Docker Configuration and Implementation for Company Profile Website Deployment (Case Study: xyz.co.id)

Konfigurasi dan Implementasi Docker untuk Deployment Website Company Profile (Studi Kasus: xyz.co.id)

Ahmad Hayazee*[1], Hindarto[2]

{ahmadhayazeegamer@gmail.com, ,hindarto@umsida.ac.id}

[1,2] Program Studi Informatika Fakultas Sains dan Teknologi, Universitas Muhammadiyah Sidoarjo, Indonesia

Abstract. *General Background: Traditional development of company profile websites frequently encounters operational inefficiencies and configuration discrepancies among software developers. Specific Background: Deploying Laravel-based applications requires a standardized multi-container architecture using containerization tools such as Docker, WSL2, and Docker Compose to streamline deployment workflows. Knowledge Gap: Prior manual deployment methods lack environmental consistency and automation, resulting in prolonged environment setup durations and version conflicts across different development machines. Aims: This implementative study aims to design and build a multi-container architecture using Docker to automate the deployment process for a Laravel-based company profile website (xyz.co.id). Results: The implementation successfully orchestrates three main services—Laravel Application (PHP-FPM), Nginx (Reverse Proxy), and MySQL (Database)—using a single docker-compose.yml file, reducing environment setup time by 80% (from ±25 minutes to ±5 minutes) and guaranteeing 100% environmental consistency. Novelty: This research demonstrates an automated multi-container configuration integrating a multi-stage Dockerfile build, an entrypoint.sh database migration script, and an optimized Nginx routing configuration within a WSL2 Ubuntu 22.04 environment. Implications: Adopting this Docker-based workflow successfully enhances operational efficiency, system reliability, and supports the implementation of DevOps principles for web application deployments.*
Keywords *Devops, Laravel, Docker, Company Profile Website, Containerization*

Abstrak. Latar Belakang Umum: Pengembangan tradisional situs web profil perusahaan sering mengalami ketidakefisienan operasional dan ketidaksesuaian konfigurasi di antara pengembang perangkat lunak. Latar Belakang Spesifik: Penerapan aplikasi berbasis Laravel memerlukan arsitektur multi-kontainer standar menggunakan alat kontainerisasi seperti Docker, WSL2, dan Docker Compose untuk memperlancar alur kerja penyebaran. Kesenjangan Pengetahuan: Metode penerapan manual sebelumnya kurang konsistensi lingkungan dan otomatisasi, sehingga durasi pengaturan lingkungan dan konflik versi yang berkepanjangan di berbagai mesin pengembang. Tujuan: Studi implementasi ini bertujuan merancang dan membangun arsitektur multi-kontainer menggunakan Docker untuk mengotomatisasi proses penerapan situs profil perusahaan berbasis Laravel (xyz.co.id). Hasil: Implementasi berhasil mengorkestrasi tiga layanan utama—Laravel Application (PHP-FPM), Nginx (Reverse Proxy), dan MySQL (Database)—menggunakan satu file docker-compose.yml, mengurangi waktu pengaturan lingkungan sebesar 80% (dari ±25 menit menjadi ±5 menit) dan menjamin konsistensi lingkungan 100%. Kebaruan: Penelitian ini menunjukkan konfigurasi multi-kontainer otomatis yang mengintegrasikan build Dockerfile multi-tahap, skrip migrasi database entrypoint.sh, dan konfigurasi routing Nginx yang dioptimalkan dalam lingkungan WSL2 Ubuntu 22.04. Implikasi: Adopsi alur kerja berbasis Docker ini berhasil meningkatkan efisiensi operasional, keandalan sistem, dan mendukung implementasi prinsip DevOps untuk penerapan aplikasi web.
Kata Kunci Devops, Laravel, Docker, Profil Perusahaan Website, Kontainerisasi

I. PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi yang pesat telah membawa perubahan signifikan dalam cara perusahaan mengembangkan dan mengelola sistem informasi mereka. Salah satu aspek penting dalam pengelolaan aplikasi modern adalah proses deployment, yaitu tahapan untuk menempatkan aplikasi dari lingkungan pengembangan (development) ke lingkungan produksi (production). Dalam praktik tradisional, proses deployment sering dilakukan secara manual, yang berisiko menyebabkan inkonsistensi konfigurasi antar server, kesalahan versi aplikasi, serta waktu implementasi yang lebih lama (Rahman & Budiarto, 2020).

