IMPLEMENTASI *CONTINUOUS INTEGRATION/CONTINUOUS DELIVERY (CI/CD)* PADA *PERFORMANCE TESTING* DEVOPS

Jaeni [1] , **Nicko Aji S.** [2] , **Arif Dwi L.** [3]

[1] *Manajemen Informatika Universitas AMIKOM Yogyakarta*
[2] *Teknik Informatika Universitas AMIKOM Yogyakarta*
[3] *Informatika Universitas AMIKOM Yogyakarta*
email : jaeni@amikom.ac.id[1], nicko.10@students.amikom.ac.id [2] , arif.laksito@amikom.ac.id [3]

Abstraksi

Secara umum proses pengembangan aplikasi terdiri dari proses perancangan, pengujian, dan deployment. Pengembangan dengan cara tradisional seringkali menimbulkan masalah seperti keterlambatan aplikasi dan kualitas produknya. Hal ini dapat terjadi karena tim operasional harus menunggu developer agar menyelesaikan proses uji coba terlebih dahulu sebelum aplikasi tersebut dapat dirilis dan diguankan. Selain itu, setelah proses pengujian selesai, sering didapati adanya konflik akibat environment yang berbeda, yang memperlambat proses rilis. Keterlambatan ini membuat klien kecewa dan memberikan efek yang tidak baik bagi perusahaan. Continuous Integration / Continuous Delivery (CI/CD) dapat dijadikan solusi untuk memecahkan masalah ini. CI/CD dapat menjadi jembatan antara tim pengembang dan tim operasional..

***Kata Kunci*:**
Continuous Integration, Continuous Delivery, Gitops, Perfomance testing

Abstract

The traditional development process used to be a repetitive cycle that consisted of developing, testing and deploying. Traditional development often raises problems such as dependency between the development and operation teams. This is because the operational team must wait for the developer to complete the testing before releasing the application. After the testing process is complete, conflicts are often found due to different development environments, which slows down the release process. These problems can cause delays in delivering the release of applications to clients. This delay disappoints the client and harms our company. Continuous Integration/Continuous Delivery (CI/CD) methods can be used as a solution to solve these problems. CI/CD can be a bridge between the development and the operational team.

***Keywords*:**
Continuous Integration, Continuous Delivery, Gitops, Perfomance testing

Pendahuluan

Pengembangan dan distribusi aplikasi dapat menimbulkan beberapa masalah terutama terkait dengan efektivitas dan efisiensi. Pada awalnya, pengembangan aplikasi dilakukan dengan prosedur tradisional, proses perancangan dan pengujian dijalankan secara terpisah oleh tim pengembang dan penguji [1]. Secara umum, proses pengembangan aplikasi terdiri dari proses perancangan, pengujian, dan *deployment.* Pada saat proses perancangan dan pengujian selesai, aplikasi akan dikirim ke tim operasi untuk melakukan *deploymen.* Namun, prosedur tersebut harus dilakukan berulang-ulang ketika adanya pembaruan. Teknik ini dapat menimbulkan masalah efektivitas kerja dan efisiensi waktu [2].

Dengan adanya masalah yang timbul tersebut dapat menyebabkan keterlambatan dalam proses perilisan aplikasi dan dapat menimbulkan penurunan kualitas produk dan kekecewaan pengguna. Oleh karena itu, dibutuhkan implementasi *devops* yang dapat distandarisasi, baik standarisasi dari bentuk alur kerja (*pipeline*), hingga penyetaraan *environment*. Hal ini diharapkan dapat menghilangkan adanya dependensi antara tim pengembang dan operasi.

Tinjauan Pustaka

Beberapa implementasi CI/CD untuk mempercepat proses integrasi dan *delivery* perangkat lunak kepada pengguna telah banyak dilakukan dengan pendekatan yang berbeda.

Danur Wijayanto (2021), meneliti tentang implementasi CI/CD untuk mempercepat proses integrasi dan *delivery* perangkat lunak kepada *user*. Fokus peneltian menggunakan *Process Manager 2* (PM2) untuk implementasi CI/CD pada Sistem Informasi Akademik (SIAKAD). Hasil penelitian ini

menunjukkan implementasi CI/CD menggunakan PM2 dapat mempercepat proses integrasi dan *delivery* perangkat lunak kepada pengguna [3].

