JNATIA Volume 4, Nomor 1, November 2025 p-ISSN: 2986-3929
Jurnal Nasional Teknologi Informasi dan Aplikasinya e-ISSN: 3032-1948

Implementasi Pipeline CI/CD dengan Github Actions dalam Mengurangi Downtime pada Aplikasi Berbasis Web

I Komang Wahyu Pranata[a1], I Wayan Santiyasa[a2]

[a]Program Studi Informatika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Udayana
Jalan Raya Kampus Udayana, Bukit Jimbaran, Kuta Selatan, Badung, Bali, Indonesia
[1]pranata.2308561026@student.unud.ac.id
[2]santiyasa@unud.ac.id

Abstract

This research focuses on optimizing the application deployment process to minimize prolonged downtime, which is often caused by the architecture that don't supports automation of program code to the deployment server. To overcome this problem, Continuous Integration/Continuous Deployment (CI/CD) pipeline using GitHub Workflow is implemented by involving three main components: local development server, GitHub (repository and actions), and deployment server. The pipeline design includes three types of branches in the GitHub repository: dev, staging, and master. The workflow process starts from local development to the dev branch, then integrated to staging for integration testing, and finally to master which triggers automatic deployment actions to the server using the FTP protocol. From the results of the system testing, the metrics used are downtime and deployment time. The results show that the implementation of the pipeline successfully achieved zero downtime (0 seconds) both in the initial deployment and the deployment of changes. However, the deployment time is still relatively long due to the use of the FTP protocol. Although the application does not experience downtime, the long deployment time has the potential to make the application not optimally usable during the deployment process.

Keywords: *CI/CD, Automation, Github Actions, Reducing Downtime, Application Deployment*.

1. Pendahuluan

Dalam pengembangan aplikasi, terdapat banyak metode pengembangan yang dapat digunakan. Pada umumnya, metode-metode yang ada dimulai dari tahapan analisis dan diakhiri dengan tahap *delivery*/*deployment*. Setelah tahap *deployment*, aplikasi dapat saja dilakukan *maintenance* yang bertujuan untuk meningkatkan efisiensi, skalabilitas, maupun pengalaman pengguna dari system [1]. *Maintenance* juga akan dilakukan apabila terjadi hal-hal yang tidak terduga seperti serangan keamanan atau *bug* terjadi pada sistem. Pada tahap ini, aplikasi akan masuk ke masa *downtime* yang di mana aplikasi belum dapat diakses oleh *end-user*. *Downtime* sistem yang berkepanjangan akan berpengaruh pada bisnis yang menggunakan sistem tersebut. Bisnis akan mengalami penurunan keuntungan, produktivitas, dan tingginya biaya yang harus dikeluarkan untuk mengganti kerugian yang dialami selama sistem *down* [2].

Oleh sebab itu, diperlukan sebuah metode yang mampu untuk mengurangi *downtime* pada sistem. Metode tradisional dengan memanfaatkan metode SDLC yang ada masih belum cukup untuk mengurangi *downtime* pada sistem. Metode tradisional cenderung memakan waktu yang lama dalam integrasi kode program, *testing*, hingga *deployment* yang akan berpengaruh pada waktu *downtime* sistem.

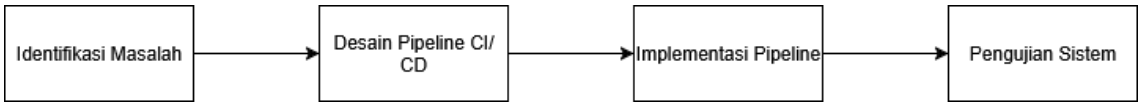
CI/CD merupakan salah satu metode dalam pengembangan perangkat lunak yang memungkinkan kode program dapat di-*deploy* secara otomatis. Terdapat dua aktivitas pada CI/CD, yaitu *continuous integration* (CI) dan *continuous delivery/deployment* (CD). CI merupakan tahapan integrasi seluruh kode program dalam satu buah repositori. Sedangkan CD adalah

