perintah pengunduhan dokumen *binary* kubectl dan memindahkan serta memberi akses permission 755 nya ke /usr/local/bin/kubectl. Terakhir yaitu memasang sops menggunakan perintah dpkg.

5.6.12 Konfigurasi SonarQube

Layaknya konfigurasi yang digunakan pada Jenkins, pada konfigurasi SonarQube juga menggunakan *helm chart* yang diperoleh dari publik repositori Github yang didapat dari dokumentasi resminya [44]. Pada konfigurasi Sonarqube, value yang digunakan adalah *default value* yang disediakan oleh *helm chart* tersebut, kecuali pada bagian *ingress*. Pada bagian ingress dilakukan modifikasi untuk menyesuaikan domain dan *TLS issuer* yang digunakan.

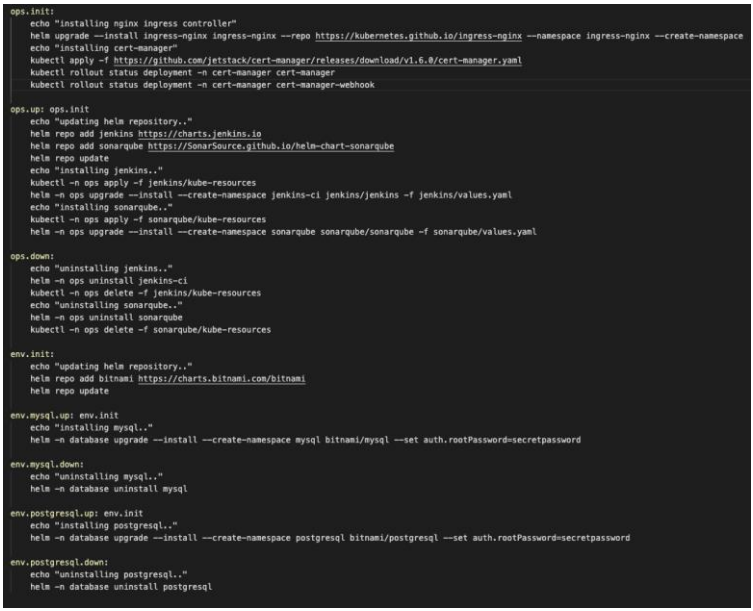


Gambar 5. 16 Konfigurasi TLS SonarQube

Serupa dengan konfigurasi TLS *issuer* pada Jenkins sebelumnya, pada Gambar 5. 16 juga diisi konfigurasi yang sama. Pembedanya ada pada *name* dan *secret name* yang digunakan sebagai identifikasi yang unik agar kedua *secret* tersebut tidak bertabrakan dan tumpang tindih dengan secret yang digunakan pada *deployment* Jenkins.

5.6.13 Otomatisasi Konfigurasi Infrastruktur

Sebelum aplikasi dapat dikonfigurasikan dan digunakan, maka sebelumnya yang perlu dilakukan adalah *provisioning service* yang dibutuhkan untuk menjalankan *pipeline* CI/CD. Dalam memudahkan proses *provisioning* pada Jenkins dan SonarQube, disini digunakan *bash script* yang dituliskan dalam *makefile* untuk otomatisasi proses tersebut. Konfigurasi makefile yang digunakan adalah seperti berikut.



```
ops.init:
    echo "installing nginx ingress controller"
    helm upgrade --install ingress-nginx ingress-nginx --repo https://kubernetes.github.io/ingress-nginx --namespace ingress-nginx --create-namespace
    echo "installing cert-manager"
    kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.6.0/cert-manager.yaml
    kubectl rollout status deployment -n cert-manager cert-manager
    kubectl rollout status deployment -n cert-manager cert-manager-webhook

ops.up: ops.init
    echo "updating helm repository.."
    helm repo add jenkins https://charts.jenkins.io
    helm repo add sonarqube https://SonarSource.github.io/helm-chart-sonarqube
    helm repo update
    echo "installing jenkins.."
    kubectl -n ops apply -f jenkins/kube-resources
    helm -n ops upgrade --install --create-namespace jenkins-ci jenkins/jenkins -f jenkins/values.yaml
    echo "installing sonarqube.."
    kubectl -n ops apply -f sonarqube/kube-resources
    helm -n ops upgrade --install --create-namespace sonarqube sonarqube/sonarqube -f sonarqube/values.yaml

ops.down:
    echo "uninstalling jenkins.."
    helm -n ops uninstall jenkins-ci
    kubectl -n ops delete -f jenkins/kube-resources
    echo "uninstalling sonarqube.."
    helm -n ops uninstall sonarqube
    kubectl -n ops delete -f sonarqube/kube-resources

env.init:
    echo "updating helm repository.."
    helm repo add bitnami https://charts.bitnami.com/bitnami
    helm repo update

env.mysql.up: env.init
    echo "installing mysql.."
    helm -n database upgrade --install --create-namespace mysql bitnami/mysql --set auth.rootPassword=secretpassword

env.mysql.down:
    echo "uninstalling mysql.."
    helm -n database uninstall mysql

env.postgresql.up: env.init
    echo "installing postgresql.."
    helm -n database upgrade --install --create-namespace postgresql bitnami/postgresql --set auth.rootPassword=secretpassword

env.postgresql.down:
    echo "uninstalling postgresql.."
    helm -n database uninstall postgresql
```

Gambar 5. 17 Script otomasi instalasi infrastruktur

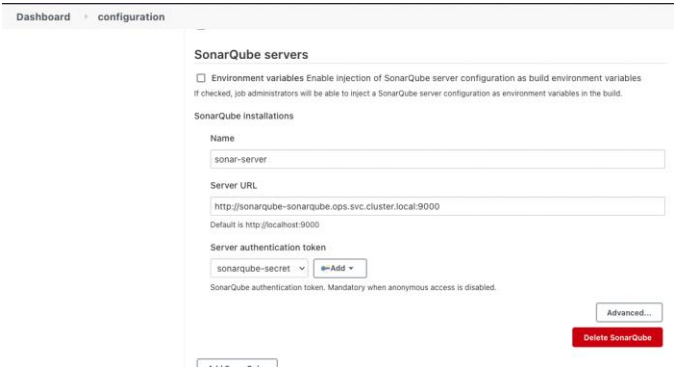
Seperti yang dapat dilihat pada Gambar 5. 17, konten makefile tersebut pada perintah *make ops.up* berisi instalasi Jenkins dan sonarqube pada *namespace ops*, dan instalasi basis data MySQL dapat menjalankan perintah *make env.mysql.up* atau *make env.postgresql.up* untuk instalasi PostgreSQL.

5.6.14 Integrasi Jenkins dan SonarQube

Setelah semua infrastruktur yang dibutuhkan ini sudah menyala dan berjalan, maka selanjutnya dapat melakukan konfigurasi agar Jenkins dan Sonarqube dapat berinteraksi. Caranya adalah dengan melakukan generasi *access token* yang digunakan pada tiap kali CI/CD dijalankan pada Jenkins.

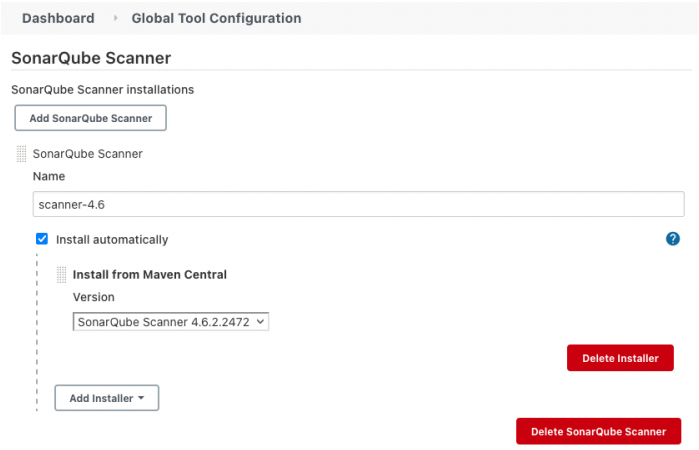
Gambar 5. 18 Generate token pada Sonarqube

Berdasarkan Gambar 5. 18, dapat dilihat bahwa *token* yang bernama *jenkins-ci* berhasil dibuat yang kemudian akan digunakan untuk konfigurasi Jenkins. Setelah *token* didapatkan, kemudian yang dapat dilakukan adalah dengan mengintegrasikan Jenkins dengan SonarQube dengan *token* tersebut.



Gambar 5. 19 Konfigurasi server SonarQube

Pada Gambar 5. 19, dilakukan pengaturan untuk mengarahkan *Server URL* SonarQube dan memberi Jenkins akses kepada *server* tersebut dengan *server authentication token* yang telah didapatkan sebelumnya.

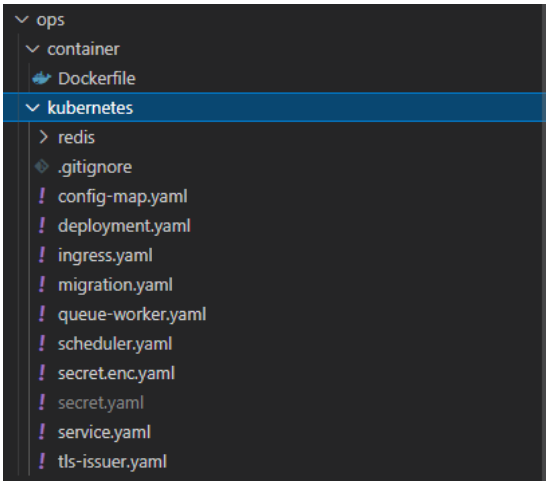


Gambar 5. 20 Pengaturan SonarQube Scanner

Selanjutnya, dapat dilakukan pengaturan versi SonarQube *scanner* yang digunakan pada saat menjalankan CI/CD *pipeline*. Di sini digunakan versi 4.6 karena versi tersebut merupakan versi terbaru pada saat percobaan dilakukan.

5.7 Konfigurasi Aplikasi CI/CD

Aplikasi juga perlu dikonfigurasikan agar dapat digunakan pada *pipeline* CI/CD. Beberapa hal yang perlu dikonfigurasikan adalah *containerization* dan Kubernetes *deployment manifests*. Beberapa *template* konfigurasi tersebut terdiri dari satu buah Jenkinsfile untuk konfigurasi pipeline dan satu buah folder bernama *ops* yang berisi *deployment manifests* seperti yang ditunjukkan pada gambar berikut.



