PERANCANGAN DAN EVALUASI MODEL CI/CD BERBASIS GITHUB ACTIONS UNTUK OTOMATISASI DEPLOYMENT BACKEND EXPRESS.JS PADA VPS

Rizky Prayatman [1], Joko Supriyanto [2], Felia Oktaviana Sekarningtias [3]
[1,2] Program Studi Teknik Informatika, Universitas Siber Muhammadiyah
Jalan HOS Cokroaminoto No. 17 RT 53 RW 12
Kota Yogyakarta Daerah Istimewa Yogyakarta
[3] Pascasarjana Pendidikan Fisika, Universitas Negeri Semarang
rizkyprayatmaan@gmail.com

ABSTRAK

Jika dilakukan perubahan kode ketika proses deployment pada infrastruktur jaringan komputer secara manual, proses tersebut menjadi tidak efisien serta rentan terhadap kesalahan akibat human error. Dalam proses pengembangan perangkat lunak modern, proses deployment aplikasi ke server produksi masih sering dilakukan secara manual. Metode ini membutuhkan beberapa tahapan konfigurasi yang berulang, memakan waktu, serta berpotensi menimbulkan kesalahan akibat human error. Permasalahan ini menyebabkan proses rilis aplikasi menjadi tidak efisien dan kurang konsisten. Oleh karena itu, penelitian ini bertujuan untuk merancang dan mengevaluasi penerapan Continuous Integration dan Continuous Deployment (CI/CD) menggunakan GitHub Actions untuk mengotomatisasi proses deployment backend sistem *ediagnostic-misconception* berbasis Express.js pada server Virtual Private Server (VPS). Metode penelitian yang digunakan adalah eksperimen terapan dengan membandingkan proses deployment manual dan deployment otomatis berbasis CI/CD. Evaluasi dilakukan melalui pengukuran waktu deployment, konsistensi proses, dan potensi kesalahan konfigurasi, yang diuji sebanyak lima kali pada kondisi server yang sama. Hasil penelitian menunjukkan bahwa penerapan CI/CD mampu menurunkan waktu deployment dari rata-rata 15 menit pada metode manual menjadi sekitar 3.2 menit pada metode otomatis, atau terjadi peningkatan efisiensi sebesar 80%. Selain itu, proses deployment menjadi lebih konsisten dan stabil. Penelitian ini membuktikan bahwa penerapan CI/CD berbasis GitHub Actions pada VPS mandiri dengan sumber daya terbatas dapat meningkatkan efisiensi, keandalan, dan produktivitas pengembangan backend aplikasi.

Kata kunci : *VPS, CI/CD, Github actions, Otomatisasi, Deployment*

1. PENDAHULUAN

Seiring dengan pertumbuhan dan berkembangnya teknologi informasi yang cepat dan semakin canggih, maka semakin ketat kebutuhan akan pengembangan berkelanjutan. Salah satu tahap penting dalam siklus proses pengembangannya adalah *deployment*, dimana hasil pengembangan akan dirilis ke lingkungan produksi [1]. Dua hal yang muncul pada pendekatan utama yaitu: *Continuous Integration/Continuous Deployment* (*CI/CD*) dan metode pengembangan manual [2]. *CI/CD* merupakan elemen utama dalam automation proses pengembangan dan *deployment* sehingga mengurangi risiko kesalahan dan mempercepat siklus rilis [3]. Pendekatan ini secara signifikan mengurangi permasalahan integrasi, sehingga tim pengembang dapat fokus pada penciptaan nilai, bukan pada kompleksitas pengelolaan kode. Dengan melakukan integrasi beberapa kali dalam sehari, tim pengembang dapat mengidentifikasi dan menyelesaikan masalah sejak dini dalam siklus pengembangan, sehingga mengurangi risiko kegagalan integrasi di tahap selanjutnya [4].

Pada praktiknya, banyak pengembang masih melakukan proses deployment secara manual melalui terminal server menggunakan SSH. Proses ini melibatkan beberapa tahapan seperti menarik kode terbaru dari repositori, menginstal dependensi, melakukan build aplikasi, serta melakukan restart layanan aplikasi. Tahapan yang dilakukan secara manual tersebut tidak hanya memerlukan waktu yang relatif lama, tetapi juga berpotensi menimbulkan berbagai permasalahan.

Permasalahan utama yang muncul dari proses deployment manual adalah tingginya risiko *human error*, seperti kesalahan konfigurasi environment, ketidaksesuaian versi dependensi, serta kesalahan dalam menjalankan perintah deployment. Kesalahan-kesalahan ini dapat menyebabkan kegagalan sistem, downtime aplikasi, bahkan gangguan pada layanan yang digunakan oleh pengguna. Selain itu, proses manual juga cenderung tidak konsisten karena bergantung pada ketelitian dan pengalaman masing-masing pengembang, sehingga hasil deployment dapat berbeda-beda pada setiap eksekusi.

Dampak dari permasalahan tersebut cukup signifikan, di antaranya adalah meningkatnya waktu deployment, menurunnya stabilitas sistem, serta terhambatnya proses rilis fitur baru ke pengguna. Kondisi ini dapat mengurangi produktivitas tim pengembang dan berpotensi menurunkan kualitas layanan aplikasi secara keseluruhan. Dalam konteks sistem yang digunakan secara berkelanjutan, ketidakefisienan ini juga dapat memperlambat respon terhadap kebutuhan pengguna dan perubahan bisnis.

