Implementasi *Continuous Integration* dan *Continuous Delivery* Pada Aplikasi myITS *Single Sign On*

Restu Agung Parama, Hudan Studiawan, dan Rizky Januar Akbar
Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember(ITS)
email: hudan@its.ac.id

***Abstrak*—Institut Teknologi Sepuluh Nopember mempunyai infrastruktur server *on-premise* atau bisa disebut dengan myITS *Cloud* yang dikelola oleh Direktorat Pengembangan Teknologi dan Sistem Informasi. Aplikasi myITS *Single Sign On* merupakan aplikasi yang digunakan ITS untuk bisa berinteraksi dengan aplikasi lainnya seperti *Classroom*, Akademik, dan Beasiswa di myITS. Dalam pengembangan myITS SSO, proses *delivery* dan *deployment* dilakukan secara manual, dimana *developer* atau pengembang melakukan *push* ke repositori kode yang kemudian dirilis ke dalam server. Pada proses CI/CD penulis menggunakan Jenkins yang akan melakukan *build* aplikasi ke dalam docker *image* yang kemudian digunakan di dalam server menjadi sebuah kontainer. Kemudian dalam serangkaian tes yang terjadi terdapat tes untuk mendeteksi masalah kualitas kode menggunakan SonarQube. Setelah itu aplikasi akan di-*deploy* ke dalam Kubernetes menggunakan Helm dan Rancher. Setelah dilakukannya uji coba, Jenkins dan SonarQube bisa diimpelementasikan kepada proses CI/CD dengan cara diintegrasikan. Aplikasi juga berhasil dikemas menjadi *image* dengan bantuan aplikasi Docker yang kemdian diunggah ke DockerHub. Dengan berhasilnya aplikasi di-*deploy* kedalam Kubernetes dan tidak ada *step pipeline* yang terlewat bisa menjadi bukti bahwa implementasi CI/CD pada aplikasi myITS *Single Sign On* sudah berhasil.**

***Kata Kunci*—CI/CD, Deployment, Docker, Jenkins, Kubernetes, SonarQube.**

I. PENDAHULUAN

CONTINUOUS Integration (CI) merupakan proses otomatisasi untuk pengembang. CI yang berhasil berarti penambahan atau perubahan kode pada aplikasi yang secara teratur dibuat, diuji, dan digabungkan ke repositori bersama. *Continuous Delivery* dan/atau *Continuous Deployment* (CD) merupakan konsep yang saling berkaitan yang terkadang digunakan secara bergantian. Keduanya berisi tentang otomatisasi lebih lanjut dari *pipeline*.

Continuous Delivery berarti perubahan yang dilakukan pengembang pada aplikasi yang secara otomatis diuji *bug*-nya dan diunggah ke repositori, kemudian dapat di-*deploy* di aplikasi utama. Tujuan dari *Continuous Delivery* adalah untuk memastikan usaha men-*deploy* kode baru seminimal mungkin. *Continuous Deployment* berarti pelepasan atau *release* secara otomatis perubahan yang dilakukan pengembang terhadap aplikasi dari repositori ke produksi yang nantinya akan digunakan oleh pengguna. Hal ini mengatasi masalah ketika tim operasi melakukan proses *deploy* manual yang akan menyebabkan aplikasi menjadi lambat.

Institut Teknologi Sepuluh Nopember (ITS) mempunyai infrastruktur *server on-premise* atau bisa disebut dengan myITS *cloud* yang dikelola oleh Direktorat Pengembangan Teknologi dan Sistem Informasi (DPTSI). Aplikasi myITS *Single Sign On* (SSO) merupakan aplikasi yang digunakan ITS untuk bisa berinteraksi dengan aplikasi lainnya seperti *Classroom*, Akademik, dan Beasiswa di myITS.

Dalam pengembangan myITS SSO, proses *delivery* dan *deployment* dilakukan secara manual, dimana developer atau pengembang melakukan *push* ke repositori kode yang kemudian dirilis ke dalam server. Namun hal ini dirasa sangat merepotkan terlebih ketika perubahan yang dirilis bersifat minor dan perubahan kode semakin sering terjadi. Berangkat dari hal tersebut, perlu diterapkan proses CI/CD pada pengembangan aplikasi myITS SSO agar pengembang bisa fokus pada aplikasi dan aplikasi dapat dirilis dengan cepat dan sudah melalui serangkaian tes pengujian tanpa perlu membuat aplikasi menjadi lambat karena proses *deploy* yang manual.