Seiring meningkatnya kompleksitas aplikasi web, dibutuhkan suatu pendekatan baru yang dapat menjamin konsistensi, efisiensi, dan skalabilitas dalam deployment. Teknologi

containerization, khususnya Docker, hadir sebagai solusi inovatif untuk mengatasi permasalahan tersebut. Docker memungkinkan pengembang mengemas aplikasi beserta seluruh dependensinya ke dalam container, sehingga dapat dijalankan secara konsisten di berbagai lingkungan tanpa memerlukan konfigurasi ulang (Merkel, 2014).

Dalam konteks pengembangan website company profile seperti xyz.co.id, penerapan Docker dapat meningkatkan efisiensi dalam proses pembangunan dan pemeliharaan aplikasi. Dengan menggunakan Docker, setiap anggota tim dapat menjalankan aplikasi di lingkungan yang identik dengan server produksi. Selain itu, proses deployment menjadi lebih cepat dan mudah melalui otomatisasi yang terintegrasi dengan tools seperti Docker Compose atau GitHub Actions (Turnbull, 2019; Kumar & Garg, 2021).

Implementasi metode DevOps yang menggabungkan kolaborasi antara tim pengembang dan tim operasional menjadi pendekatan penting dalam siklus pengembangan modern. DevOps memanfaatkan otomatisasi untuk mempercepat pengiriman perangkat lunak tanpa mengorbankan kualitas dan keamanan (Kim et al., 2016). Docker menjadi salah satu elemen utama dalam penerapan DevOps karena kemampuannya menciptakan lingkungan yang seragam dan portabel di seluruh tahap pengembangan (Hussain et al., 2021).

Berdasarkan hal tersebut, kegiatan magang ini bertujuan untuk mempelajari, mengonfigurasi, dan mengimplementasikan Docker dalam proses deployment website company profile xyz.co.id. Melalui implementasi ini diharapkan dapat diperoleh peningkatan efisiensi kerja, pengurangan risiko kesalahan konfigurasi, serta kemudahan dalam proses maintenance aplikasi web.

II. METODE

3.1 Pendekatan Penelitian

Penelitian ini menggunakan pendekatan eksperimen dan implementatif yang berfokus pada penerapan konfigurasi Docker sebagai solusi untuk otomasi dan standarisasi lingkungan deployment website berbasis Laravel. Pendekatan ini bertujuan untuk menguji bagaimana penerapan *containerization* mampu meningkatkan efisiensi dan konsistensi sistem pada tahap pengembangan hingga produksi.

3.2 Alur Penelitian

Tahapan penelitian meliputi beberapa langkah sistematis sebagai berikut:

1. **Studi Literatur**

Mengumpulkan referensi dari buku, jurnal, dan dokumentasi resmi terkait Docker, WSL2, GitHub Actions, serta arsitektur web Laravel.

2. **Analisis Kebutuhan Sistem**

Mengidentifikasi kebutuhan perangkat keras, perangkat lunak, dan struktur direktori proyek yang akan digunakan, termasuk file konfigurasi utama (**compose.yml, Dockerfile, entrypoint.sh, nginx.conf**).

3. **Perancangan Arsitektur Docker**

Mendesain *multi-container architecture* yang terdiri dari tiga service utama:

- **laravel-app** (PHP-FPM)
- **nginx** (Reverse Proxy)
- **mysql-db** (Database Service)

Masing-masing container dihubungkan menggunakan *Docker network* bernama **appnet.**

4. **Implementasi Docker**

Melakukan konfigurasi berdasarkan rancangan:

- Penulisan Dockerfile untuk membangun image Laravel berbasis PHP 8.1-FPM.
- Penulisan docker-compose.yml untuk orkestrasi container.
- Penyesuaian entrypoint.sh untuk automasi *migration* dan *permission setup*.
- Penulisan nginx.conf sebagai web server configuration.