Continuous Integration (CI) merupakan pendekatan dalam pengembangan perangkat lunak yang menekankan proses penggabungan perubahan kode secara rutin oleh pengembang ke dalam satu repositori terpusat[5] Sementara itu, *Continuous Deployment* (CD) merujuk pada praktik pengembangan di mana aplikasi secara otomatis dilanjutkan ke tahap deployment dan dirilis ke lingkungan produksi setelah proses build dan pengujian berhasil dilewati, sehingga perangkat lunak dapat langsung diakses oleh pengguna akhir [6].

CI/CD di industri pengembangan perangkat lunak memungkinkan sebuah organisasi untuk merilis fitur dan produk baru secara lebih sering dan lebih baik. Berdasarkan basis data informasi yang diperoleh masih banyak organisasi atau pengembang yang menggunakan metode pengembangan manual. Metode manual ini tentunya rawan akan potensi *human error*. Penerapan *CI/CD*, dalam proses integrasi dan pengujian menjadi lebih cepat, efisien, dan mengurangi potensi *human error* [5].

Hasil penelitian yang dilakukan oleh Jaeni menunjukkkan hasil positif dalam menggunakan *CI/CD* mampu mempersingkat proses dan meningkatkan kinerja, memberikan hasil yang lebih teliti dengan penemuan *bug* pada pengujian tersebut [7]. Penelitian-penelitian sebelumnya umumnya membahas penerapan *CI/CD* pada lingkungan *cloud-native, container-based infrastructure*, atau server terkelola dengan sumber daya yang relatif besar. Namun, kajian mengenai implementasi *CI/CD* yang bersifat *lightweight* pada server VPS mandiri dengan sumber daya terbatas, khususnya untuk aplikasi backend berbasis *Express.js*, masih relatif terbatas.

Oleh karena itu, penelitian ini berfokus pada perancangan dan evaluasi model *CI/CD* berbasis *GitHub Actions* yang diterapkan pada *backend Express.js* tanpa menggunakan *container orchestration*. Kontribusi utama penelitian ini adalah penyajian evaluasi kuantitatif terhadap efisiensi *deployment* serta pembuktian bahwa pendekatan *CI/CD* sederhana dapat diterapkan secara efektif pada infrastruktur VPS berskala kecil.

2. TINJAUAN PUSTAKA

Seiring dengan berkembangnya praktik *DevOps*, *Github* menjadi salah satu platform pengelolaan repositori kode sumber yang paling banyak digunakan dalam pengembangan perangkat lunak modern. *Github* tidak hanya berfungsi sebagai sistem kontrol versi berbasis *Git*, tetapi juga menyediakan ekosistem pendukung untuk kolaborasi, manajemen kode, serta otomasi proses pengembangan perangkat lunak secara terintegrasi [8]. Salah satu fitur yang dapat dimanfaatkan dalam implementasi *CI/CD* pada *Github* adalah *Github actions*. Layanan ini menyediakan fasilitas bagi pengembang untuk merancang serta mengotomatisasi alur kerja pengembangan perangkat lunak, termasuk proses pengujian, build, dan deployment yang dijalankan secara otomatis ketika terjadi perubahan pada kode sumber [9]. Melalui *Github actions* selaras dengan nilai-nilai *Agile* dan *DevOps* yang menekankan pentingnya kolaborasi, adaptasi yang cepat terhadap perubahan, serta percepatan proses pengiriman perangkat lunak. Dampaknya, organisasi atau perusahaan dapat merespons kebutuhan pasar dan pengguna dengan lebih cepat sekaligus mempertahankan daya saing di tengah lingkungan bisnis yang semakin kompetitif [10].

Dalam pengembangannya, penggunaan *VPS* sering digunakan dalam industri, *VPS* merupakan opsi selain menggunakan hosting. Virtual Private Server (*VPS*) merupakan salah satu bentuk layanan komputasi virtual yang menyediakan sumber daya server secara terisolasi dalam satu infrastruktur fisik menggunakan teknologi virtualisasi.

Pemanfaatannya bahasa javascript dalam backend dimudahkan dengan adanya *Node.js* dalam pengembangannya. *Node.js* merupakan sebuah *runtime* environment yang memungkinkan eksekusi JavaScript di luar peramban web dengan memanfaatkan mesin V8, sehingga banyak digunakan dalam pengembangan aplikasi sisi *back-end* [11]. Pemanfaatan *PM2* dan *Express js* sering digunakan untuk tahap *production*, *PM2* dan *Express js* saling melengkapi dalam pengembangan dan *deployment* aplikasi *backend* modern berbasis *Node.js*. *Express js* berperan sebagai *framework* minimalis yang menyediakan *routing*, *middleware*, dan fitur *HTTP utility* sehingga pengembang dapat membangun *REST API* dan aplikasi web dengan cepat dan terstruktur. *Express* dikenal sebagai *framework* yang populer, dan fleksibel, sehingga mudah diintegrasikan dengan berbagai arsitektur dan pustaka lain. *PM2* memastikan bahwa algoritma dan *API* tetap berjalan serta responsif, serta menyediakan fitur pemantauan yang komprehensif sehingga pengguna dapat melacak metrik kinerja secara *real-time* [12].