Arachchi dkk (2018), meneliti tentang penerapan otomatisasi CI/CD *pipeline* dengan *agile software project management. D*ari penelitian ini didapatkan keterangan CI/CD dapat meningkatkan efisiensi dalam proses perilisan dan produktivitas sistem. CI/CD *pipeline* dapat memudahkan proses uji coba dengan membuatnya menjadi otomatis sehingga lebih efisien. Otomatisasi yang diberikan oleh CI/CD *pipeline* memberikan hasil yang sesuai dengan target yang diharapkan[1]

Sedangakan Andrian Alpery dkk (2021), melakukan penelitian mengenai aplikasi CI/CD dalam pengembangan aplikasi *web* yang menggunakan *framework Codeigniter*. Penelitian ini bertujuan untuk mengimplementasikan dan menganalisis kualitas, waktu yang dibutuhkan, dan proses otomatisasi pada proses pengembangan aplikasi *web*. Berdasarkan hasil pengujian waktu yang dibutuhkan untuk melakukan *deployment* adalah selama 1 menit 58 detik, dengan tingkat keberhasilan sebesar 90%, dan 78 *bugs*, dan selama melakukan pengujian hanya sekali terjadinya kegagalan dalam proses *deployment* [4]

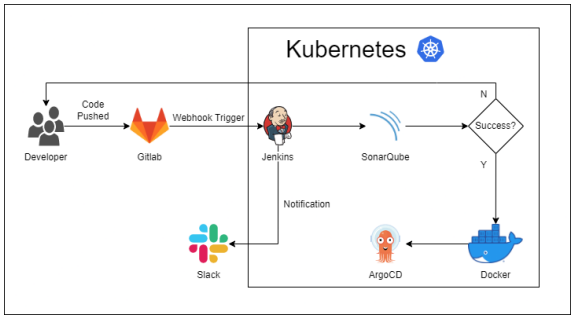
Pada penelitian lain, beberapa makalah yang berfokus pada tantangan organisasi yang dihadapi selama integrasi dalam implementasi CI/CDE/CD. Beberapa contoh adalah mengubah budaya organisasi, perluasan kualifikasi pengembang dan tim operasi, untuk "memecahkan" bisnis, sehingga bisnis benar-benar percaya pengeluaran uang akan berdampak pada laba atas investasi (ROI) [5].

Pada paper lain menjelaskan *continuous deployment* untuk *mobile applications* pada Facebook. Facebook menjelaskan prosedur pengembangan yang sebenarnya di dalam perusahaan, bukan hanya menyentuh aspek perangkat seluler, tetapi sistem penyebaran yang dibangun Facebook. Manipulasi dengan kode sumber, melakukan pemeriksaan keamanan, ruang terpisah dengan kamera yang berfokus pada analisis pembaruan pada uji coba perubahan dengan *deployment automation* CI/CDE/CD bukanlah aplikasi alat yang sederhana, tetapi tantangan luar biasa untuk mengubah pendekatan keseluruhan untuk pengembangan, memastikan pengembang dan operasi mulai berpikir secara berbeda, benar-benar mengubah cara berpikir daripa tool yang digunakan [6]

.

Metode Penelitian

Pada penelitian ini mengimplementasikan metode *agile* [7] sebagai kerangka kerja dan konsep CI/CD menggunakan *tools* Jenkins, Docker, Kubernetes dan ArgoCD. *Continuous Integration/ Continuous Delivery (CI/CD)* memberikan beberapa kelebihan seperti proses *automation testing*, mempercepat proses umpan balik antara tim pengembang dan tim operasi, dan mengurangi konflik *environment* antara tim pengembang dan tim operasi [8]. Penerapan praktik CI/CD ini dapat meningkatkan efisiensi dalam proses pengembangan dan produktivitas dari pengembang. Dengan adanya penerapan CI/CD fase uji coba dan rilis diterapkan dengan otomatisasi, sehingga pengembang diberikan kemudahan dan diharapkan kinerjanya akan meningkat [9]



