# Perancangan dan Pengembangan Infrastruktur Continuous Integration / Continuos Deployment Menggunakan Jenkins dan Docker

**Wahyu Kesuma Bakti[1], Hario Jati Setyadi[2,*], Muhammad Labib Jundillah[3]**

[1,2,3] Fakultas Teknik, Program Studi Sistem Informasi, Universitas Mulawarman, Samarinda, Indonesia
Email: [1,]kahyuwesuma@email.com, [2*]hariojati.setyadi@ft.unmul.ac.id, [3]muhammadjundillah@ft.unmul.ac.id
[*)] Email Penulis Utama

**Abstrak**– *Software Development Life Cycle* merupakan proses yang mencakup berbagai tahapan dalam pengembangan perangkat lunak, termasuk perencanaan, pengembangan, pengujian, dan perilisan. Dalam upaya meningkatkan efisiensi dan otomatisasi pada tahap pengujian serta pengiriman perangkat lunak, penelitian ini mengimplementasikan infrastruktur *Continuous Integration* / *Continuous Deployment* (CI/CD) menggunakan Jenkins dan Docker pada proyek berbasis Laravel. Metode yang digunakan adalah melibatkan perancangan *pipeline* otomatis, eksekusi *build*, pengujian, dan *deployment*, serta evaluasi performa infrastruktur. Hasil penelitian menunjukkan bahwa penerapan CI/CD mampu meningkatkan kecepatan pengembangan dan durasi *pipeline* kurang dari 10 menit. Selain itu, penggunaan Jenkins sebagai server CI/CD dan Docker sebagai lingkungan isolasi berhasil mengurangi inkonsistensi antara pengembangan, *staging*, dan produksi, memungkinkan *deployment* yang lebih andal dan minim kesalahan manusia. Namun, keterbatasan dalam pengujian unit dan *staging* masih menjadi tantangan dalam implementasi ini, sehingga diperlukan pengembangan lebih lanjut untuk memastikan validasi yang lebih menyeluruh sebelum aplikasi diterapkan di lingkungan produksi.

**Kata Kunci: Perancangan, Pengembangan, Infrastruktur, CI/CD, *Praktikum***

**Abstract**– *Software Development Life Cycle is a process that includes various stages in software development, including planning, development, testing, and release. To increase efficiency and automation at the testing and software delivery stages, this research implements Continuous Integration / Continuous Deployment (CI/CD) infrastructure using Jenkins and Docker on a Laravel-based project. The method used involves automatic pipeline design, building execution, testing and deployment, as well as infrastructure performance evaluation. The research results show that implementing CI/CD can increase development speed and pipeline duration is less than 10 minutes. Additionally, using Jenkins as a CI/CD server and Docker as an isolated environment reduces inconsistencies between development, staging, and production, enabling more reliable deployments and minimal human error. However, limitations in unit testing and staging remain a challenge in this implementation, so further development is required to ensure more thorough validation before the application is deployed in a production environment.*

**Keywords**: ***Design, Development, Infrastructure,* CI/CD, *Practicum***

## 1. PENDAHULUAN

Siklus pengembangan perangkat lunak (SDLC) merupakan kerangka kerja yang mencakup perencanaan, pengujian, hingga proses perilisan perangkat lunak [1]. Salah satu tahap penting dalam siklus ini adalah *deployment*, di mana hasil pengembangan akan dirilis ke lingkungan produksi [2]. Proses ini sering kali memerlukan waktu, tenaga, serta pengujian berulang yang umumnya dilakukan secara manual. Aktivitas tersebut menjadi tidak efisien dan rentan terhadap kesalahan, terutama dalam pengembangan aplikasi berskala menengah hingga besar [3].

Untuk mengatasi tantangan ini, diterapkan pendekatan *Continuous Integration* (CI) dan *Continuous Deployment* (CD). CI merupakan proses otomatisasi penggabungan perubahan kode ke dalam repositori utama, sedangkan CD adalah proses otomatisasi *build, test,* dan *deployment* aplikasi ke server produksi [4]. Dengan penerapan CI/CD, proses integrasi dan pengujian menjadi lebih cepat, efisien, dan mengurangi potensi *human error* [5]. Hal ini diperkuat oleh temuan DevOps *Research and Assessment* (DORA) bahwa organisasi yang menerapkan CI/CD menunjukkan kualitas kerja lebih baik, risiko lebih rendah, serta penghematan waktu dan biaya [6].

*Dashboard Application* Praktikum Program Studi Sistem Informasi merupakan aplikasi berbasis web yang dikembangkan untuk mendukung kegiatan praktikum di lingkungan akademik. Sistem ini dirancang untuk memfasilitasi pengelolaan data praktikum, distribusi informasi, dan evaluasi kegiatan. Seiring dengan meningkatnya kebutuhan pengembangan fitur baru secara cepat dan terstruktur, dibutuhkan mekanisme otomatisasi dalam pengujian dan perilisan aplikasi. Saat ini, proses tersebut masih dilakukan secara manual, sehingga rawan kesalahan dan tidak efisien dari segi waktu.

Dalam penelitian ini, dilakukan perancangan dan implementasi infrastruktur CI/CD menggunakan Jenkins dan Docker, dengan integrasi repositori melalui GitHub. Jenkins dipilih karena fleksibilitasnya sebagai alat otomasi berbasis plugin, sementara Docker memungkinkan aplikasi berjalan dalam lingkungan terisolasi yang konsisten antara pengembangan dan produksi [7][8]. Tujuan dari penelitian ini adalah merancang dan mengembangkan *pipeline* CI/CD untuk mengotomasi proses *build*, *test*, dan *deployment* aplikasi dashboard praktikum agar lebih efisien dan andal.

Penelitian ini memberikan kontribusi praktis dan ilmiah dalam bidang Sistem Informasi, khususnya dalam penerapan otomasi pengujian dan *deployment* berbasis web. Dibandingkan dengan penelitian sebelumnya seperti Adesaputra (2022) dan Dhany (2021) yang menggunakan *tools* berbeda dan fokus pada konteks yang lain, penelitian ini menitikberatkan pada dokumentasi perancangan *pipeline* serta implementasi CI/CD yang langsung diterapkan dalam konteks *dashboard* praktikum akademik. Dengan demikian, penelitian ini diharapkan dapat menjadi referensi penerapan CI/CD dalam pengembangan aplikasi pendidikan berbasis Laravel dan teknologi *open-source* lainnya [9].

# 2. METODE PENELITIAN

### 2.1 Identifikasi Masalah

Penelitian ini mengidentifikasi beberapa permasalahan teknis dalam pengembangan *Dashboard Application Praktikum Program Studi Sistem Informasi*, seperti proses *deployment* manual yang memakan waktu dan rentan kesalahan, serta pengujian yang belum terotomatisasi sehingga meningkatkan risiko *bug*. Selain itu, kurangnya kolaborasi tim dan minimnya *real-time feedback* memperlambat integrasi kode. Oleh karena itu, penelitian ini bertujuan untuk merancang dan mengimplementasikan *pipeline* CI/CD menggunakan Jenkins dan Docker guna meningkatkan efisiensi pengembangan, mempercepat perilisan aplikasi, serta mendukung kolaborasi tim secara optimal.

### 2.2 Pengumpulan Data

Pengumpulan data merupakan tahap krusial dalam metode penelitian untuk memperoleh informasi yang akurat dan relevan guna menjawab rumusan masalah dan mencapai tujuan penelitian [10]. Dalam penelitian ini, data dikumpulkan untuk menganalisis arsitektur sistem yang ada, proses pengujian, serta mengevaluasi efektivitas penerapan *pipeline* CI/CD pada *Dashboard Application Praktikum Program Studi Sistem Informasi*. Data yang dikumpulkan mencakup waktu eksekusi pada setiap tahapan seperti *build*, *testing*, dan *deployment*, serta jumlah kesalahan atau *bug* yang muncul selama proses otomatisasi. Selain itu, efisiensi sistem juga dibandingkan dengan metode manual sebelumnya untuk menilai peningkatan kinerja secara objektif [11]. Informasi ini menjadi dasar penting dalam mengukur keberhasilan penerapan CI/CD dalam konteks pengembangan aplikasi.

