Jurnal Tekno Insentif | Vol. 20 | No. 1 | Juli 2026
DOI : http://dx.doi.org/10.36787/jti.v20i1.7197 Halaman 51 - 65

Microservices Integration Using CI/CD Pipelines

BAGAS AGUNG WIYONO, FATIMAH AZZAHRA NUR FAIDAH, RONI ANDARSYAH

Universitas Logistik dan Bisnis Internasional, Indonesia
Email : 714240042@std.ulbi.ac.id

Received 30 November 201x **|** *Revised* 30 Desember 201x **|** *Accepted* 30 Januari 201x

ABSTRAK

Penelitian ini mengimplementasikan pipeline *Continuous Integration* dan *Continuous Deployment* (CI/CD) pada pengembangan aplikasi *web* Sistem Bimbingan Online menggunakan GitHub Actions dan Google Cloud Functions untuk mengatasi permasalahan *deployment* manual yang lambat, tidak konsisten, dan rawan kesalahan manusia. Pipeline dirancang dengan dua *workflow* utama yaitu *workflow* Build untuk pengujian otomatis (Jest) dan analisis kualitas kode (SonarCloud), serta *workflow* Deployment untuk rilis otomatis ke Google Cloud Functions. Pengukuran dilakukan selama 7 minggu dengan 114 eksekusi *workflow*. Hasil menunjukkan tingkat keberhasilan *build* 79,82%, frekuensi *deployment* 14 kali per minggu, dan rata-rata waktu eksekusi 2 menit 47 detik. Proyek mendapatkan *rating* A untuk *Bugs*, *Vulnerabilities*, dan *Code Smells* dari SonarCloud. Implementasi CI/CD berhasil mengurangi waktu *deployment* dari 15–30 menit secara manual menjadi rata-rata 2–3 menit secara otomatis.

Kata kunci: *CI/CD*, *GitHub Actions*, *Google Cloud Functions*, *deployment* otomatis, aplikasi *web*, *DevOps*

ABSTRACT

This research implements a Continuous Integration and Continuous Deployment (CI/CD) pipeline in the development of the Online Guidance System web application using GitHub Actions and Google Cloud Functions to address slow and error-prone manual deployment processes. The pipeline consists of two main workflows: a Build workflow for automated testing (Jest) and code quality analysis (SonarCloud), and a Deployment workflow for automated release to Google Cloud Functions. Measurements were conducted over 7 weeks with 114 workflow executions. Results showed a build success rate of 79.82%, deployment frequency of 14 times per week, and average execution time of 2 minutes 47 seconds. The project received an A rating for Bugs, Vulnerabilities, and Code Smells from SonarCloud. CI/CD implementation successfully reduced deployment time from 15–30 minutes manually to an average of 2–3 minutes automatically.

***Keywords**: CI/CD, GitHub Actions, Google Cloud Functions, automated deployment, web application, DevOps*

1. PENDAHULUAN

Perkembangan industri perangkat lunak saat ini menuntut proses pengembangan yang cepat tanpa mengorbankan kualitas produk. Aplikasi harus bisa ditingkatkan kapasitasnya, memiliki performa yang baik, dan selalu tersedia ketika dibutuhkan pengguna. Hal ini menjadi salah satu alasan mengapa penggunaan *container* semakin populer dalam pengembangan aplikasi berbasis *cloud* **(Abhishek et al., 2022)**. Rilis aplikasi yang dilakukan secara berkala mendorong penggunaan teknologi yang mampu mempercepat proses pengiriman perangkat lunak ke *server* produksi.

Arsitektur *microservices* telah menjadi pilihan banyak perusahaan karena kemampuannya dalam menawarkan kecepatan dan fleksibilitas dalam pengembangan perangkat lunak. Akan tetapi, penerapan arsitektur ini membutuhkan infrastruktur yang dirancang dengan baik agar penggunaan *container* dapat berjalan dengan maksimal. Dalam hal ini, penerapan *version control* dan infrastruktur CI/CD menjadi sangat penting untuk mempercepat proses pengiriman perangkat lunak sekaligus memastikan kode yang dibuat sesuai dengan standar yang berlaku **(Dakić, 2024)**.

Continuous Integration dan *Continuous Deployment* (CI/CD) merupakan praktik utama dalam proses *DevOps*. Pendekatan ini dapat memangkas waktu pengiriman produk dengan cara mengotomatisasi tugas-tugas yang biasanya dilakukan secara manual selama proses pengembangan dan *deployment*. Berdasarkan survei *Stack Overflow* tahun 2023, sebagian besar *developer* profesional menyatakan bahwa CI/CD sudah tersedia di tempat kerja mereka, yang menandakan bahwa teknologi ini sudah menjadi hal yang umum dalam dunia pengembangan perangkat lunak **(Teumert & Tegeler, 2025)**.

Meskipun penggunaan CI/CD terus meningkat, penerapannya masih menghadapi berbagai kendala. Cara *deployment* manual yang masih banyak digunakan sering mengakibatkan ketidakstabilan performa dan menghambat kemampuan aplikasi untuk berkembang. Proses manual juga menyulitkan pemantauan performa, pengelolaan berbagai versi aplikasi, serta proses *rollback* ketika terjadi kesalahan. Kondisi ini menyebabkan sulitnya menjaga konsistensi hasil dalam proses pengembangan **(Chandra et al., 2025)**.

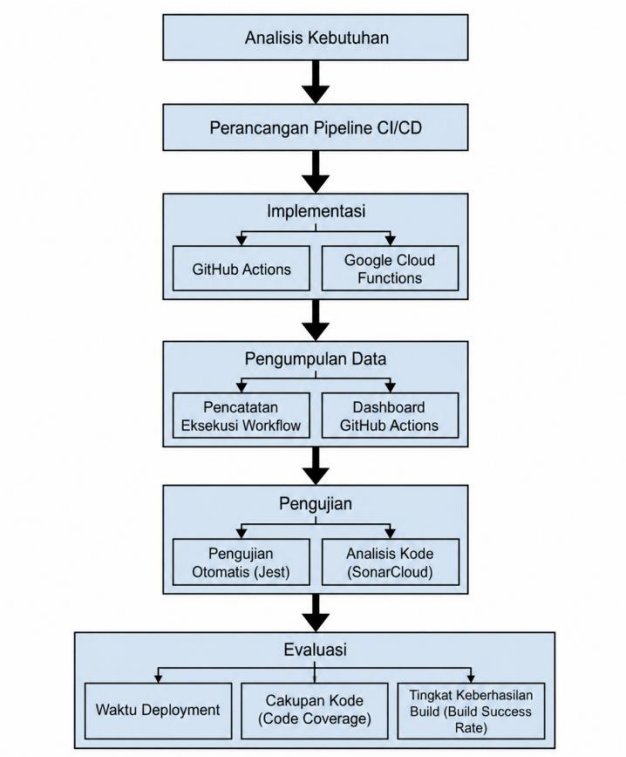
Pipeline CI/CD bekerja dengan mengotomatisasi proses *testing*, pelaporan *error*, *version control*, dan perbaikan berdasarkan *feedback* yang diterima **(Makani & Jangampeta, 2022)**. Dalam penerapannya, berbagai *tools* dapat digunakan seperti *GitHub Actions* untuk menjalankan tugas otomatis ketika kode diperbarui, serta *Docker* untuk mengemas aplikasi dalam bentuk *container* yang mudah dipindahkan **(Abhishek et al., 2022)**. *Kubernetes* kemudian berperan dalam mengelola *deployment* dan sumber daya *container* secara otomatis, sehingga aplikasi dapat berjalan dengan stabil **(Abhishek et al., 2022)**.