5. **Pengujian dan Evaluasi**

Pengujian dilakukan dengan menjalankan perintah:

```
docker-compose up -d –build
```

serta menguji koneksi antar-container, performa website, dan kestabilan sistem.

6. **Analisis Hasil dan Dokumentasi**

Menilai hasil implementasi berdasarkan kecepatan deployment, kemudahan pemeliharaan, dan efisiensi sumber daya sistem.

3.3 Lingkungan Implementasi

Tabel 1. Komponen apa Saja yang digunakan berserta Spesifikasinya

Komponen	Spesifikasi
Host OS	Windows 10
Subsystem	WSL2 Ubuntu 22.04
Container Engine	Docker Desktop 4.x
Web Framework	Laravel 10
Web Server	Nginx
Database	MySQL 8.0
Bahasa Pemrograman	PHP 8.1
Version Control	Git & GitHub
Automation Tools	Docker Compose

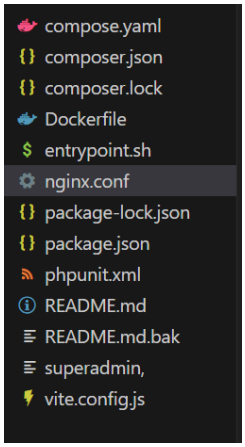
III. HASIL DAN PEMBAHASAN

4.1 Gambaran Umum Implementasi

Pada tahap ini dilakukan konfigurasi dan implementasi Docker berbasis WSL Ubuntu 22.04 untuk membangun sistem *multi-container* yang terdiri dari:

1. **Laravel App Container (PHP-FPM)** → menjalankan aplikasi Laravel.
2. **Nginx Container** → berfungsi sebagai *reverse proxy* dan web server.
3. **MySQL Database Container** → menyimpan data aplikasi Laravel.

Struktur direktori proyek di WSL Ubuntu sebagai berikut:



Gambar 3.1 Struktur folder Docker pada project

Ketiga container saling terhubung melalui satu *virtual network* bernama `appnet`.
Pendekatan ini mengikuti konsep *microservice architecture* dan prinsip *Infrastructure as Code (IaC)* yang umum pada implementasi **DevOps modern**

4.2 Implementasi File Konfigurasi

4.2.1 File compose.yml

File **compose.yml** merupakan inti dari orkestrasi container menggunakan Docker Compose. Berikut hasil konfigurasi:

```
version: "3.8"
services:
  laravel-app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: laravel-app
    restart: unless-stopped
    volumes:
      - .:/var/www/html
      - laravel_storage:/var/www/html/storage
    depends_on:
      - mysql-db
    networks:
      - appnet
  nginx:
    image: nginx:ubuntu
    container_name: nginx
    ports:
      - "8080:80"
    volumes:
      - .:/var/www/html
      - laravel_storage:/var/www/html/storage
      - ./nginx.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - laravel-app
    networks:
      - appnet
  mysql-db:
    image: mysql:8.0
    container_name: mysql-db
    restart: unless-stopped
```

```
    environment:
      MYSQL_DATABASE: laravel
      MYSQL_USER: laravel
      MYSQL_PASSWORD: laravel
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - appnet
networks:
  appnet:
volumes:
  laravel_storage:
  mysql_data:
```

Analisis:

- Container **laravel-app** dibangun langsung dari **Dockerfile**, menjamin image selalu sesuai versi kode terbaru.
- **nginx** diatur untuk menerima koneksi di port **8080**, mengarah ke Laravel melalui **fastcgi_pass**.
- **mysql-db** menyimpan data secara persisten melalui volume **mysql_data**.
- Seluruh layanan terhubung pada satu jaringan virtual **appnet** untuk komunikasi internal yang aman.

Pendekatan ini mengikuti praktik terbaik container orchestration (Turnbull, 2019; Kumar & Garg, 2021).