Penelitian ini dilakukan dengan memulai tahap perancangan dan implementasi *CI/CD* menggunakan *Github actions*, dengan integrasi repositori melalui *Github*, *framework* yang digunakan yakni *Express js* dalam *VPS* digunakan *PM2*. Pemilihan *Github actions* dikarenakan *Github actions* terintegrasi langsung dengan *Github repository*, dan juga *Github actions* tidak memerlukan server sendiri, dan adanya *marketplace actions*, serta manajemen *secrets* terintegrasi manjadikan *Github actions* dipilih untuk implementasi otomatisasi ini. Beberapa proses siklus pengembangan penting yang dapat diotomatisasi menggunakan *Github actions* diantaranya yaitu: pengujian unit, integrasi berkelanjutan, dan penyebaran ke lingkungan produksi [13]. Pada Studi kasus pada proyek *backend* berbasis Golang yang dilakukan oleh Anggraeni menunjukkan bahwa *pipeline CI/CD* dengan *Github actions* mampu menghasilkan *deployment* otomatis yang stabil dan efisien, membuktikan bahwa pendekatan ini tidak

hanya cocok untuk aplikasi web *frontend* tetapi juga *backend* [14].

Penelitian ini, diharapkan dapat memberikan wawasan terkait *CI/CD* khususnya dalam menggunakan *Github actions* untuk *framework Express js* pada *VPS*. Penelitian yang dilakukan menitikberatkan implementasi *CI/CD backend* sistem *ediagnostic-misconception* yang menggunakan *framework Express js* pada server *VPS*. Kontribusi utama penelitian ini adalah penyusunan dan evaluasi model implementasi *CI/CD* berbasis *Github Actions* pada lingkungan VPS mandiri untuk aplikasi backend *Express.js*. Penelitian ini menitikberatkan pada pendekatan *CI/CD* ringan (*lightweight*) yang dapat diterapkan pada infrastruktur dengan sumber daya terbatas. Selain itu, penelitian ini menyajikan evaluasi kuantitatif terhadap efisiensi deployment, konsistensi konfigurasi, dan potensi pengurangan human error, yang belum banyak dibahas dalam penelitian *CI/CD* berbasis *Github Actions* sebelumnya.

Beberapa penelitian sebelumnya telah membahas penerapan *CI/*CD menggunakan berbagai *platform* dan *tools*. Penelitian oleh Zulhakim dan Kurniawan mengkaji implementasi CI/CD berbasis *Docker* dan *Github Actions* pada aplikasi web, dengan fokus pada otomatisasi berbasis *container* [15]. Peters serta Sannino dan Bruneo menekankan aspek *reproducibility* dan *workflow automation* pada *pipeline CI/CD* menggunakan *Github Actions* [16] [13]. Selain itu, Anggraeni dll menerapkan *CI/CD* pada *backend* berbasis *Golang-Echo* yang berjalan pada lingkungan server terkelola [14].

Meskipun demikian, sebagian besar penelitian tersebut berfokus pada lingkungan *cloud-native*, *containerized* infrastructure, atau platform terkelola, sehingga kurang membahas penerapan *CI/CD* pada infrastruktur VPS mandiri dengan sumber daya terbatas. Selain itu, evaluasi kuantitatif yang membandingkan *deployment* manual dan otomatis pada aplikasi *backend* berbasis *Express.js* masih relatif terbatas. Oleh karena itu, penelitian ini mengisi celah tersebut dengan merancang dan mengevaluasi model *CI/CD* berbasis *Github Actions* yang diterapkan pada *backend Express.js* di lingkungan *VPS* mandiri. Kebaruan penelitian ini terletak pada pendekatan *CI/CD lightweight* yang tidak bergantung pada *container orchestration*, namun tetap mampu meningkatkan efisiensi *deployment*, konsistensi konfigurasi, serta mengurangi risiko *human error* secara signifikan.

3. METODE PENELITIAN

3.1. Desain Penelitian

Penelitian ini menggunakan pendekatan eksperimen terapan (applied experimental research), di mana sistem CI/CD dirancang, diimplementasikan, dan dievaluasi berdasarkan metrik kinerja deployment. Evaluasi dilakukan dengan membandingkan proses deployment manual dan otomatis berbasis CI/CD.

3.2. Lingkungan dan Spesifikasi Sistem

Implementasi sistem dilakukan pada server Virtual Private Server (VPS) dengan spesifikasi RAM 1 GB, CPU 1 vCPU, dan storage 20 GB dengan sistem operasi Ubuntu 22.04 LTS. Tools yang digunakan meliputi Node.js, Nginx, PM2, Git, GitHub Actions, SSL (Let's Encrypt), serta Postman untuk pengujian API.

3.3. Perancangan Pipeline CI/CD

Pipeline CI/CD dirancang menggunakan GitHub Actions yang terintegrasi dengan repositori GitHub. Setiap perubahan kode pada branch utama akan memicu workflow otomatis yang terdiri dari beberapa tahapan, yaitu:

a. Trigger workflow
b. Checkout repository
c. Setup Node.js
d. Install dependencies
e. Deploy ke VPS melalui SSH
f. Restart aplikasi menggunakan PM2

3.4. Metode Pengujian Sistem

Pengujian dilakukan untuk memastikan bahwa setiap tahapan dalam pipeline CI/CD berjalan dengan baik. Pengujian dilakukan dengan metode pengujian fungsional yang meliputi trigger workflow, instalasi dependensi, proses deployment, serta pengujian aplikasi melalui API.