p-ISSN: 2986-3929
e-ISSN: 3032-1948

tahapan lanjutan dari CD yang memungkinkan kode program yang sudah terintegrasi langsung di-*deploy* secara otomatis [3]. Pada penelitian yang dilakukan oleh Praveen pada 2022, implementasi CI/CD pada aplikasi berbasis bahasa pemrograman Java berhasil untuk mengurangi *downtime* pada aplikasi, hingga mencapai *zero downtime* [4]. Oleh sebab itu, pada penelitian ini penulis melakukan implementasi *deployment pipeline* dengan CI/CD yang diharapkan mampu untuk mengurangi downtime pada aplikasi. Platform yang digunakan oleh penulis dalam implementasi CI/CD adalah *Github Actions*. *Github Actions* merupakan platform integrasi CI/CD dalam sebuah repositori GitHub [5]. Github Actions digunakan pada penelitian ini karena sudah terintegrasi dengan repositori yang sudah ada dan dukungan terhadap beberapa pustaka maupun bahasa pemrograman yang sudah ada. Pada penelitian ini, penulis melakukan integrasi pipeline pada sebuah sistem informasi pengunggahan proyek mahasiswa Informatika yang akan disambungkan dengan *web server.*

2. Metode Penelitian

Pada penelitian ini, terdapat metode yang dilakukan:



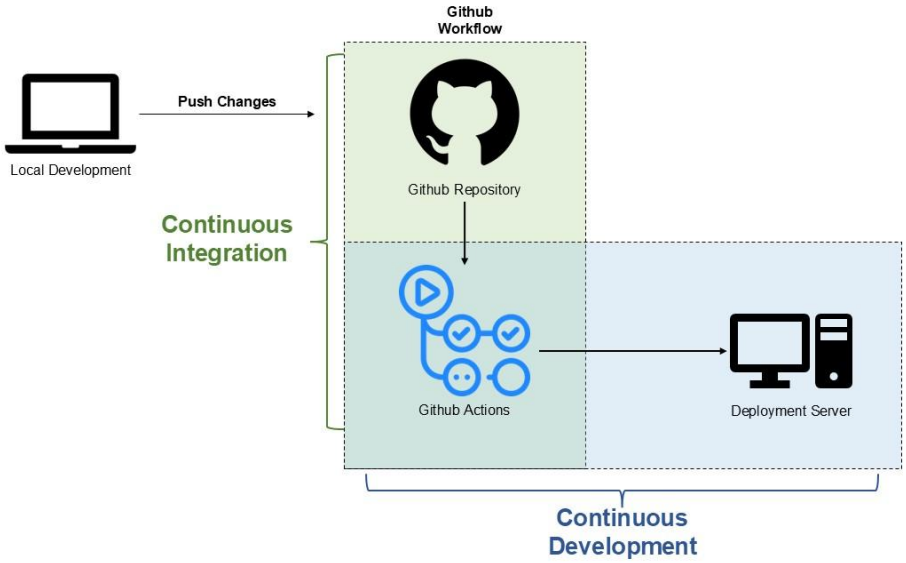
Gambar 1. Alur Penelitian

2.1. Identifikasi Masalah

Masalah yang diangkat pada penelitian ini adalah *downtime* aplikasi yang berkepanjangan yang umumnya disebabkan karena tidak adanya arsitektur dari aplikasi yang memungkinkan adanya otomasi kode program ke *deployment server.* Oleh sebab itu, akan diimplementasikan

2.2. Desain Pipeline CI/CD

Sebelum melakukan implementasi pada sistem, dilakukan desain terhadap *pipeline* yang akan digunakan. Terdapat 3 buah komponen penting pada pipeline, yaitu: *local development server*, *GitHub workflow* (*repository* dan *actions*), dan *deployment server*.