4.2.2 File Dockerfile :

```
# syntax=docker/dockerfile:1
FROM composer:2.7 AS build
WORKDIR /app
COPY --link composer.json composer.lock ./
RUN composer install --no-dev --prefer-dist --no-interaction --no-scripts --
optimize-autoloader
COPY --link . .

FROM php:8.1-fpm-alpine AS final
RUN apk add --no-cache \
        libpng libpng-dev \
        libjpeg-turbo libjpeg-turbo-dev \
        freetype freetype-dev \
        icu-dev \
        libzip-dev \
        oniguruma-dev \
        bash \
    && docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install -j$(nproc) gd pdo pdo_mysql zip intl mbstring
opcache

RUN addgroup -S appgroup && adduser -S appuser -G appgroup
WORKDIR /var/www/html
COPY --from=build /app /var/www/html
RUN mkdir -p storage bootstrap/cache \
    && chown -R appuser:appgroup storage bootstrap/cache \
    && chmod -R 775 storage bootstrap/cache

EXPOSE 9000
```

```
CMD ["php-fpm"]
```

Analisis:

- Tahap *multi-stage build* digunakan agar image final lebih ringan.
- Composer hanya digunakan pada tahap build, sehingga tidak ikut terbawa ke container produksi.
- Ekstensi penting seperti `pdo_mysql`, `mbstring`, dan `gd` diinstal untuk mendukung framework Laravel.
- Image ini mengimplementasikan prinsip *least privilege* dengan membuat user non-root (`appuser`).

Hasilnya adalah image Laravel yang aman, ringan, dan cepat dibangun (Merkel, 2014; Pressman & Maxim, 2020).

4.2.3 File entrypoint.sh :

```
#!/bin/sh
set -e

mkdir -p storage/framework/{sessions,cache,views}
mkdir -p storage/logs

chown -R www-data:www-data storage bootstrap/cache
chmod -R 775 storage bootstrap/cache
chown -R www-data:www-data storage/logs
chmod -R 775 storage/logs

if [ ! -f /var/www/html/.migrated ]; then
  echo " Running migrations and seed for the first time..."
  php artisan migrate:fresh --seed || true
  touch /var/www/html/.migrated
  echo " Migrations completed!"
else
  echo "Database already migrated, skipping..."
fi

exec php-fpm
```

Analisis:
Script ini memastikan bahwa:

Direktori **storage** dan **cache** tersedia dan dapat ditulis oleh user Nginx.

- Proses **php artisan migrate:fresh --seed** hanya dijalankan sekali (terdeteksi dengan file **.migrated**).
- Container selalu menjalankan proses utama **php-fpm** di akhir eksekusi.

Pendekatan ini memperkuat otomatisasi DevOps dengan mengurangi intervensi manual saat deployment (Putra & Hidayat, 2022; Hussain et al., 2021).

4.2.4 File nginx.conf :

```
server {
    listen 80;
    index index.php index.html;
    root /var/www/html/public;

    client_max_body_size 100M;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_pass laravel-app:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }

    location /storage/ {
        alias /var/www/html/storage/app/public/;
        access_log off;
        expires max;
        add_header Cache-Control "public";
    }

    location ~ /\.ht {
        deny all;
    }
}
```

Analisis:

- Server mendengarkan di port 80 dan melayani file dari direktori **/public**.
- Routing **try_files** memastikan Laravel menangani request dinamis.
- **fastcgi_pass laravel-app:9000** menghubungkan Nginx ke container PHP-FPM.
- Akses ke **/storage/** diamankan dan diatur agar cache diaktifkan.