Tabel 1. Metode Pengujian Fungsional

Jenis Uji	Metode	Deskripsi	Keterangan
Uji *Trigger Workflow*	*Push* kode baru ke *branch* utama	Mengecek apakah *Github* action otomatis berjalan	Workflow otomatis aktif dan *pipeline* dijalankan
Uji Instalasi Dependensi	Observasi log tahapan npm *install*	Memastikan semua dependensi berhasil diunduh	Semua dependensi terpasang tanpa error
Uji *Deploy*	Observasi tahapan SSH dan upload File	Mengecek apakah *Github actions* berhasil mengirim file ke *VPS*	Koneksi SSH berhasil dan file aplikasi tersalin di *VPS*
Uji jalankan Aplikasi di *VPS*	Mengecek proses *PM2* status di server	Memastikan aplikasi berjalan setelah *deploy*	Aplikasi berstatus online dan dapat diakses
Uji Akses Aplikasi	Akses aplikasi melalui pengujian *API* Postman	Memastikan Aplikasi berjalan dengan baik	Hasil aplikasi memberikan response baik

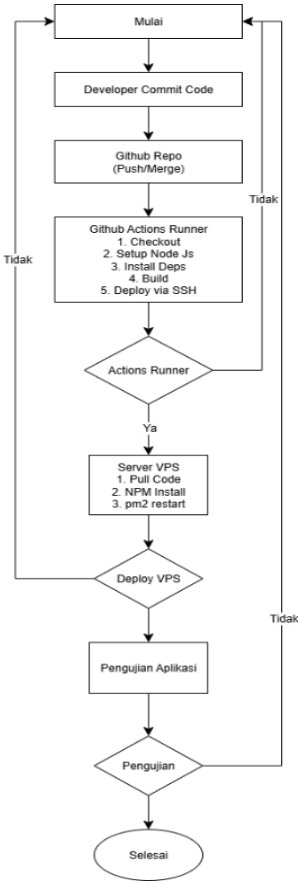
3.5. Metode Evaluasi

Evaluasi dilakukan dengan membandingkan proses deployment manual dan otomatis berdasarkan waktu, konsistensi, dan risiko kesalahan. Penelitian ini bertujuan untuk merancang dan mengimplementasikan *CI/CD Github actions* untuk meningkatkan efisiensi *deployment*. Data dalam penelitian dikumpulkan yang bertujuan untuk pengujian fungsional dalam penerapan *CI/CD* pada sistem *ediagnostic-misconception* yang menggunakan *framework Express js*.

Pengujian dilakukan untuk membandingkan proses deployment manual dan deployment otomatis berbasis CI/CD. Setiap metode deployment diuji sebanyak lima kali pada kondisi server VPS yang sama, yaitu dalam keadaan idle tanpa beban trafik pengguna.

Waktu deployment dihitung sejak proses pengiriman kode ke repositori utama hingga aplikasi backend kembali aktif dan dapat diakses melalui endpoint API. Pengukuran waktu dilakukan berdasarkan log proses deployment dan waktu eksekusi pipeline GitHub Actions.

Parameter evaluasi yang digunakan meliputi waktu deployment, jumlah langkah manual, konsistensi proses, serta potensi kesalahan konfigurasi. Hasil pengujian kemudian dibandingkan untuk menilai efektivitas penerapan CI/CD terhadap proses deployment aplikasi backend.Data yang dikumpulkan.



Gambar 1. Diagram Proses Pengujian

Diagram alur *CI/CD* menunjukkan pemisahan yang jelas antara tahapan integrasi dan *deployment*, sehingga setiap perubahan kode tervalidasi sebelum diterapkan ke lingkungan produksi. Pendekatan ini mendukung prinsip DevOps dengan meningkatkan *traceability* dan *repeatability* proses *deployment*. Kode ke server *VPS*. Dimulai dari push code ke branch utama, *Github* mendeteksi adanya push atau merge ke branch utama. Event ini akan menjadi trigger untuk menjalankan workflow *Github actions*. Kemudian *Github actions* mulai menjalankan *pipeline* otomatis sesuai file konfigurasi dari *Github*/workflows/*deploy*.yml, adapun tahapan ini yakni checkout (mengambil kode dari *repository*), setup *Node.js*, *install* dependencies, *build*, kemudian *deploy* via *ssh* jika berhasil maka proses di sisi server dijalankan secara otomatis yakni *pull code*, *install* dependencies dan restart *PM2*. Tahap selanjutnya yakni memastikan aplikasi berhasil *deploy* dengan menjalankan aplikasi *backend* system di postman.

Tahap evaluasi dilakukan untuk memastikan bahwa implementasi *CI/CD* yang telah dirancang sebelumnya telah berjalan dengan baik dan sesuai. Sehingga tujuan untuk meningkatkan proses *deployment* dalam tahap pengembangan dan pengurangan resiko *human error* dapat teratasi dengan baik. Evaluasi ini dilakukan melalui monitoring langsung terhadap proses setiap *pipeline* dijalankan. Rincian evaluasinya sebagai berikut:

Table 2. Rencana Hasil Evaluasi Implementasi

Tahapan	Status	Keterangan
Trigger Workflow	-	Workflow aktif saat *push*/*merge*
Checkout Repository	-	Berhasil mengambil sumber kode
Setup Node.js & Install Deps	-	Semua paket berhasil ter*install*
Deploy ke VPS	-	Berhasil *Deploy*
Restart PM2	-	Aplikasi *Express js* berjalan kembali
Pengujian Aplikasi	-	Pengujian Aplikasi *Backend* melalui *Postman*

Tabel di atas menggambarkan tahapan proses *Continuous Integration/Continuous Deployment* (*CI/CD*) menggunakan *Github actions* untuk aplikasi *Express*.js yang di-*deploy* ke server *VPS*. Setiap tahapan merepresentasikan langkah penting dalam otomatisasi *pipeline* — dimulai dari pemicu workflow ketika terjadi push atau merge pada branch utama, hingga proses pengujian akhir aplikasi setelah *deployment*.

