HOAQ: JURNAL TEKNOLOGI INFORMASI, Volume 17 Nomor 2, Desember 2026, p400-407
p-ISSN: 2337-5280, e-ISSN: 26207427
DOI: https://doi.org/10.52972/hoaq.vol17no2

PENERAPAN DEVOPS DAN CI/CD MENGGUNAKAN GITHUB WEBHOOK PADA SISTEM PEMANTAUAN KESEHATAN KARYAWAN DI PT XYZ

Muhammad Salman Alfarisi[1], Esi Putri Silmina[2]

[1,2] Program Studi Teknologi Informasi, Universitas Aisyiyah Yogyakarta
Jl. Siliwangi, Sleman – Daerah Istimewa Yogyakarta, Indonesia
Email: [1]alfarizim168@gmail.com, [2]esiputrisilmina@unisayogya.ac.id

ABSTRAK

Penelitian ini bertujuan untuk mengoptimalkan proses pengembangan dan *deployment* pada sistem pemantauan kesehatan karyawan di PT XYZ melalui penerapan metode DevOps dan *Continuous Integration/Continuous Deployment* (CI/CD) menggunakan *GitHub webhook*. Meskipun sistem pemantauan kesehatan telah tersedia, proses integrasi, pengujian, dan *deployment* sebelumnya belum sepenuhnya terotomatisasi, sehingga berpotensi menimbulkan keterlambatan pembaruan, kesalahan teknis, dan ketidakkonsistenan lingkungan sistem. Metode penelitian yang digunakan adalah rekayasa perangkat lunak dengan pendekatan deskriptif dan eksperimental melalui implementasi siklus DevOps yang terintegrasi dengan pipeline CI/CD. *GitHub webhook* dikonfigurasi sebagai mekanisme pemicu otomatis untuk menjalankan proses *build*, pengujian, dan *deployment* setiap terjadi perubahan kode pada repositori. Hasil penelitian menunjukkan peningkatan signifikan pada metrik *deployment*, yaitu *Build Success Rate* meningkat dari 70% menjadi 90%, *Build Duration* berkurang dari 20 menit menjadi 10 menit. Selain itu, penerapan mekanisme otomatis ini juga meningkatkan konsistensi dan stabilitas sistem dalam mendukung pengembangan berkelanjutan. Penelitian ini memberikan kontribusi ilmiah berupa evaluasi kuantitatif efektivitas pendekatan CI/CD berbasis *GitHub Webhook* sebagai alternatif *lightweight* terhadap *platform* CI/CD konvensional, serta menyediakan data perbandingan metrik *deployment* sebelum dan sesudah implementasi yang dapat dijadikan acuan bagi penelitian selanjutnya.

Kata kunci: DevOps, CI/CD, *GitHub webhook*, metrik DORA, *deployment* otomatis

ABSTRACT

This study aims to optimize the development and deployment processes of the employee health monitoring system at PT XYZ through the implementation of DevOps and Continuous Integration/Continuous Deployment (CI/CD) using GitHub webhooks. Although the health monitoring system was already available, the integration, testing, and deployment processes had not been fully automated, which potentially caused update delays, technical errors, and inconsistencies in system environments.The research method employed was software engineering with descriptive and experimental approaches by implementing the DevOps lifecycle integrated with a CI/CD pipeline. GitHub webhooks were configured as an automatic triggering mechanism to execute build, testing, and deployment processes whenever changes occurred in the code repository.The results indicate a significant improvement in deployment metrics, with the Build Success Rate increasing from 70% to 90% and the Build Duration decreasing from 20 minutes to 10 minutes. Furthermore, the implementation of this automation mechanism enhanced system consistency and stability in supporting continuous development. This study provides a scientific contribution through a quantitative evaluation of the effectiveness of a GitHub Webhook-based CI/CD approach as a lightweight alternative to conventional CI/CD platforms, along with comparative deployment metric data before and after implementation that can serve as a reference for future research.

Keywords: DevOps, CI/CD, GitHub webhook, DORA metrics, automated deployment

1. PENDAHULUAN

Kesehatan karyawan memiliki peranan penting dalam menjaga produktivitas dan keberlangsungan operasional perusahaan. PT XYZ telah memiliki sistem informasi untuk mendukung pendataan kesehatan karyawan, namun sistem tersebut belum sepenuhnya menerapkan mekanisme otomatis dalam proses pengelolaan dan pengembangannya, khususnya pada tahap *build*, pengujian, dan *deployment*. Proses *deployment* yang masih bergantung pada intervensi manual berpotensi menimbulkan berbagai permasalahan, seperti keterlambatan pembaruan sistem, risiko kesalahan teknis, perbedaan konfigurasi lingkungan, serta menurunnya efisiensi

Copyright (c) HOAQ (High Education of Organization Archive Quality) : Jurnal Teknologi Informasi

pemeliharaan aplikasi. Kondisi serupa juga ditemukan pada sistem informasi lain yang belum menerapkan otomasi, di mana keterbatasan tersebut berdampak pada kualitas dan keandalan sistem [1], [2].