Gambar 2. Diagram Pipeline CI/CD

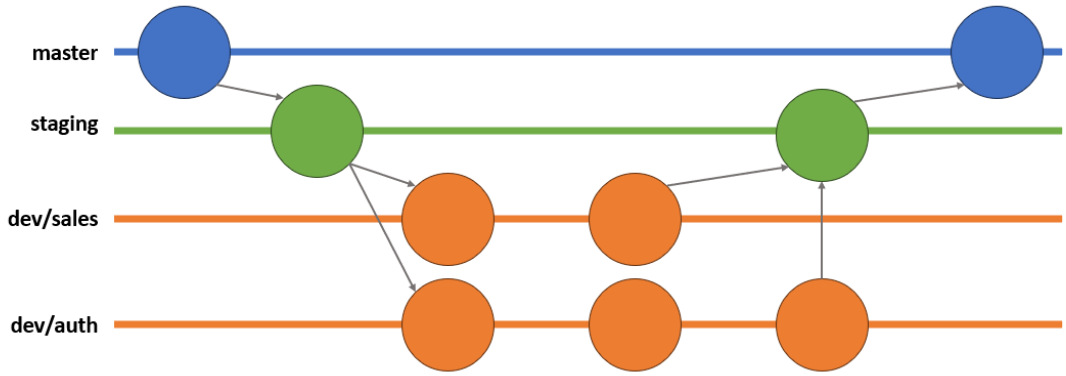
p-ISSN: 2986-3929
e-ISSN: 3032-1948

Pada repositori di GitHub terdapat 3 jenis *branch*:

a. Dev
   Merupakan *branch* untuk menyimpan perubahan yang di-*push* dari *local development*. Nama *branch* disesuaikan dengan fitur yang dibangun (contoh: dev/transaction, dev/authentication, dev/sales, dll.)

b. Staging
   Merupakan *branch* yang digunakan sebagai tempat integrasi dari banyak *branch dev* sebelum masuk ke *deployment*. Branch *staging* sekaligus menjadi tempat *testing* dari sistem.

c. Master
   Merupakan *branch* yang digunakan oleh aplikasi yang sudah siap untuk di-*deploy* ke server.



Gambar 3. Diagram *Branch* Repositori GitHub

Kode program yang ada di *local* akan dibawa ke *repository* GitHub pada branch dev yang sesuai. Setelah satu fitur selesai, maka perubahan pada *branch* dev tersebut akan dibawa ke *branch staging* untuk diintegrasikan. Setelah kode program terintegrasi, dilakukan *integration test* pada aplikasi yang ada di *branch staging*. Setelah itu, kode program pada *branch staging* akan dibawa ke branch *master*.

Pada branch master, disiapkan sebuah *deployment actions* yang akan dijalankan apabila terdapat perubahan pada kode program. Deployment actions berisi perintah yang akan menjalankan *deployment* ke *server*. *Deployment* akan dilakukan dengan protokol FTP.

2.3. Implementasi Pipeline

Sebelum melakukan integrasi pipeline, terlebih dahulu menyiapkan akun FTP pada web server. Setelah membuat akun FTP, buat sebuah folder bernama ".github/workflows" pada repositori GitHub. Pada folder tersebut, dibuat sebuah file dengan tipe yml yang terlihat pada tabel 1.

p-ISSN: 2986-3929
e-ISSN: 3032-1948

Tabel 1. Kode *GitHub Actions*

No.	Kode Github Actions
1	name: Laravel CI/CD Pipeline
2	on:
3	push:
4	branches: [master]
5	pull_request:
6	branches: [master]
7	jobs:
8	- name: Create FTP deployment script
9	run: |
10	#!/bin/bash
11	FTP_HOST="${{ secrets.FTP_HOST }}"
12	FTP_USER="${{ secrets.FTP_USERNAME }}"
13	FTP_PASS="${{ secrets.FTP_PASSWORD }}"
14	FTP_PORT="${{ secrets.FTP_PORT }}"
15	REMOTE_DIR="${{ secrets.FTP_SERVER_DIR }}"
16	cat > ftp_commands.txt << EOF_LFTP
17	open -u $FTP_USER,$FTP_PASS -p $FTP_PORT $FTP_HOST
18	cd $REMOTE_DIR
19	mirror --reverse --delete --verbose --exclude-glob .env --exclude-glob storage/logs/*.log ./ ./
20	chmod 755 public
21	chmod -R 755 storage
22	chmod -R 755 bootstrap/cache
23	quit
24	EOF_LFTP
25	lftp -f ftp_commands.txt rm -f ftp_commands.txt
26	if [$? -eq 0]; then
27	echo "FTP deployment successful!"
28	else
29	echo "FTP deployment failed!"
30	exit 1
31	fi
32	rm -f ftp_commands.txt

Pada kode tersebut, terdapat dua bagian yaitu “on” dan “jobs”. *On* menandakan kapan *actions*

ini akan dijalankan. Sedangkan “jobs” menandakan hal apa saja yang akan dijlankan.