Pada tahap awal, workflow diaktifkan secara otomatis, kemudian sistem melakukan checkout *repository* untuk mengambil kode sumber dari *Github*. Selanjutnya, *Github actions* menyiapkan lingkungan *Node.js* dan menginstal seluruh dependensi proyek agar siap dijalankan. Setelah semua dependensi berhasil diinstal, proses *deploy* dilakukan ke server *VPS*, diikuti dengan restart layanan menggunakan

PM2 untuk memastikan aplikasi berjalan kembali tanpa gangguan. Tahap terakhir adalah pengujian aplikasi *backend* menggunakan Postman, yang berfungsi untuk memverifikasi bahwa seluruh endpoint *API* berfungsi sesuai ekspektasi setelah proses *deployment* selesai.

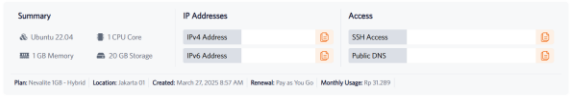
4. HASIL DAN PEMBAHASAN

4.1. Implementasi Infrastruktur Sistem

Implementasi sistem dilakukan pada server VPS yang digunakan sebagai lingkungan produksi aplikasi backend berbasis Express.js. Spesifikasi VPS yang digunakan dapat dilihat pada Tabel 3.

Tabel 3. Spesifikasi Virtual Private Server (VPS)

Aspek	Deskripsi
Provider	NevaCloud
CPU	1 vCPU
RAM	1 GB
Storage	20 GB
OS	*Ubuntu 22.04 LTS*



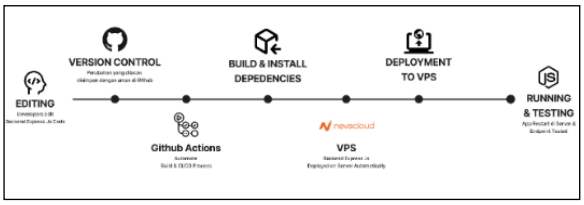
Gambar 2. Spesifikasi VPS

Penggunaan VPS ini memberikan fleksibilitas dalam pengelolaan server serta mendukung penerapan otomatisasi CI/CD. Penggunaan *VPS* dengan pilihan spesifikasi diatas karena mampu memenuhi kebutuhan operasional aplikasi *backend* untuk system *ediagnostic-misconception framework Express js* yang relative ringan namun membutuhkan kestabilan dan control penuh terhadap lingkungan server. Dengan dukungan *VPS* tersebut dapat dilakukan instalasi, konfigurasi, serta pengelolaan layanan seperti *PM2*, Nginx, dan *Node.js* secara independen tanpa keterbatasan yang sering dijumpai pada layanan hosting bersama (shared hosting). Selain itu, sistem operasi Ubuntu 22.04 LTS yang bersifat stabil dan banyak didukung komunitas open-source menjadikan *VPS* ini cocok sebagai lingkungan produksi untuk penerapan otomatisasi *CI/CD* menggunakan *Github actions*.

4.2. Implementasi Pipeline CI/CD

Pipeline CI/CD diimplementasikan menggunakan GitHub Actions yang terintegrasi dengan repositori GitHub. Setiap perubahan kode pada branch utama akan memicu proses otomatis. Alur implementasi CI/CD meliputi tahapan editing, version control, build & install dependencies, deployment ke VPS, serta running dan testing aplikasi. Alur ini dapat dilihat pada Gambar 3.

Alur implementasi *Continuous Integration* dan *Continuous Deployment* (*CI/CD*) pada pengembangan sistem *backend* berbasis *Express*.js dengan memanfaatkan layanan *Github actions* dan Virtual Private Server (*VPS*). Alur tersebut terdiri atas lima tahapan utama, yaitu editing, version control, *build* & *install* dependencies, *deployment* to *VPS*, serta running & testing.



Gambar 3. Alur Implementasi *CI/CD*

Pada tahap editing, pengembang melakukan proses perancangan dan pengembangan kode sumber aplikasi *backend* menggunakan bahasa pemrograman JavaScript dengan kerangka kerja *Express*.js. Tahap ini merupakan fase awal dalam siklus pengembangan perangkat lunak, di mana setiap perubahan kode ditulis dan diuji secara lokal.

Selanjutnya, tahap version control dilakukan untuk mengelola perubahan kode menggunakan sistem kontrol versi *Git* yang terintegrasi dengan repositori *Github*. Proses ini memungkinkan sinkronisasi kode antar pengembang dan mendukung kolaborasi dalam tim pengembang melalui pelacakan setiap versi perubahan yang dilakukan.

Tahap *build* & *install* dependencies dijalankan secara otomatis melalui *Github actions* setelah kode dikirim (push) ke repositori utama. Pada fase ini, sistem secara otomatis melakukan instalasi seluruh dependensi yang diperlukan serta proses *build* untuk memastikan kode siap diimplementasikan di lingkungan server. Otomatisasi ini bertujuan untuk menjamin konsistensi lingkungan pengembangan dan mengurangi kesalahan manusia (*human error*).

Berikutnya, tahap *deployment* to *VPS* merupakan proses pengiriman dan penerapan kode hasil *build* ke server produksi, dalam hal ini menggunakan layanan *VPS* dari penyedia Nevacloud. Melalui *Github actions*, proses ini berlangsung secara otomatis sehingga aplikasi *backend* dapat diperbarui di server tanpa intervensi manual dari pengembang.