### 2.3 Studi Literatur

Penelitian ini menggunakan metode penelitian kualitatif. Penelitian kualitatif adalah metode penelitian yang bertujuan untuk memahami kenyataan melalui proses penalaran dari hal-hal khusus menuju kesimpulan umum [12]. Fokus utama dari penelitian ini adalah memahami fenomena atau gejala sosial dengan memberikan gambaran menyeluruh tentang fenomena tersebut, daripada memecahnya menjadi variabel-variabel yang terpisah**.**

### 2.4 Analisis Arsitektur Sistem

Pada tahap awal penelitian, dilakukan analisis terhadap infrastruktur dan alur kerja manual yang digunakan dalam proses pengembangan *Dashboard Application Praktikum Program Studi Sistem Informasi*. Tujuan analisis ini adalah untuk memahami struktur sistem yang ada dan mengidentifikasi hambatan-hambatan yang menyebabkan proses *development* dan *deployment* berjalan secara tidak efisien [13].

Analisis mencakup kondisi server, alur pengujian, alat pengembangan, serta proses integrasi dan perilisan kode. Hasil identifikasi menunjukkan bahwa seluruh proses masih dilakukan secara manual, yang menyebabkan tingginya potensi kesalahan, miskomunikasi tim, serta tidak konsistennya aplikasi saat dirilis. Ringkasan perencanaan analisis sistem ditampilkan pada Tabel 1 berikut:

**Tabel 1.** Usulan Analisa Arsitektur Sistem

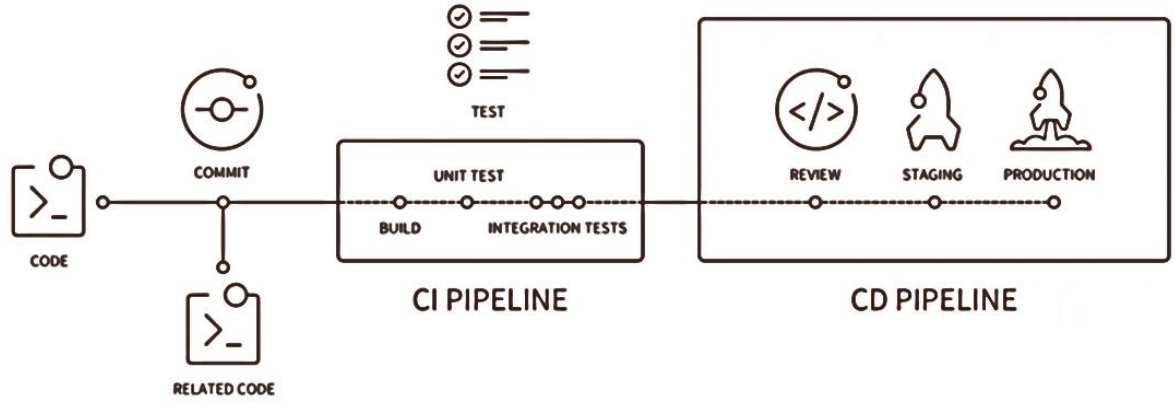
| Komponen Sistem | Kondisi | Kelemahan Yang Ditemukan |
|---|---|---|
| Server Perilisan | Belum Diketahui | Belum Diketahui |
| Pengujian Aplikasi | Belum Diketahui | Belum Diketahui |
| Alat Pengembangan | Belum Diketahui | Belum Diketahui |
| Proses Integrasi Kode | Belum Diketahui | Belum Diketahui |
| Proses Perilisan Aplikasi | Belum Diketahui | Belum Diketahui |

### 2.5 Perancangan *Pipeline* CI/CD

Setelah dilakukan analisis terhadap sistem yang berjalan secara manual, tahap selanjutnya dalam penelitian ini adalah merancang arsitektur pipeline CI/CD yang mampu mengotomatisasi proses *build*, *testing*, dan *deployment*. Tujuan dari perancangan ini adalah untuk mempercepat siklus pengembangan perangkat lunak, mengurangi kesalahan manusia, serta menjaga konsistensi antara lingkungan pengembangan dan produksi.

Tahapan awal dimulai dengan mendesain alur pipeline yang terdiri dari proses *build* otomatis setelah kode dikirim (*commit*) ke repositori, menjalankan pengujian secara otomatis, dan melakukan *deployment* ke lingkungan produksi tanpa intervensi manual. Alat yang digunakan dalam pipeline ini meliputi GitHub sebagai sistem kontrol versi, Jenkins sebagai server otomasi CI/CD, dan Docker untuk membangun lingkungan kontainer yang konsisten.

Alur kerja *pipeline* yang dirancang dapat dilihat secara visual pada Gambar 2, yang menggambarkan urutan tahapan mulai dari *commit* kode hingga *deployment* ke server produksi.




**Gambar 1.** Rancangan *Pipeline* CI/CD

Untuk mendukung implementasi secara sistematis, setiap tahapan *pipeline* dirancang dengan detail sebagai berikut:

**Tabel 2.** Perancangan Tahapan Pipeline

| Tahapan | Peralatan | Deskripsi Proses |
|---|---|---|
| Commit Change | GitHub | Commit kode pada GitHub akan memicu pipeline CI/CD secara otomatis. |

| Tahapan | Peralatan | Deskripsi Proses |
|---|---|---|
| Trigger Build | Jenkins | Jenkins mendeteksi commit baru dan memulai proses *build*. |
| Build | Jenkins, Docker | Proses *build* dijalankan dalam kontainer Docker untuk menjamin konsistensi. |
| Run Test | Jenkins, Docker, PHPUnit | Unit dan integrasi test dijalankan otomatis di dalam kontainer. |
| Outcome | Jenkins | Kode hanya dilanjutkan jika pengujian berhasil, menandakan stabilitas. |
| Delivery to Staging | Jenkins, Docker, Staging Server | Hasil *build* dikirim ke server staging untuk pengujian tambahan. |
| Deployment | Jenkins, Docker, Production Server | Jenkins melakukan deployment otomatis ke server produksi dengan metode *recreated*. |

### 2.6 Implementasi *Pipeline* CI/CD

*Pipeline* CI/CD yang telah dirancang sebelumnya diterapkan secara bertahap ke dalam sistem untuk memastikan bahwa setiap proses pengembangan, pengujian, dan deployment berjalan secara otomatis dan konsisten. Tahapan implementasi dimulai dengan konfigurasi Jenkins sebagai server utama CI/CD, yang berperan dalam menjalankan proses *build*, *testing*, dan *deployment* secara terotomatisasi. Jenkins diintegrasikan dengan GitHub sebagai sistem *version control*, sehingga setiap *commit* yang dilakukan oleh pengembang akan langsung memicu eksekusi *pipeline* CI/CD tanpa intervensi manual.

Selain itu, teknologi Docker diimplementasikan untuk proses *containerization*, yaitu membungkus aplikasi beserta seluruh dependensi dan konfigurasi ke dalam satu container yang dapat dijalankan secara konsisten di berbagai lingkungan, baik *development*, *staging*, maupun *production*. Hal ini penting untuk menghindari permasalahan yang sering terjadi akibat perbedaan lingkungan.

Implementasi ini diterapkan pada *Dashboard Application Praktikum Program Studi Sistem Informasi*, yaitu aplikasi berbasis web yang digunakan dalam kegiatan praktikum mahasiswa. Dengan otomatisasi pipeline, aplikasi dapat selalu berada dalam kondisi terbaru dan teruji, sehingga waktu proses *build* dan *deployment* menjadi lebih singkat serta mampu meningkatkan produktivitas dan efisiensi tim pengembang.

Tahapan implementasi *pipeline* CI/CD yang dilakukan dirangkum dalam Tabel 3 berikut:

**Tabel 3.** Implementasi *Pipeline* CI/CD

| Komponen | Alat |
|---|---|
| Jenkins Setup | Jenkins, GitHub |
| Docker Setup | Docker, Docker Compose |
| Automation Build | Jenkins, GitHub |
| Automation Testing | Jenkins, Docker, PHPUnit |
| Automation Deployment | Jenkins, Docker, Production Environment |

### 2.7 Validasi dan Pengujian

Setelah implementasi pipeline CI/CD selesai dilakukan, tahap berikutnya adalah proses pengujian menyeluruh terhadap pipeline tersebut. Tujuan dari pengujian ini adalah untuk memastikan bahwa setiap tahapan dalam *pipeline*, mulai dari *build*, *testing*, hingga *deployment*—berjalan sesuai perancangan, serta untuk mendeteksi potensi bug atau kesalahan selama proses berlangsung.