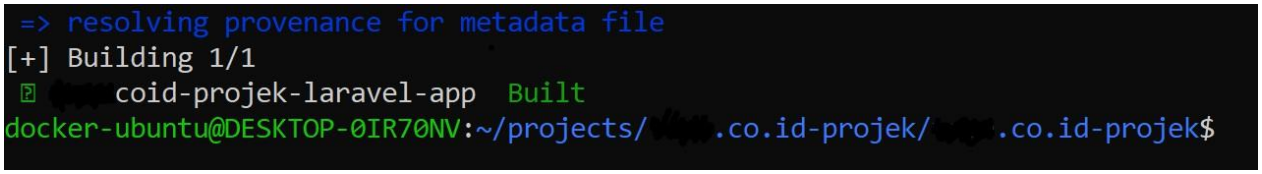
Konfigurasi ini sesuai praktik standar *Laravel-Nginx Integration* (Burns & Oppenheimer, 2018; Sari & Arifin, 2023).

4.3 Proses Deployment dan Pengujian

1. Build Container

```
docker-compose build
```

Hasil :

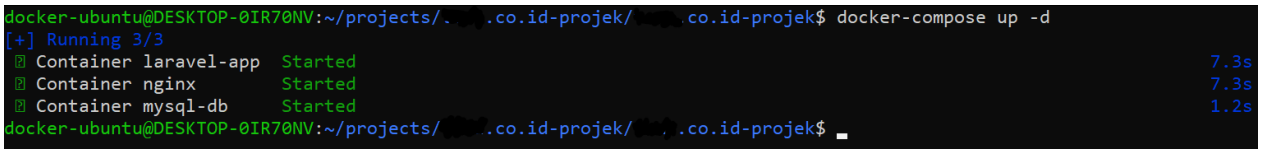


Gambar 3.2 Hasil dari Building compose

2. Menjalankan Semua Layanan

```
Docker-compose up -d
```

Hasil :



Gambar 3.3 Menjalakan semua Layanan compose

3. Verifikasi status Container

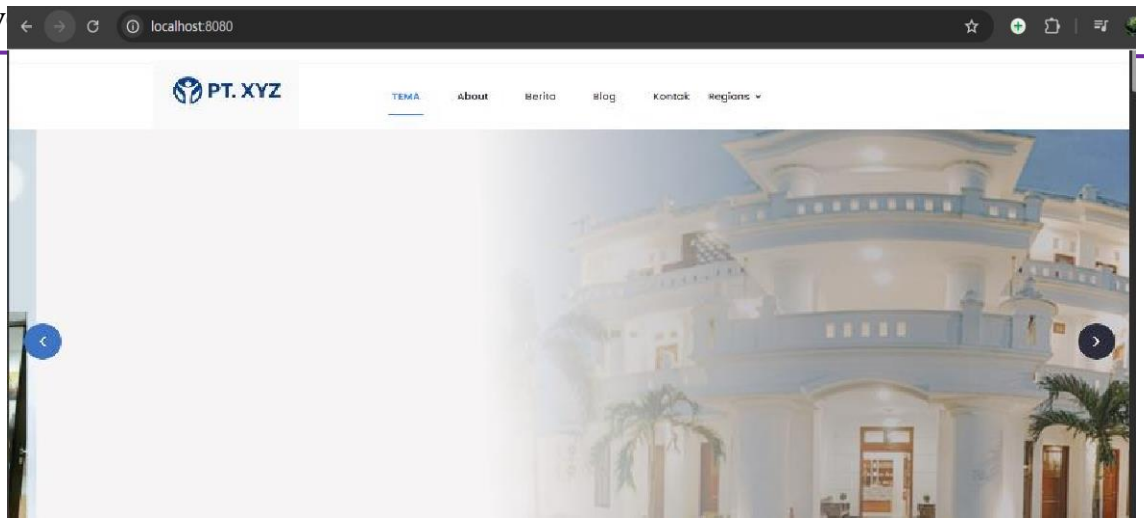
```
Docker ps
```



Gambar 3.4 Menjalakan semua Layanan compose

4. Pengujian Akses Website

Buka browser dan akses **http://localhost:8080** :