II. DAFTAR PUSTAKA

A. Jenkins

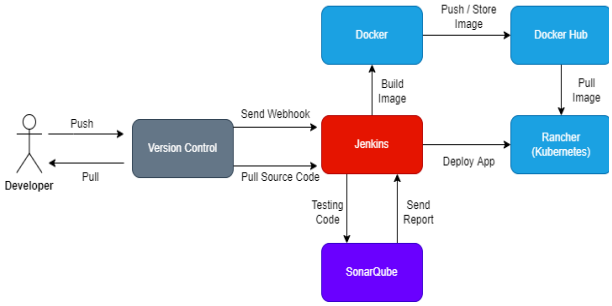
Jenkins adalah sebuah *open source automation* server untuk mengotomatisasi tugas-tugas di dalam proses *continuous integration* dan *continuous delivery* pada perangkat lunak. Jenkins merupakan aplikasi berbasis Java yang dapat dipasang dari repositori Ubuntu atau dengan mengunduh dan menjalankan file Web application ARchive (WAR), sebuah koleksi file yang sudah lengkap dan tinggal dijalankan di sebuah server.

B. CI/CD

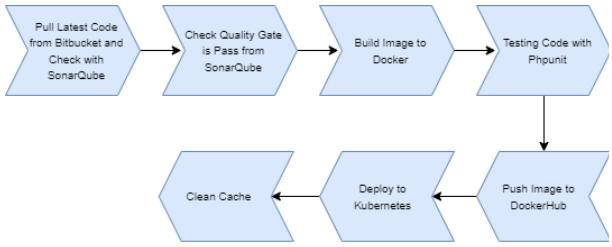
CI/CD adalah proses automatisasi yang dilakukan dalam pengiriman suatu versi dari lingkungan development ke dalam lingkungan produksi atau bisa disebut proses perilisan. Proses tersebut terdiri dari beberapa tahapan yang dijalankan atau dieksekusi secara berurutan, CI/CD merupakan akronim dari continuous integration dan continuous *delivery*.

Tujuan dari CI/CD adalah untuk meningkatkan penemuan cacat pada kode, meningkatkan produktivitas, dan menyediakan siklus perilisan yang lebih cepat. Prosesnya kontras dengan metode tradisional di mana kumpulan pembaruan perangkat lunak diintegrasikan ke dalam satu kumpulan besar sebelum menyebarkan versi yang lebih baru [1].

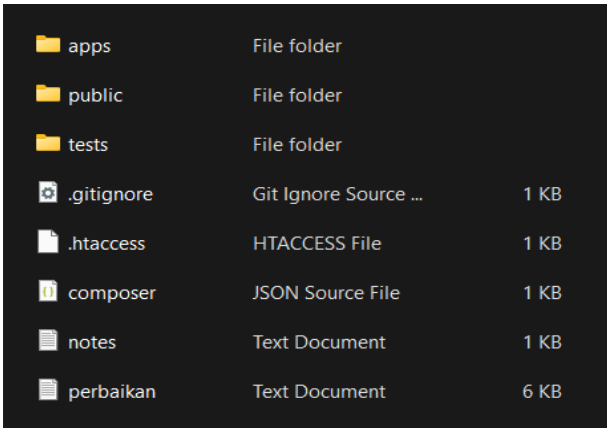
Continuous Integration (CI) merupakan proses otomatisasi untuk pengembang. CI yang berhasil berarti penambahan atau perubahan kode pada aplikasi yang secara teratur dibuat, diuji, dan digabungkan ke repositori bersama. *Continuous Delivery* dan/atau *Continuous Deployment* (CD) merupakan



Gambar 1. Arsitektur Alur Pipeline.



Gambar 2. Alur Step Pada Script.



Gambar 3. Direktori Phalcon.