Pendekatan DevOps merupakan metode pengembangan perangkat lunak yang mengintegrasikan proses pengembangan *(development)* dan operasional *(operations)* untuk meningkatkan kolaborasi tim, efisiensi kerja, serta kualitas sistem yang dihasilkan. DevOps menekankan penerapan otomatisasi, komunikasi lintas fungsi, dan integrasi berkelanjutan dalam siklus pengembangan perangkat lunak. Berbagai penelitian menunjukkan bahwa penerapan DevOps mampu meningkatkan kinerja sistem di berbagai bidang. Penerapan DevOps pada sistem akademik [3], sistem e-Skrining Covid-19 [4] , sistem monitoring pembimbing tugas akhir [5], dan sistem informasi kepegawaian [6] terbukti mampu mempercepat proses pengembangan serta meningkatkan keandalan aplikasi. Implementasi DevOps didukung oleh penerapan *Continuous Integration dan Continuous Deployment* (CI/CD). CI/CD memungkinkan setiap perubahan kode yang dilakukan oleh pengembang dapat diintegrasikan, diuji, dan *zero downtime deployment* [7]. Dengan mekanisme ini, kesalahan dapat dideteksi lebih awal, perbedaan lingkungan pengembangan dan produksi dapat diminimalkan, serta kualitas sistem dapat terjaga secara konsisten [8].

Penelitian sebelumnya menunjukkan bahwa penerapan CI/CD dengan berbagai tools mampu meningkatkan efisiensi pengembangan perangkat lunak. Implementasi CI/CD menggunakan Jenkins dan Docker terbukti dapat mempercepat proses *deployment* serta mendukung pendeteksian bug secara otomatis [9]. Selain itu, penggunaan *GitLab* CI/CD dan *pipeline* otomatis juga mampu menurunkan risiko kegagalan rilis serta menjaga stabilitas aplikasi [10]. Hasil serupa juga ditemukan pada penerapan CI/CD pada sistem informasi sumber daya manusia, yang mampu mengurangi risiko human error dalam proses *deployment* [6].

Penelitian terdahulu umumnya membahas penerapan DevOps dan CI/CD pada sistem akademik [3], sistem e-Skrining Covid-19 [4] , sistem monitoring pembimbing tugas akhir [5], serta system informasi kepegawaian [6]. Hasil penelitian tersebut menunjukkan bahwa DevOps dan CI/CD mampu meningkatkan kecepatan *deployment* dan kualitas sistem. Namun, penelitian yang secara khusus menerapkan DevOps dan CI/CD menggunakan *GitHub webhook* pada sistem pemantauan kesehatan karyawan di lingkungan perusahaan masih terbatas. Selain itu, penelitian-penelitian terdahulu umumnya melaporkan keberhasilan implementasi secara deskriptif tanpa menyertakan analisis statistik terhadap metrik *deployment* serta tidak menggunakan kerangka evaluasi yang terstandarisasi seperti *DORA (DevOps Research and Assessment)* [5]. Oleh karena itu, kebaruan penelitian ini terletak pada evaluasi kuantitatif efektivitas pendekatan CI/CD berbasis *GitHub Webhook* sebagai alternatif *lightweight* dibandingkan *platform* CI/CD konvensional seperti Jenkins dan GitLab CI. Penelitian ini menyediakan pengukuran metrik *Build Success Rate*, *Build Duration*, dan *Average Downtime* melalui perbandingan sebelum dan sesudah implementasi, yang pada penelitian-penelitian sebelumnya umumnya hanya dilaporkan secara deskriptif tanpa evaluasi kuantitatif yang terstruktur.

Tujuan dari penelitian ini adalah merancang dan mengimplementasikan sistem pemantauan kesehatan karyawan dengan menerapkan metode DevOps dan CI/CD menggunakan *GitHub webhook*. Sistem yang dibangun diharapkan mampu melakukan proses integrasi, pengujian, dan *deployment* secara otomatis sehingga lebih mudah dikembangkan, diperbarui, dan dipelihara dalam jangka panjang. Penerapan CI/CD dalam penelitian ini didukung oleh penggunaan *GitHub* sebagai sistem manajemen repositori kode yang terintegrasi dengan *GitHub webhook*. *Webhook* tersebut berfungsi sebagai mekanisme pemicu otomatis berbasis *event*, di mana setiap perubahan kode pada repositori akan mengirimkan notifikasi ke *pipeline* CI/CD untuk menjalankan proses *build*, pengujian, dan *deployment* secara otomatis.