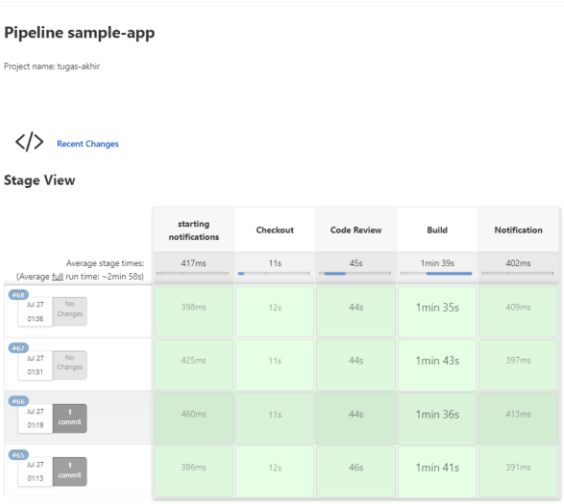
Gambar 1 Diagram *flowchart* system

Dari gambar 1 alur kerja sistem yang dibuat adalah *developer* melakukan *push* ke dalam repository di Gitlab. Proses push tersebut akan ditangkap oleh *webhook* untuk melakukan trigger otomatisasi di dalam Jenkins. Jenkins akan menjalankan *code review* dan unit test ke dalam Sonarqube. Apabila proses ini gagal, maka Jenkins tidak akan melanjutkan *build.*

Sedangkan jika code tersebut berada di atas standar dari Sonarqube, maka Jenkins akan menjalankan *build image* menggunakan Docker. Setelah itu, image yang telah dibuat akan dikirim ke ArgoCD untuk dilakukan *deployment.* Ketika seluruh proses berhasil, maka Jenkins akan menjalankan *trigger* untuk mengirim pesan notifikasi ke *channel* di dalam Slack. Dengan penggunaan Docker sebagai *build image*, maka diharapakan akan terjadi standarisasi *environment* yang dapat mengeliminasi adanya kesalahan konfigurasi ketika *deployment*

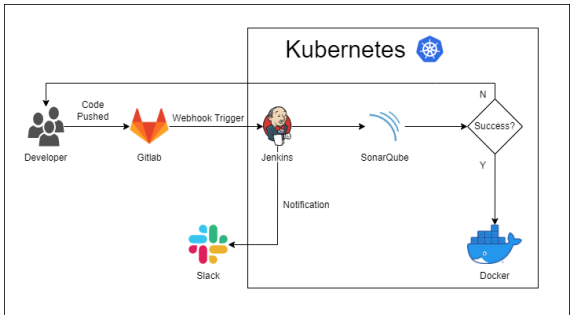
Hasil dan Pembahasan

Dalam implementasi Jenkins untuk otomatisasi langkah-langkah (*pipeline*) dari aplikasi. *Pipeline* ini memiliki beberapa *stage* (tahapan) yang dimulai dari mengirim notifikasi untuk *build*, dilanjutkan dengan *checkout code* dari *Version Control System* (Gitlab), lalu *code review* menggunakan Sonarqube,



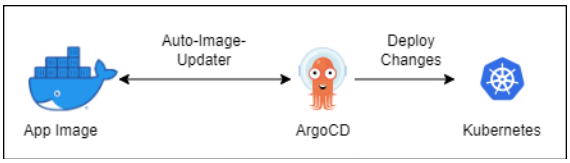
Gambar 2 Implemntasi Jenkins

Apabila *code review* lolos dilanjutkan dengan *build image* menggunakan Docker dan dilanjutkan mengirim notifikasi *build* selesai dengan status selesai. Seluruh langkah di atas berjalan secara otomatis tanpa campur tangan tim pengembang maupun operasi. Dengan penggunaan jenkins dan juga docker sebagai *platform* otomatisasi dan *virtualization*, maka alur kerja dapat distandarisasi, baik standarisasi dari bentuk alur kerja (*pipeline*), hingga penyetaraan *environment*. Hal ini dapat menghilangkan adanya dependensi antara tim pengembang dan operasi.



Gambar 3 Diagram *Continuous Integration*

Pada gambar 3 menggambarkan alur kerja dari *Continuous Integration* menggunakan Jenkins adalah pengembang melakukan perubahan kode, lalu dilanjutkan dengan melakukan *push* ke dalam Git *repository*. Jenkins akan menangkap perubahan dari kode menggunakan *webhook*, setelah itu dilanjutkan dengan menjalankan *pipeline* dengan beberapa *stage* (tahapan) yang dimulai dari mengirim notifikasi untuk *build*, dilanjutkan dengan *checkout code* dari *version control system* (Gitlab), lalu *code review* menggunakan Sonarqube, apabila *code review* lolos dilanjutkan dengan *build image* menggunakan *docker* dan dilanjutkan mengirim notifikasi *build* selesai dengan status selesai.