Gambar 5. 21 Isi direktori ops

Berdasarkan Gambar 5. 21, terdapat dua direktori utama yaitu *container* yang berisi Dockerfile yang berfungsi untuk

containerization aplikasi dan direktori Kubernetes yang berisi Kubernetes *deployment manifests* berformat yaml.

5.7.15 Containerization

Kebutuhan untuk aplikasi dapat berjalan di atas *container* merupakan suatu kewajiban yang perlu dipenuhi sebelum melakukan *deployment* di atas Kubernetes. *Container* di sini didefinisikan menggunakan sebuah *dockerfile* untuk *image* yang berisikan php-fpm dan nginx.

```
FROM ghcr.io/rendyananta/php-docker:8.0-nginx AS base

LABEL org.opencontainers.image.source=https://github.com/rendyananta/urlhub

FROM base AS build-fpm

WORKDIR /var/www/html

COPY --from=composer:2 /usr/bin/composer /usr/bin/composer
COPY /artisan artisan

# Prevent composer install again if the composer.json file doesn't changed
COPY /composer.json composer.json
COPY /composer.lock composer.lock

RUN composer install --prefer-dist --no-ansi --no-autoloader

# Optimize for image building cache
COPY /bootstrap bootstrap
COPY /app app
COPY /config config
COPY /routes routes
COPY . /var/www/html
RUN chmod -R 775 /var/www/html

RUN composer dump-autoload -o
RUN php artisan test

FROM node:17-alpine AS build-asset

WORKDIR /var/www/html

COPY /package.json package.json
COPY /yarn.lock yarn.lock

COPY . /var/www/html

RUN yarn install --frozen-lockfile
RUN yarn run production

FROM build-fpm AS fpm

COPY --from=build-fpm --chown=www-data:www-data /var/www/html /var/www/html
COPY --from=build-asset --chown=www-data:www-data /var/www/html/public /var/www/html/public

USER www-data
```

Gambar 5. 22 Konfigurasi Dockerfile untuk aplikasi

Berdasarkan Gambar 5. 22, basis *image* yang digunakan adalah versi 8.0, namun basis image tersebut juga tersedia

dalam versi 7.3, 7.4, dan 8.1. Hal ini bertujuan untuk kustomisasi *template* sesuai dengan kebutuhan aplikasi. Selain itu, digunakan juga fitur *container multistage build* agar *cache* tiap *stage* dapat digunakan kembali para proses *build* berikutnya. Pada konfigurasi tersebut, terdapat inisialisasi project PHP dengan *composer*, dan dilanjutkan dengan dijalankannya pengujian secara otomatis. Pengujian tersebut bertujuan untuk mencegah aplikasi yang tidak layak untuk rilis (memiliki *defect*) untuk disimpan dalam *container registry*. Kemudian dilanjutkan dengan *build asset* menggunakan *image* NodeJS.

5.7.16 Definisi Kubernetes *Resource*

Ketika telah dilakukan containerization pada aplikasi Laravel, maka hal selanjutnya adalah dengan mendefinisikan *resource* Kubernetes yang dibutuhkan.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
spec:
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - name: redis
        image: redis
        resources:
          limits:
            memory: "128Mi"
            cpu: "100m"
        ports:
        - containerPort: 6379
          name: redis
```

Gambar 5. 23 Konfigurasi deployment redis

Pada percobaan kali ini, terdapat resource tersendiri untuk *deployment* redis dengan sederhana dengan satu buah replika seperti yang ditunjukkan pada Gambar 5. 23.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis
spec:
  ports:
  - port: 6379
  selector:
    app: redis
  clusterIP: None
```

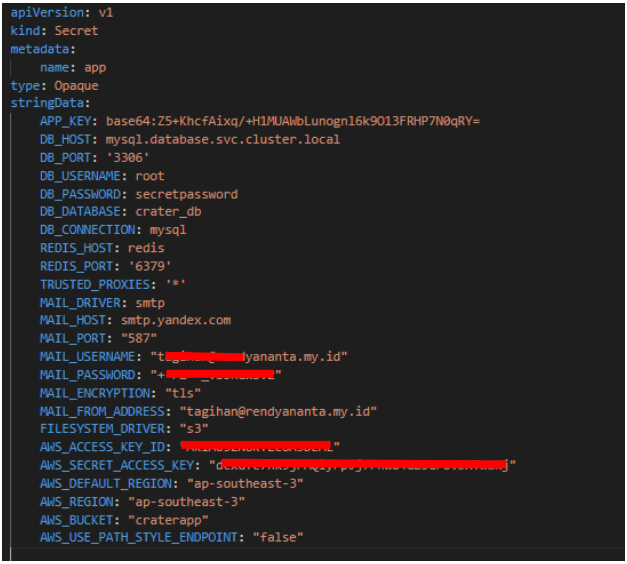
Gambar 5. 24 Konfigurasi service redis

Agar *deployment* dari redis tersebut dapat diakses oleh *service* lain, maka diperlukan definisi *manifest* seperti pada Gambar 5. 24.

Pada sisi aplikasi, juga terdapat macam *manifest* yang perlu didefinisikan sebagaimana terlihat berikut.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app
data:
    APP_NAME: "Crater"
    APP_ENV: local
    APP_DEBUG: "true"
    APP_URL: https://crater.k8s.rendyananta.my.id
    ASSET_URL: https://crater.k8s.rendyananta.my.id
    LOG_CHANNEL: stderr
    LOG_LEVEL: debug
    BROADCAST_DRIVER: redis
    CACHE_DRIVER: redis
    QUEUE_CONNECTION: redis
    SESSION_DRIVER: cookie
    SESSION_LIFETIME: "120"
    MAIL_FROM_NAME: "Crater Invoice"
    SANCTUM_STATEFUL_DOMAINS: crater.k8s.rendyananta.my.id
    SESSION_DOMAIN: crater.k8s.rendyananta.my.id
    TRUSTED_PROXIES: "*"
```

Gambar 5. 25 Konfigurasi template config map

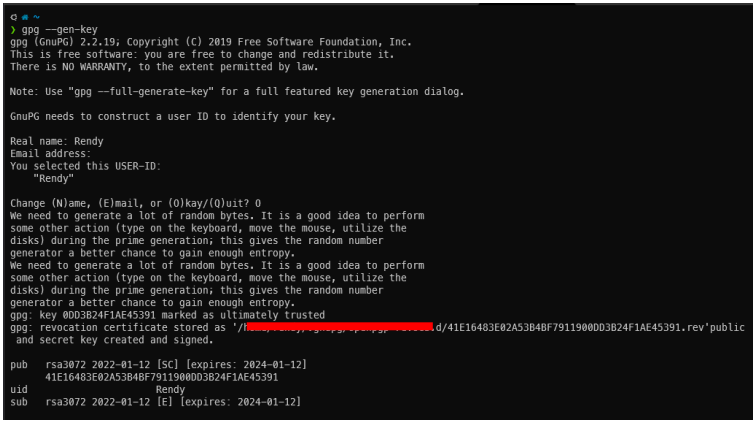