Berdasarkan permasalahan yang telah diuraikan, muncul beberapa pertanyaan penelitian yang perlu dijawab. Pertama, bagaimana pipeline CI/CD dapat dirancang dan diimplementasikan secara efektif pada pengembangan aplikasi web Sistem Bimbingan Online menggunakan GitHub Actions dan Google Cloud Functions? Kedua, seberapa efektif penerapan pipeline CI/CD tersebut ditinjau dari metrik *Build Success Rate* (BSR), frekuensi *deployment*, serta rata-rata waktu eksekusi *workflow*? Ketiga, bagaimana pengaruh integrasi CI/CD terhadap kualitas kode yang diukur melalui analisis SonarCloud?

2. METODE

2.1 Tahapan Penelitian

Penelitian ini menggunakan pendekatan **kuantitatif deskriptif** dengan metode **studi kasus** (*case study*). Metode studi kasus dipilih karena memungkinkan investigasi empiris terhadap fenomena rekayasa perangkat lunak secara kontemporer dalam konteks nyatanya **(Wohlin & Rainer, 2022)**. Pengumpulan data dilakukan melalui observasi langsung terhadap eksekusi *workflow* yang tercatat di *dashboard* GitHub Actions dan SonarCloud selama periode pengembangan aktif. Data yang dikumpulkan bersifat kuantitatif, meliputi metrik *Build Success Rate* (BSR), frekuensi *deployment*, dan rata-rata waktu eksekusi *workflow*.



Gambar 1. Tahapan Penelitian yang Diusulkan

Analisis Kebutuhan dilakukan untuk mengidentifikasi kebutuhan teknis aplikasi sebelum pipeline dirancang, mencakup struktur proyek, teknologi yang digunakan (Node.js, Express.js, MySQL), *branch strategy*, serta konfigurasi *environment variable* dan *deployment* yang diperlukan.

Perancangan Pipeline CI/CD menghasilkan dua *workflow* utama yang berjalan terpisah, yaitu *workflow* Build untuk *quality assurance* dan *workflow* Deployment untuk proses rilis ke Google Cloud Functions, beserta penentuan *trigger event* dan integrasi dengan SonarCloud.

Implementasi dilakukan dengan menerapkan rancangan *workflow* ke repositori GitHub menggunakan konfigurasi YAML pada direktori .github/workflows, mengintegrasikan GitHub Actions sebagai platform CI/CD dan Google Cloud Functions sebagai target *deployment* **(Manolov et al., 2025)**.

Pengumpulan **Data** dilakukan melalui observasi langsung terhadap setiap eksekusi *workflow* yang tercatat di *dashboard* GitHub Actions dan SonarCloud, meliputi status eksekusi, durasi waktu, dan metrik kualitas kode selama 7 minggu dengan total 114 kali eksekusi.

Pengujian dijalankan secara otomatis melalui Jest untuk pengujian fungsional dan SonarCloud untuk analisis kualitas kode, yang keduanya terintegrasi dalam *workflow* Build dan berjalan setiap ada *push* ke *branch* dev maupun main **(Kim et al., 2022)**.

Evaluasi mengukur efektivitas pipeline berdasarkan tiga metrik utama yaitu *Build Success Rate* (BSR), frekuensi *deployment*, dan rata-rata waktu eksekusi *workflow*, yang kemudian dibandingkan dengan kondisi sebelum implementasi CI/CD.

2.2 Objek Penelitian

Penelitian ini mengambil studi kasus pada Sistem Bimbingan Online, sebuah aplikasi *backend API* untuk mengelola proses bimbingan akademik di Program Studi D4 Teknik Informatika Universitas Logistik dan Bisnis Internasional. Aplikasi ini dibuat untuk mendigitalisasi proses bimbingan yang sebelumnya dilakukan secara manual, sehingga bisa meningkatkan efisiensi dan kemudahan akses bagi mahasiswa dan dosen.

Aplikasi ini dibangun menggunakan *Node.js* versi 22.x dengan *framework Express.js* versi 5.x yang sudah mendukung *ES Modules*. *Database* yang dipakai adalah *MySQL* dengan *library mysql2* untuk koneksinya. Sistem ini menggunakan arsitektur *REST API* dengan autentikasi *JWT* dan enkripsi *password* menggunakan *bcryptjs*. Tabel 1 menunjukkan spesifikasi teknis aplikasi yang dijadikan objek penelitian.

Tabel 1. Spesifikasi Teknik Aplikasi

Komponen	Spesifikasi
Runtime	Node.js 22.x
Framework	Express.js 5.x (ES Module)
Database	MySQL dengan mysql2
Autentikasi	JSON Web Token (JWT)
Enkripsi Password	bcryptjs
Dokumentasi API	Swagger UI

Sistem Bimbingan Online punya enam role pengguna dengan hak akses yang berbeda-beda. Tabel 2 menunjukkan daftar role beserta fitur yang bisa diakses.

Tabel 2. Role Pengguna dan Hak Akses

Role	Deskripsi
Mahasiswa	Mahasiswa D4 Teknik Informatika
Dosen	Dosen Pembimbing
Kaprodi	Kepala Program Studi
Koordinator	Koordinator Proyek/Internship
Penguji	Penguji sidang
Admin	Administrator sistem

Aplikasi ini punya beberapa fitur utama untuk mendukung proses bimbingan akademik. Ada fitur manajemen *user* untuk registrasi dan *login multi-role*, sistem bimbingan *online* dengan maksimal 8 kali pertemuan, *submit proposal* dengan *upload* dokumen dan usulan pembimbing, serta penjadwalan sidang yang dikelola koordinator.

2.3 Tools dan Teknologi

Implementasi CI/CD pada penelitian ini menggunakan beberapa *tools* yang saling terintegrasi. Penelitian oleh **(Wessel et al. 2023)** menunjukkan bahwa hampir 30% dari 5000 *repository* populer di *GitHub* sudah mengadopsi *GitHub Actions*, yang menandakan *tool* ini sudah banyak digunakan *developer*. Tabel 3 menunjukkan daftar *tools* beserta fungsi dan versinya.