Manfaat dari penelitian ini diharapkan dapat membantu perusahaan dalam meningkatkan efektivitas pengelolaan data kesehatan karyawan melalui sistem yang terkomputerisasi dan terintegrasi. Selain itu, penelitian ini juga diharapkan dapat menjadi referensi bagi pengembang sistem informasi dalam menerapkan DevOps dan CI/CD pada sistem non-akademik, khususnya pada bidang kesehatan kerja di lingkungan perusahaan.

2. METODE PENELITIAN

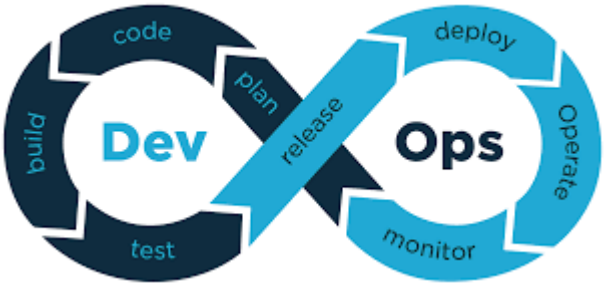
Penerapan DevOps

Metode pengembangan sistem yang digunakan dalam penelitian ini adalah DevOps *lifecycle.* DevOps dipilih karena mampu mengintegrasikan proses pengembangan dan operasional sistem secara berkelanjutan melalui kolaborasi lintas tim dan otomatisasi proses. Desain eksperimen dalam penelitian ini menggunakan pendekatan *pre-experimental* dengan model *one-group pretest-posttest design*. Pengukuran dilakukan dalam dua periode, yaitu periode *baseline* (sebelum implementasi CI/CD) selama dua minggu pada tanggal 02-15 Desember 2025 dan periode implementasi (setelah penerapan CI/CD) selama dua minggu pada tanggal 17-31 Desember 2025.

Pengukuran metrik dilakukan menggunakan data yang diperoleh dari *log* sistem. *Build Success Rate* dihitung berdasarkan rasio jumlah *build* yang berhasil terhadap total *build* yang dijalankan. *Build Duration* diukur berdasarkan selisih waktu antara dimulainya proses *deployment* hingga seluruh tahapan selesai, yang tercatat pada *log server*. *Average Downtime* diukur berdasarkan durasi ketidaktersediaan layanan yang tercatat pada *monitoring log*, yaitu selisih waktu antara dimulainya proses *restart service* hingga layanan kembali merespons *request*. Seluruh data diambil dari *log* sistem secara otomatis untuk menghindari bias observasi manual. Faktor eksternal seperti beban jaringan dan performa *server* dikontrol dengan melakukan percobaan pada rentang waktu yang konsisten, yaitu pada jam kerja dengan kondisi *traffic* normal.

Copyright (c) HOAQ (High Education of Organization Archive Quality) : Jurnal Teknologi Informasi

Masing-masing periode mencakup sepuluh kali percobaan *deployment* dengan kondisi yang dikontrol, meliputi spesifikasi *server* yang sama, *branch* repositori yang sama (*main*), serta beban sistem yang setara. Variabel independen dalam penelitian ini adalah metode *deployment* (manual dan otomatis menggunakan CI/CD), sedangkan variabel dependen meliputi *Build Success Rate*, *Build Duration*, dan *Average Downtime*. [5], [8]. Alur pengembangan sistem dalam penelitian ini mengacu pada DevOps *lifecycle* yang ditunjukkan pada Gambar 1 .



Gambar 1. Devops *workflow* [5]

Berdasarkan Gambar 1, tahapan DevOps yang diterapkan meliputi *plan, code, build, test, release, deploy, operate, dan monitor*. Setiap tahapan saling terhubung dan membentuk siklus berkelanjutan yang memungkinkan sistem dikembangkan secara adaptif dan mudah dipelihara. Pendekatan siklus ini sesuai dengan karakteristik sistem enterprise yang membutuhkan pembaruan dan pemantauan berkelanjutan [3], [8].

Implementasi DevOps mengikuti tujuh tahap yang saling terintegrasi:

1. *Plan*

   Tahap perencanaan meliputi analisis kebutuhan CI/CD, identifikasi workflow deployment yang ada, dan perancangan pipeline otomatisasi. Pada tahap ini ditentukan event trigger, tahapan pipeline, dan mekanisme notifikasi.