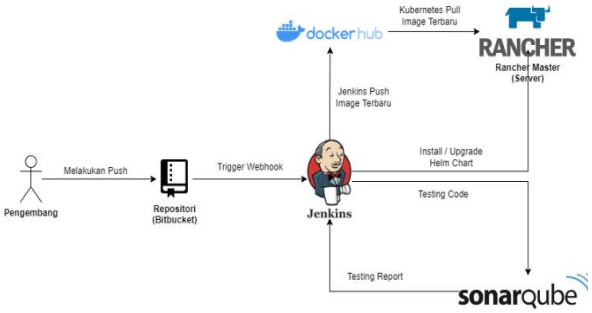
konsep yang saling berkaitan yang terkadang digunakan secara bergantian. Keduanya berisi tentang otomatisasi lebih lanjut dari pipeline.

Continuous Delivery berarti perubahan yang dilakukan pengembang pada aplikasi yang secara otomatis diuji *bug*-nya dan diunggah ke repositori, kemudian dapat di-*deploy* di aplikasi utama. Tujuan dari *Continuous Delivery* adalah untuk memastikan usaha men-deploy kode baru seminimal mungkin. *Continuous Deployment* berarti pelepasan atau *release* secara otomatis perubahan yang dilakukan pengembang terhadap aplikasi dari repositori ke produksi yang nantinya akan digunakan oleh pengguna. Hal ini mengatasi masalah ketika tim operasi melakukan proses *deploy* manual yang akan menyebabkan aplikasi menjadi lambat [2]

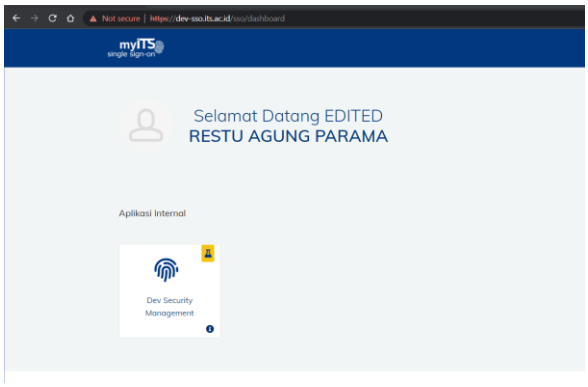
Pada dasarnya, *Continuous Delivery* dan/atau *Continuous Deployment* mulai saat CI berakhir. Jadi intinya CD mencakup *staging*, *testing* dan *deployment* kode CI. Sementara CI berada di bawah praktik pengembang. *Continuous Delivery* dan/atau *Continuous Deployment* sepenuhnya berada di ranah operasi.

C. SonarQube

SonarQube merupakan sebuah *tool* terkemuka yang dapat membantu para *developer* mengecek kualitas dan keamanan kode dari aplikasi. SonarQube tak hanya mendukung 27



Gambar 4. Diagram Komponen Uji Coba.



Gambar 5. Aplikasi setelah Perubahan lokal.

Gambar 6. Pipeline Berjalan.

bahasa pemrograman seperti Java, Phyton, dan Ruby, tetapi juga mendukung pengintegrasian CI/CD dalam DevOps. Selain itu, SonarQube mampu mendeteksi *bugs*, kerentanan, dan *code smell* yang ada di *codebase*. Dengan kapabilitas tersebut, tool ini dapat memastikan reliabilitas, keamanan, dan *maintainability* dari aplikasi dan *codebase* yang dimiliki [3].

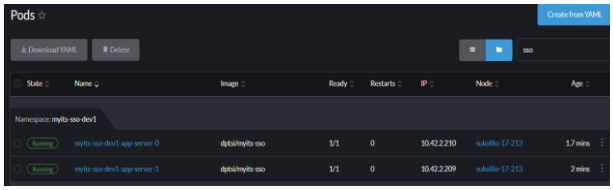
D. Docker

Docker merupakan *open-source* platform yang digunakan untuk pengembangan, shipping, dan menjalankan aplikasi. Docker memungkinkan sebuah aplikasi dapat dipisahkan dari infrastruktur utama karena Docker dibangun berdasarkan teknologi *container*. Berbeda dengan virtualisasi yang harus berjalan didalam infrastruktur utamanya.