Tahap terakhir yaitu running & testing, di mana aplikasi *backend* *Express*.js dijalankan pada lingkungan server dan dilakukan pengujian terhadap endpoint atau antarmuka pemrograman aplikasi (*API*) untuk memastikan sistem berfungsi sesuai dengan rancangan. Proses pengujian ini juga bertujuan untuk memastikan keberhasilan penerapan *pipeline CI/CD* secara keseluruhan.

Secara keseluruhan, alur ini menunjukkan penerapan otomatisasi pengembangan perangkat lunak yang efektif dan efisien melalui integrasi *Github actions* dan *VPS*. Implementasi *CI/CD* tersebut meningkatkan keandalan sistem, mempercepat siklus rilis, serta meminimalkan risiko kesalahan pada saat proses *deployment*.

4.3. Hasil Evaluasi Implementasi

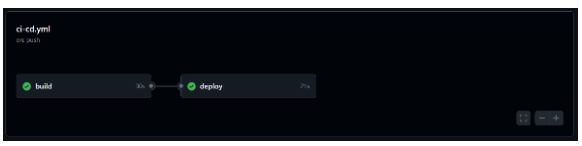
Berdasarkan hasil implementasi, seluruh tahapan pipeline CI/CD berhasil dijalankan dengan baik. Setiap tahapan seperti trigger workflow, checkout repository, instalasi dependensi, deployment ke VPS, serta restart aplikasi menggunakan PM2 menunjukkan status berhasil.

Hasil evaluasi implementasi ditunjukkan pada Tabel 3 dan Gambar 4.

Table 3. Hasil Evaluasi Implementasi

Tahapan	Status	Waktu Eksekusi (detik)	Keterangan
Trigger Workflow	Berhasil	2-3	Workflow aktif saat *push*/*merge*
Checkout Repository	Berhasil	5-7	Berhasil mengambil sumber kode
Setup Node.js & Install Deps	Berhasil	20-30	Semua paket berhasil ter*install*
Deploy ke *VPS*	Berhasil	10-15	Berhasil *Deploy*
Restart PM2	Berhasil	3-5	Aplikasi *Express js* berjalan kembali
Pengujian Aplikasi	Berhasil	2-3	Pengujian Aplikasi *Backend* melalui Postman

Berdasarkan Tabel 3, seluruh tahapan dalam pipeline CI/CD berhasil dijalankan dengan waktu eksekusi yang relatif singkat. Tahap instalasi dependensi menjadi proses yang paling memakan waktu, yaitu sekitar 20–30 detik, karena sistem harus mengunduh dan mengonfigurasi paket Node.js. Secara keseluruhan, total waktu eksekusi pipeline berkisar antara 40 hingga 60 detik. Hal ini menunjukkan bahwa proses deployment otomatis dapat dilakukan dalam waktu kurang dari satu menit, sehingga jauh lebih efisien dibandingkan metode manual.



Gambar 4. *Deploy CI/CD* Berhasil

Hasil implementasi *Continuous Integration* dan *Continuous Deployment* (*CI/CD*) menggunakan *Github actions* menunjukkan bahwa proses *deployment* aplikasi *backend* pada system *ediagnostic-misconception* berbasis *Express*.js pada server *VPS* dapat berjalan secara otomatis dan konsisten. Setiap kali pengembang melakukan commit atau merge ke branch utama, *Github actions* akan men-trigger proses otomatis yang terdiri dari beberapa tahap utama: checkout *repository*, setup *Node.js*, *install* dependencies, *build* project, serta *deployment* ke *VPS* melalui koneksi SSH yang aman.

Setelah proses *deployment* selesai, server *VPS* akan menarik versi kode terbaru, melakukan instalasi dependensi melalui npm *install*, dan menjalankan ulang aplikasi menggunakan *PM2*. Dengan otomatisasi ini, seluruh proses *deployment* dapat diselesaikan dalam waktu kurang dari satu menit tanpa perlu intervensi manual melalui terminal SSH. Hal ini meningkatkan efisiensi kerja tim dan mengurangi risiko *human error* yang umum terjadi pada proses manual.

Pada Tabel 3 dan Gambar 4 menunjukkan hasil evaluasi implementasi proses *Continuous Integration* dan *Continuous Deployment* (*CI/CD*) pada sistem *backend* berbasis *Express*.js yang diotomatisasi menggunakan *Github actions* dan server *Virtual Private Server* (*VPS*). Berdasarkan hasil pengujian, seluruh tahapan dalam *pipeline CI/CD* menunjukkan status "berhasil", mulai dari pemicu workflow yang aktif secara otomatis saat proses push atau merge di repositori *Github*, pengambilan kode sumber melalui checkout *repository*, hingga proses instalasi dependensi *Node.js* yang berjalan tanpa kendala. Proses *deployment* ke *VPS* juga berhasil dilakukan, diikuti dengan restart otomatis menggunakan *PM2* yang memastikan aplikasi *Express*.js berjalan kembali setelah pembaruan.

Tahap akhir pengujian aplikasi yang dilakukan melalui Postman dan juga integrasi dengan *frontend* menunjukkan bahwa seluruh endpoint *backend* berfungsi sesuai dengan rancangan dapat dilihat pada gambar 5.