```
apiVersion: v1
kind: Secret
metadata:
    name: app
type: Opaque
stringData:
    APP_KEY: base64:Z5+KhcfAixq/+H1MUAWbLunogn16k9O13FRHP7N0qRY=
    DB_HOST: mysql.database.svc.cluster.local
    DB_PORT: '3306'
    DB_USERNAME: root
    DB_PASSWORD: secretpassword
    DB_DATABASE: crater_db
    DB_CONNECTION: mysql
    REDIS_HOST: redis
    REDIS_PORT: '6379'
    TRUSTED_PROXIES: '*'
    MAIL_DRIVER: smtp
    MAIL_HOST: smtp.yandex.com
    MAIL_PORT: "587"
    MAIL_USERNAME: "t[redacted]yananta.my.id"
    MAIL_PASSWORD: "+[redacted]"
    MAIL_ENCRYPTION: "tls"
    MAIL_FROM_ADDRESS: "tagihan@rendyananta.my.id"
    FILESYSTEM_DRIVER: "s3"
    AWS_ACCESS_KEY_ID: "[redacted]"
    AWS_SECRET_ACCESS_KEY: "[redacted]"
    AWS_DEFAULT_REGION: "ap-southeast-3"
    AWS_REGION: "ap-southeast-3"
    AWS_BUCKET: "craterapp"
    AWS_USE_PATH_STYLE_ENDPOINT: "false"
```

Gambar 5. 26 Konfigurasi template secret

Potongan kode konfigurasi yang tampak pada Gambar 5. 25 (*configmap*) dan Gambar 5. 26 (*secret*) memiliki kegunaan yang berbeda. Konfigurasi yang ada pada *configmap* cenderung pada konfigurasi aplikasi, sedangkan pada konfigurasi *secret* adalah *credentials* yang kontennya sangat sensitif terhadap keamanan aplikasi. Oleh karena itu, digunakan Mozilla SOPS untuk enkripsi *file secret* tersebut agar tidak tersimpan ke dalam repositori git.

Sebelum dilakukan enkripsi, sebelumnya diperlukan untuk membuat sebuah PGP *key* baru terlebih dahulu dengan menjalankan perintah *gpg --gen-key*, untuk membuat sebuah *key pair* PGP.



Gambar 5. 27 Perintah pembuatan key baru

Seperti yang tampak pada Gambar 5. 27, terdapat *fingerprint* yang terlihat dari proses pembuatan *key* tersebut. *Fingerprint* tersebut akan digunakan untuk ekspor PGP *private key*. Cara ekspor *private key* tersebut adalah dengan menggunakan perintah berikut.

gpg --export-secret-keys --armor "fingerprint" > outputfile.rsa

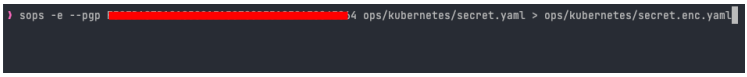
Gambar 5. 28 Perintah ekspor private key

Setelah dilakukan ekspor *private key*, dapat juga dilihat bentuk dari *private key* tersebut seperti yang ada pada Gambar 5. 28. *File private-key.rsa* ini akan dipakai ketika *deployment* untuk dekripsi *file secret* yang telah terenkripsi melalui Jenkins *credentials*.

Dalam proses enkripsi, memiliki sebuah PGP *key* dan *fingerprint* adalah sebuah keharusan. Enkripsi menggunakan Mozilla SOPS dengan memanfaatkan *fingerprint* yang telah dibuat sebelumnya dengan perintah seperti berikut.

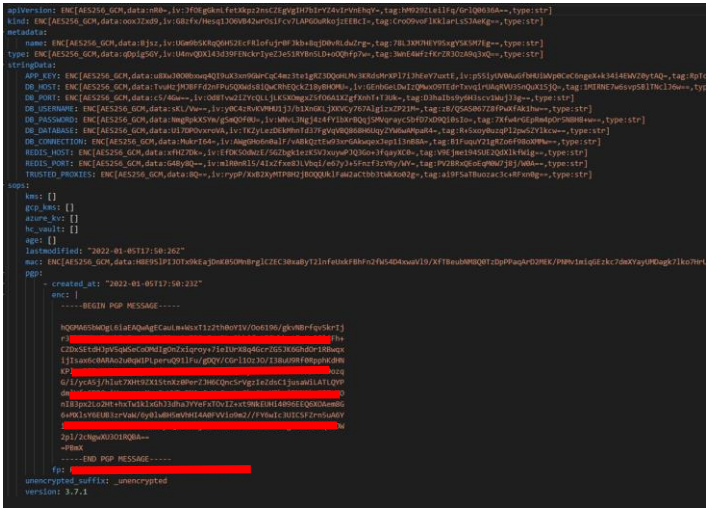
sops -e –pgp "fingerprint" target-enkripsi > output-file

Perintah tersebut dapat diganti sesuai dengan fingerprint yang diperoleh dari generasi PGP key sebelumnya. Target enkripsi dapat diganti dengan *file* yang hendak dienkripsi, serta output file adalah ekspektasi output file yang dibuat dari hasil enkripsi tersebut. Berikut adalah contoh penggunaan perintah pada enkripsi *file secret*.



Gambar 5. 29 Perintah enkripsi file secret

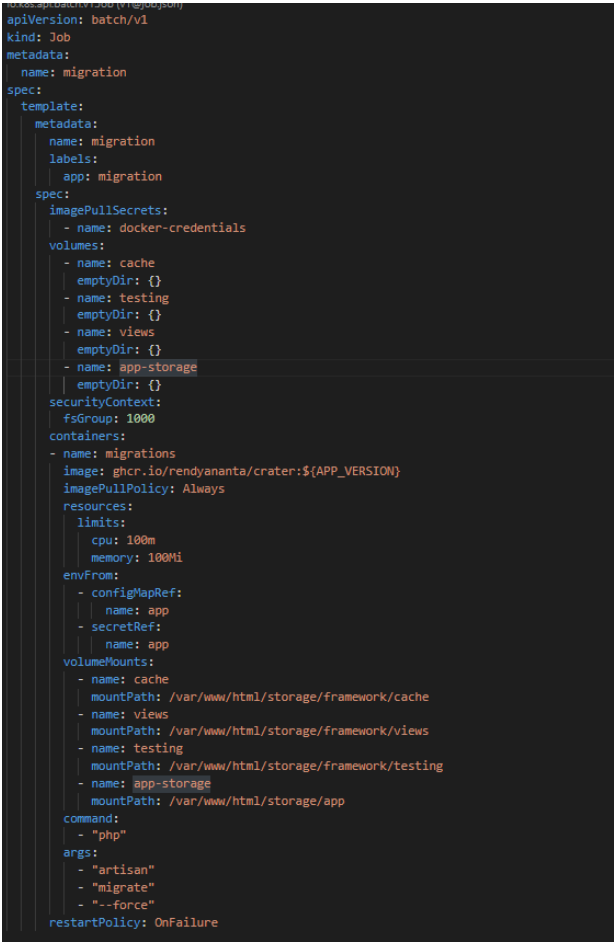
Setelah dilakukan enkripsi menggunakan perintah pada Gambar 5. 29, selanjutnya *file* terenkripsi pada direktori ops/kubernetes akan terbentuk. *File* tersebut bernama secret.enc.yaml yang tampak seperti pada gambar berikut.



Gambar 5. 30 Hasil enkripsi secret menggunakan sops

Seperti yang tampak pada gambar Gambar 5. 30, konfigurasi pada file secret sudah tersamarkan sehingga aman jika disimpan dalam repositori git.

Selain itu, terdapat juga *file* konfigurasi pada *template* terkait *deployment* yang terdiri dari *database migration job*, *web deployment*, *queue-worker deployment*, *scheduler cronjob*, *service*, *TLS issuer*, dan *ingress* yang dapat diketahui dari potongan kode berikut.



```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: migration
spec:
  template:
    metadata:
      name: migration
      labels:
        app: migration
    spec:
      imagePullSecrets:
        - name: docker-credentials
      volumes:
        - name: cache
          emptyDir: {}
        - name: testing
          emptyDir: {}
        - name: views
          emptyDir: {}
        - name: app-storage
          emptyDir: {}
      securityContext:
        fsGroup: 1000
      containers:
      - name: migrations
        image: ghcr.io/rendyananta/crater:${APP_VERSION}
        imagePullPolicy: Always
        resources:
          limits:
            cpu: 100m
            memory: 100Mi
        envFrom:
          - configMapRef:
              name: app
          - secretRef:
              name: app
        volumeMounts:
          - name: cache
            mountPath: /var/www/html/storage/framework/cache
          - name: views
            mountPath: /var/www/html/storage/framework/views
          - name: testing
            mountPath: /var/www/html/storage/framework/testing
          - name: app-storage
            mountPath: /var/www/html/storage/app
        command:
          - "php"
        args:
          - "artisan"
          - "migrate"
          - "--force"
      restartPolicy: OnFailure
```

Gambar 5. 31 Konfigurasi template migration

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name:  web
  labels:
    app:  web
spec:
  selector:
    matchLabels:
      app: web
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app:  web
    spec:
      imagePullSecrets:
        - name: docker-credentials
      volumes:
        - name: cache
          emptyDir: {}
        - name: testing
          emptyDir: {}
        - name: views
          emptyDir: {}
        - name: app-storage
          emptyDir: {}
      containers:
      - name: web
        image: ghcr.io/rendyananta/crater:${APP_VERSION}
        imagePullPolicy: Always
        resources:
          limits:
            cpu: 200m
            memory: 300Mi
        ports:
        - containerPort: 8000
          name: web
        volumeMounts:
          - name: cache
            mountPath: /var/www/html/storage/framework/cache
          - name: views
            mountPath: /var/www/html/storage/framework/views
          - name: testing
            mountPath: /var/www/html/storage/framework/testing
          - name: app-storage
            mountPath: /var/www/html/storage/app
        envFrom:
          - configMapRef:
              name: app
          - secretRef:
              name: app
```

Gambar 5. 32 Konfigurasi template deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: queue-worker
  labels:
    app: queue-worker
spec:
  strategy:
    type: Recreate
  replicas: 1
  selector:
    matchLabels:
      app: queue-worker
  template:
    metadata:
      labels:
        app: queue-worker
    spec:
      volumes:
        - name: cache
          emptyDir: {}
        - name: testing
          emptyDir: {}
        - name: views
          emptyDir: {}
        - name: app-storage
          emptyDir: {}
      imagePullSecrets:
          - name: docker-credentials
      securityContext:
        fsGroup: 1000
      containers:
        - name: queue
          image: ghcr.io/rendyananta/crater:${APP_VERSION}
          imagePullPolicy: Always
          resources:
            limits:
              cpu: 100m
              memory: 100Mi
          volumeMounts:
          - name: cache
            mountPath: /var/www/html/storage/framework/cache
          - name: views
            mountPath: /var/www/html/storage/framework/views
          - name: testing
            mountPath: /var/www/html/storage/framework/testing
          - name: app-storage
            mountPath: /var/www/html/storage/app
          envFrom:
            - configMapRef:
                name: app
            - secretRef:
                name: app
          command:
            - "php"
          args:
            - "artisan"
            - "queue:work"
```

Gambar 5. 33 Konfigurasi template queue worker

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: scheduler
  labels:
    app: scheduler
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          volumes:
          - name: cache
            emptyDir: {}
          - name: testing
            emptyDir: {}
          - name: views
            emptyDir: {}
          - name: app-storage
            emptyDir: {}
          imagePullSecrets:
          - name: docker-credentials
          securityContext:
            fsGroup: 1000
          containers:
            - name: scheduled
              image: ghcr.io/rendyananta/crater:${APP_VERSION}
              imagePullPolicy: Always
              resources:
                limits:
                  cpu: 100m
                  memory: 100Mi
              volumeMounts:
              - name: cache
                mountPath: /var/www/html/storage/framework/cache
              - name: views
                mountPath: /var/www/html/storage/framework/views
              - name: testing
                mountPath: /var/www/html/storage/framework/testing
              - name: app-storage
                mountPath: /var/www/html/storage/app
              envFrom:
                - configMapRef:
                    name: app
                - secretRef:
                    name: app
              command:
                - "php"
              args:
                - "artisan"
                - "schedule:run"
          restartPolicy: OnFailure
```

Gambar 5. 34 Konfigurasi template scheduler

```yaml
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  selector:
    app: web
  type: ClusterIP
  sessionAffinity: None
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 10800
  ports:
  - name: web
    protocol: TCP
    port: 80
    targetPort: 8000
```

Gambar 5. 35 Konfigurasi template service

```yaml
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: rendyananta66@gmail.com
    privateKeySecretRef:
      name: tls-private-key
    solvers:
    - http01:
        ingress:
          class: nginx
```