Tabel 3. Tools dan Teknologi yang Digunakan

Komponen	Versi	Fungsi
Github	-	Repository untuk menyimpan source code
Github Action	v4	Platform CI/CD untuk menjalankan pipeline
Node.js	22.x	Runtime untuk menjalankan aplikasi
npm	11.x	Package manager untuk mengelola dependencies
Jest	29.7.0	Testing Framework
SonarCloud	-	Analisis kualitas kode dan code coverage
Google Cloud Function	Gen2	Platform serverless untuk hosting backend

GitHub Actions dipilih sebagai platform CI/CD karena beberapa alasan. Pertama, mudah diintegrasikan dengan *repository* GitHub yang sudah dipakai untuk *version control*, sehingga tidak memerlukan konfigurasi tambahan untuk menghubungkan antara kode sumber dan pipeline **(Wessel et al., 2023)**. Integrasi ini memungkinkan setiap perubahan kode langsung memicu eksekusi *workflow* secara otomatis tanpa perantara layanan pihak ketiga. Kedua, GitHub Actions memungkinkan automasi tugas berdasarkan berbagai *trigger* seperti *commits*, *pull requests*, *issues*, dan *comments*, sehingga tim dapat mengonfigurasi kapan dan bagaimana pipeline dijalankan sesuai kebutuhan alur kerja pengembangan **(Wessel et al., 2023)**. Ketiga, menyediakan *runner* gratis 2000 menit per bulan untuk *repository* publik, yang menjadikannya pilihan ekonomis bagi proyek pengembangan skala kecil hingga menengah tanpa biaya infrastruktur tambahan. Keempat, mendukung berbagai *workflow* yang bisa dikustomisasi melalui file YAML, memungkinkan konfigurasi pipeline yang fleksibel dan dapat di-*version control* bersama kode sumber **(Makani & Jangampeta, 2022)**.

(Manolov et al. 2025) menyatakan bahwa memilih platform CI/CD yang tepat merupakan keputusan strategis yang bisa mempengaruhi produktivitas tim *development* dan efisiensi operasional. *Google Cloud Functions* dipilih sebagai platform *deployment* karena sifatnya yang *serverless*, jadi *developer* bisa fokus ke kode tanpa harus mengelola *server*. *Google Cloud Functions Gen 2* mendukung *runtime Node.js* sampai versi 20, punya *timeout* maksimal 60 menit, dan mendukung *HTTP trigger* yang cocok dengan arsitektur *REST API* aplikasi.

2.4 Arsitektur Pipeline CI/CD

Pipeline CI/CD yang dibuat terdiri dari dua *workflow* utama yang berjalan terpisah berdasarkan *trigger* yang berbeda. Arsitektur ini dirancang untuk memisahkan proses *quality assurance* dan *deployment*, jadi kode bisa divalidasi dulu sebelum di-*deploy* ke *production*.

2.4.1 Workflow Build dan Quality Assurance

Workflow pertama bernama "*Build*" yang fokus pada proses *build*, *testing*, dan analisis kualitas kode. **(Kim et al. 2022)** menunjukkan bahwa implementasi *continuous integration* dengan *GitHub Actions* mampu mengurangi tingkat *error* dengan memberikan notifikasi cepat kepada *developer* ketika terjadi kesalahan. *Workflow* ini jalan otomatis ketika ada *event* tertentu di *repository*. Tabel 4 menunjukkan *trigger events* yang mengaktifkan *workflow Build*.

Tabel 4. Trigger Events Workflows Build

Event	Branch	Deskripsi
Push	dev	Ketika ada branch push ke branch development
Push	main	Ketika ada branch push ke branch utama
Pull Request	*	Ketika pull request di buka atau di update

Workflow Build terdiri dari beberapa langkah yang dijalankan berurutan. Pertama, *checkout repository* untuk mengambil *source code*. Kedua, *setup Node.js* versi 22 dengan *cache npm* untuk mempercepat instalasi. Ketiga, *install dependencies* pakai *npm ci*. Keempat, jalankan *test* dengan *coverage* pakai *Jest*. Terakhir, analisis kode pakai *SonarCloud*.

2.4.2 Workflow Deployment

Workflow kedua bernama "*Google Cloud Function Deployment*" yang fokus pada proses *deployment* ke *Google Cloud Functions*. *Workflow* ini jalan khusus ketika ada *push* ke *branch main*. **(Fluri et al. 2024)** menekankan bahwa adopsi CI/CD memungkinkan *developer* merespons cepat terhadap perubahan kebutuhan dan membantu menghasilkan *software* yang sudah teruji tanpa perlu pekerjaan manual tambahan. Tabel 5 menunjukkan langkah-langkah dalam *workflow* ini.

Tabel 5. Langkah-langkah Workflow Deployment

Komponen	Versi
Checkout	actions/checkout@v4
Authenticate	google-github-actions/auth@v2
Setup Cloud SDK	google-github-actions/setup-gcloud@v2
Deploy	google function deploy
Verify	google function describe

Konfigurasi *deployment* menggunakan *parameter* yang disesuaikan dengan kebutuhan aplikasi. Tabel 6 menunjukkan *parameter deployment* yang dipakai.

Tabel 6. Parameter Deployment Google Cloud Function

Parameter	Nilai
Region	asia-southeast2
Runtime	nodejs20
Generation	Gen 2
Trigger	HTTP
Authentication	Unauthenticated
Timeout	540 detik

Environment variables yang diperlukan aplikasi disimpan sebagai *GitHub Secrets* dan dimasukkan ke *Cloud Functions* saat *deployment*. Cara ini memastikan *kredensial* sensitif seperti koneksi *database*, *JWT secret*, dan *API keys* tidak terekspos di *source code*.

2.5 Metrik Pengukuran

Untuk mengevaluasi efektivitas implementasi CI/CD, penelitian ini mengukur beberapa *metrik* yang relevan. Tabel 7 menunjukkan daftar *metrik* beserta definisi dan cara mengukurnya.