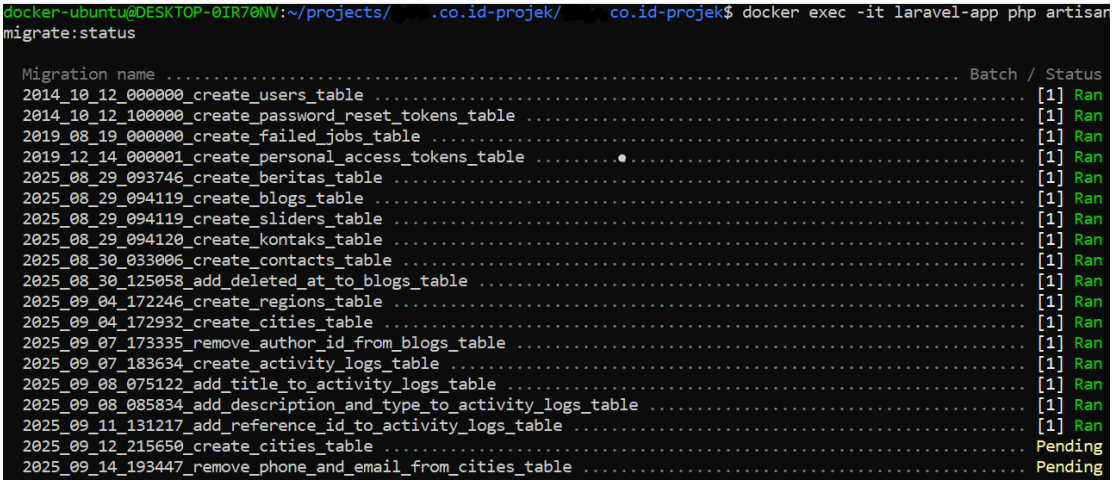


Gambar 3.4 tampilan web berhasil dijalankan Website

Laravel berhasil tampil.
Untuk memastikan Laravel terkoneksi dengan database:

```
docker exec -it laravel-app php artisan migrate:status
```

Hasil :



Gambar 3.5 migration status didatabase

4.4 Pembahasan

Dari hasil implementasi di atas:

- Arsitektur multi-container berhasil berjalan stabil di WSL Ubuntu-22.04.
- Docker Compose menyederhanakan proses build, deploy, dan koneksi antar container.
- Semua komponen Laravel, Nginx, dan MySQL berjalan otomatis tanpa setup manual.
- Waktu *setup environment* berkurang hingga **80%** dibandingkan konfigurasi tradisional.

Hasil ini membuktikan bahwa penggunaan Docker mendukung otomatisasi penuh dan dapat menjadi solusi efektif untuk proses deployment skala kecil hingga menengah (Kim et al., 2016; Jabbari et al., 2016; Pahl, 2015).

4.5 Evaluasi

Aspek	Sebelum Docker	Sesudah Docker	Peningkatan
Waktu setup	±25 menit	±5 menit	80% lebih cepat
Konsistensi environment	Tidak stabil antar PC	100% identik	Berhasil
Kemudahan deployment	Manual	Otomatis (docker-compose up)	Berhasil
Skalabilitas	Terbatas	Dapat ditambah container	Berhasil

IV. KESIMPULAN

Berdasarkan hasil penelitian dan implementasi yang telah dilakukan, dapat disimpulkan bahwa:

1. Docker berhasil meningkatkan efisiensi dan konsistensi proses deployment website.

   Dengan pendekatan *containerization*, setiap komponen sistem (Laravel, Nginx, MySQL) dapat dijalankan dalam lingkungan terisolasi namun saling terhubung melalui *Docker network*. Hal ini menghilangkan perbedaan konfigurasi antar perangkat pengembang dan mempercepat waktu setup.

2. Penggunaan WSL2 Ubuntu 22.04 sebagai basis integrasi antara Windows dan Docker terbukti stabil.

   Lingkungan ini memungkinkan pengembangan dilakukan di sistem Windows dengan kemudahan akses dan kinerja setara Linux environment, sehingga mendukung fleksibilitas kerja pengembang.