Gambar 5. 36 Konfigurasi template TLS issuer

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/issuer: "letsencrypt-prod"
spec:
  defaultBackend:
    service:
      name: web
      port:
        number: 80
  rules:
  - host: "crater.k8s.rendyananta.my.id"
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: web
            port:
              number: 80
  tls:
  - secretName: tls
    hosts:
      - crater.k8s.rendyananta.my.id
```

Gambar 5. 37 Konfigurasi template ingress

Beberapa *resource manifest* seperti pada *migration* (Gambar 5. 31), *deployment* (Gambar 5. 32), *queue worker* (Gambar 5. 33), *scheduler* (Gambar 5. 34), merupakan hal yang ada fungsi umum yang mencakup fitur utama dari kerangka kerja Laravel.

Setelah aplikasi dapat berjalan pada Kubernetes, aplikasi tersebut belum dapat diakses secara publik sehingga diperlukan *file manifest* tambahan. *File manifest* seperti *service* (Gambar 5. 35), *TLS issuer* (Gambar 5. 36), dan *ingress* (Gambar 5. 37) untuk membuka akses aplikasi tersebut di internet secara publik.

5.7.17 Konfigurasi Jenkins

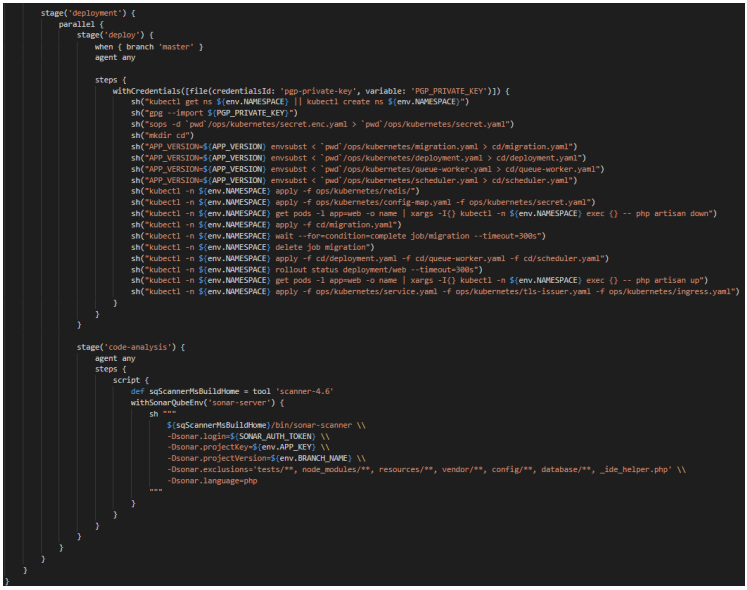
Selanjutnya dilakukan konfigurasi *pipeline* Jenkinsfile sebelum dapat menjalankan *pipeline* pada Jenkins, oleh karena itu perlu dilakukan seperti konfigurasi berikut.

```
pipeline {

    environment {
        APP_KEY = "crater"
        APP_VERSION = "${BUILD_NUMBER}"
        APP_REGISTRY_DESTINATION = 'ghcr.io/rendyananta/crater'
        NAMESPACE = "app-${APP_KEY}"
    }

    agent {
        kubernetes {
            yaml """
kind: Pod
metadata:
  name: php
spec:
  containers:
  - name: kaniko
    image: gcr.io/kaniko-project/executor:debug
    command:
    - sleep
    args:
    - 9999999
    volumeMounts:
    - name: docker-config
      mountPath: /kaniko/.docker
  volumes:
  - name: docker-config
    secret:
      secretName: docker-credentials
      items:
        - key: .dockerconfigjson
          path: config.json
              """
        }
    }

    stages {
        stage('build') {
            steps {
                // build
                container(name: 'kaniko') {
                    script {
                        try {
                            sh """
                              /kaniko/executor --dockerfile=`pwd`/ops/container/Dockerfile \\
                                --context=`pwd` \\
                                --destination=${APP_REGISTRY_DESTINATION}:latest \\
                                --destination=${APP_REGISTRY_DESTINATION}:${APP_VERSION} \\
                                --cache \\
                                --cache-copy-layers
                            """
                        } catch (err) {
                            error('Build aborted. Reason: Cannot build container image')
                        }
                    }
                }
            }
        }
    }
```



```
        stage('deployment') {
            parallel {
                stage('deploy') {
                    when { branch 'master' }
                    agent any

                    steps {
                        withCredentials([file(credentialsId: 'pgp-private-key', variable: 'PGP_PRIVATE_KEY')]) {
                            sh("kubectl get ns ${env.NAMESPACE} || kubectl create ns ${env.NAMESPACE}")
                            sh("gpg --import ${PGP_PRIVATE_KEY}")
                            sh("sops -d `pwd`/ops/kubernetes/secret.enc.yaml > `pwd`/ops/kubernetes/secret.yaml")
                            sh("mkdir cd")
                            sh("APP_VERSION=${APP_VERSION} envsubst < `pwd`/ops/kubernetes/migration.yaml > cd/migration.yaml")
                            sh("APP_VERSION=${APP_VERSION} envsubst < `pwd`/ops/kubernetes/deployment.yaml > cd/deployment.yaml")
                            sh("APP_VERSION=${APP_VERSION} envsubst < `pwd`/ops/kubernetes/queue-worker.yaml > cd/queue-worker.yaml")
                            sh("APP_VERSION=${APP_VERSION} envsubst < `pwd`/ops/kubernetes/scheduler.yaml > cd/scheduler.yaml")
                            sh("kubectl -n ${env.NAMESPACE} apply -f ops/kubernetes/redis/")
                            sh("kubectl -n ${env.NAMESPACE} apply -f ops/kubernetes/config-map.yaml -f ops/kubernetes/secret.yaml")
                            sh("kubectl -n ${env.NAMESPACE} get pods -l app=web -o name | xargs -I{} kubectl -n ${env.NAMESPACE} exec {} -- php artisan down")
                            sh("kubectl -n ${env.NAMESPACE} apply -f cd/migration.yaml")
                            sh("kubectl -n ${env.NAMESPACE} wait --for=condition=complete job/migration --timeout=300s")
                            sh("kubectl -n ${env.NAMESPACE} delete job migration")
                            sh("kubectl -n ${env.NAMESPACE} apply -f cd/deployment.yaml -f cd/queue-worker.yaml -f cd/scheduler.yaml")
                            sh("kubectl -n ${env.NAMESPACE} rollout status deployment/web --timeout=300s")
                            sh("kubectl -n ${env.NAMESPACE} get pods -l app=web -o name | xargs -I{} kubectl -n ${env.NAMESPACE} exec {} -- php artisan up")
                            sh("kubectl -n ${env.NAMESPACE} apply -f ops/kubernetes/service.yaml -f ops/kubernetes/tls-issuer.yaml -f ops/kubernetes/ingress.yaml")
                        }
                    }
                }