Pertama, pada bagian *on* diatur *branch* yang akan dipasang *actions* adalah *branch master*. *Actio*ns akan dijalankan setiap *branch master* menerima kode baru yang berasal dari unggahan langsung (*push*) atau dari *branch* lain (*pull_request*).

Selanjutnya, pada bagian *jobs* diberikan sebuah *action* yang menjalankan perintah *bash*. Disiapkan beberapa variabel: FTP_HOST, FTP_USER, FTP_PASS, FTP_POST, REMOTE_DIR. Variabel tersebut berfungsi untuk melakukan koneksi FTP pada *web server*. Nilai dari variabel tersebut diambil dari tempat penyimpanan rahasia yang disediakan oleh GitHub.

Setelah itu, FTP diatur dalam mode pasif. Dibuka koneksi dengan menggunakan perintah “open” dengan nilai konfigurasi yang sudah diatur sebelumnya. Pada koneksi FTP, dijalankan perintah *mirror* yang akan me-*copy* seluruh *file* pada repositori dengan *web server*. Setelah *mirror* berjalan, jika berjalan dengan sukses maka akan menampilkan pesan sukses. Sebaliknya, akan menampilkan pesan gagal.

2.4. Pengujian Sistem

Setelah sistem terintegrasi, sebelum dilakukan *deployment* akan dilakukan pengujian. Pengujian yang dilakukan adalah *integration test.*

Berikut metriks yang digunakan pada *integration test*:

Tabel 2. Metriks *Integration Test*

No.	Nama	Keterangan
1.	Downtime	Semakin lama, semakin buruk.
2.	Deployment Time	Semakin lama, semakin buruk

3. Hasil dan Pembahasan

Setelah dilakukan *deployment* dengan GitHub Actions didapatkan hasil untuk mengunggah aplikasi untuk pertama kali seperti berikut

Tabel 2. Hasil Test Unggahan Pertama Kali

No.	Nama	Nilai
1.	Downtime	0s
2.	Deployment Time	1 jam, 15 menit

Adapun hasil untuk mengunggah perubahan pada aplikasi:

Tabel 3. Hasil Test Perubahan

No.	Nama	Nilai
1.	*Downtime*	0s
2.	*Deployment Time*	1 jam, 14 menit

Hasil menunjukkan bahwa *deployment* yang dilakukan berhasil untuk tidak membuat *downtime*. Namun, *deployment time* masih cukup lama. Meskipun tidak dalam *downtime*, aplikasi memiliki kemungkinan belum dapat digunakan secara maksimal karena masih ada perubahan yang belum

di-*deploy* dengan sempurna.

4. Kesimpulan

Implementasi berhasil untuk dilakukan. *Downtime* pada aplikasi berhasil diturunkan hingga menjadi 0 detik (*zero downtime*). Namun, terdapat efek samping yang didapat yang di mana *deployment time* menjadi sangat lama karena masih menggunakan protokol FTP. Hal ini dapat menyebabkan aplikasi yang masih berada pada masa *deployment* belum dapat digunakan secara sempurna.

Untuk ke depannya, pada penelitian selanjutnya dapat untuk menggunakan protokol SSH atau HTTP yang lebih cepat dalam mengunggah *file* sehingga waktu *deployment* dapat diturunkan.

Daftar Pustaka

[1] A. Noor, B. Talal, I. Muhammad, “A Comprehensive Framework for Intelligent, Scalable, and Performance-Optimized Software Development” IEEE Access, p 74063, 2025
[2] C. Adcock, Portnox, 5 June 2025. [Online]. Available: https://www.portnox.com/blog/network-security/the-business-operational-impacts-of-system-downtime/. [2 July 2025]
[3] “CI/CD Pipelines In Kubernetes: Accelerating Software Development And Deployment” EPH-International Journal of Science and Engineering. vol. 8, no. 3, 2022
[4] P. K. Koppanati, “Achieving Zero-Downtime Deployment for Java Applications Using GitLab CI/CD” Journal of Scientific and Engineering Research. vol. 9
[5] A. Decan, “On the Use of GitHub Actions in Software Development Repositories”. https://doi.org/10.1109/ICSME55016.2022.00029