Untuk menilai efektivitas dan keandalan pipeline, digunakan beberapa metrik pengukuran, antara lain durasi waktu pengujian, persentase keberhasilan eksekusi pipeline, serta jumlah error atau bug yang teridentifikasi selama proses berjalan. Evaluasi berbasis metrik ini memberikan gambaran objektif terhadap performa pipeline dalam mendukung proses pengembangan perangkat lunak yang berkelanjutan.

### 2.8 Evaluasi

Tahap evaluasi dilakukan untuk mengukur efektivitas dari implementasi *pipeline* CI/CD yang telah dirancang dan diterapkan sebelumnya. Evaluasi ini mencakup perbandingan antara proses pengembangan yang dilakukan secara manual dengan proses otomatis yang telah terintegrasi dalam *pipeline*. Tujuannya adalah untuk menilai peningkatan efisiensi kerja, penghematan waktu dalam setiap tahap pengembangan, serta pengurangan risiko kesalahan (error) yang biasanya terjadi pada proses *deployment* manual.

Evaluasi dilakukan melalui monitoring langsung terhadap aktivitas *pipeline* saat dijalankan, termasuk pada tahap *build*, *testing*, dan keseluruhan durasi *pipeline*. Hasil yang diperoleh menunjukkan bahwa proses pengembangan berjalan lebih cepat dan terstruktur setelah CI/CD diimplementasikan. Rincian pengukuran metrik performa setelah implementasi *pipeline* ditampilkan pada Tabel 4 berikut:

**Tabel 4.** Detail Metrik Pengukuran

| Aspek | Durasi | Status |
|---|---|---|
| Build Estimation | Belum Diketahui | Belum Diketahui |
| Testing Estimation | Belum Diketahui | Belum Diketahui |
| Pipeline Estimation | Belum Diketahui | Belum Diketahui |

## 3. HASIL DAN PEMBAHASAN

### 3.1 Hasil Pengumpulan Data

Tahapan pengumpulan data dilakukan untuk memperoleh informasi teknis yang menjadi dasar dalam perancangan dan implementasi infrastruktur CI/CD. Data yang dikumpulkan mencakup teknologi inti yang digunakan pada aplikasi, serta komponen-komponen pendukung yang relevan dalam proses integrasi dan *deployment*.

#### 3.1.1 Spesifikasi Teknologi Aplikasi

Berdasarkan hasil identifikasi, pengembangan *Dashboard Application* Praktikum Program Studi Sistem Informasi masih menggunakan arsitektur monolitik berbasis Laravel 11. Sistem manajemen kode telah menggunakan Git dan GitHub sebagai *version control*, serta MySQL sebagai sistem manajemen basis data. Namun, aplikasi masih dijalankan secara lokal (localhost) pada komputer pengembang, dan belum dihosting pada lingkungan server produksi. Selain itu, belum tersedia alat pengujian otomatis, sehingga proses pengujian masih dilakukan secara manual. Komponen dam tech stack dapat di lihar pada Tabel 5.1 dibawah :

**Tabel 5.** Spesifikasi *Tech Stack*

| **Komponen** | ***Tech Stack*** |
|---|---|
| Framework | Laravel 11 |
| Version Control | Git dan GitHub |
| Database | MySQL |
| Server Hosting | Belum ada |
| Alat Pengujian | Belum ada |

Kondisi ini menunjukkan bahwa proses *deployment* dan pengujian belum optimal. Oleh karena itu, diperlukan penerapan CI/CD untuk mengotomasi proses pengujian serta integrasi layanan *hosting* yang lebih stabil agar mendukung efisiensi dan skalabilitas sistem.

### 3.1.2 Dependensi Aplikasi

Selain spesifikasi utama, penelitian juga mencatat dependensi aplikasi yang digunakan untuk membangun dan menjalankan sistem. Dependensi dan versi aplilasi dapat dilihat pada Tabel 6 beirkut :

**Tabel 6.** Dependensi Aplikasi

| **Dependensi** | **Versi** |
|---|---|
| Laravel Framework | 11 |
| Node JS | 18 |

Informasi ini menjadi landasan dalam menyusun konfigurasi pipeline CI/CD yang kompatibel dengan struktur dan kebutuhan sistem yang sedang dikembangkan.

## 3.2 Analisis Arsitektur Sistem

Hasil analisis arsitektur sistem menunjukkan sejumlah kelemahan yang menjadi hambatan dalam proses *development* dan *deployment* aplikasi. Permasalahan utama yang diidentifikasi mencakup keterbatasan aksesibilitas aplikasi, tidak adanya proses pengujian otomatis, serta kurangnya mekanisme validasi kode yang terintegrasi dalam alur pengembangan. Informasi detail terkait kondisi sebelum penerapan CI/CD dapat dilihat pada Tabel 7.

**Tabel 7.** Arsitektur Sistem

| **Komponen** | **Kondisi Sebelum Penerapan** | **Kelemahan yang Ditemukan** |
|---|---|---|
| Server Perilisan | Hosting Lokal | Aplikasi hanya berjalan di lingkungan pengembangan sehingga belum dapat diakses oleh pengguna umum |
| Pengujian Aplikasi | Manual | Rentan terjadi bug karena pengujian tidak menyeluruh dan tidak terdokumentasi |
| Alat Pengembangan | Visual Studio Code dan WSL | Tidak Ada |
| Proses Integrasi Kode | Git dan GitHub | Tidak ada validasi otomatis saat *commit* sehingga berisiko konflik dan bug saat integrasi. |
| Proses Perilisan Aplikasi | Belum Tersedia | Tidak Ada |

Kondisi tersebut memperlihatkan bahwa sistem masih bergantung pada proses manual yang berisiko terhadap kesalahan teknis dan tidak mendukung kolaborasi pengembangan secara efisien. Oleh karena itu, dibutuhkan infrastruktur CI/CD yang mampu mengotomasi alur kerja utama seperti integrasi, pengujian, dan deployment untuk meningkatkan keandalan dan kecepatan siklus pengembangan perangkat lunak.

## 3.3 Perancangan Infrastruktur *Pipeline* CI/CD

Berdasarkan analisis arsitektur sistem yang telah dilakukan, dirancang sebuah *pipeline Continuous Integration* / *Continuous Deployment* (CI/CD) guna mengotomatisasi proses *build*, *testing*, dan *deployment*. *Pipeline* ini disusun untuk menjamin setiap perubahan kode yang dikirim ke repositori akan melalui proses validasi dan pengujian otomatis sebelum diimplementasikan ke lingkungan produksi.

### 3.3.1 Desain Arsitektur *Pipeline* CI/CD

*Pipeline* yang dirancang terdiri dari dua tahapan utama, yaitu *Continuous Integration* (CI) dan *Continuous Deployment* (CD). CI berfokus pada proses otomatisasi *build* dan pengujian unit setiap kali ada perubahan kode, sementara CD bertujuan untuk merilis kode ke lingkungan staging dan produksi secara otomatis setelah seluruh pengujian berhasil dilakukan.

Tahapan-tahapan dalam *pipeline* ini dijalankan oleh Jenkins yang terintegrasi dengan GitHub dan Docker, serta diuji menggunakan PHPUnit. Rangkaian proses ini dijelaskan dalam Tabel 8 berikut:

**Tabel 8.** Tahapan *Pipeline* CI/CD

| Tahapan | Penjelasan |
|---|---|
| Commit and Trigger Build | Jenkins mendeteksi commit baru di GitHub dan memicu pipeline CI/CD. |
| Build | Jenkins membangun image aplikasi Laravel 11 dengan perintah ‘docker build’. |
| Run Test | Jenkins menjalankan perintah ‘php artisan test’ dalam container untuk pengujian otomatis. |
| Outcome | Jika pengujian berhasil, Jenkins mencetak log dan melanjutkan ke proses deployment. |
| Deployment | Jika sukses, Jenkins memperbarui kode di server produksi dan menjalankan deployment otomatis. |
| Post Actions | Jenkins mencetak log akhir untuk *debugging* dan audit, baik saat *pipeline* berhasil maupun gagal. |

### 3.3.2 Infrastruktur dan Spesifikasi Server

Pipeline diimplementasikan menggunakan Virtual Private Server (VPS) dari IDCloudHost. Pemilihan platform ini didasarkan pada dukungan terhadap Docker dan Jenkins, performa stabil, serta lokasi server yang strategis. Spesifikasi infrastruktur dapat dilihat pada Tabel 9. berikut:

**Tabel 9.** Spesifikasi *Virtual Private Server*

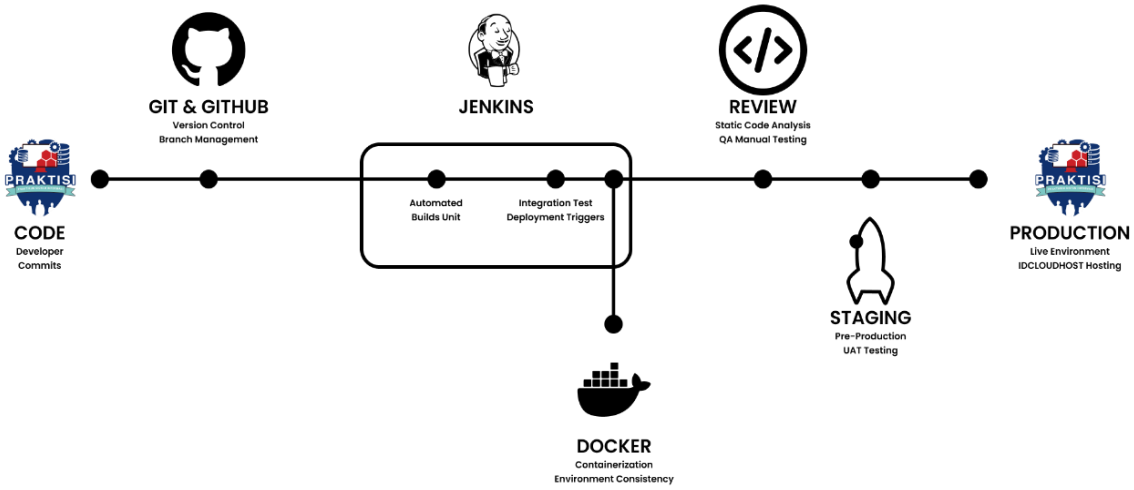
| Komponen | Spesifikasi |
|---|---|
| Cloud Provider | IDCloudHost |
| CPU | 2 Core / 2vCPU |
| RAM | 2 GB |
| Storage | 20 GB |
| OS | Debian 11 |
| Docker Version | 20.10 |
| Jenkins Version | Jenkins LTS |
| Biaya | Rp 61.399,- |

Penggunaan VPS berbasis *cloud* memberikan keunggulan dari sisi skalabilitas, keamanan, serta kemudahan dalam pemantauan sistem secara *real-time* dibandingkan dengan lingkungan pengembangan lokal.

### 3.3.3 Rancangan Workflow *Pipeline* CI/CD

Arsitektur CI/CD yang dirancang bertujuan untuk mengintegrasikan seluruh proses pengembangan perangkat lunak ke dalam satu alur kerja otomatis. Setiap perubahan kode yang dikirimkan ke repositori akan secara otomatis memicu Jenkins untuk memulai proses *build* dan *testing* menggunakan Docker dan PHPUnit. Setelah berhasil melalui tahap staging, aplikasi akan secara otomatis dirilis ke server produksi.

Implementasi ini bertujuan untuk meningkatkan efisiensi, mengurangi potensi kesalahan akibat proses manual, serta mempercepat waktu rilis fitur atau perbaikan (*bug fix*). Pada Gambar 2 menunjukkan visualisasi rancangan workflow pipeline yang dilakuakn pada penelitian ini/




**Gambar 2.** Hasil Akhir Rancangan *Pipeline* CI/CD

Desain *pipeline* ini memastikan bahwa setiap perubahan kode telah melalui validasi dan pengujian yang ketat sebelum sampai ke pengguna akhir. Dengan workflow yang terstruktur dan otomatis, tim pengembang dapat bekerja lebih efektif dan sistem dapat tetap stabil seiring dengan berjalannya pengembangan aplikasi.

### 3.4 Jenkins *Setup*

Jenkins dikonfigurasi sebagai server utama untuk mengelola dan menjalankan pipeline Continuous Integration / Continuous Deployment (CI/CD). Tujuan dari setup ini adalah agar setiap perubahan kode yang dikirimkan ke repositori GitHub dapat secara otomatis memicu *pipeline*, tanpa memerlukan intervensi manual, serta memastikan proses *build*, *testing*, dan *deployment* berjalan secara otomatis dan konsisten.

### 3.4.1 Instalasi Jenkins

Instalasi Jenkins dilakukan pada server virtual berbasis Debian 11, dimulai dengan pembaruan sistem dan instalasi dependensi, termasuk Java Development Kit (JDK) sebagai komponen utama untuk menjalankan Jenkins. Setelah itu, dilakukan konfigurasi repository resmi Jenkins serta verifikasi layanan menggunakan perintah `systemctl status jenkins` untuk memastikan Jenkins aktif dan berjalan dengan status “active (running)”.

```
kahyuwesuma@DevOps-PRAKTISI:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service;
enabled; vendor prese>
     Active: active (running) since Mon 2025-02-10 02:50:21
UTC; 16s ago
   Main PID: 472 (java)
      Tasks: 50 (limit: 2322)
     Memory: 455.4M
        CPU: 36.773s
     CGroup: /system.slice/jenkins.service
```

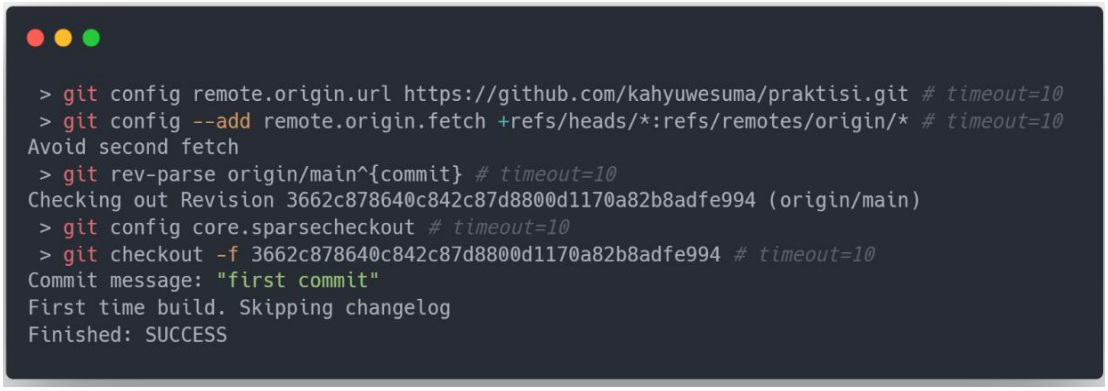
**Gambar 3.** Status Jenkins

Setelah layanan aktif, Jenkins siap digunakan untuk mengatur *pipeline* CI/CD. Tahapan ini dikonfirmasi melalui pengujian awal dengan menjalankan proses build sederhana dan meninjau log layanan Jenkins untuk memverifikasi bahwa sistem berjalan tanpa kendala pada virtual.

### 3.4.2 Integrasi Jenkins dan GitHub

Integrasi Jenkins dengan GitHub dilakukan melalui konfigurasi Secure Shell (SSH), yang memungkinkan Jenkins mengakses repositori GitHub secara otomatis dan aman. SSH key dipasang pada GitHub sebagai *public key*, sedangkan *private key* disimpan di Jenkins Credential Manager. Proses ini memungkinkan Jenkins menarik kode dari GitHub tanpa perlu memasukkan kredensial secara manual.

Langkah selanjutnya adalah membuat Jenkins Job, yaitu unit kerja otomatis yang digunakan untuk menjalankan pipeline. Dalam konfigurasi Job, ditentukan URL repositori GitHub, credential yang telah dikonfigurasi sebelumnya, serta branch target (seperti `main`) yang akan menjadi pemicu pipeline. Job ini juga mengatur tahapan *build*, pengujian dengan PHPUnit, hingga *deployment* menggunakan Docker.