III. METODOLOGI

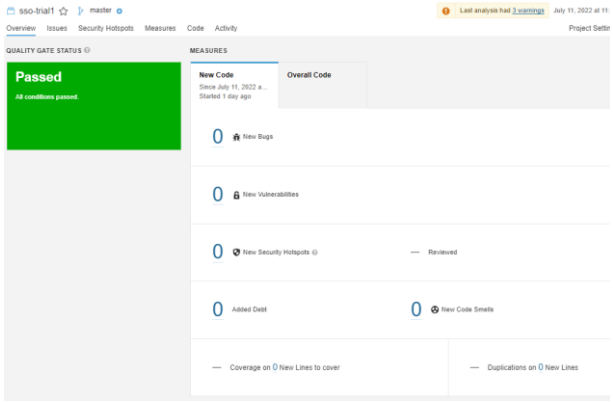
A. Analisis Permasalahan

Permasalahan yang diangkat pada tugas akhir ini adalah automatisasi proses *deploy* suatu aplikasi pada server menggunakan *pipeline* Jenkins yang didalamnya terdapat juga beberapa pengujian untuk mengecek hasil *commit* yang dilakukan tim pengembang. Proses automatisasi ini digunakan agar proses *update* atau *deploy* tidak dilakukan secara manual dengan masuk ke server aplikasi. Terdapat beberapa tahapan dalam *pipeline* yang akan diimplementasi yaitu pengujian kode dengan SonarQube, pengujian unit dengan PHPUnit, *build* dan *push image* Docker, dan *deploy* aplikasi di server.



Gambar 7. Aplikasi yang sudah dideploy.

Gambar 8. Aplikasi setelah perubahan di Production.



Gambar 9. Hasil SonarQube Lolos Pengujian pada New Code.

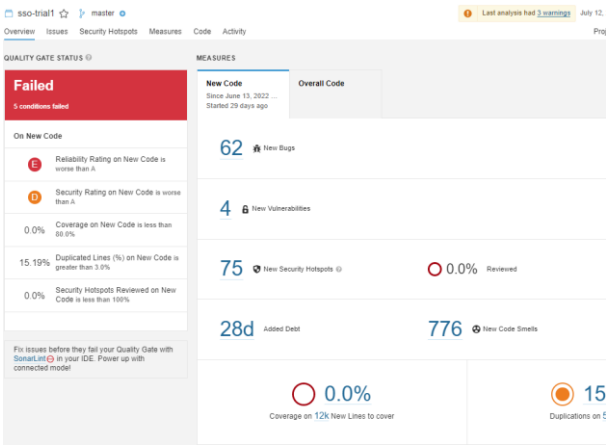
B. *Deskripsi Umum Perangkat Lunak*

Pada tugas akhir ini akan dilakukan integrasi repositori Bitbucket, SonarQube, dan Rancher melalui *pipeline* CI/CD dari lingkungan pengembang kepada lingkungan produksi *platform* myITS menggunakan Jenkins.

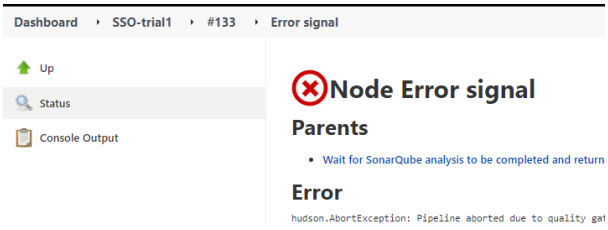
Dalam proses integrasi, Jenkins akan berperan sebagai integrator utama dari kedua lingkungan, proses utama dari *pipeline* dieksekusi pada server Jenkins. Proses berjalannya *pipeline* diawali dengan *programmer* yang melakukan *commit* kode sumber terbaru pada repositori Bitbucket, *webhook* Jenkins yang terhubung pada Bitbucket akan mendeteksi perubahan yang terjadi, setelah perubahan terdeteksi Jenkins akan menjalankan perintah untuk melakukan pengujian kode kepada SonarQube, lalu memberikan perintah melakukan *pull* dan *build image* terbaru dari aplikasi, kemudian menyimpannya pada *registry* DockerHub, selanjutnya Jenkins akan mengeksekusi perintah Helm untuk melakukan pemasangan ataupun perbaruan dari sebuah aplikasi mengikuti konfigurasi yang dibuat kedalam Rancher.