                stage('code-analysis') {
                    agent any
                    steps {
                        script {
                            def sqScannerMsBuildHome = tool 'scanner-4.6'
                            withSonarQubeEnv('sonar-server') {
                                sh """
                                    ${sqScannerMsBuildHome}/bin/sonar-scanner \\
                                    -Dsonar.login=${SONAR_AUTH_TOKEN} \\
                                    -Dsonar.projectKey=${env.APP_KEY} \\
                                    -Dsonar.projectVersion=${env.BRANCH_NAME} \\
                                    -Dsonar.exclusions='tests/**, node_modules/**, resources/**, vendor/**, config/**, database/**, _ide_helper.php' \\
                                    -Dsonar.language=php
                                """
                            }
                        }
                    }
                }
            }
        }
    }
}
```

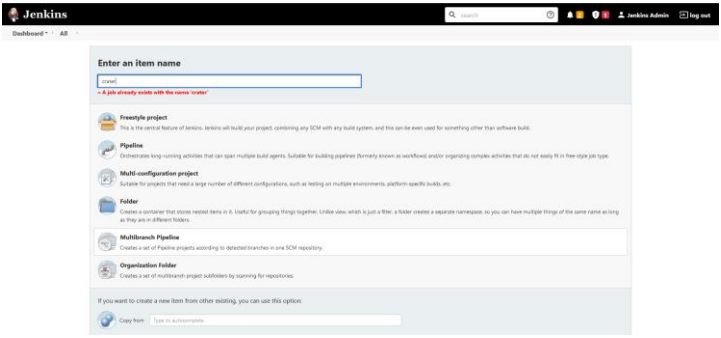
Gambar 5. 38 Konfigurasi Jenkinsfile

Dari Gambar 5. 38 pada bagian *environment*, terdapat beberapa nilai *environment variables* yang akan digunakan pada *pipeline* yang perlu diubah. *Environment variabels* tersebut adalah APP_KEY yang berisikan *project key* dan nantinya digunakan sebagai *deployment namespace* di Kubernetes, sehingga tidak boleh mengandung spasi pada nilainya. Pada saat *pipeline* dijalankan, terdapat proses untuk *build* dan *push container* pada sebuah *container registry*, oleh karena itu perlu dilakukan konfigurasi pada *environment variable* APP_REGISTRY_DESTINATION untuk menentukan target *container registry server* yang dituju.

Jika dilihat dari sisi alur, sudah sesuai dengan apa yang ada pada bab perancangan. Mulai dari awal dilakukannya proses

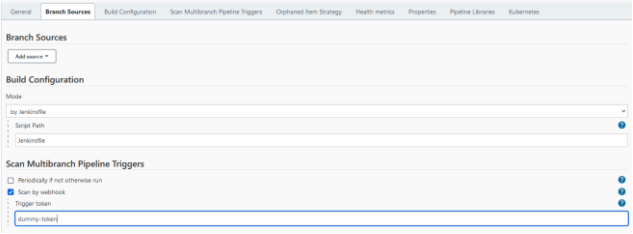
build kemudian dilanjutkan *stage* yang dilakukan secara paralel yaitu proses *deployment* dan proses *code-analysis*.

Selanjutnya, agar pipeline dapat dijalankan pada Jenkins, diperlukan adanya *personal access token* yang didapatkan dari *penyedia server* git seperti GitHub/Bitbucket/Gitlab, untuk membaca repositori privat. Permission yang perlu ada pada *personal access token* adalah diperbolehkan untuk membaca git repositori. Setelah disiapkan *personal access token*, maka dapat dilanjutkan dengan membuat sebuah item baru dengan tipe *multibranch pipeline* seperti yang tampak pada Gambar 5. 39.



Gambar 5. 39 Pembuatan multibranch pipeline

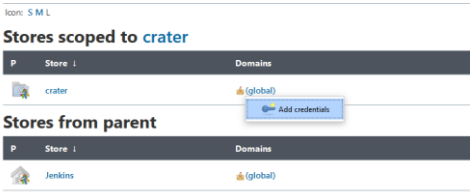
Setelah mengisi *item name* dan memilih tipe item, maka dapat dilanjutkan untuk pengaturan *branch sources* sesuai dengan repositori dan git *credentials* yang telah dimiliki.



Gambar 5. 40 Pengaturan credentials dan webhook

Pada Gambar 5. 40, pada bagian *Scan Multibranch Pipeline Triggers*, pastikan untuk cek *checkbox* "*Scan by webhook"* dan mengisi *trigger token*. Nilai dari *trigger token* ini dapat berupa nilai sembarang, namun lebih baik jika *token* yang digunakan dapat berasal dari *generated random key*. Setelah itu simpan *item*, maka Jenkins akan memulai pemindaian branch pada repositori tersebut. Jika terdapat Jenkinsfile pada *branch*, maka *build job* akan dipicu.

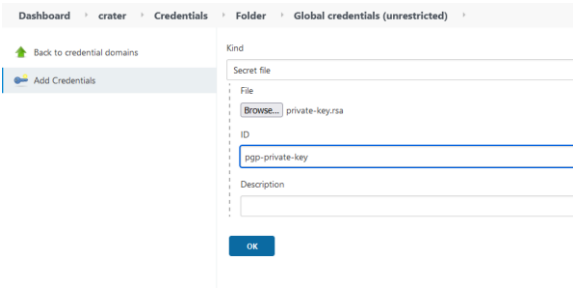
Selanjutnya adalah memasukkan PGP *private key* yang tadi telah diekspor sebelumnya. Pada menu *credentials*, akan tampil tabel *credentials* seperti berikut.



Gambar 5. 41 Item credentials menu

Seperti yang juga terlihat pada Gambar 5. 41, pilih *stores scoped* to nama item yang telah dibuat sebelumnya kemudian

klik *Add credentials* maka akan muncul tampilan seperti berikut.



Gambar 5. 42 Tambah credentials PGP private key

Seperti yang ada pada Gambar 5. 42, *kind credentials* yang dipilih adalah *secret file* dengan *file* yang digunakan adalah private key yang telah diekspor sebelumnya. Pada input ID perlu dipastikan isinya adalah *pgp-private-key* agar tidak perlu mengubah nama variabel pada Jenkinsfile dari templat. Selain itu, perlu dipastikan untuk *docker-credentials secret* telah ada pada namespace aplikasi. Setelah selesai, maka *pipeline* CI/CD dapat dijalankan.

** Halaman ini sengaja dikosongkan **

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini menjelaskan hasil penelitian dan pembahasan secara keseluruhan selama melakukan penelitian.

6.1 Percobaan *Template*

Percobaan *template* CI/CD ini meliputi dua bagian, yaitu pengujian CI/CD dan pengujian *deployment*. Pengujian CI/CD dilakukan untuk memastikan apakah CI/CD berhasil mengotomatisasikan proses pengujian otomatis, *deployment*, dan *static code analysis* dari aplikasi tersebut.

6.1.1 UrlHub

Template yang telah dibuat pada implementasi, kemudian diujicobakan secara langsung dengan menambahkan *resource manifest* untuk *containerization* dan Kubernetes *deployment* dengan menggunakan *image* PHP versi 8. Kustomisasi yang dilakukan hanya pada versi *image* PHP dan penambahan package predis/predis agar aplikasi UrlHub dapat manajemen *session*, *queue*, dan *cache* menggunakan redis.

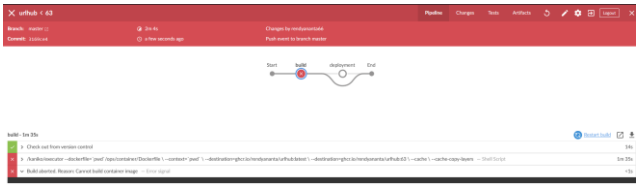
A. *Pipeline* CI/CD

Percobaan *template* pipeline CI/CD ini berlangsung menggunakan beberapa kali iterasi. Pipeline tersebut berjalan dengan sukses seperti yang ditunjukkan berikut.

Gambar 6. 1 Pipeline CI/CD UrlHub

Seperti yang tampak pada Gambar 6. 1, dapat diketahui bahwa total waktu yang diperlukan untuk menjalankan Pipeline adalah 6 menit.

Pengujian selanjutnya adalah dengan menguji menggunakan *source code* yang gagal pada *unit test*. Proses pengujian ini dilakukan untuk memastikan bahwa kode yang memiliki *defect* tidak sampai terdeploy pada Kubernetes *cluster*. Hal ini dapat menyebabkan *blocker* atau tidak dapat diaksesnya aplikasi yang merupakan kejadian fatal yang patut untuk dihindari.

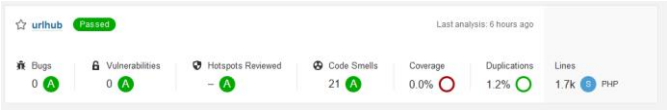


Gambar 6. 2 Percobaan UrlHub dengan pipeline gagal

Seperti yang dapat diketahui melalui Gambar 6. 2 bahwa terdapat pesan kesalahan bahwa *stage* tersebut dilewati karena tidak sukses melalui *stage build*. Hal tersebut menyebabkan tidak ada *container image* yang dibuat dan tidak terjadi *deployment*.

B. SonarQube *static code analysis*

Percobaan ini dilakukan untuk memastikan apakah *pipeline* CI/CD terintegrasi dengan SonarQube. Caranya adalah dengan mengakses *dashboard* SonarQube dan melihat apakah terdapat hasil analisis kode dari repositori UrlHub.

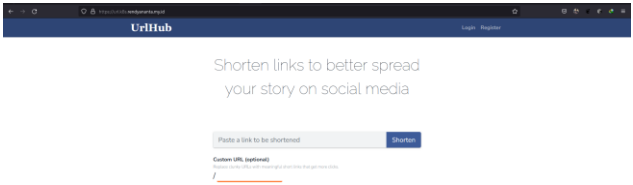


Gambar 6. 3 Hasil analisis UrlHub di SonarQube

Seperti yang tampak pada Gambar 6. 3, dapat diketahui bahwa *pipeline* CI/CD telah terintegrasi dengan SonarQube. Kode sumber dari hasil analisis juga menunjukkan bahwa tidak terdapat *bug*, tidak terdapat *vulnerabilities*, memiliki *code smells* sebanyak 21, dan nilai duplikasi yang sangat rendah yaitu 1.2%.

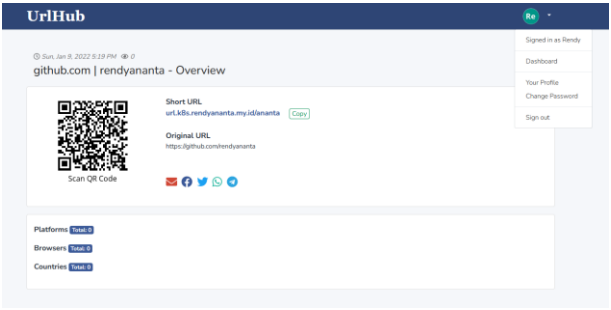
C. Deployment

Pada pengujian *deployment*, UrlHub dapat bekerja dengan baik dengan total 1 replika. Seperti yang dapat dilihat pada Gambar 6. 4.



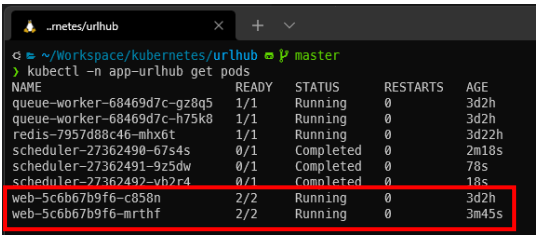
Gambar 6. 4 Laman utama UrlHub

Selain itu, dapat dilihat juga pada bar alamat, bahwa *deployment* pada UrlHub sudah menggunakan HTTPS (logo kunci) dengan menggunakan *certificate* dari *Let's Encrypt.*



Gambar 6. 5 Percobaan fitur link shortener

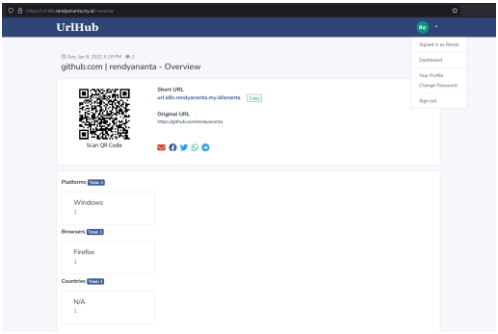
Seperti yang tampak pada Gambar 6. 5, dapat dilihat bahwa fitur *login*, *register*, dan *link shortener* dapat bekerja dengan baik. Selanjutnya dilakukan percobaan dengan menambah jumlah replika pada *deployment* menjadi dua buah.



Gambar 6. 6 Percobaan UrlHub dengan 2 replika

Setelah dilakukan percobaan menggunakan dua buah replika seperti Gambar 6. 6, kedua buah pod

berjalan (*running*) tanpa adanya status *error*. Ketika laman UrlHub diakses, tidak ada kesalahan yang terjadi sehingga *template* yang telah dibuat dapat dijalankan dengan baik pada aplikasi UrlHub.



Gambar 6. 7 Percobaan dua replika UrlHub

Pada percobaan dengan dua buah replika yang ditunjukkan Gambar 6. 7, hal yang penting untuk diketahui adalah aplikasi dapat berjalan secara normal dan tidak ada perubahan. Status *session* pengguna tidak berubah, serta tautan yang telah dipendekkan sebelumnya tetap dapat diakses dengan baik.