**Gambar 4.** Integration Tests

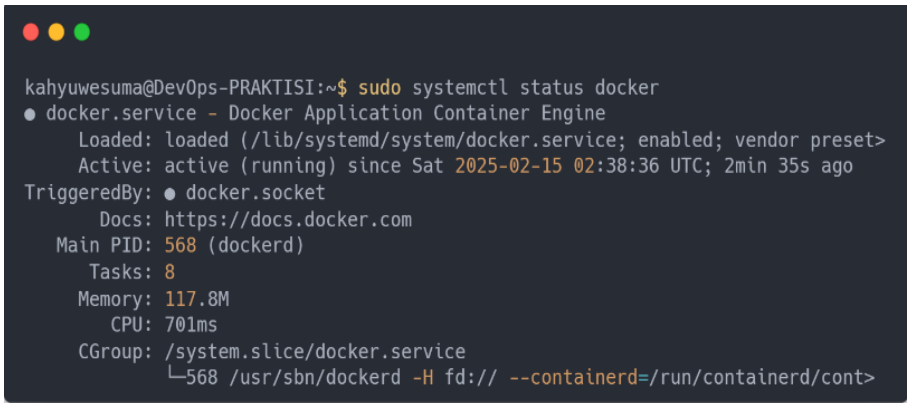
Setelah seluruh konfigurasi selesai, dilakukan pengujian awal integrasi. Jenkins berhasil menarik kode dari repositori GitHub dan menjalankan proses build sesuai skenario *pipeline* yang telah dirancang. Hal ini menunjukkan bahwa integrasi antara Jenkins dan GitHub berjalan dengan baik yang dapat dilihat pada Gambar 4 diatas, istem siap digunakan untuk mendukung proses pengembangan perangkat lunak secara otomatis dan efisien.

## 3.5 Docker *Setup*

Docker digunakan sebagai teknologi containerisasi untuk menciptakan lingkungan pengembangan yang konsisten serta memastikan aplikasi dapat dijalankan secara stabil di berbagai platform. Dalam konteks implementasi pipeline CI/CD, Docker berperan dalam mengemas seluruh dependensi aplikasi ke dalam satu *container*, sehingga proses build, pengujian, dan deployment menjadi lebih mudah diatur dan direplikasi.

### 3.5.1 Instalasi Docker

Instalasi Docker dilakukan pada Virtual Private Server (VPS) berbasis Debian 11, tempat Jenkins juga diinstal. Tahapan instalasi dimulai dengan pembaruan dependensi sistem, pemasangan `ca-certificates`, `curl`, dan pustaka pendukung lainnya. Setelah itu, Docker Engine dan Docker Compose diinstal untuk memungkinkan pengelolaan container dan layanan multikontainer.




**Gambar 5.** Status Docker

Pada Gambar 5 di atas Setelah instalasi selesai, dilakukan aktivasi layanan Docker serta verifikasi statusnya menggunakan perintah `systemctl status docker`. Jika status menunjukkan "active (running)", maka Docker siap digunakan sebagai bagian dari pipeline CI/CD.

### 3.5.2 Konfigurasi Dockerfile

Dockerfile dirancang untuk membangun image Laravel 11 yang dilengkapi dengan seluruh dependensi sistem dan pustaka aplikasi. Base image yang digunakan adalah `php:8.2-cli`, kemudian dilengkapi dengan pustaka untuk pengolahan gambar (libpng-dev, libjpeg-dev, libfreetype6-dev), database (pdo_mysql, libmariadb-dev-compat), serta pustaka tambahan lainnya seperti intl dan zip.

```
FROM php:8.2-cli

RUN apt-get update && apt-get install -y \
    libpng-dev libjpeg-dev libfreetype6-dev zip git curl \
    libmariadb-dev-compat libicu-dev pkg-config libzip-dev \
    && docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install gd pdo pdo_mysql intl zip \
    && apt-get clean

RUN curl -fsSL https://deb.nodesource.com/setup_18.x | bash - \
    && apt-get install -y nodejs

RUN node -v && npm -v

RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

RUN npm install -g yarn

WORKDIR /var/www/html

COPY . /var/www/html

RUN composer install --no-interaction --prefer-dist

RUN yarn install && yarn build

EXPOSE 8083

CMD ["php", "artisan", "serve", "--host=0.0.0.0", "--port=8083"]
```

**Gambar 6.** Dockerfile

Untuk mendukung pengelolaan aset dapat dilihat pada Gambar 6 di atas untuk front-end Laravel 11, Node.js versi 18, NPM, dan Yarn juga diinstal. File aplikasi kemudian disalin ke dalam direktori kerja di container, diikuti dengan instalasi dependensi menggunakan Composer (`composer install`) dan build aset front-end (`yarn install && yarn build`). Dockerfile juga mengekspos port 8083 untuk mengakses aplikasi Laravel dari luar container, serta menjalankan server Laravel melalui perintah `php artisan serve`.

### 3.5.3 Integrasi docker-compose.yml

Untuk mempermudah orkestrasi layanan dalam pengembangan dan deployment aplikasi, digunakan file `docker-compose.yml` dengan versi 3.8. File ini digunakan untuk menjalankan dua layanan utama, yaitu `app` sebagai aplikasi Laravel dan `db` sebagai layanan basis data MySQL.

Layanan `app` dibangun dari direktori kerja saat ini (`.`), dan menghasilkan sebuah container dengan nama `praktisi_app`. Container ini dikonfigurasi agar selalu aktif (`restart: always`) dan menggunakan volume `./:/var/www` untuk menyinkronkan direktori kerja di host dan container. Aplikasi ini juga diatur agar otomatis memuat konfigurasi dari file `.env`, dan dapat diakses melalui port 8083. Untuk mengatur komunikasi antar layanan, container `app` menggunakan jaringan internal bernama `praktisi_network` dan akan menunggu hingga layanan `db` siap sebelum dijalankan (`depends_on: db`).

Sementara itu, layanan `db` menggunakan image `mysql:8` dan menghasilkan container bernama `praktisi_db`. Container ini dikonfigurasi dengan kredensial yang dimuat melalui environment variables, seperti nama basis data (`MYSQL_DATABASE=praktisi`), pengguna (`MYSQL_USER=root`), serta sandi (`MYSQL_PASSWORD=root`, `MYSQL_ROOT_PASSWORD=root`). Layanan database ini menjalankan perintah tambahan `--default-authentication-plugin=mysql_native_password` untuk menjamin kompatibilitas autentikasi. Penyimpanan data basis data disimpan secara persisten menggunakan volume `dbdata` yang dipetakan ke direktori `/var/lib/mysql`, serta mengekspos port 3306 untuk konektivitas eksternal.

Kedua layanan ini terhubung melalui jaringan internal `praktisi_network` yang didefinisikan secara eksplisit, sehingga memastikan koneksi aman dan efisien antar container selama proses pipeline CI/CD berlangsung. Konfigurasi ini memungkinkan pengembangan, pengujian, dan deployment aplikasi dilakukan secara terisolasi dan konsisten di berbagai lingkungan.

```yaml
version: '3.8'

services:
  app:
    build: .
    container_name: praktisi_app
    restart: always
    volumes:
      - .:/var/www
    depends_on:
      - db
    env_file:
      - .env
    ports:
      - "8083:8083"
    networks:
      - praktisi_network

  db:
    image: mysql:8
    container_name: praktisi_db
    restart: always
    environment:
      MYSQL_DATABASE: praktisi
      MYSQL_USER: root
      MYSQL_PASSWORD: root
      MYSQL_ROOT_PASSWORD: root
    command: --default-authentication-plugin=mysql_native_password
    volumes:
      - dbdata:/var/lib/mysql
    ports:
      - "3306:3306"
    networks:
      - praktisi_network
volumes:
  dbdata:
networks:
  praktisi_network:
```

**Gambar 7.** Docker-compose.yml

### 3.6 Implementasi *Automation Process*

Implementasi automation process dalam pipeline CI/CD pada Gambar 7 di atas bertujuan untuk mengotomatisasi proses *build*, *pengujian*, dan *deployment* dari *Dashboard Application Praktikum Program Studi Sistem Informasi*. Dengan memanfaatkan Jenkins, Docker, dan PHPUnit, setiap perubahan kode pada repositori GitHub akan diproses secara otomatis, sehingga integritas dan kualitas aplikasi dapat terjaga sebelum diterapkan ke lingkungan produksi.

#### 3.6.1 *Automation Build*

Tahap Automation Build pada Gambar 8 dan Gambar 9 dibawah mengotomatiskan proses membangun aplikasi setelah setiap *commit* dilakukan ke repositori GitHub. Jenkins dikonfigurasi agar secara otomatis menjalankan pipeline menggunakan GitHub Webhook yang mengarah ke endpoint Jenkins, memicu eksekusi pipeline setiap kali terdapat perubahan kode.