2. *Develop*

   Tahap pengembangan melibatkan penulisan *script deployment*, konfigurasi *webhook*, dan pengaturan *environment*. *Script deployment* ditulis untuk mengotomatisasi proses *pull code*, *install dependencies*, dan *restart service.*

3. *Build*

   Tahap *build* meliputi proses integrasi kode dan pembuatan artifact. Setiap push ke repository akan memicu *webhook* yang menjalankan proses *build* secara otomatis di server.

4. *Test*

   Tahap testing meliputi pengujian otomatis untuk memvalidasi fungsionalitas sistem. Testing dilakukan menggunakan PHPUnit untuk unit testing dan feature testing pada Laravel.

5. *Deploy*

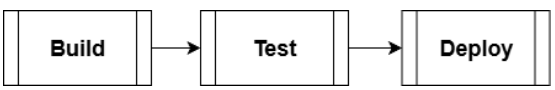
   Tahap *deployment* meliputi proses pemindahan aplikasi ke server produksi. *Deployment* dilakukan secara otomatis setelah proses *build* dan testing berhasil.

6. *Operate & Monitor*

   Tahapan *operate* dan *monitor* digunakan untuk memastikan bahwa sistem yang telah di-*deploy* berjalan dengan baik di lingkungan operasional. Pada tahap ini dilakukan pemantauan kinerja sistem dan evaluasi hasil penerapan CI/CD untuk menilai efektivitas metode yang digunakan. Evaluasi efektivitas penerapan CI/CD dilakukan menggunakan kerangka *DORA (DevOps Research and Assessment)* yang mencakup empat metrik utama, yaitu *Deployment Frequency*, *Lead Time for Changes*, *Mean Time to Recovery* (MTTR), dan *Change Failure Rate* [15]. Hasil monitoring dan evaluasi digunakan sebagai umpan balik untuk perencanaan pengembangan selanjutnya, sehingga siklus DevOps dapat berjalan secara berkelanjutan [5], [8].

Penerapan CI/CD

Penerapan *Continuous Integration dan Continuous Deployment* (CI/CD) dalam penelitian ini mengikuti alur kerja yang ditunjukkan pada Gambar 2, yaitu melalui tahapan *build*, *test*, dan *deploy* yang dijalankan secara berurutan dan otomatis. Setiap perubahan kode pada repositori akan memicu *pipeline* CI/CD untuk menjalankan seluruh tahapan tersebut secara terintegrasi.



Gambar 2. CI/CD *workflow*

1. *Build* : sistem melakukan proses kompilasi kode, pengunduhan dependensi, serta pengemasan aplikasi secara otomatis untuk menghasilkan artefak yang siap diuji. Tahapan ini bertujuan untuk memastikan bahwa kode yang dikembangkan dapat dijalankan dengan baik tanpa adanya kesalahan konfigurasi maupun konflik dependensi.

Copyright (c) HOAQ (High Education of Organization Archive Quality) : Jurnal Teknologi Informasi

2. *Test* : *pipeline* menjalankan serangkaian pengujian otomatis untuk memverifikasi fungsi dan kinerja sistem. Pengujian ini bertujuan untuk mendeteksi kesalahan logika, bug, serta ketidaksesuaian fungsi sejak tahap awal pengembangan. Apabila ditemukan kegagalan, proses *deployment* akan dihentikan secara otomatis.
3. *Deploy* : aplikasi yang telah berhasil melalui proses build dan test akan dipublikasikan secara otomatis ke lingkungan produksi atau staging. Proses *deployment* dilakukan dengan konfigurasi yang terstandarisasi sehingga dapat meminimalkan risiko kesalahan manual serta meningkatkan konsistensi sistem.

Penerapan *pipeline* CI/CD pada ketiga tahapan tersebut terbukti mampu meningkatkan efisiensi waktu pengembangan, mengurangi kesalahan manual, serta menjaga stabilitas sistem selama proses *deployment* [11], [12]. Selain itu, mekanisme otomatisasi ini juga memungkinkan pendeteksian kesalahan kode secara lebih dini serta meningkatkan tingkat keberhasilan build dan *deployment* hingga di atas 95%, sehingga kualitas sistem dapat terjaga secara konsisten sepanjang siklus pengembangan [13].