Gambar 5. Integrasi *Backend* dan Frontend

Secara keseluruhan, hasil evaluasi ini membuktikan bahwa penerapan *CI/CD* dengan *Github actions* mampu meningkatkan efisiensi dan keandalan proses *deployment* aplikasi *backend*. Otomatisasi yang diterapkan meminimalkan potensi kesalahan manusia, mempercepat siklus rilis, serta memastikan kestabilan sistem di lingkungan server produksi.

4.4. Analisis Perbandingan Deployment

Perbandingan antara deployment manual dan deployment berbasis CI/CD ditunjukkan pada Tabel 4.

Table 4. Tabel Perbandingan Manual vs CI/CD

Parameter	Manual	CI/CD
Waktu deployment	±15 menit	±3.2 menit
Langkah manual	8 langkah	1 trigger
Risiko error	Tinggi	Rendah
Konsistensi	Variatif	Konsisten

Berdasarkan hasil pengujian, metode deployment manual memerlukan waktu rata-rata sekitar 15 menit, sedangkan metode CI/CD hanya memerlukan sekitar 3 menit. Hal ini menunjukkan peningkatan efisiensi sebesar sekitar 80%. Penerapan CI/CD mampu menurunkan waktu deployment secara signifikan dibandingkan metode manual. Hal ini terjadi karena proses CI/CD menghilangkan langkah-langkah manual seperti login ke server melalui SSH, konfigurasi ulang dependensi, dan restart aplikasi secara manual. Otomatisasi melalui GitHub Actions memastikan bahwa setiap proses deployment dijalankan dengan alur yang sama dan tervalidasi, sehingga mengurangi potensi kesalahan konfigurasi. Selain itu, penggunaan PM2 memungkinkan aplikasi backend untuk dijalankan kembali secara otomatis tanpa downtime yang signifikan.

Hasil ini sejalan dengan konsep DevOps yang menekankan otomatisasi, konsistensi, dan kecepatan dalam siklus pengembangan perangkat lunak. Menunjukkan pengurangan waktu deployment sebesar 80%, yang berdampak langsung pada peningkatan produktivitas pengembang. Selain itu, eliminasi langkah manual menurunkan potensi kesalahan konfigurasi yang sering terjadi pada proses deployment tradisional.

Selain itu, jumlah langkah manual berkurang dari delapan langkah menjadi satu trigger otomatis. Proses deployment juga menjadi lebih konsisten dan memiliki risiko kesalahan yang lebih rendah dibandingkan metode manual.

Tabel 5. Hasil Pengujian Waktu Deployment

Percobaan	Manual (menit)	CI/CD (menit)
1	15	3
2	14	3
3	16	4
4	15	3
5	15	3
Rata-rata	14	3.2

Berdasarkan hasil pengujian yang ditunjukkan pada Tabel 4 dan Tabel 5, terdapat perbedaan yang signifikan antara metode deployment manual dan deployment berbasis CI/CD. Pada metode manual, waktu deployment rata-rata yang dibutuhkan adalah sekitar 15 menit. Hal ini disebabkan oleh banyaknya tahapan yang harus dilakukan secara manual, seperti login ke server melalui SSH, melakukan pull repository, instalasi dependensi, serta restart aplikasi.

Sebaliknya, pada metode CI/CD, waktu deployment rata-rata hanya sekitar 3.2 menit. Proses ini dijalankan secara otomatis oleh GitHub Actions, sehingga menghilangkan intervensi manual pada setiap tahapan deployment. Dengan demikian, terjadi pengurangan waktu deployment sebesar sekitar 11.8 menit atau sekitar 78–80% lebih cepat dibandingkan metode manual.

Selain itu, jumlah langkah deployment juga berkurang secara signifikan dari delapan langkah manual menjadi satu trigger otomatis. Hal ini tidak hanya meningkatkan efisiensi waktu, tetapi juga mengurangi potensi kesalahan konfigurasi yang sering terjadi pada proses manual. Dari sisi konsistensi, deployment manual cenderung bersifat variatif karena bergantung pada kondisi dan ketelitian pengembang. Sementara itu, deployment berbasis CI/CD memiliki alur yang terstandarisasi, sehingga menghasilkan proses yang konsisten pada setiap eksekusi. Dengan demikian, hasil penelitian ini menunjukkan bahwa penerapan CI/CD tidak hanya meningkatkan efisiensi waktu deployment, tetapi juga meningkatkan keandalan dan stabilitas sistem secara keseluruhan.

5. KESIMPULAN DAN SARAN

Berdasarkan hasil implementasi dan evaluasi yang telah dilakukan, dapat disimpulkan bahwa penerapan Continuous Integration dan Continuous Deployment (CI/CD) menggunakan GitHub Actions pada sistem backend berbasis Express.js berhasil meningkatkan efisiensi dan keandalan proses deployment.

Hasil pengujian menunjukkan bahwa waktu deployment pada metode manual memerlukan rata-rata sekitar 15 menit, sedangkan dengan penerapan CI/CD waktu deployment dapat dipersingkat menjadi sekitar 3.2 menit. Hal ini menunjukkan adanya peningkatan fisiensi waktu sebesar sekitar 78–80%. Selain itu, jumlah langkah deployment juga berkurang secara signifikan dari delapan langkah manual menjadi satu trigger otomatis pada pipeline GitHub Actions.

Dari sisi konsistensi, metode deployment manual menunjukkan hasil yang variatif dan bergantung pada proses yang dilakukan oleh pengembang, sedangkan metode CI/CD menghasilkan proses yang lebih konsisten dan terstandarisasi. Risiko kesalahan konfigurasi juga dapat diminimalkan karena seluruh tahapan deployment dijalankan secara otomatis.