Tabel 7. Metrik Pengukuran Efektivitas CI/CD

Metrik	Definisi	Cara Pengukuran
Waktu Deployment	Waktu dari push kode sampai aplikasi berhasil di-deploy	Durasi eksekusi workflow di Github Actions

Tingkat keberhasilan Build	Persentase keberhasilan proses build	Rumus (1)
Frekuensi Deployment	Jumlah deployment dalam peroide tertentu	Rumus (2)
Rata-rata Waktu Deployment	Waktu rata-rata yang dibutuhkan untuk deployment	Rumus (3)
Code Coverage	Persentase kode yang tercakup dalam testing	Output coverage daru Jest
Code Quality Score	Skor kualitas kode	Rating dari SonarCloud (A-E)

Berikut adalah rumus yang digunakan untuk menghitung *metrik* pengukuran:

Tingkat Keberhasilan *Build* (*Build Success Rate*) dihitung menggunakan rumus (1), dimana BSR adalah *Build Success Rate* dalam persen, B_sukses adalah jumlah *build* yang berhasil, dan B_total adalah total jumlah *build* yang dijalankan.

$$BSR = \frac{B_{success}}{B_{total}} \times 100\% \quad (1)$$

Frekuensi *Deployment* dihitung menggunakan rumus (2), dimana DF adalah *Deployment Frequency*, D_total adalah total jumlah *deployment* yang dilakukan, dan T_periode adalah periode waktu pengukuran dalam satuan hari, minggu, atau bulan.

$$DF = \frac{D_{total}}{T_{periode}} \quad (2)$$

Rata-rata Waktu *Deployment* dihitung menggunakan rumus (3), dimana $\bar{T}$_deploy adalah rata-rata waktu *deployment*, T_i adalah waktu *deployment* ke-i, dan n adalah jumlah total *deployment* yang diukur.

$$\bar{T}_{deploy} = \frac{\sum_{i=1}^{n} T_i}{n} \quad (3)$$

Pengukuran dilakukan dengan mencatat data dari setiap eksekusi *pipeline* yang tercatat di *dashboard GitHub Actions* dan *SonarCloud*. Data waktu *deployment* diambil dari durasi total *workflow Deployment*. Data tingkat keberhasilan *build* diambil dari status eksekusi *workflow Build*. Data *code coverage* dan *metrik* kualitas kode diambil dari *dashboard SonarCloud*.

Periode pengukuran dilakukan selama proses pengembangan aktif aplikasi yang mencakup beberapa siklus *push*, *build*, *test*, dan *deployment*. Data yang terkumpul kemudian dianalisis untuk mengevaluasi performa *pipeline* CI/CD yang sudah diimplementasikan.

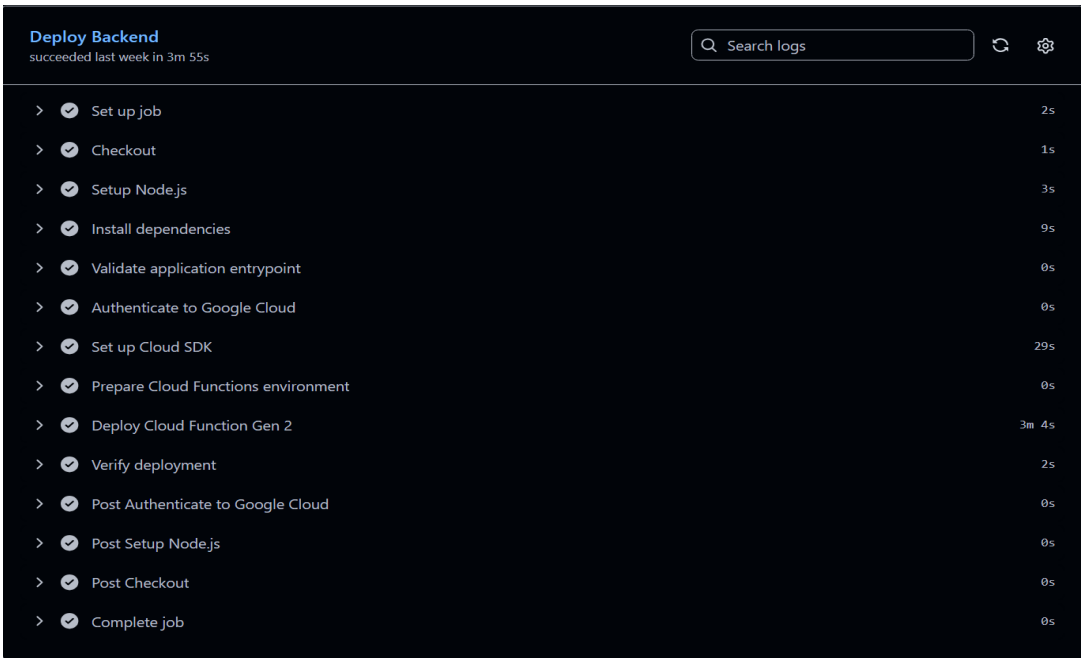
3. HASIL DAN PEMBAHASAN

3.1 Hasil Implementasi Pipeline CI/CD

Pipeline CI/CD berhasil diterapkan pada proyek Sistem Bimbingan Online menggunakan *GitHub Actions* dan *Google Cloud Functions*. Implementasi terdiri dari dua *workflow* utama yaitu *workflow Build* untuk *quality assurance* dan *workflow Deployment* untuk proses *deployment* otomatis ke *production*. Kedua *workflow* ini berjalan secara independen berdasarkan *trigger* yang sudah di konfigurasi file *YAML*. Pendekatan ini sejalan dengan konsep

automated deployment yang dikemukakan oleh **(DesLauriers et al. 2025)** dimana *deployment descriptor* digunakan sebagai *input* untuk mengelola eksekusi aplikasi secara otomatis.

Workflow Build jalan setiap kali ada *push* ke *branch dev* atau *main*, serta ketika ada *pull request*. *Workflow* ini menjalankan *automated testing* pakai *Jest* dan analisis kualitas kode pakai *SonarCloud*. Sementara itu, *workflow Deployment* jalan khusus ketika ada *push* ke *branch main* dan secara otomatis melakukan *deployment* aplikasi ke *Google Cloud Functions Gen 2* di *region asia-southeast2*. Gambar 2 menunjukkan contoh eksekusi *workflow Deployment* yang berhasil di *GitHub Actions*.

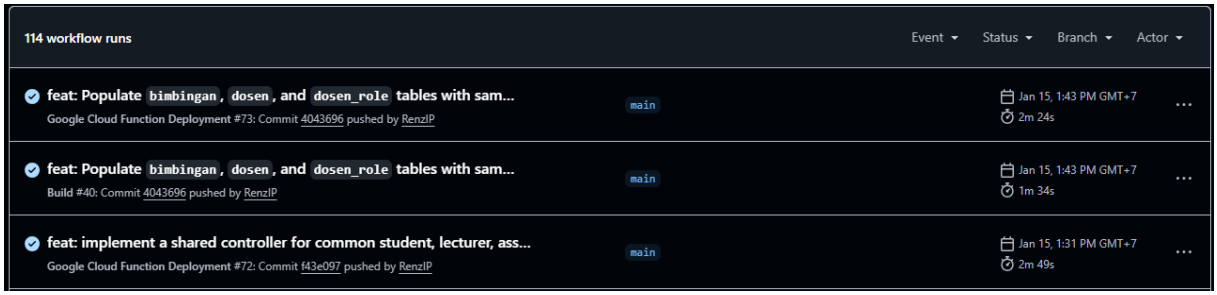


Gambar 2. Eksekusi Workflow Deployment di Github Action

Gambar 2 menunjukkan eksekusi *workflow* Deployment (*Deploy Backend*) di GitHub Actions yang berhasil dijalankan dengan status *succeeded* dalam waktu 3 menit 55 detik. *Workflow* ini terdiri dari 13 langkah berurutan, yaitu *Set up job*, *Checkout*, *Setup Node.js*, *Install dependencies*, *Validate application entrypoint*, *Authenticate to Google Cloud*, *Set up Cloud SDK*, *Prepare Cloud Functions environment*, *Deploy Cloud Function Gen 2*, *Verify deployment*, *Post Authenticate to Google Cloud*, *Post Setup Node.js*, *Post Checkout*, dan *Complete job*. Langkah terlama adalah *Deploy Cloud Function Gen 2* dengan durasi 3 menit 4 detik, menunjukkan bahwa proses *deployment* aktual ke Google Cloud Functions mendominasi total waktu eksekusi *workflow*.