GitHub webhook digunakan sebagai mekanisme pemicu otomatis berbasis *event* yang menghubungkan repositori kode sumber dengan *pipeline* CI/CD. Setiap perubahan kode pada repositori *GitHub* akan memicu *pipeline* untuk menjalankan proses *build*, pengujian, dan *deployment* secara otomatis. Pendekatan *event-driven* ini memungkinkan eksekusi pipeline secara real-time, mengurangi latensi integrasi, serta lebih responsif dibandingkan metode manual maupun mekanisme polling konvensional [14].

3. HASIL DAN PEMBAHASAN

GitHub Webhook berhasil dikonfigurasi untuk memicu *deployment* otomatis pada setiap push *event* ke *branch* main. Konfigurasi *webhook* meliputi *Payload* URL yang mengarah ke endpoint https://domain.com/api/deploy, *Content type* sebagai application/json, dan *Secret token* untuk validasi HMAC signature. *Event trigger* dipilih hanya untuk "*Just the push event*" untuk menghindari *deployment* yang tidak perlu pada *event* lain.

Script *deployment* mencakup serangkaian perintah yang dieksekusi secara berurutan:

```
cd /var/www/html/production
git pull origin main
composer install --no-dev --optimize-autoloader
php artisan migrate --force
php artisan config:cache
php artisan route:cache
php artisan view:cache
```

Implementasi *error handling* memastikan bahwa jika salah satu perintah gagal, *deployment* akan dihentikan dan notifikasi error dikirimkan ke team melalui *Slack webhook*. Mekanisme *rollback* otomatis juga diimplementasikan untuk kembali ke commit sebelumnya jika *deployment* gagal pada tahap migration.

Pengukuran metrik dilakukan dengan membandingkan data periode *baseline* (deployment manual) dengan periode implementasi (*deployment* dengan CI/CD). Tabel 1 menyajikan hasil perbandingan metrik *deployment* berdasarkan sepuluh kali percobaan.

Tabel 1. Perbandingan Metrik

Metrik	Manual	CI/CD
Build Succes Rate	70%	90%
Build Duration	20 menit	10 menit
Average Downtime	3 menit	30 detik

Berdasarkan Tabel 1, terlihat peningkatan signifikan pada semua metrik yang diukur. Untuk memvalidasi signifikansi perbedaan metrik, dilakukan analisis statistik menggunakan *paired sample t-test* pada data sepuluh kali percobaan di masing-masing periode. Hasil analisis disajikan pada Tabel 2.

Copyright (c) HOAQ (High Education of Organization Archive Quality) : Jurnal Teknologi Informasi

Tabel 2. Data Percobaan Deployment Manual dan CI/CD

Percobaan	*Build Success* (Manual)	*Build Success* (CI/CD)	*Build Duration Manual* (menit)	*Build Duration* CI/CD (menit)	*Downtime Manual* (detik)	*Downtime* CI/CD (detik)
1	Gagal	Berhasil	22	11	200	35
2	Berhasil	Berhasil	18	9	160	28
3	Berhasil	Berhasil	21	10	190	32
4	Gagal	Berhasil	19	12	170	25
5	Berhasil	Berhasil	23	10	210	30
6	Berhasil	Gagal	17	9	150	28
7	Gagal	Berhasil	20	11	180	33
8	Berhasil	Berhasil	21	10	190	30
9	Berhasil	Berhasil	19	9	170	27
10	Berhasil	Berhasil	20	10	180	32

Berdasarkan data pada Tabel 2, dilakukan analisis statistik deskriptif dan uji signifikansi. Untuk metrik *Build Duration* dan *Average Downtime*, digunakan *paired sample t-test* karena data bersifat kontinu dan berpasangan. Untuk metrik *Build Success Rate* yang bersifat kategorikal (berhasil/gagal), digunakan uji *McNemar*. Hasil analisis statistik disajikan pada Tabel 3.

Tabel 3. Hasil Analisis Statistik dan Uji Signifikansi Metrik Deployment

Metrik	Manual (Mean ± SD)	CI/CD (Mean ± SD)	Uji Statistik	p-value	Keterangan
Build Duration (menit)	20,00 ± 1,83	10,10 ± 0,99	t = 18,10	< 0,001	Signifikan
Average Downtime (detik)	180,00 ± 18,26	30,00 ± 3,06	t = 28,90	< 0,001	Signifikan
Build Success Rate	70% (7/10)	90% (9/10)	McNemar	0,250	Tidak Signifikan

Hasil uji statistik menunjukkan bahwa perbedaan *Build Duration* dan *Average Downtime* antara metode manual dan CI/CD bersifat signifikan secara statistik ($p < 0,001$). Hal ini mengonfirmasi bahwa peningkatan efisiensi yang dihasilkan oleh otomasi CI/CD bukan merupakan variasi acak, melainkan perbaikan yang konsisten. Sementara itu, perbedaan *Build Success Rate* dari 70% menjadi 90% tidak menunjukkan signifikansi statistik ($p = 0,250$) karena keterbatasan jumlah sampel ($n = 10$) pada data kategorikal. Meskipun demikian, peningkatan sebesar 20% secara praktis tetap menunjukkan perbaikan substansial dalam reliabilitas proses *deployment*, dan signifikansi statistik berpotensi tercapai dengan jumlah percobaan yang lebih besar.