Pada tugas akhir ini studi kasus yang digunakan adalah aplikasi *Single Sign On* berbasis web menggunakan *framework* PHP Phalcon yang dijalankan oleh web server Nginx.

Gambar 10. Status Step Quality Gateway Pada Jenkins Berhasil.



Gambar 11. Hasil SonarQube Tidak Lolos Pengujian.



Gambar 12. Status Step Quality Gateway Pada Jenkins Error.

C. *Perancangan Rancher*

Pada komponen ini akan dirancang konfigurasi Rancher *deployment* untuk menjalankan aplikasi myITS *Single Sign On* pada lingkungan produksi. Rancher *deployment* terdiri dari kumpulan *pod*. *Pod* adalah kumpulan kontainer yang menjalankan *image*, disini *image* akan dijalankan sebagai kontainer yang dikemas dalam sebuah *pod*.

Selanjutnya, untuk menghubungkan antar *deployment* memerlukan *service*. *Service* berguna untuk mengekspos suatu port pada sebuah *pod*, hal ini dapat digunakan untuk menghubungkan *service* aplikasi myITS *Single Sign On* dengan aplikasi lainnya di dalam Rancher. Selanjutnya konfigurasi akan dikemas dalam sebuah paket yang bernama Helmchart, kemudian Helmchart akan dijalankan melalui Jenkins.

D. *Perancangan Pipeline*

Pipeline akan dibangun untuk mengintegrasikan aplikasi-aplikasi dari repositori hingga lingkungan Rancher. *Pipeline* terdiri dari beberapa komponen seperti Jenkins, Bitbucket, SonarQube, DockerHub, Helm, dan Rancher.

Jenkins menjadi komponen utama pada rancangan *pipeline* ini. Jenkins akan berjalan pada server yang nantinya akan terhubung dengan Bitbucket, SonarQube, DockerHub, dan Rancher. Jenkins akan menjalankan perintah yang ditulis pada *script* yang ada di dalam Jenkins. *Script* yang digunakan pada *pipeline* ini secara garis besar berisikan tahapan perintah yang dibagi menjadi beberapa tahapan yaitu:

Tabel 1.
Spesifikasi setiap komputer

Komponen	Spesifikasi
Nodes Kubernetes	
Nama	sukolilo-17-211
CPU	32 core
Sistem Operasi	Ubuntu 20.04.2 LTS
Memori	32 GB
Penyimpanan	64 GB
Konfigurasi Jaringan	External IP: 10.199.17.211 Internal IP: 10.199.17.211
Komputer	
CPU	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)
Sistem Operasi	Microsoft Windows 11 Home Single Language
Memori	16 GB
Penyimpanan	118 GB
Server	
CPU	Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz, 4 Core
Sistem Operasi	Ubuntu 20.04.2 LTS
Memori	4 GB
Penyimpanan	68 GB

Tabel 2.
Waktu Uji Coba Pipeline

Step Pipeline	Waktu		
	Min	Maks	Rata-rata
SonarQube Analysis	48 s	50 s	48,6 s
Quality Gate	2,12 s	2,41 s	2,284 s
Build	132 s	348 s	212,2 s
Test Phpunit	15 s	37 s	25,8 s
Push	38 s	61 s	45,4 s
Deploy	4 s	5 s	4,6 s
Cleanup	2 s	4 s	2,4 s
Total	241,12 s	507,41 s	340,984 s

1. Melakukan *pull* kode terbaru pada *branch* yang sudah ditentukan dan melakukan pengecekan kode sumber menggunakan SonarQube.
2. Melakukan pengecekan apakah hasil dari SonarQube lolos dengan bantuan *quality gate* SonarQube.
3. Melakukan *build* Dockerfile menjadi *docker image*.
4. Melakukan pengujian menggunakan PHPUnit.
5. Melakukan *push* ke repositori DockerHub.
6. Melakukan *deployment* aplikasi ke Kubernetes *cluster*.
7. Memberishkan *cache* dari proses *build*.