Pipeline diawali dengan tahapan Checkout Code, di mana Jenkins menarik kode terbaru dari cabang `main`. Selanjutnya, Jenkins menjalankan perintah `docker build` untuk membuat image Laravel 11 berdasarkan konfigurasi Dockerfile. Setelah berhasil dibangun, container dijalankan melalui perintah `docker run`, dengan variabel lingkungan yang disesuaikan (seperti DB_HOST, APP_KEY, dll), dan Laravel dijalankan dengan `php artisan serve`.

```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE = 'praktisi-app'
        DOCKER_REGISTRY = 'docker.io'
        DOCKER_TAG = 'latest'
        CONTAINER_NAME = 'praktisi-app-container'
    }

    stages {
         stage('Checkout Code') {
            steps {
                git branch: 'main', credentialsId: 'github-praktisi',
                url: 'git@github.com:kahyuwesuma/DevOps-PRAKTISI.git'
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    sh "docker build -t ${DOCKER_IMAGE}:${DOCKER_TAG} ."
                }
            }
        }
```

**Gambar 8.** Automation Build Stages

```
stage('Run Laravel Container') {
            steps {
                script {
                    withCredentials([
                        string(credentialsId: 'DB_HOST', variable: 'DB_HOST'),
                        string(credentialsId: 'APP_KEY', variable: 'APP_KEY'),
                        string(credentialsId: 'DB_USERNAME', variable: 'DB_USERNAME'),
                        string(credentialsId: 'DB_PASSWORD', variable: 'DB_PASSWORD'),
                        string(credentialsId: 'DB_DATABASE', variable: 'DB_DATABASE')
                    ]) {
                        withEnv([
                            "DB_HOST=${DB_HOST}",
                            "APP_KEY=${APP_KEY}",
                            "DB_USERNAME=${DB_USERNAME}",
                            "DB_PASSWORD=${DB_PASSWORD}",
                            "DB_DATABASE=${DB_DATABASE}"
                        ]) {
                            sh '''
                            echo "Starting Docker Container..."
                            echo "Container Name: $CONTAINER_NAME"
                            echo "Docker Image: ${DOCKER_REGISTRY}/${DOCKER_IMAGE}:${DOCKER_TAG}"
                            docker run -d -p 8083:8083 --name $CONTAINER_NAME \
                            -e DB_HOST=$DB_HOST \
                            -e APP_KEY=$APP_KEY \
                            -e DB_USERNAME=$DB_USERNAME \
                            -e DB_PASSWORD=$DB_PASSWORD \
                            -e DB_DATABASE=$DB_DATABASE \
                            ${DOCKER_REGISTRY}/${DOCKER_IMAGE}:${DOCKER_TAG} \
                            bash -c "php artisan serve --host=0.0.0.0 --port=8083"

                            echo "Container Started Successfully!"
                            '''
                        }
                    }
                }
            }
        }
    }
```

**Gambar 9.** Automation Build Stages

### 3.6.2 *Automation Testing*

Tahap Automation Test pada Gambar 10 bertujuan untuk memastikan stabilitas dan fungsionalitas aplikasi sebelum deployment. Jenkins menjalankan perintah `docker exec` untuk menjalankan `php artisan test` di dalam container, memanfaatkan PHPUnit sebagai kerangka kerja pengujian. Jika pengujian berhasil, proses dilanjutkan ke tahap selanjutnya; sebaliknya, pipeline akan dihentikan dan log kesalahan ditampilkan.

```
stage('Run Tests') {
            steps {
                script {
                    sh '''
                    echo "Running Tests..."
                    docker exec $CONTAINER_NAME php artisan test
                    echo "Tests Completed!"
                    '''
                }
            }
        }
    }
```

**Gambar 10.** Automation Test Stages

Unit test yang digunakan merupakan feature test sederhana untuk memastikan bahwa halaman utama aplikasi dapat diakses dan mengembalikan status HTTP 200 yang dapat di lihat pada Gambar 11 . Meskipun cakupan pengujian masih terbatas, langkah ini menjadi dasar untuk mengintegrasikan praktik pengujian otomatis di pengembangan aplikasi berikutnya.

```php
<?php

namespace Tests\Feature;

use Tests\TestCase;

class ExampleTest extends TestCase
{

    public function test_the_application_returns_a_successful_response(): void
    {
        $response = $this->get('/');
        $response->assertStatus(200);

}
```

**Gambar 11.** Unit Test File

### 3.6.3 *Automation Deployment*

Tahap Automation Deployment pada Gambar 12 memastikan bahwa aplikasi yang telah diuji dapat secara otomatis dirilis ke server produksi. Jenkins mengelola logika success dan failure pada blok `post`, di mana pesan keberhasilan akan dicetak ketika semua tahapan berjalan lancar. Sebaliknya, jika terjadi kegagalan, Jenkins menjalankan skrip *cleanup* untuk menghentikan dan menghapus container yang gagal, serta melakukan `docker system prune` guna menjaga efisiensi sistem.

```
post {
        success {
            echo 'Pipeline Succeeded!'
        }

        failure {
            echo 'Pipeline Failed!'
            echo 'Cleaning up Docker Containers...'
            script {
                sh '''
                docker ps -a
                docker stop $CONTAINER_NAME || true
                docker rm $CONTAINER_NAME || true
                docker system prune -f
                '''
            }
        }
    }
}
```

**Gambar 12.** *Post Action*

Konfigurasi ini penting untuk mencegah penumpukan container yang tidak terpakai dan memastikan sumber daya tetap optimal. Dengan demikian, proses *deployment* menjadi lebih cepat, andal, dan minim intervensi manual.

### 3.7 Validasi dan Pengujian

Validasi dan pengujian dilakukan untuk memastikan bahwa *pipeline Continuous Integration* (CI) dan *Continuous Deployment* (CD) berjalan sesuai dengan fungsinya. Pengujian ini mencakup skenario keberhasilan dan kegagalan pada seluruh tahapan proses otomasi, yaitu build, test, dan deployment. Tujuan dari pengujian ini adalah untuk memastikan bahwa pipeline dapat menangani proses secara otomatis dan dapat menolak perubahan yang gagal melewati tahapan validasi.

#### 3.7.1 Uji Keberhasilan

Uji keberhasilan dilakukan dengan mensimulasikan proses commit kode ke repositori GitHub. Webhook GitHub akan memicu Jenkins untuk menjalankan pipeline. Tahapan *build* dilakukan dengan Docker, menghasilkan image aplikasi Laravel yang berjalan dalam container. Langkah berikutnya PHPUnit menjalankan unit test untuk memastikan bahwa kode tidak mengandung error. Jika semua tahapan berhasil, pipeline akan melakukan deployment ke staging (jika tersedia), lalu di lanjutkanke server produksi.

Seluruh tahapan pipeline ditandai berhasil jika status tiap proses (build, run container, migration, testing, deployment) tercatat sebagai sukses. Proses ini diverifikasi melalui tampilan Jenkins dan log eksekusi.

**Tabel 10.** *Success Pipeline*

| Pipeline Stage | Keterangan | Status |
|---|---|---|
| Checkout SCM | Mengambil kode sumber dari repositori GitHub | Sukses |
| Checkout Code | Memastikan kode baru tersedia | Sukses |
| Build Docker Image | Membangun image Docker dari aplikasi Laravel | Sukses |
| Run Container | Menjalankan Container Laravel | Sukses |
| Run Migrations | Menyesuaikan skema databases | Sukses |
| Run Tests | Menjalankan Unit Test dengan PHPUnit | Sukses |
| Deployent | Merilis aplikasi ke server | Sukses |

Pada Tabel 10 di atas, *pipeline* yang berhasil menghasilkan aplikasi dapat diakses melalui *port* layanan tanpa *error*, dengan status log Jenkins menampilkan pesan sukses pada setiap tahap. Dari 7 tahapan *pipeline*, seluruhnya berhasil dijalankan (7/7), sehingga tingkat keberhasilan mencapai 100%. Keberhasilan ini menunjukkan bahwa *pipeline* telah berfungsi sesuai rancangan.