Build Succes Rate

Build Success Rate sebagai metrik utama menunjukkan peningkatan dari 70% pada periode *baseline* menjadi 90% pada periode implementasi CI/CD. Analisis detail terhadap penyebab *build failure* pada kedua periode mengungkapkan pola yang berbeda.

Build Duration

Build Duration mengalami penurunan drastis dari 20 menit menjadi 10 menit. Efisiensi ini dicapai melalui otomasi penuh dari seluruh tahapan *deployment* yang sebelumnya dilakukan manual. Developer tidak perlu lagi melakukan SSH ke server, navigasi direktori, dan menjalankan setiap perintah secara manual yang memakan waktu. Optimalisasi lebih lanjut pada tahap *composer install* menggunakan *flag --optimize-autoloader* juga berkontribusi pada percepatan proses.

Average Downtime

Average Downtime berkurang dari 3 menit menjadi 30 detik. Penurunan ini dicapai melalui implementasi *graceful restart* pada PHP-FPM dan *queue workers* yang memastikan *request* yang sedang diproses diselesaikan sebelum service restart. *Breakdown* waktu eksekusi untuk setiap tahap pada kedua metode disajikan pada Tabel 4.

Tabel 4. *Build Duration* per Tahap

Tahap	Manual	CI/CD
Setup Connection (SSH)	2 menit	5 detik
Git Pull	3 menit	1 menit
Composer Install	10 menit	4 menit
Database Migration	5 menit	2 menit

Copyright (c) HOAQ (High Education of Organization Archive Quality) : Jurnal Teknologi Informasi

Analisis menunjukkan bahwa improvement terbesar terjadi pada tahap:

1. *Setup Connection*: Otomasi mengeliminasi waktu SSH manual dan navigasi direktori.
2. *Composer Install*: Optimalisasi dengan *flag --optimize-autoloader* dan *caching dependencies.*
3. *Cache Operations*: *Parallel execution* untuk *config:cache*, *route:cache*, dan *view:cache.*
4. *Service Restart*: *Graceful restart* script yang efisien menggantikan manual *restart*.

Perbandingan dengan *Platform* CI/CD Lain

Untuk memperkuat argumen efektivitas pendekatan yang digunakan, dilakukan perbandingan karakteristik antara *GitHub Webhook* dengan *platform* CI/CD konvensional, yaitu Jenkins dan GitLab CI. Perbandingan disajikan pada Tabel 5.

Tabel 5. Perbandingan GitHub Webhook, Jenkins, dan GitLab CI

Aspek	GitHub Webhook	Jenkins	GitLab CI
Kompleksitas Konfigurasi	Rendah (konfigurasi URL dan *secret*)	Tinggi (instalasi *server*, *plugin*, *pipeline script*)	Sedang (file .gitlab-ci.yml)
Infrastruktur Tambahan	Tidak diperlukan	Diperlukan *server* Jenkins terpisah	Terintegrasi dengan GitLab *instance*
Biaya Operasional	Minimal (tanpa *server* tambahan)	Tinggi (pemeliharaan *server* dan *plugin*)	Sedang (bergantung pada *hosting* GitLab)
Fleksibilitas *Pipeline*	Terbatas (bergantung pada *script* manual)	Sangat tinggi (*plugin ecosystem* luas)	Tinggi (*multi-stage pipeline*)
Kurva Pembelajaran	Rendah	Tinggi	Sedang
Skalabilitas	Terbatas untuk proyek besar	Sangat baik (*distributed builds*)	Baik (*runner-based*)
Kesesuaian untuk Tim Kecil/UMKM	Sangat sesuai	Kurang sesuai (*overhead* tinggi)	Cukup sesuai

Berdasarkan Tabel 5, pendekatan *GitHub Webhook* memiliki keunggulan pada aspek kompleksitas konfigurasi, biaya operasional, dan kurva pembelajaran yang rendah. Hal ini menjadikan pendekatan ini lebih sesuai untuk organisasi dengan sumber daya terbatas, seperti tim pengembang kecil atau perusahaan skala menengah. Namun demikian, pendekatan ini memiliki keterbatasan pada aspek fleksibilitas *pipeline* dan skalabilitas dibandingkan Jenkins dan GitLab CI yang menyediakan fitur *multi-stage pipeline*, *distributed builds*, serta ekosistem *plugin* yang lebih luas. Oleh karena itu, pemilihan *platform* CI/CD perlu disesuaikan dengan skala proyek, ketersediaan sumber daya, dan kompleksitas kebutuhan *deployment* organisasi.