Pipeline dimulai ketika programmer atau pengembang melakukan *push* pada repositori Bitbucket. *Webhook* mendeteksi adanya perubahan dan Jenkins menjalankan *step pipeline* yang sudah didefinisikan pada Jenkinfile. Jenkins melakukan *pull code* terbaru dan mentrigger SonarQube untuk mengecek kode sumber yang baru di-*pull*. Kemudian *Quality Gate* akan cek lolos tidak tahap sebelumnya. *Image* di *build* pada *worker* docker, melakukan pengujian PHPUnit kemudian disimpan pada *registry* dockerhub. Setelah proses tersebut selesai Jenkins akan menjalankan perintah Helm *install* atau *upgrade* yang membuat klaster Kubernetes mengimplementasikan spesifikasi yang sudah didefinisikan pada Helmchart. Arsitektur alir *pipeline* tertera pada Gambar 1. Gambar 2 merupakan alur step pada *script*.

IV. IMPLEMENTASI

A. *Implementasi PHPunit*

PHPUnit atau *unit testing* adalah tes terkecil dalam serangkaian test untuk menguji sebuah fungsi atau kelas pada kode sumber pada aplikasi berbasis PHP. Dalam

Gambar 13. Lima Hasil Pipeline Yang Sudah Dijalankan.

implementasinya, telah dibuat 16 *test* pada sebuah file UpdatePasswordTest.php yang digunakan untuk mengetes apakah kelas pada aplikasi myITS *Single Sign On* untuk *update password* sudah bisa berjalan dsengan baik atau belum.

B. *Implementasi Dockerfile*

Pada tahap ini Dockerfile akan dibuat untuk menjalankan aplikasi myITS *Single Sign On* yang menggunakan *framework* Phalcon pada kontainer docker. Daftar direktori dari aplikasi myITS *Single Sign On* dapat dilihat pada Gambar 3. Aplikasi myITS *Single Sign On* dapat berjalan pada lingkungan yang sudah memiliki web server Nginx, Bahasa pemrograman PHP versi 7.4 dan Composer. Oleh karena itu *base image* yang digunakan disini adalah php:7.4-fpm. Selanjutnya *web server* Nginx dan *Composer* akan ditambahkan pada *base image* melalui Dockerfile. Direktori phalcon dapat dilihat pada Gambar 3.

C. *Implementasi Deployment*

Sesuai dengan tahap perancangan, lingkungan yang digunakan adalah klaster Kubernetes, aplikasi myITS *Single Sign* On beserta aplikasi pendukungnya akan dijalankan pada klaster Kubernetes. Rancher digunakan untuk manajemen klaster Kubernetes, dengan menggunakan Rancher pengguna dapat dengan mudah mengelola klaster Kubernetes melalui GUI.

Pada lingkungan produksi Helmchart digunakan untuk menyimpan konfigurasi dari aplikasi yang akan berjalan pada klaster Kubernetes, konfigurasi yang telah dikemas dalam Helmchart akan disimpan pada repositori pribadi milik DPTSI.

D. *Implementasi Pipeline*

Pada tahap ini Jenkins akan diintegrasikan dengan Bitbucket repositori, SonarQube, DockerHub, dan klaster Kubernetes. Untuk menghubungkan Bitbucket dengan Jenkins pengembang perlu menambahkan webhook Jenkins pada repositori Bitbucket. *Webhook* adalah salah satu sarana berkomunikasi antar aplikasi atau sistem dengan lebih efektif. Komunikasi yang terjadi saat menggunakan sarana *webhook* tersebut berbasis *event*. Artinya, transfer informasi baru terjadi saat ada input atau tindakan, yang kemudian memicu tindakan lain.