Gambar 4 Diagram *Continuous Delivery*

Dari gambar 4, menjelasakn alur kerja *Continuous Delivery* menggunakan ArgoCD adalah ArgoCD melakukan pengecekan *hash* dari *image* secara berkala ke Docker *registry* menggunakan fitur *auto-image-updater*, apabila *hash image* dari aplikasi berbeda dengan yang ada di *registry* ArgoCD akan melakukan *deployment* ulang aplikasi agar aplikasi dapat secara otomatis berada di versi terbaru.

Time-based metrics

Metrik ini mengukur waktu yang dibutuhkan oleh proses CI/CD dan tradisional dalam melakukan seluruh tahapan proses seperti *notifications*, *testing*, *build*, dan *deployment* pada pengembangan aplikasi *web* [10] .

Berdasarkan hasil pengujian menggunakan proses CI/CD, didapati bahwa rata-rata total *build time* bersifat stagnan di angka 3 menit 13 detik hingga 4 menit 2 detik. Hal ini terjadi dikarenakan *pipeline* untuk pembuatan aplikasi bersifat statis dan jarang berubah-ubah.

Sedangkan berdasarkan proses tradisional mendapatkan hasil rata-rata 11 menit 57 detik. Dibanding dengan proses CI/CD tadi hasil ini cukup jauh perbandingannya, ditambah dari masing-masing percobaan dengan proses tradisional terdapat jarak waktu yang cukup beragam. Di dalam percobaan dengan proses tradisional terdapat beberapa faktor yang mempengaruhi lamanya durasi waktu, seperti manual *testing* dan kemungkinan terjadinya *human error* seperti kurang telitinya pengembang saat mengecek kode atau salah mengunggah *file* saat proses mengunggah ke *server*

Berdasarkan hasil percobaan, didapatkan hasil bahwa proses *deployment* menggunakan CI/CD mampu mempersingkat proses *deployment* hingga 8 menit 43 detik. Hal ini dikarenakan proses *deployment* dengan CI/CD memiliki standarisasi yang telah dibuat sebelumnya lalu berjalan secara otomatis dan menggunakan *tools* jadi hanya ada sedikit campur tangan dari manusia saat terjadi proses *deployment.* Berbeda dengan proses tradisional yang sedari awal sudah bergantung dengan campur tangan manusia dari proses *testing* sampai proses *upload* ke *serve.*

Quality-based metrics

Pengukuran kualitas dengan *Quality-based metrics* [11] dilakukan dengan memperhatikan 2 aspek sebagai berikut. yakni *Test Past Rate dan Number of Bugs*

Berdasarkan pengujian keberhasilan dengan menggunakan proses CI/CD adalah 80%. Hasil ini didapatkan dari 5 kali percobaan yang ada , terdapat 4 percobaan yang berhasil dan 1 percobaan yang gagal karena tidak lolos *quality gate* dari Sonarqube. Hasil dari tidak lolos *quality gate* ini membuat proses *pipeline* berhenti lalu memberi pemberitahuan kepada pengembang untuk segera memperbaiki kodenya dari *bug* yang ada.

Sedangkan tingkat keberhasilan dengan menggunakan proses tradisional adalah 60%. Hasil ini didapatkan dari 5 kali percobaan yang ada , terdapat 2 percobaan yang gagal karena penguji kode menemukan adanya *bug* lalu 3 percobaan sisanya mendapatkan hasil sukses *deployment*.

Berdasarkan dari observasi adalah ketika menggunakan proses CI/CD, *bug-bug* yang terdapat di dalam aplikasi dapat secara langsung ditangkap pada proses *code review* di dalam *pipeline*, hal ini merupakan keunggulan dari proses CI/CD dikarenakan jika *bug-bug* itu dapat dideteksi sebelum masuk ke dalam aplikasi yang sudah dirilis, maka akan dapat segera dikoreksi oleh pihak pengembang. Hasil temuan *bug* dari proses CI/CD berjumlah 3 yang ditemukan pada percobaan pertama. Untuk sisa 4 percobaan sudah tidak ditemukan *bug* lagi dan proses CI/CD berjalan dengan sukses.