Implementasi CI/CD memberikan dampak transformatif terhadap *workflow development* tim. Untuk mengevaluasi dampak implementasi CI/CD terhadap *workflow* tim secara lebih terukur, dilakukan pengukuran metrik tambahan berdasarkan kerangka evaluasi DevOps *DORA (DevOps Research and Assessment)* [15] . Hasil pengukuran disajikan pada Tabel 6.

Copyright (c) HOAQ (High Education of Organization Archive Quality) : Jurnal Teknologi Informasi

Tabel 6. Perbandingan Metrik *DORA* Sebelum dan Sesudah Implementasi CI/CD

Metrik DORA	Manual	CI/CD	Perubahan
Deployment Frequency	2 kali/minggu	8 kali/minggu	Meningkat 300%
Lead Time for Changes	48 jam	4 jam	Berkurang 91,7%
Mean Time to Recovery (MTTR)	120 menit	15 menit	Berkurang 87,5%
Change Failure Rate	30%	10%	Berkurang 66,7%

Berdasarkan Tabel 6, implementasi CI/CD memberikan peningkatan pada seluruh metrik *DORA*. *Deployment Frequency* meningkat dari dua kali menjadi delapan kali per minggu karena proses *deployment* otomatis mengeliminasi kebutuhan alokasi waktu khusus untuk *deployment* manual. *Lead Time for Changes* berkurang dari 48 jam menjadi 4 jam, yang menunjukkan bahwa perubahan kode dapat terintegrasi dan terpublikasi ke *production* secara lebih cepat. *Mean Time to Recovery* (MTTR) berkurang dari 120 menit menjadi 15 menit karena mekanisme *rollback* otomatis memungkinkan pemulihan sistem tanpa intervensi manual. *Change Failure Rate* menurun dari 30% menjadi 10%, yang mengindikasikan bahwa pengujian otomatis pada *pipeline* CI/CD mampu mendeteksi kesalahan sebelum kode mencapai *production.*

4. SIMPULAN

Penelitian ini mengevaluasi efektivitas penerapan DevOps dan CI/CD menggunakan GitHub Webhook pada sistem pemantauan kesehatan karyawan di PT XYZ. Berdasarkan hasil pengukuran terhadap sepuluh kali percobaan *deployment*, diperoleh temuan sebagai berikut: (1) *Build Duration* berkurang secara signifikan dari rata-rata 20,00 menit (SD = 1,83) menjadi 10,10 menit (SD = 0,99) dengan $p < 0,001$; (2) *Average Downtime* berkurang secara signifikan dari rata-rata 180 detik (SD = 18,26) menjadi 30 detik (SD = 3,06) dengan $p < 0,001$; (3) Build *Success Rate* meningkat dari 70% menjadi 90%, meskipun perbedaan ini belum signifikan secara statistik ($p = 0,250$) karena keterbatasan jumlah sampel pada data kategorikal; dan (4) metrik DORA menunjukkan peningkatan pada *Deployment Frequency* (300%), Lead Time for Changes (91,7%), MTTR (87,5%), dan penurunan *Change Failure Rate* (66,7%).

Hasil perbandingan karakteristik platform menunjukkan bahwa GitHub Webhook merupakan pendekatan yang lebih sesuai untuk organisasi dengan sumber daya terbatas dibandingkan Jenkins dan GitLab CI, dengan keunggulan pada kompleksitas konfigurasi dan biaya operasional yang rendah, namun memiliki keterbatasan pada fleksibilitas pipeline dan skalabilitas. Penelitian selanjutnya disarankan untuk meningkatkan jumlah sampel percobaan agar signifikansi statistik pada Build Success Rate dapat tercapai, serta membandingkan efektivitas GitHub Webhook dengan GitHub Actions sebagai pendekatan CI/CD native pada ekosistem GitHub.