3.2 Hasil Pengukuran Metrik

Pengukuran dilakukan selama periode 7 minggu pengembangan aktif aplikasi. Selama periode tersebut, tercatat total 105 kali eksekusi *workflow* yang terdiri dari *workflow* di *branch main* sebanyak 100 kali dan *branch dev* sebanyak 13 kali. Badampudi et al. (2025) dalam penelitiannya di *Ericsson* menemukan bahwa penggunaan CI/CD secara efektif dapat meningkatkan produktivitas dan kualitas *software* dalam jangka panjang. Berikut adalah data yang sudah dikumpulkan dalam bentuk gambar.



Gambar 3. Total Workflow Run

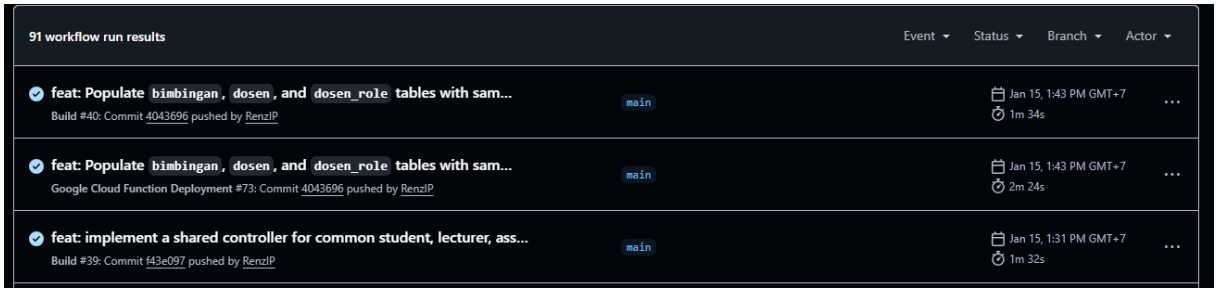
Gambar 3 menampilkan keseluruhan eksekusi *workflow* selama periode pengukuran dengan total 114 kali *workflow run*. Data ini mencakup seluruh eksekusi dari kedua *branch* (main dan dev) yang dipicu oleh berbagai *event* seperti *push* dan *pull request*. Informasi yang ditampilkan meliputi nama *commit*, *branch* asal, waktu eksekusi, dan durasi *workflow*, yang menjadi dasar perhitungan metrik BSR dan frekuensi *deployment*.

Gambar 4. Total Workflow Run Main Branch

Gambar 4 menampilkan eksekusi *workflow* khusus pada *branch* main dengan total 100 kali *workflow run*. Seluruh eksekusi pada *branch* ini dipicu oleh *event push* yang berasal dari aktivitas pengembangan aktif. Data dari *branch* main digunakan sebagai dasar perhitungan frekuensi *deployment* karena setiap eksekusi pada *branch* ini berkorelasi langsung dengan proses rilis aplikasi ke Google Cloud Functions.

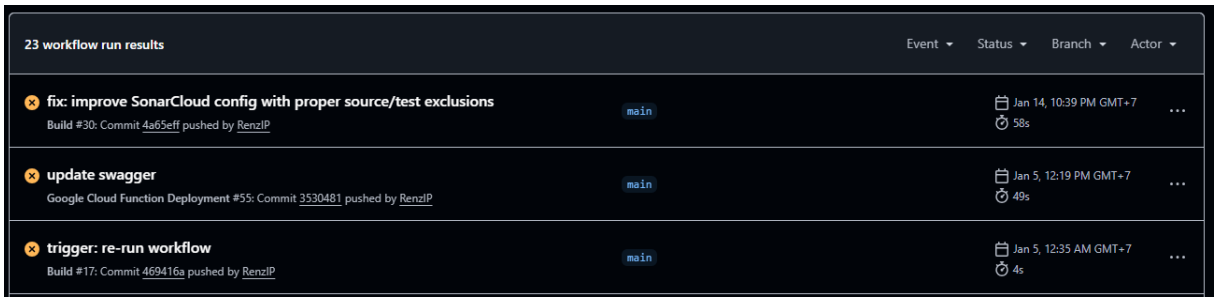
Gambar 5. Total Workflow Run Dev Branch

Gambar 5 menampilkan eksekusi *workflow* pada *branch* dev dengan total 13 kali *workflow run*. Eksekusi pada *branch* ini umumnya dipicu oleh aktivitas pengembangan fitur baru dan perbaikan kode sebelum di-*merge* ke *branch* main. Terlihat bahwa beberapa eksekusi pada *branch* dev mengalami kegagalan, yang mengindikasikan adanya proses deteksi *error* sebelum perubahan kode masuk ke lingkungan produksi.



Gambar 6. Success Total Workflow Run

Gambar 6 menampilkan seluruh eksekusi *workflow* yang berhasil (*success*) dengan total 91 kali dari keseluruhan 114 eksekusi. Eksekusi yang berhasil ditandai dengan ikon hijau dan mencakup *workflow* dari kedua *branch* main dan dev. Data ini digunakan sebagai nilai B_sukses dalam perhitungan *Build Success Rate* (BSR) menggunakan rumus (1).



Gambar 7. Failed Total Workflow Run

Gambar 7 menampilkan eksekusi *workflow* yang gagal (*failed*) dengan total 23 kali dari keseluruhan 114 eksekusi. Kegagalan ditandai dengan ikon kuning dan umumnya disebabkan oleh *error* pada proses *testing* atau konfigurasi *workflow* yang belum sesuai. Adanya kegagalan ini justru membuktikan bahwa pipeline berfungsi dengan benar dalam mendeteksi permasalahan sebelum kode masuk ke lingkungan produksi, sehingga kualitas aplikasi tetap terjaga.