Sedangkan pada proses tradisional percobaan pertama & kedua mendapatkan hasil gagal karena terdapat *bug* di dalamnya yang diketahui setelah melalui manual *testing*. Manual *testing* ini dilakukan oleh pengembang yang bertanggung jawab untuk meninjau kode atau disebut juga penguji. Penemuan *bug* pada percobaan pertama oleh penguji lalu disampaikan kepada pengembang yang bertanggung jawab atas pembuatan kode tersebut untuk segera diperbaiki, begitupun juga penemuan *bug* pada percobaan kedua.

Dengan 5 kali percobaan pada masing-masing proses CI/CD dan tradisional, hasil yang didapatkan adalah proses CI/CD memberikan hasil yang lebih teliti dengan penemuan 3 *bug* hanya dengan 1 kali proses percobaan, sedangkan proses tradisional hanya menemukan 2 *bug* dengan 2 kali percobaan. Total *bug* yang terdapat di dalam kode adalah 3 namun dengan proses tradisional hanya mampu mendapatkan hasil 2 *bug*, karena proses *testing* dilakukan secara manual oleh manusia. Hasil ini membuktikan bahwa terjadi *human error* pada proses tradisional

Kesimpulan dan Saran

Dengan implementasi CI/CD alur kerja dapat distandarisasi, baik standarisasi dari bentuk alur kerja (*pipeline*) maupun penyetaraan *environment*. Sehingga hal ini dapat menghilangkan adanya dependensi antara tim pengembang dan operasi.

Dari hasil peneltian didapatkan kesimpulan bahwa proses *deployment* menggunakan CI/CD mampu mempersingkat proses dan meningkatkan kinerja. Selain itu CI/CD memberikan hasil yang lebih teliti dengan penemuan *bug* pada pengujian tersebut.

Daftar Pustaka

[1] S. A. I. B. S. Arachchi and I. Perera, “Continuous integration and continuous delivery pipeline automation for agile software project management,” *MERCon 2018 - 4th Int. Multidiscip. Moratuwa Eng. Res. Conf.*, pp. 156–161, 2018, doi: 10.1109/MERCon.2018.8421965.

[2] C. Trubiani, P. Jamshidi, J. Cito, W. Shang, Z. M. Jiang, and M. Borg, “Performance issues? Hey DevOps, mind the uncertainty,” *IEEE Softw.*, vol. 36, no. 2, pp. 110–117, 2019, doi: 10.1109/MS.2018.2875989.

[3] Danur Wijayanto, Arizona Firdonsyah, and Faisal Dharma Adhinata, “Implementasi Continous Integration/Continous Delivery Menggunakan Process Manager 2 (Studi Kasus: SIAKAD Akademi Keperawatan Bina Insan),” *Teknika*, vol. 10, no. 3, pp. 181–188, 2021, doi: 10.34148/teknika.v10i3.400.

[4] A. Alpery and M. A. F. Ridha, “Implementasi CI-CD dalam Pengembangan Aplikasi Web Menggunakan Docker dan Jenkins,” *9th Appl. Bus. Eng. Conf.*, pp. 287–296, 2021.

[5] M. Shahin, M. A. Babar, M. Zahedi, and L. Zhu, “Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges,” *Int. Symp. Empir. Softw. Eng. Meas.*, vol. 2017-Novem, no. November, pp. 111–120, 2017, doi: 10.1109/ESEM.2017.18.

[6] C. Rossi, E. Shibley, S. Su, K. Beck, T. Savor, and M. Stumm, “Continuous deployment of mobile software at facebook (showcase),” *Proc.*

ACM SIGSOFT Symp. Found. Softw. Eng., vol. 13-18-November-2016, pp. 12–23, 2016, doi: 10.1145/2950290.2994157.

[7] I.-C. Donca, O. P. Stan, M. Misaros, D. Gota, and L. Miclea, "Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects," *Sensors*, vol. 22, no. 12, p. 4637, 2022, doi: 10.3390/s22124637.

[8] L. Zhu, L. Bass, and G. Champlin-Scharff, "DevOps and Its Practices," *IEEE Softw.*, vol. 33, no. 3, pp. 32–34, 2016, doi: 10.1109/MS.2016.81.

[9] "A perspective on Performance Testing in DevOps environment".

[10] "Evaluation of different runner set - ups for CI / CD pipelines," 2022.

[11] A. Mishra and Z. Otaiwi, "DevOps and software quality: A systematic mapping," *Comput. Sci. Rev.*, vol. 38, p. 100308, 2020, doi: 10.1016/j.cosrev.2020.100308.