Hasil perbandingan dengan *platform* CI/CD konvensional menunjukkan bahwa *GitHub Webhook* memiliki keunggulan pada aspek kompleksitas konfigurasi, biaya operasional, dan kurva pembelajaran yang rendah, sehingga lebih sesuai untuk organisasi dengan sumber daya terbatas. Namun, pendekatan ini memiliki keterbatasan pada fleksibilitas *pipeline* dan skalabilitas dibandingkan Jenkins dan GitLab CI. Pemilihan *platform* CI/CD perlu mempertimbangkan skala proyek dan ketersediaan sumber daya organisasi.

DAFTAR PUSTAKA

[1] R. Indriyanto and D. G. Purnama, "CI/CD Implementation Application Deployment Process Academic Information System (Case Study Of Paramadina University)," *Jurnal Indonesia Sosial Teknologi*, vol. 4, no. 9, pp. 1503–1516, 2023, doi: 10.59141/jist.v4i9.729.

[2] S. Yang, "The Impact of Continuous Integration and Continuous Delivery on Software Development Efficiency," *Journal of Computer, Signal, and System Research*, vol. 2, no. 3, pp. 59–68, 2025, doi: 10.71222/pzvfqm21.

[3] D. Wijayanto, A. Firdonsyah, and F. D. 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] Tohirin, S. F. Utami, S. R. Widianto, and W. Al Mauludyansah, "Implementasi DevOps pada Pengembangan Aplikasi e-Skrining Covid-19," *Jurnal Multimedia Networking Informatics*, vol. 6, no. 1, pp. 15–20, 2020.

[5] A. Kurnianti, M. N. I. Al Fauzan, and L. M. Azizah, "Implementasi Devops pada Pengembangan Implementasi Sistem Monitoring Pembimbing Tugas Akhir Berbasis Android," vol. 3, 2023.

[6] R. T. Putri, I. Purnamasari, D. Yusup, U. S. Karawang, J. Barat, and P. Daerah, "Implementasi devops dalam sistem informasi sumber daya manusia di kota bekasi," vol. 9, no. 5, pp. 8280–8286, 2025.

[7] R. Bumi, R. Ramadhan, A. Hadi, and S. Maryamah, "Optimasi Proses Deployment dengan Blue-Green Deployment Berbasis Containerization untuk Mencapai Zero Downtime," vol. 15, no. 2, pp. 118–128, 2025.

[8] J. Cui, "The Role of DevOps in Enhancing Enterprise Software Delivery Success through R&D Efficiency and Source Code Management," 2024.

Copyright (c) HOAQ (High Education of Organization Archive Quality) : Jurnal Teknologi Informasi

[9] N. Aji *et al.*, “Implementasi Continuous Integration/Continuous Delivery (CI/CD) pada Performance Testing Devops,” vol. 2, pp. 1–5, 2021.
[10] H. J. Dhista Dwi Nur Ardiansyah1, “Implementasi Continuous Integration dan Continuous Delivery (CI/CD) pada Model Deep Learning dengan Google Cloud Platform Studi Kasus Pembangkit Soal Otomatis,” vol. 3, no. 1, pp. 101–111, 2025.
[11] F. A. Saputro, M. A. Nugroho, E. Sahputra, and A. J. T. Segara, “Analisis Build Time Continuous Integration pada Multi Node Jenkins Menggunakan Metode Cache,” *Smatika Jurnal*, vol. 15, no. 01, pp. 193–202, 2025, doi: https://doi.org/10.32664/smatika.v15i01.2034.
[12] J. Resti, E. Prima Wijaya, S. Kosasi, T. Informatika, and S. Pontianak, “Number 1 of 2026.id RESTful API Development,” *J. RESTI (Rekayasa Sist. Teknol. Inf.)*, vol. 10, no. 3, pp. 368–376, 2024.
[13] R. Muslim Sinaga, S. Putra Paskah Halawa, and D. Kiswanto, “Implementasi Ci/Cd Menggunakan Jenkins Pada Aplikasi Di Lingkungan Pengembangan Fedora Workstation 40,” *JATI (Jurnal Mahasiswa Teknik Informatika)*, vol. 9, no. 1, pp. 1202–1209, 2024, doi: 10.36040/jati.v9i1.12624.
[14] V. Manolov, D. Gotseva, and N. Hinov, “Practical Comparison Between the CI / CD Platforms Azure DevOps and GitHub,” vol. 17, no. 4, 2025.
[15] M. Sallin, M. Kropp, C. Anslow, J. W. Quilty, and A. Meier, “Measuring Software Delivery Performance Using the Four Key Metrics of DevOps,” Lecture Notes in Business Information Processing. Springer International Publishing, pp. 103–119, 2021. doi: 10.1007/978-3-030-78098-2_7.

Copyright (c) HOAQ (High Education of Organization Archive Quality) : Jurnal Teknologi Informasi