3.2.1 Tingkat Keberhasilan Build

Berdasarkan data yang dikumpulkan, tingkat keberhasilan *build* dihitung pakai rumus (1). Dengan jumlah *workflow* berhasil sebanyak 91 kali dari total 114 kali eksekusi, didapatkan hasil BSR sebesar 79.82%. Kegagalan *workflow* sebanyak 23 kali disebabkan oleh berbagai faktor seperti kesalahan *syntax*, *test* yang gagal, atau masalah konfigurasi. Meskipun tingkat keberhasilan belum optimal, ini menunjukkan bahwa *pipeline* CI/CD berhasil mendeteksi *error* sebelum kode masuk ke *production*, sesuai dengan temuan **(Dakić 2024)** yang menekankan pentingnya *version control* dan CI/CD untuk memastikan kode sesuai dengan standar yang berlaku.

$$BSR = \frac{91}{114} \times 100\% = 79.82\% \quad (4)$$

3.2.2 Tingkat Keberhasilan Build

Berdasarkan data yang dikumpulkan, tingkat keberhasilan *build* dihitung menggunakan rumus (1). Dengan jumlah *workflow* berhasil sebanyak 91 kali dari total 114 kali eksekusi, didapatkan hasil BSR sebesar 79,82%.

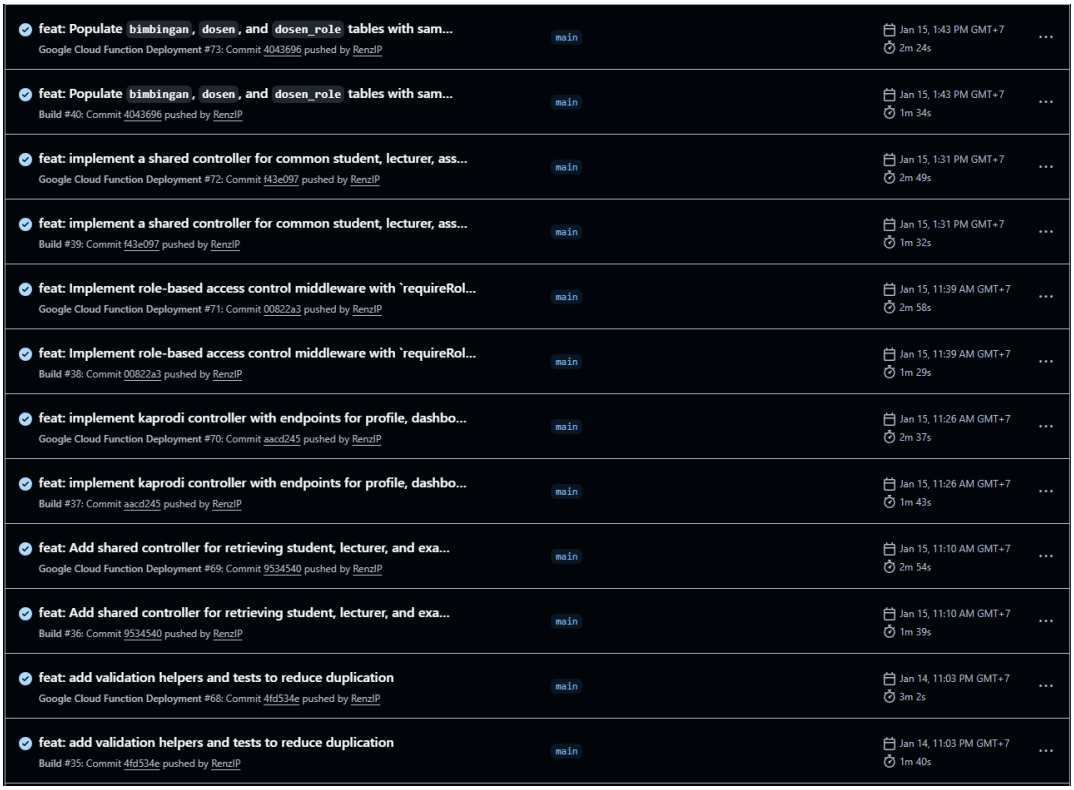
Nilai BSR 79,82% menunjukkan bahwa sekitar 1 dari 5 eksekusi mengalami kegagalan. Dari 23 kegagalan yang tercatat, sebagian besar terjadi pada *branch* dev yang disebabkan oleh

error pada proses *testing* Jest akibat perubahan kode yang belum memenuhi kriteria pengujian, serta *misconfiguration* pada parameter *deployment* ke Google Cloud Functions di *branch* main. Meskipun demikian, kondisi ini membuktikan bahwa pipeline berfungsi dengan benar dalam mendeteksi *error* sebelum kode masuk ke lingkungan produksi, sejalan dengan tujuan utama implementasi CI/CD untuk menggeser deteksi kesalahan ke tahap lebih awal dalam siklus pengembangan **(Fluri et al., 2024)**.

$$DF = \frac{D_{total}}{T_{periode}} = \frac{100}{7} = 14.3 \quad (5)$$

3.2.3 Rata-rata Waktu Eksekusi Workflow

Waktu *deployment* dicatat dari setiap eksekusi *workflow Deployment* di *GitHub Actions*. gambar 7 menyajikan ringkasan data waktu *deployment* untuk 6 kali eksekusi (hanya mengambil Google Cloud Function Deployment) yang dilakukan selama periode pengukuran. **(Abhishek et al. 2022)** menyatakan bahwa *container* bersifat ringan dalam hal *deployment* karena aplikasi dikemas dalam satu paket lengkap termasuk *library* dan *dependencies* yang diperlukan.