### 3.7.2 Uji Kegagalan

Uji kegagalan dilakukan dengan menyisipkan kesalahan sintaks (typo) pada `Dockerfile`. Hal ini mensimulasikan skenario umum dalam pengembangan aplikasi, di mana kesalahan kecil dapat menyebabkan *build* gagal. Jenkins pipeline yang telah dikonfigurasi akan otomatis menghentikan eksekusi ketika mendeteksi error saat proses *build*.

Pipeline akan mengeksekusi blok `post` untuk melakukan proses pembersihan terhadap container yang berjalan dan membebaskan resource Docker. Langkah ini menjaga kebersihan lingkungan dan mencegah gangguan pada proses berikutnya.

**Tabel 11.** *Failure Pipeline*

| Pipeline Stage | Keterangan | Status |
|---|---|---|
| Checkout SCM | Mengambil kode sumber dari repositori GitHub | Sukses |
| Checkout Code | Memastikan kode baru tersedia | Sukses |
| Build Docker Image | Gagal karena kesalahan peulisan sintaks pada Dockerfile | Gagal |
| Run Container | Tidak Dijalankan | - |
| Run Migrations | Tidak Dijalankan | - |
| Run Tests | Tidak Dijalankan | - |
| Deployent | Tidak Dijalankan | - |

Hasil pengujian pada Tabel 11 menunjukkan bahwa *pipeline* berhasil menghentikan proses otomatisasi ketika mendeteksi error. Dari total 7 tahapan pipeline, hanya 2 yang berhasil dieksekusi sebelum terhenti (2/7), sehingga tingkat keberhasilan pada skenario ini adalah 28,57%, dengan tingkat kegagalan 71,43%. Hal ini membuktikan bahwa *pipeline* bekerja secara optimal dalam menolak perubahan yang tidak valid, serta mencegah kode bermasalah masuk ke lingkungan produksi. Mekanisme validasi ini menjadi indikator bahwa pipeline mampu menjaga kualitas sistem dan mengurangi risiko *error* saat *deployment*.

### 3.8 Evaluasi

Evaluasi dilakukan untuk mengukur keberhasilan perancangan dan implementasi infrastruktur *Continuous Integration / Continuous Deployment (CI/CD)* menggunakan Jenkins dan Docker. Proses evaluasi memanfaatkan Prometheus dan Grafana sebagai alat pemantauan kinerja untuk memperoleh metrik *real-time* selama proses *build*, pengujian, dan *deployment*. Tujuan utama evaluasi ini adalah untuk menilai efektivitas pipeline dalam meningkatkan efisiensi pengembangan dan mendeteksi potensi masalah teknis.

Beberapa metrik utama digunakan untuk mengevaluasi performa pipeline, di antaranya Build Duration, Unit Test Duration, dan Pipeline Duration. Data diperoleh dari aktivitas pipeline pada tanggal 12 Februari 2025 melalui observasi terhadap log Jenkins dan visualisasi metrik melalui Grafana.

**Tabel 12.** Metrik Pengukuran

| Metrik | Waktu yang Diperoleh | Status |
|---|---|---|
| Build Duration | 7 Menit 5 Detik | Sukses |
| Unit Test Duration | 13 Detik | Sukses |
| Pipeline Duration | 10 Menit 43 Detik | Sukses |

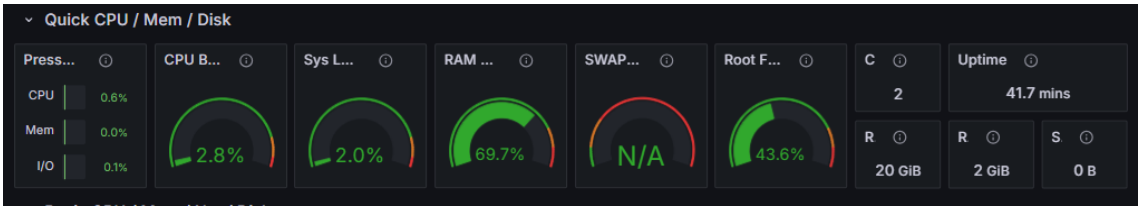
Pada Tabel 12 Hasil evaluasi menunjukkan bahwa pipeline berjalan secara stabil dan efisien. Waktu build selama 7 menit 5 detik masih tergolong optimal dan dapat ditingkatkan lebih lanjut dengan teknik caching. Durasi pengujian unit test selama 13 detik mengindikasikan bahwa proses pengujian berjalan cepat dan responsif. Pipeline secara keseluruhan mengeksekusi semua tahapan dalam waktu 10 menit 43 detik, yang mencakup build, test, hingga deployment.

Evaluasi ini membuktikan bahwa penerapan CI/CD berhasil mengotomatisasi proses pengembangan dengan baik. Meskipun demikian, terdapat ruang untuk optimasi lebih lanjut guna meningkatkan performa pipeline, khususnya pada tahap deployment.

### 3.9 Monitoring

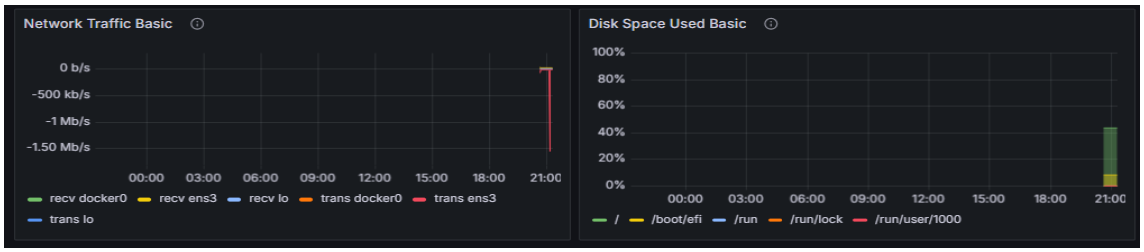
Monitoring merupakan bagian integral dalam pengelolaan infrastruktur Continuous Integration / Continuous Deployment (CI/CD). Pemantauan performa sistem secara real-time diperlukan untuk memastikan pipeline berjalan dengan stabil, mendeteksi gangguan sedini mungkin, serta menjaga efisiensi sumber daya server. Dalam penelitian ini, digunakan Prometheus sebagai *time-series data collector* dan Grafana sebagai alat visualisasi yang diintegrasikan langsung dengan server CI/CD.

*Dashboard* yang ditampilkan pada Grafana mencakup beberapa panel pemantauan utama yang difokuskan pada sumber daya sistem serta aktivitas container Docker.




**Gambar 13.** *Basic Dashboard*

Pada Gambar 13 Panel ini menampilkan informasi real-time terkait penggunaan CPU, memori, dan disk. Grafik ini digunakan untuk mengamati lonjakan beban sistem selama eksekusi *pipeline* seperti saat proses *build* atau *deployment*. Lonjakan mendadak dapat menjadi indikasi proses berat atau kesalahan konfigurasi. *Dashboard* ini menyajikan statistik yang lebih mendalam, mencakup performa CPU, konsumsi memori, lalu lintas jaringan, dan aktivitas disk I/O. Tampilan data historis memungkinkan pengembang melakukan analisis pasca-deployment, serta memahami pola penggunaan sumber daya selama aktivitas CI/CD berlangsung.




**Gambar 14.** Network Traffic

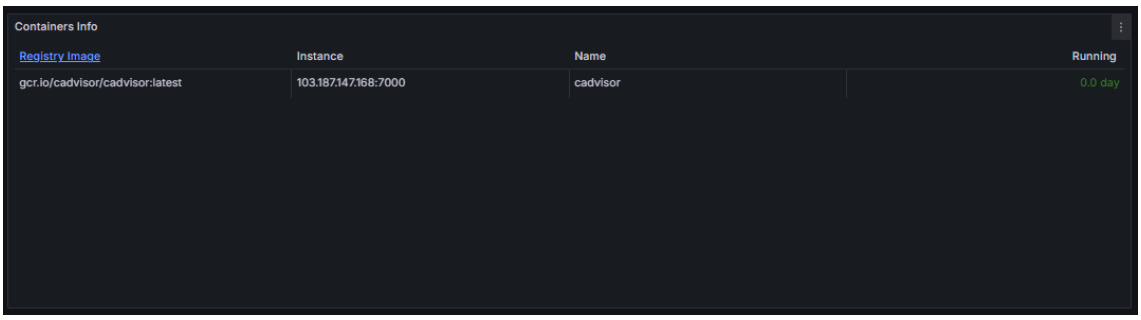
Pada gambar 14 memperlihatkan panel monitoring dasar yang menampilkan dua indikator utama, yaitu *Network Traffic Basic* dan *Disk Space Used Basic*. Panel *Network Traffic Basic* menyajikan data pergerakan lalu lintas jaringan pada beberapa antarmuka seperti docker0, ens3, dan lo dalam bentuk grafik waktu nyata (real-time). Indikator ini mempermudah pengembang dalam memantau aktivitas jaringan selama proses pipeline berlangsung, serta mengidentifikasi adanya potensi gangguan atau lonjakan lalu lintas yang tidak normal pada antarmuka tertentu.