Untuk menambahkan *webhook* pada repositor, pengembang dapat mengakses menu *webhook* dari menu *settings*. Tambahkan alamat Jenkins server yang sudah dikemas dengan bantuan aplikasi pihak ketiga smee.io dengan URL https://smee.io/PXyoI3oNaXz5FYup. Smee.io merupakan aplikasi yang berguna untuk meneruskan pesan

dari *webhook* ke dalam lokal. Aplikasi pihak ketiga ini diperlukan karena server Jenkins yang digunakan berada di lingkungan ITS dan tidak bisa diakses begitu saja dari luar ITS. Oleh karena itu digunakan aplikasi smee.io ini agar Bitbucket bisa mengirim *webhook* ke server Jenkins tanpa perlu menggunakan VPN. Pengguna dapat memilih *events* apa saja pada yang akan melakukan trigger berjalannya *webhook*.

Setelah *webhook* ditambahkan, Bitbucket akan secara otomatis melakukan *request* ke alamat Jenkins, Bitbucket akan menampilan kode respon 200 jika *webhook* berhasil terhubung.

E. Implementasi SonarQube

Aplikasi SonarQube diintegrasikan dengan Jenkins untuk melakukan pengecekan kode sumber melalui *pipeline*. Untuk melakukan integrasi dengan Jenkins diperlukan Token dari Jenkins yang kemudian disimpan pada *Project Setting* > *Webhooks* SonarQube.

Dalam SonarQube terdapat *Quality Gates* yang berisi tentang aturan untuk mengecek apakah kode sumber lulus tes atau tidak. *Quality Gates* yang akan digunakan merupakan *default* dari sistem SonarQube.

V. UJI COBA DAN EVALUASI

Uji coba dilakukan dengan menggunakan total 1 komputer, 1 server, dan 1 Nodes Kubernetes, dimana 1 komputer bertindak sebagai komputer pengembang, dan 1 klaster Kubernetes bertindak sebagai lingkungan produksi. Spesifikasi setiap komputer dapat dan klaster Kubernetes dilihat pada Tabel 1.

A. Skenario Pengujian

Uji coba ini dilakukan untuk menguji apakah fungsionalitas sistem sudah sesuai dan bekerja seperti yang seharusnya. Uji coba akan didasarkan pada beberapa skenario untuk menguji kesesuaian respon sistem. Skenario pengujian dibedakan menjadi 2 yaitu:

1. Uji Fungsionalitas bertujuan untuk memastikan fitur-fitur yang ada pada sistem telah berjalan sebagaimana mestinya
2. Uji Performa bertujuan untuk membuktikan apakah infrastruktur yang dibuat dapat mempercepat proses *deployment* aplikasi.

Pada uji coba ini, pengembangan akan dilakukan pada 1 komputer dengan sistem operasi Windows 11. Selanjutnya pengembang akan melakukan *push* repositori Bitbucket. Pengembang akan melakukan perubahan pada repositori, kemudian akan dilihat apakah perubahan tersebut berhasil berjalan pada lingkungan produksi. Diagram komponen dari alur uji coba yang ditunjukkan pada Gambar 4.

B. Analisis Hasil Uji Coba

Pada bagian ini, akan dibahas mengenai hasil uji coba sistem sesuai skenario yang telah didefinisikan. Hasil uji coba dibagi menjadi dua, hasil uji fungsionalitas dan hasil uji performa.

1) Skenario Uji Fungsionalitas Pipeline

Pada uji ini akan pengembang melakukan perubahan kode pada lingkungan pengembangan, perubahan akan dilakukan pada file id.php. File ini merupakan kumpulan kata-kata yang digunakan dalam aplikasi myITS *Single Sign On* agar bisa menggunakan fitur *translate*, di sini dilakukan peubahan pada kata '*welcome'* yang sebelumnya 'Selamat Datang' menjadi 'Selamat Datang EDITED'. Hal ini dapat dilihat pada Gambar 5. Setelah pengembang melakukan *push* Bitbucket mengirimkan *webhook* kepada Jenkins. Pengiriman *webhook* ini akan membuat Jenkins melakukan *pipeline job*-nya.

Gambar 6 adalah *pipeline* yang menunjukan push terbaru pada repositori Bitbucket akan mentrigger berjalannya pipeline dan melakukan pembaruan aplikasi pada lingkungan produksi.

Pipeline berlanjut sampai berakhir dan aplikasi akan di-*deploy* kedalam Rancher yang ditunjukkan pada Gambar 7. Bisa dilihat tampilan aplikasi yang telah diperbarui pada Gambar 8 setelah melewati proses CI/CD dengan Jenkins.