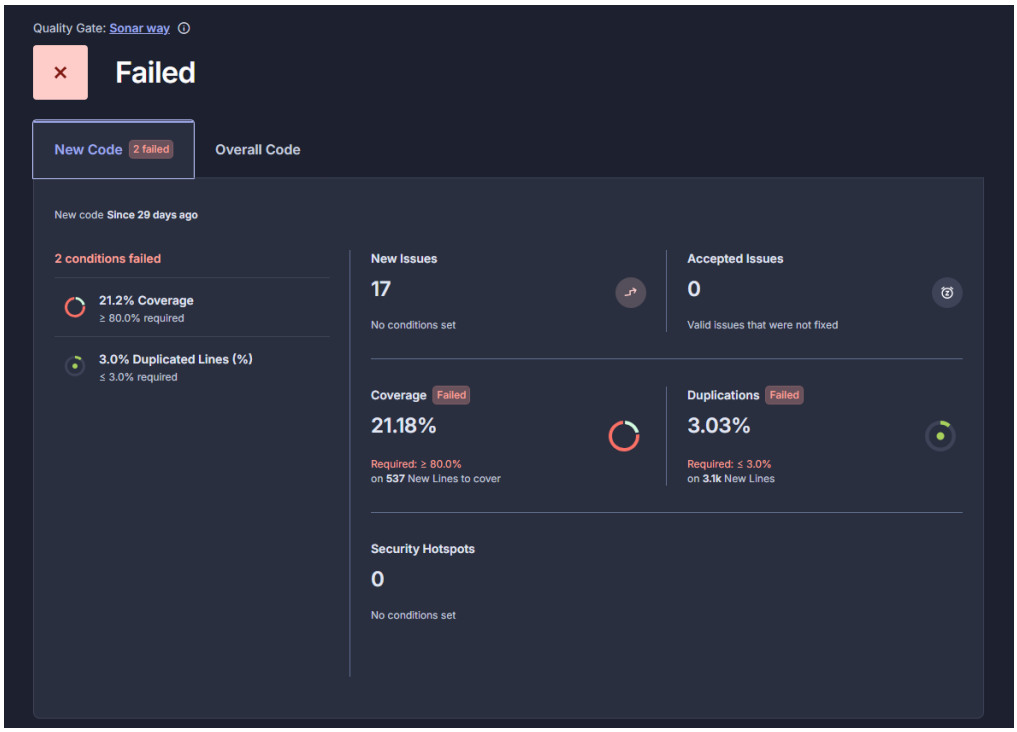
Gambar 8. Data Waktu Eksekusi Workflow

Rata-rata waktu eksekusi *workflow* dihitung pakai rumus (3). Dengan menjumlahkan seluruh waktu eksekusi (144+169+178+157+174+182=1004 detik) dan membaginya dengan jumlah sampel, didapatkan rata-rata waktu eksekusi sebesar 167.3 detik atau sekitar 2 menit 47 detik.

$$\overline{T}deploy = \frac{1004}{7} = 167.3 \quad (6)$$

3.2.4 Code Coverage dan Code Quality

Hasil analisis *SonarCloud* menunjukkan *metrik* kualitas kode dari proyek Sistem Bimbingan Online. Gambar 8 menyajikan hasil analisis kualitas kode yang didapat dari *dashboard SonarCloud* setelah integrasi dengan *pipeline* CI/CD.



Gambar 9. Hasil Analisis Kualitas Kode dari SonarCloud

Hasil menunjukkan bahwa dari sisi keamanan dan kualitas kode, proyek mendapatkan *rating* A untuk semua *metrik* utama yaitu *Bugs*, *Vulnerabilities*, dan *Code Smells* dengan nilai 0. Ini berarti tidak ditemukan potensi *bug*, kerentanan keamanan, maupun masalah *maintainability* pada kode. Namun, *code coverage* masih rendah yaitu 21.18% dan tingkat duplikasi kode sebesar 5.6% yang menyebabkan *Quality Gate* gagal. Kondisi ini menunjukkan bahwa masih diperlukan penambahan *unit test* untuk meningkatkan *code coverage*.

3.2 Pembahasan

Implementasi CI/CD pada Sistem Bimbingan Online memberikan beberapa manfaat yang cukup signifikan bagi proses pengembangan aplikasi. Pertama, proses *deployment* jadi lebih cepat dan konsisten dibanding sebelumnya. Dengan rata-rata waktu eksekusi *workflow* sekitar 2 menit 47 detik, tim *developer* bisa melakukan rilis kapan saja tanpa perlu melakukan konfigurasi manual yang memakan waktu. Temuan ini sejalan dengan penelitian **(Fluri et al. 2024)** yang menyatakan bahwa CI/CD memungkinkan *developer* merespons cepat terhadap perubahan kebutuhan pengguna.

Kedua, meskipun tingkat keberhasilan *build* sebesar 79.82% masih perlu ditingkatkan, hal ini justru membuktikan bahwa *pipeline* berfungsi dengan benar dalam mendeteksi *error* sebelum masuk ke *production*. Kegagalan *workflow* sebanyak 23 kali menunjukkan bahwa *automated testing* berhasil menangkap masalah pada tahap awal. Hal ini sesuai dengan temuan **(Kim et al. 2022)** yang menunjukkan bahwa implementasi *continuous integration* dengan *GitHub*

Actions mampu mengurangi tingkat *error* karena *developer* langsung dapat notifikasi ketika ada kesalahan.

Ketiga, frekuensi *deployment* yang tinggi yaitu sekitar 14 kali per minggu menandakan bahwa tim bisa melakukan iterasi dengan sangat cepat. Ini penting dalam pengembangan aplikasi *web* dimana *feedback* dari pengguna perlu segera direspons dengan perbaikan atau fitur baru. **(Wessel et al. 2023)** mencatat bahwa *GitHub Actions* sudah diadopsi oleh hampir 30% *repository* populer di *GitHub*, yang menandakan *tool* ini sudah terbukti efektif untuk automasi *workflow* dalam berbagai jenis proyek.

Keempat, hasil analisis *SonarCloud* menunjukkan kualitas kode yang baik dari sisi keamanan dengan tidak ditemukannya *bugs* atau *vulnerabilities* (*rating* A). Namun, *code coverage* sebesar 21.18% masih perlu ditingkatkan untuk memenuhi standar *Quality Gate*. **(Stötzner et al. 2025)** menekankan bahwa pemilihan teknologi *deployment* yang tepat dapat meningkatkan kualitas *deployment* secara keseluruhan. **(Manolov et al. 2025)** juga menekankan bahwa pemilihan platform CI/CD yang tepat bisa mempengaruhi produktivitas tim *development*, dan hasil penelitian ini membuktikan bahwa kombinasi *GitHub Actions* dan *Google Cloud Functions* merupakan pilihan yang efektif untuk *deployment* aplikasi *web* berbasis *Node.js*.

Tabel 8 menyajikan perbandingan kondisi sebelum dan sesudah implementasi CI/CD pada proyek Sistem Bimbingan Online untuk memberikan gambaran yang lebih jelas mengenai dampak implementasi.

Tabel 8. Perbandingan Sebelum dan Sesudah Implementasi CI/CD

Aspek	Sebelum CI/CD	Sesudah CI/CD
Waktu Deployment	15-30 Menit (manual)	2-3 menit (otomatis)
Frekuensi Rilis	5x per minggu	14x per minggu
Deteksi Error	Saat testing manual	Otomatis saat push
Konsistensi Deployment	Tergantung Developer	Selalu konsisten
Code Quality Check	Manual Review	Otomatis via SonarCloud

Berdasarkan perbandingan pada Tabel 8, dapat dilihat bahwa implementasi CI/CD berhasil meningkatkan efisiensi proses pengembangan dan rilis aplikasi *web* secara signifikan. Waktu *deployment* berkurang drastis dari 15-30 menit menjadi hanya 2-3 menit. Frekuensi rilis meningkat hampir 3 kali lipat dari 5x menjadi 14x per minggu. Deteksi *error* yang sebelumnya dilakukan manual sekarang berjalan otomatis setiap kali ada *push* ke *repository*. Konsistensi *deployment* juga meningkat karena proses sudah terotomatisasi dan tidak lagi tergantung pada *developer* yang menjalankannya. Hasil ini sejalan dengan temuan **(Badampudi et al. 2025)** yang menunjukkan bahwa penggunaan CI/CD secara efektif dapat meningkatkan kualitas, produktivitas, dan cara kerja tim dalam jangka panjang.