Sementara itu, panel *Disk Space Used Basic* menampilkan tingkat pemakaian ruang penyimpanan pada berbagai direktori penting dalam sistem seperti /, /boot/efi, /run, dan /run/user/1000. Panel ini memberikan informasi visual mengenai kapasitas disk yang digunakan, yang berguna untuk mencegah kemungkinan kegagalan build atau deployment akibat kekurangan ruang penyimpanan. Melalui panel ini, tim pengembang dapat memastikan bahwa sumber daya sistem dalam kondisi aman dan optimal untuk menjalankan seluruh proses otomasi dalam pipeline CI/CD.

Pada gambar 15 menunjukkan panel monitoring *CPU Basic* dan *Memory Basic* yang merekam penggunaan sumber daya sistem secara real-time. Dari panel ini terlihat bahwa aktivitas CPU didominasi oleh status idle, yang menandakan sistem tidak dalam kondisi beban tinggi, sedangkan penggunaan memori juga masih berada dalam batas wajar dengan RAM Free yang cukup besar, menunjukkan bahwa *pipeline* berjalan efisien tanpa membebani performa server secara berlebihan.


**Gambar 15.** CPU Monitoring

Panel ini dihasilkan dari integrasi Prometheus dengan *cAdvisor Exporter*. Informasi yang ditampilkan mencakup nama container, image Docker yang digunakan, status container (running/stopped), serta *instance* tempat container berjalan. Dashboard ini sangat berguna untuk memastikan bahwa seluruh layanan seperti aplikasi Laravel, database MySQL, dan *pipeline service* berhasil dijalankan dalam container pasca deployment pada Gambar 16.




**Gambar 16.** Container Monitoring

Implementasi monitoring ini mendukung deteksi dini terhadap masalah performa serta menjadi dasar evaluasi untuk peningkatan infrastruktur pipeline CI/CD secara berkelanjutan.

## 4. KESIMPULAN

Penelitian ini telah berhasil merancang dan mengimplementasikan infrastruktur *Continuous Integration / Continuous Deployment* (CI/CD) pada *Dashboard Application* Praktikum Program Studi Sistem Informasi dengan menggunakan Jenkins dan Docker. Implementasi pipeline CI/CD memungkinkan proses *build* dan *deployment* berjalan secara otomatis, sehingga meningkatkan efisiensi pengembangan, mengurangi risiko kesalahan manual, serta mempercepat proses integrasi dan pengujian kode. Meskipun proses otomasi telah berjalan sesuai rencana, tahap pengujian seperti *unit test* dan *staging* belum sepenuhnya diterapkan karena keterbatasan pada kondisi aplikasi yang masih dalam tahap pengembangan. Dari 7 tahapan *pipeline*, seluruhnya berhasil dijalankan (7/7), sehingga tingkat keberhasilan mencapai 100%. Keberhasilan ini menunjukkan bahwa *pipeline* telah berfungsi sesuai rancangan.Total 7 tahapan pipeline, hanya 2 yang berhasil dieksekusi sebelum terhenti (2/7), sehingga tingkat keberhasilan pada skenario ini adalah 28,57%, dengan tingkat kegagalan 71,43%.

Hasil implementasi menunjukkan bahwa CI/CD memiliki potensi besar dalam meningkatkan kualitas dan kecepatan pengembangan perangkat lunak secara keseluruhan. Saran yang dapat diberikan untuk pengembangan lebih lanjut adalah menyelesaikan terlebih dahulu pengembangan aplikasi secara utuh agar seluruh tahapan pipeline, termasuk *unit test* dan *staging*, dapat diterapkan secara efektif. Selain itu, implementasi *unit test* dan *integration test* sangat dianjurkan untuk memastikan kestabilan aplikasi sebelum masuk ke produksi. Pembuatan lingkungan *staging* juga perlu dipertimbangkan untuk melakukan pengujian lebih menyeluruh sebelum aplikasi dirilis ke server produksi.

## UCAPAN TERIMAKASIH

Penulis mengucapkan terima kasih kepada Program Studi Sistem Informasi, Fakultas Teknik, Universitas Mulawarman atas dukungan fasilitas dan bimbingan yang telah diberikan selama proses penelitian ini berlangsung. Ucapan terima kasih juga disampaikan kepada semua pihak yang mendukung dalam penyusunan

penelitian ini. Tak lupa, penulis juga mengapresiasi seluruh pihak yang telah membantu secara langsung maupun tidak langsung dalam penyelesaian penelitian dan penulisan artikel ini.

# REFERENCES

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

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

[3] A. Alpery and M. A. F. Ridha, "Implementasi CI/CD Dalam Pengembangan Aplikasi Web Menggunakan Docker dan Jenkins," *Appl. Bus. Eng. Conf.*, pp. 287–296, 2021.

[4] R. A. Parama, H. Studiawan, and R. J. Akbar, "Implementasi Continuous Integration dan Continuous Delivery Pada Aplikasi myITS Single Sign On," *J. Tek. ITS*, vol. 11, no. 3, 2022, doi: 10.12962/j23373539.v11i3.99436.

[5] H. Toba, T. K. Gautama, J. Narabel, A. Widjaja, and S. F. Sujadi, "Evaluasi Metodologi CI/CD untuk Pengembangan Perangkat Lunak dalam Perkuliahan," *J. Edukasi dan Penelit. Inform.*, vol. 8, no. 2, p. 227, 2022, doi: 10.26418/jp.v8i2.51992.

[6] V. H. Adesaputra, "Implementasi Ci/Cd Pada Microservice Kompres Gambar Menggunakan Drone.Io," 2022.

[7] Creative Commons Attribution-ShareAlike 4.0, "Jenkins User Documentatio," Jenkins. Accessed: Sep. 27, 2024. [Online]. Available: https://www.jenkins.io/doc/

[8] S. Dwiyatno, E. Rakhmat, and O. Gustiawan, "Implementasi virtualisasi server berbasis docker container," *PROSISKO*, vol. 7, no. 2, pp. 165–175, 2020, doi: https://doi.org/10.30656/prosisko.v7i2.2520.

[9] A. DHANY, "Implementation of Docker and Continuous Integration / Continuous Delivery for Management Information System Development," *IJEEIT Int. J. Electr. Eng. Inf. Technol.*, vol. 3, no. 2, pp. 20–24, 2021, doi: 10.29138/ijeeit.v3i2.1208.

[10] M. Waruwu, "Metode Penelitian dan Pengembangan (R&D): Konsep, Jenis, Tahapan dan Kelebihan," *J. Ilm. Profesi Pendidik.*, vol. 9, no. 2, pp. 1220–1230, 2024, doi: 10.29303/jipp.v9i2.2141.

[11] P. R. Perkasa and E. Mailoa, "Adopsi Devsecops Untuk Mendukung Metode Agile Menggunakan Trivy Sebagai Security Scanner Docker Image Dan Dockerfile," *J. Indones. Manaj. Inform. dan Komun.*, vol. 4, no. 3, pp. 856–863, 2023, doi: 10.35870/jimik.v4i3.291.

[12] D. S. Muhammad Rizal Pahleviannur, Anita De Grave, Dani Nur Saputra, Dedi Mardianto, Lis Hafrida, Vidriana Oktoviana Bano, Eko Edy Susanto, Ardhana Januar Mahardhani, Amruddin, Mochamad Doddy Syahirul Alam, Mutia Lisya, Dasep Bayu Ahyar, *Metodologi Penelitian Kualitatif*. 2022.

[13] T. Tohirin, S. F. Utami, S. R. Widianto, and W. Al Mauludyansah, "Implementasi DevOps Pada Pengembangan Aplikasi e-Skrining Covid-19," *Multinetics*, vol. 6, no. 1, pp. 15–20, 2020, doi: 10.32722/multinetics.v6i1.2764.