6.1.2 Crater

Pada implementasi templat, terdapat beberapa penyesuaian yang dilakukan. Beberapa penyesuaian tersebut antara lain adalah versi PHP yang digunakan adalah versi 7.4. Selain itu, terdapat penambahan *package* predis/predis seperti pada kasus aplikasi UrlHub, untuk memanajemen *session*, *queue*, dan *cache*. Selain itu, pada kode sumber, terdapat bagian untuk modifikasi *file* .env yang mana fitur tersebut tidak akan bekerja karena Laravel akan *override*

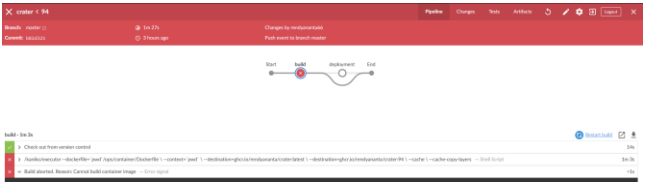
environment variables tersebut jika *environment variables* tersebut ada pada sistem.

A. *Pipeline* CI/CD

Pengujian pertama yang dilakukan pada repositori Crater adalah memastikan CI/CD berjalan tanpa ada kendala.

Gambar 6. 8 Pipeline CI/CD Crater

Pada Gambar 6. 8, dapat diketahui bahwa pipeline berjalan dengan baik waktu 7 menit. Selanjutnya dilakukan juga percobaan ketika tidak sukses melakukan *unit test* atau *build container image* maka pipeline perlu digagalkan untuk memastikan tidak ada aplikasi yang memiliki *defect*.



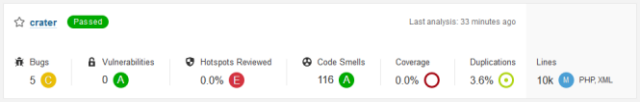
Gambar 6. 9 Percobaan Crater dengan pipeline gagal

Pada Gambar 6. 9, dapat diketahui bahwa terdapat pesan *error* yang menandakan bahwa proses selanjutnya yaitu *deployment* harus dilewati. Hal ini

menunjukkan bahwa CI/CD berjalan sesuai dengan ekspektasi.

B. SonarQube *static code analysis*

Pada pengujian ini, pengecekan integrasi CI/CD dengan SonarQube, seperti yang dapat diketahi melalui Gambar 6. 10 berikut.



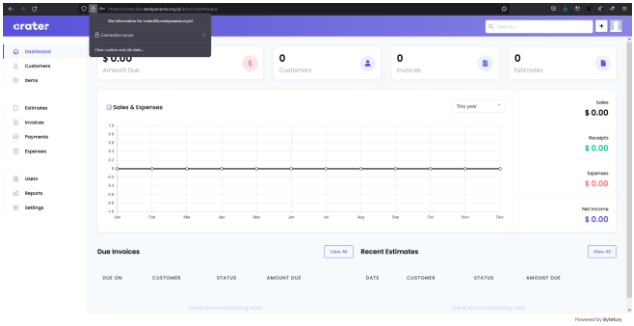
Gambar 6. 10 Hasil analisis Crater di SonarQube

Berdasarkan Gambar 6. 10 tersebut, dapat diketahui bahwa hasil analisis kode pada Crater memiliki 5 *bug*, dan 116 *Code Smells*. Namun, Crater juga memiliki nilai yang cukup baik pada aspek *vulnerabilities* yang bernilai nol dan memiliki 3% duplikasi.

C. *Deployment*

Pada pengujian *deployment*, pertama kali crater memerlukan adanya inisialisasi yaitu dengan menjalankan *database seeder* yang berisikan populasi data konfigurasi secara default yang disediakan, termasuk pada akun yang dibuat.

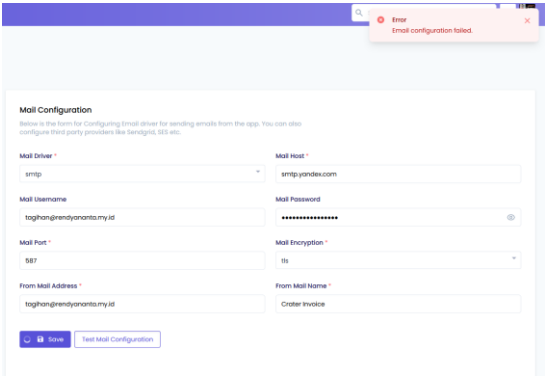
Setelah dijalankan *database seeder*, maka *dashboard crater* dapat diakses dengan normal. Laman *dashboard* dapat diakses melalui protokol HTTPS seperti yang tampak pada Gambar 6. 11.



Gambar 6. 11 Laman dashboard crater

Dari sisi keamanan, *deployment* aplikasi Crater juga dapat diakses menggunakan protokol HTTPS sehingga data yang dikirim maupun diterima oleh pengguna akan terenkripsi.

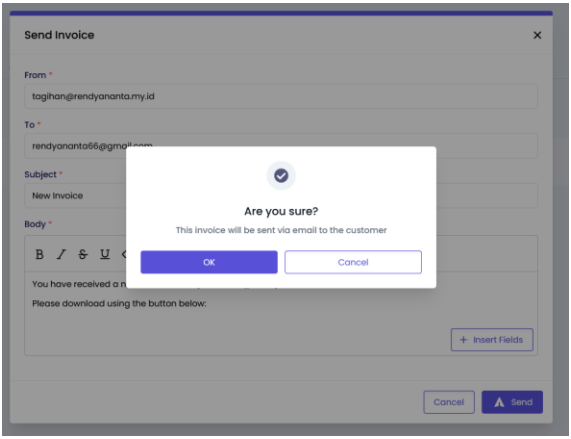
Setelah dicoba, terdapat salah satu fitur yang tidak dapat digunakan yaitu fitur Mail Configuration seperti yang terlihat pada Gambar 6. 12 berikut.



Gambar 6. 12 Gagal simpan mail configuration

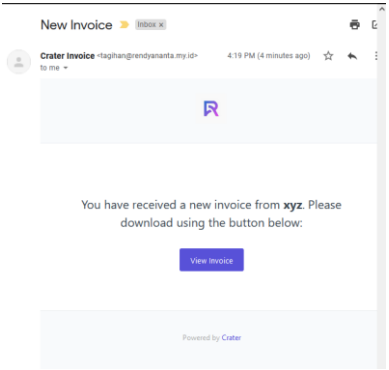
Gagal ketika simpan mail configuration tersebut sudah menjadi diekspektasikan karena dari *source code* ingin menimpa konfigurasi pada *file* .env, sedangkan *environment variables* yang dimuat pada aplikasi berasal dari container.

Namun, meskipun terdapat kegagalan pada fitur *mail configuration*, hal ini tidak terlalu mengganggu jalannya aplikasi. Hal tersebut ditunjukkan pada percobaan pengiriman *invoice* melalui surel.



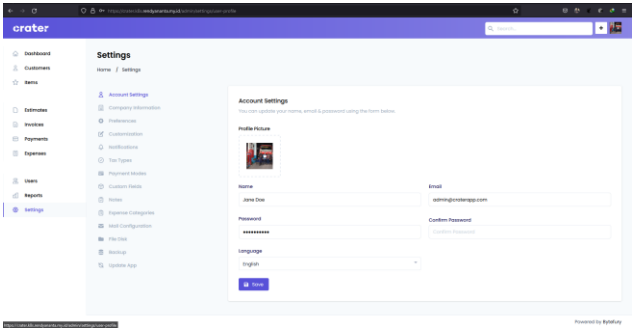
Gambar 6. 13 Percobaan pengiriman invoice

Aksi yang dilakukan pada Gambar 6. 13 dapat diterima dengan alamat surel yang dituju Gambar 6. 14. Hal tersebut menunjukkan bahwa dari sisi fungsi, aplikasi tidak mengalami kendala yang fatal dan dapat berfungsi dengan baik.



Gambar 6. 14 Kotak masuk surel berisi invoice

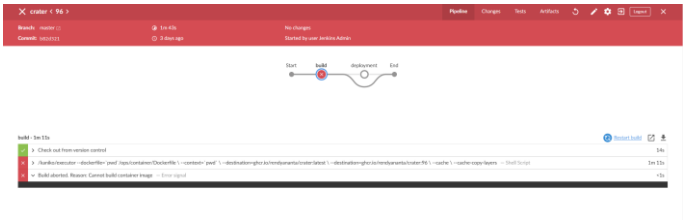
Percobaan selanjutnya adalah dengan menggunakan dua buah replika. Percobaan dengan dua buah replika web berjalan dengan baik, seperti yang dapat dilihat pada Gambar 6. 15.



Gambar 6. 15 Sesi pengguna menggunakan dua replika

Berdasarkan percobaan dua buah replika dengan beberapa kali *reload* halaman dengan disertai pengecekan *log*, tidak ada masalah yang terjadi dan sesi pengguna masih terautentikasi.

Percobaan kedua *template* CI/CD ini juga mengalami permasalahan yang terjadi secara *intermittent* seperti yang tampak pada gambar berikut.



Gambar 6. 16 Intermittent error pada pipeline

Pada Gambar 6. 16, dapat dilihat bahwa *job* gagal dilaksanakan. Hal tersebut terjadi karena adanya *error* pada jaringan yang menyebabkan *pipeline* tidak dapat mengunggah image manifest yang sedang *dibuild*.

6.2 Analisis Hasil Pengujian Templat

Pada bagian ini berisi hasil dan analisis dari percobaan *template* CI/CD yang telah dibuat. Berikut rangkuman implementasi *template* pada dua buah repositori percobaan yaitu UrlHub dan Crater.

Tabel 6. 1 Percobaan implementasi templat

No	Keterangan Implementasi	UrlHub	Crater
1.	Kesesuaian dengan prasyarat *template*	Sesuai	Sesuai
2.	Perubahan konfigurasi templat	Tidak	Penambahan initContainer
3.	Perubahan kode	Menambahkan *package predis/predis*	Menambahkan *package predis/predis*

4.	*Init action*	Tidak	Menjalankan *database seeder* dan konfigurasi *filesystem* untuk menggunakan AWS S3

Seperti yang tampak pada Tabel 6. 1, dari kedua aplikasi yang diujicobakan sesuai dengan prasyarat templat. Meskipun sesuai, terdapat perubahan pada kode pada kedua aplikasi tersebut agar dapat menggunakan *template* yang telah dibuat. Perubahan kode tersebut adalah penambahan *package* predis/predis agar kerangka kerja Laravel dapat menggunakannya sebagai *session* dan *queue driver* sehingga dapat menghilangkan dependency terhadap *filesystem.* Selain perubahan kode, terdapat juga perubahan konfigurasi *template* yang dilakukan pada aplikasi Crater, yaitu penambahan initContainer. Kemudian dari kedua aplikasi tersebut, hanya Crater yang mengharuskan adanya *init action* yaitu aksi yang perlu dilakukan setelah melakukan *deployment* untuk pertama kali dengan menjalankan *database seeder* dan juga konfigurasi *filesystem* untuk menggunakan AWS S3 pada *dashboard*.

Setelah dilakukan implementasi, dilakukan juga pengujian *acceptance* untuk memastikan bahwa *deployment* secara otomatis menggunakan CI/CD dan Kubernetes tidak memiliki perbedaan dengan *deployment* yang dilakukan secara manual yang ditunjukkan pada Tabel 6. 2 berikut.

Tabel 6. 2 Pengujian Acceptance keseluruhan

Kategori	***Acceptance***	**UrlHub**	**Crater**
Pipeline CI/CD	*Unit testing*	Sukses	Sukses
	Build container image	Sukses	Sukses

Analisis Kode	Terintegrasi	Sukses	Sukses
Deployment	Aplikasi dapat diakses menggunakan protocol HTTPS	Sukses	Sukses
	Aplikasi berjalan normal	Sukses	Sukses dengan catatan
	Aplikasi normal menggunakan dua buah replika	Sukses	Sukses dengan catatan

Seperti yang dapat diketahui dari Tabel 6. 2, kedua aplikasi sukses untuk menjalankan *unit testing* dan *build container* pada *pipeline* CI/CD. Kemudian, pada kategori analisis kode, kedua aplikasi sukses menjalankan *pipeline*. Selanjutnya pada kategori deployment, kedua aplikasi sama-sama dapat diakses menggunakan protokol HTTPS, kedua aplikasi berjalan normal pada satu maupun dua buah replika. Hal ini menunjukkan bahwa *template pipeline* CI/CD dan Kubernetes *deployment* efektif digunakan pada kerangka kerja Laravel yang telah memenuhi syarat pada metodologi.

6.3 Perbandingan Proses *Deployment*

Sebelum dilakukan perbandingan proses *deployment* tanpa menggunakan CI/CD dan menggunakan CI/CD, perlu diidentifikasi terlebih dahulu perbedaan mendasar terkait prasyarat untuk melakukan proses tersebut. Berikut adalah pemetaan perbedaan prasyarat sebelum dilakukan deployment.

Tabel 6. 3 Perbandingan prasyarat deployment manual dan otomatis

No	**Manual**	**Otomatis**
1.	*Database*	Kubernetes cluster
2.	Web server	Jenkins

3.	Redis	SonarQube
4.	PHP	Konfigurasi jenkins
5.	Composer	*Database*
6.	NodeJS	*Generate private key*
7.	Supervisord	Konfigurasi Kubernetes *manifests*
8.	Konfigurasi virtualhost	
9.	Konfigurasi supervisor	
10.	Konfigurasi crontab	
11.	Konfigurasi TLS	
Total	11	7

Dari apa yang ada pada Tabel 6. 3, dapat diketahui bahwa prasyarat yang diperlukan untuk melakukan *deployment* secara manual lebih banyak yaitu 11 dibandingkan *deployment* secara otomatis yang hanya berjumlah 7. Selanjutnya adalah perbandingan operasi yang dilakukan ketika *deployment* yang ditunjukkan pada tabel berikut.