2) Skenario Uji Fungsionalitas SonarQube

Hasil uji fungsionalitas SonarQube yang menunjukan apabila *quality gateway*-nya lulus dan tidak, *pipeline* akan berhenti beroperasi apabila kode sumber yang dicek tidak lolos dari pengujian SonarQube.

Pada Gambar 9 dan Gambar 10 dapat dilihat hasil pengecekan SonarQube pada kode sumber terbaru telah *passed* dan pada Jenkins menerima informasi bahwa *Quality Gate is "OK"*, sehingga *pipeline* bisa meneruskan *job* ke *step* berikutnya.

Pada Gambar 11 bisa diketahui bahwa SonarQube sudah mengecek kode sumber terbaru dan hasilnya *failed.* Hal ini menyebabkan Jenkins menerima informasi bahwa *Quality Gate failure* dan *error* yang dapat dilihat pada Gambar 12, sehingga *pipeline* akan terhenti dan tidak bisa melanjutkan ke *step* berikutnya.

3) Skenario Uji Performa

Gambar 13 adalah hasil dari uji coba untuk melihat berapa waktu yang dibutuhkan pada satu proses pipeline. Setelah dilakukan 5 kali pengujian, rata rata berjalannya *pipeline* dibutuhkan waktu 355.4 detik atau sekitar 5.93 menit. Untuk detailnya bisa dilihat pada Tabel 2.

VI. KESIMPULAN

Kesimpulan yang diperoleh dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

Telah diimplementasikan CI/CD untuk melakukan proses *deployment* aplikasi myITS *Single Sign On.* Proses *pull*, *build*, *push*, dan *deploy* berjalan otomatis setelah pengembang melakukan pembaruan terhadap repositori.

Telah diimplementasikan Docker sebagai standar untuk mengemas aplikasi myITS *Single Sign On* menjadi *image* serta melakukan virtualisasi pada proses *build* hingga *deploy* pada klaster Kubernetes.

Telah diimplementasikan Jenkins untuk mengintegrasikan lingkungan kerja pengembang dengan lingkungan produksi yang berjalan pada klaster Kubernetes, Jenkins sebagai pusat integrasi dan tempat *pipeline* berjalan, SonarQube sebagai analisis kode sumber, dan Kubernetes sebagai tempat aplikasi di-*deploy* terhubung menggunakan plugin dan *script* pada server Jenkins.

Telah diimplementasikan SonarQube sebagai alat pengecekan kode sumber pada proses *pipeline*. Menggunakan bantuan *Quality* Gateway untuk melakukan

pengecekan lolos tidaknya kode sumber dari pengecekan SonarQube.

Telah dilakukan pengujian performa dan pengujian fungsional pada implementasi CI/CD. Didapatkan hasil pengujian fungsional yang baik dengan semua step *pipeline* berjalan dengan normal. Untuk pengujian performa didapatkan waktu rata rata berjalannya *pipeline* dibutuhkan waktu 355.4 detik atau sekitar 5.93 menit

Saran yang diberikan dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut: sistem dapat dikembangkan dengan melakukan optimasi saat melakukan *build image* untuk mempersingkat durasi *build* pada *pipeline*.

Pada tugas akhir ini, sistem hanya dapat mengakomodir perubahan pada aplikasi tidak pada *database*. Untuk pengembangan kedepannya sistem dapat diimplementasikan untuk mengakomodir perubahan yang terjadi pada *database*.

Pada tugas akhir ini, sistem tidak dapat mengakomodir apabila terdapat kegagalan pada *pipeline job*, untuk pengembang kedepannya sistem dapat mengatasi hal tersebut.

DAFTAR PUSTAKA

[1] B. El Khalyly, A. Belangour, M. Banane, and A. Erraissi, "A New Metamodel Approach of CI/CD Applied to Internet of Things Ecosystem," in *IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*, 2020, pp. 1--6.

[2] I. Red Hat, "What is CI/CD?," *Red Hat, Inc.*, 2022. .

[3] SonarSource, "SonarQube Documentation," *SonarSource S.A*, 2022.