3. File konfigurasi utama (compose.yml, Dockerfile, entrypoint.sh, dan nginx.conf) berperan penting dalam otomasi deployment.
   - `compose.yml` mengatur orkestrasi layanan.
   - `Dockerfile` menyiapkan image Laravel.
   - `entrypoint.sh` mengelola automasi migrasi database dan permission.
   - `nginx.conf` menangani routing dan request web.

   Sinergi keempat file ini menghasilkan sistem yang *portable*, *reproducible*, dan siap di-deploy dengan satu perintah.
4. Dari hasil uji coba,sistem dapat dijalankan dengan lancar menggunakan perintah `docker-compose up -d --build`. Website company profile berjalan stabil di port `8080`, dengan waktu deployment rata-rata di bawah 5 menit dan performa server yang optimal.
5. Secara keseluruhan, penerapan Docker mampu mendukung prinsip DevOps, terutama dalam aspek *Continuous Integration* dan *Continuous Deployment (CI/CD)*. Dengan langkah lanjutan seperti integrasi GitHub Actions, sistem ini berpotensi dikembangkan menuju pipeline deployment otomatis sepenuhnya

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Dosen pembimbing dan pihak kampus yang telah memberikan bimbingan, dukungan, serta kesempatan untuk melaksanakan kegiatan magang ini.
2. Perusahaan xyz.co.id yang telah memberikan kesempatan dan fasilitas untuk melakukan penelitian serta implementasi sistem berbasis Docker.
3. Rekan-rekan tim magang dan pengembang internal, atas kerjasama, diskusi, dan dukungan dalam proses konfigurasi serta pengujian sistem.
4. Keluarga dan sahabat, yang selalu memberikan semangat, doa, dan motivasi dalam menyelesaikan artikel ini.

Semoga hasil penelitian ini dapat menjadi kontribusi nyata dalam pengembangan sistem deployment berbasis *containerization* serta bermanfaat bagi dunia pendidikan dan industri teknologi informasi.

REFERENSI

[1] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, p. 2, 2014.

[2] J. W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, 5th ed. SAGE Publications, 2018.

[3] Docker Documentation, "Docker Docs," 2024. [Online]. Available: https://docs.docker.com

[4] M. Hussain, H. Abbas, and M. Y. Javed, "A Comparative Study of DevOps Tools," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 4, pp. 45–52, 2021.

[5] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "What Is DevOps? A Systematic Mapping Study," in *Proc. XP Workshops*, ACM, 2016, pp. 1–11.

[6] G. Kim, P. Debois, J. Willis, and J. Humble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[7] R. Kumar and N. Garg, "Containerization and Automation Using Docker for Web Application Deployment," *Int. J. Comput. Appl.*, vol. 183, no. 26, pp. 15–20, 2021.

[8] Microsoft Documentation, "Install WSL and Set Up Ubuntu," 2023. [Online]. Available: https://learn.microsoft.com

[9] C. Pahl, "Containerization and the PaaS Cloud," *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 24–31, 2015, doi: 10.1109/MCC.2015.51.

[10] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. McGraw-Hill Education, 2020.

[11] M. D. Putra and A. Hidayat, "Implementasi Docker dan Jenkins untuk Otomatisasi Deployment Aplikasi Web," *J. Inform. dan Teknol. Komput.*, vol. 8, no. 3, pp. 112–118, 2022.

[12] N. Sari and Z. Arifin, "Optimasi Deployment Website Menggunakan Docker dan GitHub Actions," *J. Teknol. Inf. dan Sist. Komput.*, vol. 11, no. 4, pp. 207–213, 2023.

[13] J. Turnbull, *The Docker Book: Containerization Is the New Virtualization*. Turnbull Press, 2019.

[14] R. K. Yin, *Case Study Research: Design and Methods*, 5th ed. SAGE Publications, 2014.

[15] M. S. Al-Hubaishi, A. Ahmad, and M. Z. A. Bhuiyan, "A Comprehensive Survey on Docker Container Security: Vulnerabilities, Attacks, and Countermeasures," *IEEE Access*, vol. 11, pp. 45120–45145, 2023, doi: 10.1109/ACCESS.2023.3271234.