Tabel 6. 4 Perbandingan operasi ketika deployment

No	Manual	Otomatis
1.	*Set maintenance mode*	*Pull* repositori
2.	*Pull* repositori	*Build container* (termasuk *automated testing dan build asset*)
3.	*Composer install*	*Set maintenance mode*
4.	Migrasi *database*	Migrasi *database*
5.	*Build asset*	*Apply kubernetes manifest*
6.	*Reload supervisor*	*Static code analysis*

7.	*Set production mode*	*Set production mode*

Seperti yang tampak pada Tabel 6. 4, jumlah operasi yang ada pada *deployment* manual tanpa menggunakan CI/CD terlihat sama dengan *deployment* otomatis menggunakan CI/CD. Meskpun terlihat sama, dari sini dapat terlihat bahwa *output* yang dihasilkan dari proses *deployment* menggunakan CI/CD akan lebih banyak. Tambahannya adalah dijalankannya *automated testing* dan *static code analysis*.

Selain itu, jika diperhatikan lagi pada Tabel 6. 4, aplikasi mengalami *down* (tidak dapat diakses, mendapat *status code* 503) pada proses *deployment* secara manual dimulai dari operasi nomor satu hingga nomor 7. Sedangkan, pada proses *deployment* secara otomatis, aplikasi mengalami *down* hanya pada operasi ketiga hingga operasi terakhir.

6.4 Analisis Perbandingan Proses *Deployment*

Pada perbandingan proses *deployment* sebelumnya dijelaskan perbadaan mendasar terkait prasyarat dan pemetaan jenis operasi yang dilakukan pada *deployment* tanpa menggunakan CI/CD dan dengan menggunakan CI/CD.

Percobaan pertama dilakukan menggunakan 5 kali iterasi dengan membandingkan waktu yang diperlukan untuk melakukan operasi *deployment*. Proses *deployment* dilakukan oleh subyek yang sebisa mungkin untuk fokus dalam menjalankan operasi yang ada pada proses *deployment*. Berikut adalah data yang didapatkan dari proses operasi *deployment* tanpa menggunakan CI/CD dan dengan menggunakan CI/CD pada kedua buah objek percobaan.

Tabel 6. 5 Perbandingan waktu deployment

Iterasi	UrlHub		Crater	
	CI/CD (s)	**Manual (s)**	**CI/CD (s)**	**Manual (s)**
1.	380	100	465	412
2.	393	78	447	381
3.	379	87	455	429
4.	379	98	506	458
5	438	103	480	394
Rata-rata	393.8	93.2	470.6	414.8

Data yang diperoleh pada Tabel 6. 5, berupa detik. Berdasarkan hasil tersebut, angka yang didapatkan cukup konsisten, sehingga iterasi berhenti sampai di sini. Dari data yang diperoleh, dapat diketahui bahwa durasi *deployment* manual tanpa CI/CD pada objek penelitian pertama yaitu UrlHub mempunyai proses *deployment* yang cukup cepat jika dibandingkan durasi keseluruhan pada *deployment* dengan CI/CD. Total durasi *pipeline* CI/CD terpaut lebih dari 250 detik atau empat menit lebih lambat dibandingkan *deployment* tanpa CI/CD.

Fenomena tersebut berbeda dengan apa yang terjadi pada objek penelitian kedua yaitu Crater. Pada percobaan ini, *deployment* secara manual memerlukan waktu yang cukup lama, sehingga tidak terpaut cukup jauh dengan total waktu yang dibutuhkan CI/CD. Selisih rata-rata dari percobaan menggunakan Crater lebih dari 60 detik atau satu menit. Pada kasus ini tetap total keseluruhan *pipeline* CI/CD lebih lambat daripada proses *deployment* secara manual.

Jika dibandingkan, faktor yang menyebabkan deployment pada Crater lebih lama daripada UrlHub adalah pada proses *build asset*. *Build asset* bisa menjadi lama jika terdapat banyak *resource* Javascript yang perlu dikompilasi, seperti halnya Crater yang menggunakan VueJS sebagai kerangka kerja pada tampilan, sedangkan UrlHub tidak menggunakan kerangka kerja Javascript.

Untuk mengurangi bias subyek, dilakukan percobaan lainnya dengan menjalankan *bash script* sederhana untuk membantu eksekusi operasi pada proses *deployment* secara manual. Penggunaan *bash script* ini sekaligus mengabaikan kecepatan pengetikan perintah pada proses *deployment*.

Tabel 6. 6 Perbandingan waktu deployment dengan script

Iterasi	UrlHub			Crater		
	CI/CD (s)	**Manual (s)**	**Script (s)**	**CI/CD (s)**	**Manual (s)**	**Script (s)**
1.	380	100	48	465	412	323
2.	393	78	46	447	381	291
3.	379	87	38	455	429	365
4.	379	98	43	506	458	384
5	438	103	41	480	394	304
Rata-rata	393.8	93.2	43.2	470.6	414.8	333.4

Setelah ditambahkan perhitungan waktu dengan menggunakan *script*, rentang waktu antara *deployment* menggunakan CI/CD dan *deployment* manual (menggunakan menggunakan *script*) menjadi semakin jauh. Hal ini terlihat mencolok pada UrlHub yang terpaut waktu rata-rata hampir 6 menit. Sedangkan pada kasus Crater, perbedaannya ada di kisaran dua menit.

Ketika diamati, CI/CD memiliki waktu yang sangat paling lama baik pada UrlHub maupun pada Crater dan masing-masing pada proses *deployment* manual dan proses *deployment* dengan bantuan *script.*

Analisis selanjutnya adalah bagian yang paling krusial pada waktu *deployment*, yaitu perihal *downtime*. Jika diperhatikan pada operasi pada CI/CD *pipeline* dan operasi manual pada yang tampak pada Tabel 6. 4, terdapat perbedaan urutan operasi yang dijalankan. Terlihat jika operasi *set maintenance mode* pada proses *deployment* secara manual dilakukan sedari awal dilakukan *deployment*. Hal ini berbeda dengan proses *deployment* menggunakan CI/CD, yang operasi *set maintenance mode* hanya ketika hendak *apply kubernetes manifest* yang di dalamnya memanggil container yang telah dilakukan *build* sebelumnya. Berikut adalah total downtime dari perbandingan proses *deployment* secara otomatis menggunakan CI/CD, secara manual, ataupun secara manual dengan bantuan *script.*

Tabel 6. 7 Perbandingan downtime aplikasi

Iterasi	UrlHub			Crater		
	CI/CD (s)	**Manual (s)**	**Script (s)**	**CI/CD (s)**	**Manual (s)**	**Script (s)**
1.	45	100	48	63	412	323
2.	55	78	46	39	381	291
3.	42	87	38	48	429	365
4.	38	98	43	43	458	384
5	47	103	41	48	394	304
Rata-rata	45.4	93.2	43.2	48.2	414.8	333.4

Setelah ditelurusi lebih spesifik terkait *downtime* pada proses *deployment*, hasil yang diperoleh menjadi berbeda. Pada UrlHub, rata-rata *downtime* pada proses *deployment* manual menggunakan bantuan *script* kurang lebih sama dengan rata-rata *downtime* yaitu di sekitar 40 detik. Hal ini membuat CI/CD terlihat kurang begitu efektif pada proyek berskala kecil seperti UrlHub. Berbeda pada kasus Crater yang merupakan proyek dengan skala yang lebih besar dan menggunakan *asset* lebih banyak daripada UrlHub, penggunaan CI/CD terlihat sangat efektif dalam mengurangi *downtime* pada proses *deployment*. Terlihat cukup lama perbedaan *downtime* yaitu hampir 5 menit pada *deployment* manual menggunakan bantuan *script* dan 6 menit pada *deployment* manual.

Dari proses analisis perbedaan sebelumnya, dapat diketahui bahwa penggunaan *template* CI/CD dan Kubernetes memiliki *output* yang lebih banyak daripada proses *deployment* tanpa menggunakan CI/CD. Berikut adalah perbedaan *output* pada proses *deployment*.

- Adanya proses pengecekan validitas *source code* apakah sesuai dengan pengujian terotomatisasi yang telah didefinisikan untuk mencegah perangkat lunak yang memiliki *defect* dilakukan *deployment*.
- Pembuatan *container* sangat berguna jika digunakan pada *platform container orchestration* seperti pada percobaan kali ini yaitu Kubernetes. *Container* memungkinkan adanya *deployment* dengan menggunakan mereplika aplikasi dengan cepat.
- *Static code analysis* yang memberikan informasi kepada stakeholder terkait pengembangan perangkat lunak mengerti bahwa terdapat isu

keamanan pada *source code*, atau terdapat *bug* potensial yang mungkin terjadi.

- Semua dijalankan secara otomatis sehingga minim atau bahkan jarang interaksi dengan manusia, sehingga pengembang perangkat lunak dapat fokus mengembangkan perangkat lunak dan menghemat atau mengeliminasi waktu yang dihabiskan untuk *deployment*.

** Halaman ini sengaja dikosongkan **

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini merupakan bagian akhir dari pengerjakan Tugas Akhir, penjelasan pada bab ini berisi mengenai kesimpulan dari percobaan yang telah dilakukan, kemudian terdapat penjelasan terkait saran dari penulis untuk penelitian selanjutnya.

7.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, terdapat beberapa hal yang dapat disimpulkan dari rancang bangun *template* infrastruktur CI/CD, pipeline CI/CD, dan Kubernetes *manifest deployment* untuk aplikasi yang menggunakan kerangka kerja Laravel, adalah sebagai berikut:

- *Template* infrastruktur terkait *deployment* Jenkins dan SonarQube yang dibuat dapat menjalankan *pipeline* CI/CD dan menganalisis kode dengan baik.
- Dari dua buah percobaan *template* CI/CD dan *deployment* yang diimplementasikan pada repositori publik UrlHub dan Crater dapat dijalankan dengan baik, meskipun diperlukan adanya sedikit perubahan kode.
- Hasil deployment pada aplikasi Crater terdapat salah satu fitur yang tidak dapat digunakan, yaitu *mail configuration*. Namun fitur tersebut bukanlah fitur yang krusial dan vital pada Crater karena fungsi-fungsi yang lain bekerja normal.
- Penggunaan *template* CI/CD dapat bermanfaat untuk pengecekan validitas *source code* sebelum dilakukan *deployment* untuk mencegah perangkat lunak yang memiliki *defect* untuk naik hingga *server*. Selain itu, *template* CI/CD tersebut dapat menghasilkan *container*

image yang dapat memudahkan deployment pada *platform container orchestration* seperti Kubernetes. Adanya *static code analysis* dapat digunakan sebagai laporan dan informasi kualitas *source code* kepada stakeholder yang berkaitan dengan pengembangan perangkat lunak. Pengembang perangkat lunak dapat lebih fokus untuk mengembangkan perangkat lunak karena proses *deployment* tersebut sudah dilakukan secara otomatis. Terakhir yaitu CI/CD dapat meminimalisir *downtime* dengan memanfaatkan *container*, sehingga proses mulai aplikasi jadi lebih cepat.

7.2 Saran

Saran yang dapat diberikan untuk penelitian berikutnya adalah sebagai berikut.

- Adanya penelitian terpisah untuk membandingkan implementasi teknologi CI/CD yang ada seperti Gitlab CI, Travis CI, dan Github Actions.
- Menambahkan fitur *key-rotation* untuk mengganti *key* lama yang telah digunakan dalam enkripsi *file secret*. Fitur tersebut berguna untuk meningkatkan keamanan dengan mengganti *key* lama dengan *key* yang baru demi menjaga kerahasiaan konten dari *file secret* yang masuk ke dalam repositori git.

** Halaman ini sengaja dikosongkan **

DAFTAR PUSTAKA

[1] B. Fitzgerald and K. J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017, doi: 10.1016/j.jss.2015.06.063.

[2] StateOfAgile, "14th annual STATE OF AGILE REPORT," *Annual Report for the STATE OF AGILE*, vol. 14, no. 14, pp. 2–19, 2020, [Online]. Available: https://explore.digital.ai/state-of-agile/14th-annual-state-of-agile-report%0Ahttps://stateofagile.com/#

[3] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016, doi: 10.1109/MS.2016.68.

[4] P. Perera, R. Silva, and I. Perera, "Improve software quality through practicing DevOps," *17th International Conference on Advances in ICT for Emerging Regions, ICTer 2017 - Proceedings*, vol. 2018-Janua, no. 1, pp. 13–18, 2017, doi: 10.1109/ICTER.2017.8257807.

[5] J. Díaz, D. López-Fernández, J. Pérez, and Á. González-Prieto, "Why are many businesses instilling a DevOps culture into their organization?," *Empirical Software Engineering*, vol. 26, no. 2, 2021, doi: 10.1007/s10664-020-09919-3.

[6] Puppet Labs, "2020 State of DevOps Report," *Puppetlabs*, 2020.

[7] CollabNet VersionOne, “The 13th annual STATE OF AGILE Report - 2018,” *CollabNet | VersionOne*, vol. 13, p. 16, 2019.

[8] B. George and L. Williams, “A structured experiment of test-driven development,” *Information and Software Technology*, vol. 46, no. 5 SPEC. ISS., pp. 337–342, 2004, doi: 10.1016/j.infsof.2003.09.011.

[9] C. Amrit and Y. Meijberg, “Effectiveness of Test Driven Development and Continuous Integration: A Case Study,” *IT Professional*, no. February, pp. 27–35, 2017, doi: 10.1109/MITP.2017.265104251.

[10] A. N. A. Thohari and A. E. Amalia, “Implementasi Test Driven Development Dalam Pengembangan Aplikasi Berbasis Web,” *SITECH : Jurnal Sistem Informasi dan Teknologi*, vol. 1, no. 1, pp. 1–10, 2018, doi: 10.24176/sitech.v1i1.2255.

[11] I. B. K. Manuaba, “Combination of test-driven development and behavior-driven development for improving backend testing performance,” *Procedia Computer Science*, vol. 157, pp. 79–86, 2019, doi: 10.1016/j.procs.2019.08.144.

[12] F. Zampetti, A. Di Sorbo, C. A. Visaggio, G. Canfora, and M. Di Penta, “Demystifying the adoption of behavior-driven development in open source projects,” *Information and Software Technology*, vol. 123, no. March, p. 106311, 2020, doi: 10.1016/j.infsof.2020.106311.

[13] S. Stolberg, “Enabling agile testing through continuous integration,” *Proceedings - 2009 Agile Conference, AGILE 2009*, pp. 369–374, 2009, doi: 10.1109/AGILE.2009.16.

[14] M. Fowler, “Continuous Integration,” 2006. http://martinfowler.com/articles/continuousIntegration.html

[15] M. Fowler, J. Highsmith, and others, “The agile manifesto,” *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[16] M. Layton, *Agile Project Management For Dummies: Cheat Sheet*. Canada: John Wiley & Sons, Inc., 2014.

[17] P. Debois, “Agile infrastructure and operations: How infra-gile are you?,” *Proceedings - Agile 2008 Conference*, pp. 202–207, 2008, doi: 10.1109/Agile.2008.42.

[18] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010.

[19] M. Fowler, “ContinuousDelivery,” 2013. https://martinfowler.com/bliki/ContinuousDelivery.html

[20] S. Prince, “The Product Managers’ Guide to Continuous Delivery and DevOps,” 2016. https://www.mindtheproduct.com/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/

[21] Y. Sundmand, “Continuous Delivery vs Continuous Deployment,” 2013.

https://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment

[22] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access*, vol. 5, no. Ci, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.

[23] B. Dustin, Elfriede; Garrett, Thom; Gauf, *Implementing Automated Software Testing*. 2009.

[24] KBBI Daring, "Templat," 2020. https://kbbi.kemdikbud.go.id/entri/templat (accessed Jul. 12, 2021).

[25] Docker, "What is a Container," 2021. https://www.docker.com/resources/what-container

[26] Linux Foundation, "Open Container Initiative," 2021. https://opencontainers.org/ (accessed Jul. 12, 2021).

[27] Kubernetes, "What is Kubernetes?," 2021. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[28] Jenkins, "Jenkins," 2021.

[29] SonarQube, "SonarQube," 2021. https://www.sonarqube.org/

[30] S. Bergmann, "PHPUnit," 2021. https://phpunit.de/index.html

[31] D. Huizinga and A. Kolawa, *Automated Defect Prevention: Best Practices in Software Management*. 2007. doi: 10.1002/9780470165171.

[32] Laravel, "Laravel - Introduction," *Laravel Documentation*, 2021. https://laravel.com/docs/8.x

[33] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "OpenPGP Message Format," 2007. Accessed: Jan. 08, 2022. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4880.txt

[34] Mozilla, "Mozilla Secret Operations." https://github.com/mozilla/sops (accessed Jan. 08, 2022).

[35] Redis Ltd., "Redis." https://redis.io/ (accessed Jan. 13, 2022).

[36] nginx.org, "About nginx." https://nginx.org/en/ (accessed Jan. 21, 2022).

[37] supervisord.org, "Supervisor: A Process Control System." http://supervisord.org/ (accessed Jan. 21, 2022).

[38] L. Linas, "Cron Job: A Comprehensive Guide for Beginners 2022." https://www.hostinger.com/tutorials/cron-job (accessed Jan. 21, 2022).

[39] Electronic Frontier Foundation, "About Certbot." https://certbot.eff.org/pages/about (accessed Jan. 21, 2022).

[40] Amazon Web Services, “FAQ S3 Umum.” https://aws.amazon.com/id/s3/faqs/ (accessed Jan. 30, 2022).

[41] B. Hermawan, “UrlHub.” https://github.com/realodix/urlhub (accessed Jan. 13, 2022).

[42] Craterapp, “Crater.” https://docs.craterapp.com/#features (accessed Jan. 13, 2022).

[43] Jenkins, “Kubernetes,” Nov. 2021. https://www.jenkins.io/doc/book/installing/kubernetes/ (accessed Jan. 03, 2022).

[44] SonarQube, “Deploy SonarQube on Kubernetes,” 2021. https://docs.sonarqube.org/latest/setup/sonarqube-on-kubernetes/ (accessed Jan. 03, 2022).

** Halaman ini sengaja dikosongkan **

LAMPIRAN

Repositori percobaan penelitian terkait *template* infrastruktur.

https://github.com/rendyananta/jenkins-agent-k8s

https://github.com/rendyananta/k8s-infra-template

Repositori basis *container image* yang digunakan dalam *template* aplikasi

https://github.com/rendyananta/php-docker

Hasil implementasi aplikasi menggunakan templat

https://github.com/rendyananta/crater

https://github.com/rendyananta/urlhub