Dengan demikian, penerapan CI/CD berbasis GitHub Actions pada server VPS terbukti mampu meningkatkan efisiensi, konsistensi, dan stabilitas dalam proses deployment aplikasi backend. Meskipun hasil penelitian menunjukkan peningkatan yang signifikan, penelitian ini masih memiliki keterbatasan, yaitu hanya dilakukan pada satu sistem dengan beban trafik yang rendah. Oleh karena itu, penelitian selanjutnya dapat dilakukan pada sistem dengan skala yang lebih besar serta mempertimbangkan aspek keamanan dan skalabilitas secara lebih mendalam. Seluruh tahapan dalam *pipeline*, mulai dari proses trigger workflow, instalasi dependensi, *deployment* ke server *VPS*, hingga restart aplikasi dan pengujian endpoint, menunjukkan hasil yang berhasil tanpa adanya kesalahan. Hal ini menunjukkan bahwa

integrasi antara *Github actions* dan lingkungan *VPS* dapat mendukung otomatisasi proses pengembangan dan penerapan aplikasi secara efektif.

Secara keseluruhan, penerapan *CI/CD* ini mampu meningkatkan efisiensi kerja pengembang, mempercepat waktu *deployment*, serta mengurangi risiko kesalahan manual yang sering terjadi pada proses penerapan tradisional. Dengan demikian, sistem *CI/CD* berbasis *Github actions* dapat dijadikan solusi yang andal untuk menjaga konsistensi, kestabilan, dan kualitas dalam pengelolaan serta pembaruan sistem *backend* di lingkungan produksi.

Meskipun hasil penelitian menunjukkan peningkatan signifikan, penelitian ini terbatas pada satu studi kasus aplikasi backend dengan beban trafik rendah. Evaluasi skalabilitas dan keamanan tingkat lanjut belum dilakukan. Keterbatasan ini membuka peluang penelitian lanjutan pada lingkungan produksi berskala besar.

DAFTAR PUSTAKA

[1] R. Setiawan, "Metode SDLC Dalam Pengembangan Software," *Dicoding Online*, Jul. 2021, [Online]. Available: https://www.dicoding.com/blog/metode-sdlc/

[2] R. Setiabudi, "Penerapan Continuous Integration dan Continuous Delivery pada Sistem Informasi Akademik," *Jurnal Ismetek*, vol. 17, no. 2, 2024.

[3] J. Humble and D. Farley, "Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education," 2020.

[4] S. Gujar and S. Patil, "Continuous Integration and Continuous Deployment (CI/CD) Optimization," *International Journal of Innovative Science and Research Technology (IJISRT)*, pp. 429–437, Oct. 2024, doi: 10.38124/ijisrt/ijisrt24oct014.

[5] H. Toba, T. K. Gautama, J. Narabel, A. Widjaja, and S. F. Sujadi, "Evaluasi Metodologi CI/CD untuk Pengembangan Perangkat Lunak dalam Perkuliahan," *Jurnal Edukasi Dan Penelitian Informatika*, vol. 8, no. 2, 2022.

[6] I. Guna Noviantama and A. W. Purno Wahyu, "MILLENNIA SOLUSI INFORMATIKA," *Jurnal Ilmiah Teknologi Informasi Terapan*, vol. 8, no. 1, 2021.

[7] N. A. Jaeni, "Implementasi Continuous Integration/Continuous Delivery (Ci/Cd) Pada Performance Testing Devops," *Joism : Jurnal Of Information System Management*, 2022.

[8] Jon Loeliger and Matthew McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, Inc., 2012.

[9] M. Wessel, J. Vargovich, M. A. Gerosa, and C. Treude, "GitHub Actions: The Impact on the Pull Request Process," *Empir. Softw. Eng.*, vol. 28, no. 6, 2023, doi: https://doi.org/10.1007/s10664-023- 10369-w.

[10] A. Dwi Praba and T. Santoso, "Pengembangan Aplikasi Point Of Sales Menggunakan Metode AGILE Dengan Pola Scrum," *JIKA : Jurnal Informatika*, vol. 7, no. 2, 2023.

[11] D. Herron, *Node.js web development server-side web development made easy with Node 14 using practical examples by David Herron*, 5th ed. Packt Publishing, 2020.

[12] D. Yulianto, A. F. Nugraha, and F. F. Rahani, "Automated Hydroponics System using the Internet of Things," *Jurnal Edukasi Elektro*, vol. 8, no. 2, Nov. 2024, doi: 10.21831/jee.v8i2.76816.

[13] L. Sannino and D. Bruneo, "Improving The Reproducibility Of Continuous Integration And Continuous Deployment Pipelines With Github Actions," *J. Open Res. Softw.*, p. 33, 2022.

[14] M. D. Anggraeni, F. S. Utomo, and H. Marcos, "Integrasi Backend Golang-Echo pada Aplikasi Greenly sebagai Solusi Teknologi Pengelolaan Sampah Digital," *Jurnal Informatika: Jurnal pengembangan IT*, vol. Vol. 10, No. 2, 2025, 2025.

[15] Z. Zulhakim and A. Kurniawan, "Implementasi Continuous Integration Dan Continuous Deployment Pada Pengembangan Aplikasi Website Menggunakan Docker Dan Github Actions," *Jurnal Managemen Informatika*, vol. 13, no. 1, pp. 1–11, 2024.

[16] B. Peters, "Automating The Ci/Cd Workflow With Github Actions," *J. Open Source Softw.*, p. 3061, 2021