4. KESIMPULAN

Berdasarkan hasil penelitian implementasi CI/CD pada proses pengembangan dan rilis aplikasi *web* Sistem Bimbingan Online, dapat disimpulkan beberapa hal berikut.

Pertama, *pipeline* CI/CD berhasil diimplementasikan menggunakan *GitHub Actions* dan *Google Cloud Functions* dengan dua *workflow* utama yaitu *workflow Build* untuk *quality assurance* dan *workflow Deployment* untuk proses *deployment* otomatis. Selama periode 7 minggu

pengembangan, tercatat total 114 kali eksekusi *workflow* dengan tingkat keberhasilan 79.82%.

Kedua, implementasi CI/CD memberikan dampak positif terhadap efisiensi proses pengembangan. Waktu *deployment* berkurang drastis dari 15-30 menit secara manual menjadi rata-rata 2 menit 47 detik secara otomatis. Frekuensi rilis meningkat signifikan menjadi sekitar 14 kali per minggu, yang menandakan tim *developer* bisa melakukan iterasi dengan sangat cepat.

Ketiga, dari sisi kualitas kode, integrasi dengan SonarCloud berhasil membantu memonitor *metrik* kualitas. Proyek mendapatkan *rating* A untuk *Bugs*, *Vulnerabilities*, dan *Code Smells* dengan nilai 0. Namun, *code coverage* sebesar 21.18% masih perlu ditingkatkan untuk memenuhi standar *Quality Gate*, yang dapat dilakukan dengan menambahkan *unit test* pada modul-modul yang belum tercakup seperti fitur manajemen bimbingan dan autentikasi multi-*role*, serta menetapkan target minimum *coverage* per *sprint* menggunakan fitur *Quality Gate* pada SonarCloud.

Keempat, kombinasi *GitHub Actions* dan *Google Cloud Functions* terbukti efektif untuk *deployment* aplikasi *web* berbasis *Node.js* dengan arsitektur *REST API*. Penggunaan *GitHub Secrets* untuk menyimpan *environment variables* memastikan keamanan *kredensial* sensitif.

Untuk pengembangan lebih lanjut, disarankan untuk meningkatkan *code coverage* dengan menambahkan lebih banyak *unit test* dan *integration test*. Selain itu, perlu dilakukan optimasi pada *workflow* untuk mengurangi tingkat kegagalan *build* dan mempersingkat waktu eksekusi.

DAFTAR RUJUKAN

Abhishek, M. K., Rao, D. R., & Subrahmanyam, K. (2022). *Framework to deploy containers using Kubernetes and CI/CD pipeline*. *International Journal of Advanced Computer Science and Applications*, *13*(4), 522-528.

Badampudi, D., Usman, M., & Chen, X. (2025). *Large scale reuse of microservices using CI/CD and InnerSource practices - a case study*. *Empirical Software Engineering*, *30*, 41. https://doi.org/10.1007/s10664-024-10595-w

Chandra, R., Ranjan, K., & Lulla, K. (2025). *Optimizing LLM performance through CI/CD pipelines in cloud-based environments*. *International Journal of Advanced Research in Science and Engineering*, *38*(2s). *ISSN*: 1311-1728.

Dakić, P. (2024). *Software compliance in various industries using CI/CD, dynamic microservices, and containers*. *Open Computer Science*, *14*, 20240013. https://doi.org/10.1515/comp-2024-0013

DesLauriers, J., Kovacs, J., Kiss, T., Stork, A., Serna, S. P., & Ullah, A. (2025). *Automated generation of deployment descriptors for managing microservices-based applications in the cloud to edge continuum*. *Future Generation Computer Systems*, *166*, 107628. https://doi.org/10.1016/j.future.2024.107628

Fluri, J., Fornari, F., & Pustulka, E. (2024). *On the importance of CI/CD practices for database applications*. *Journal of Software: Evolution and Process*, *36*, e2720. https://doi.org/10.1002/smr.2720

Jiang, B., Tang, Z., Xiao, X., Yao, J., Cao, R., & Li, K. (2022). *Efficient and automated deployment architecture for OpenStack in TianHe SuperComputing environment*. *IEEE Transactions on Parallel and Distributed Systems*, *33*(12), 4255-4267. https://doi.org/10.1109/TPDS.2021.3127128

Kim, A. Y., Herrmann, V., Barreto, R., Calkins, B., Gonzalez-Akre, E., Johnson, D. J., Jordan, J. A., Magee, L., McGregor, I. R., Montero, N., Novak, K., Rogers, T., Shue, J., & Anderson-Teixeira, K. J. (2022). *Implementing GitHub Actions continuous integration to reduce error rates in ecological data collection*. *Methods in Ecology and Evolution*, *13*, 2572-2585. https://doi.org/10.1111/2041-210X.13982

Kongarana, S. R., Rao, A. A., & Raju, P. R. (2025). *Test case prioritization with smart adaptation and transparent decisions for data-constrained CI/CD pipelines*. *International Journal of Intelligent Engineering and Systems*, *18*(10), 868-880. https://doi.org/10.22266/ijies2025.1130.55

Manolov, V., Gotseva, D., & Hinov, N. (2025). *Practical comparison between the CI/CD platforms Azure DevOps and GitHub*. *Future Internet*, *17*, 153. https://doi.org/10.3390/fi17040153

Santos, Á., Bernardino, J., & Correia, N. (2023). *Automated application deployment on multi-access edge computing: A survey*. *IEEE Access*, *11*, 89606-89625. https://doi.org/10.1109/ACCESS.2023.3307023

Stötzner, M., Krieger, N., Speth, S., Weller, M., Becker, S., Weder, B., Soldani, J., & Morlock, V. (2025). *A method for the quality-aware automated selection of deployment technologies*. *Software: Practice and Experience*. https://doi.org/10.1002/spe.3398

Teumert, S., & Tegeler, T. (2025). *Towards X-as-models in the context of CI/CD*. *Electronic Communications of the EASST*, *84*. https://doi.org/10.14279/eceasst.v84.2680

Wessel, M., Vargovich, J., Gerosa, M. A., & Treude, C. (2023). *GitHub Actions: The impact on the pull request process*. *Empirical Software Engineering*, *28*, 131. https://doi.org/10.1007/s10664-023-10369-w

Wohlin, C., & Rainer, A. (2022). Is it a case study? — A critical analysis and guidance. *Journal of Systems and Software*, *192*, 111395. https://doi.org/10.1016/j.jss.2022.111395