SAIN TEKOM
ISSN 2088-1770 (Print)
ISSN 2503-3247 (Online)
Vol. 15 No.2 (2025) 118-128

JURNAL
SAINTEKOM
Sains, Teknologi, Komputer, dan Manajemen

Optimasi Proses Deployment dengan Blue-Green Deployment Berbasis Containerization untuk Mencapai Zero Downtime

Restu Bumi Ryan Ramadhan[1], *Abdul Hadi[2], Siti Maryamah[3]
[1)]Teknik Informatika, Universitas Nusa Putra
Jl. Raya Cibolang Cisaat - Sukabumi No.21, Cibolang Kaler, Kec. Cisaat, Kabupaten Sukabumi, Jawa Barat
[2,3)]Teknik Informatika, STMIK Palangkaraya
Jl. G. Obos No. 114, Kel. Menteng, Kec. Jekan Raya, Palangka Raya
Email: [1]restu.bumi_ti21@nusaputra.ac.id, [2]abdulhadi@stmikplk.ac.id, [3]sitikemag@gmail.com

ABSTRACT

Downtime during deployment remains a major challenge that can disrupt service availability and reduce user trust in websites. This study aims to implement and evaluate the effectiveness of a blue-green deployment strategy based on containerization to achieve zero downtime. The implementation utilizes Docker and Traefik on an infrastructure of separate Virtual Private Servers for frontend and backend services, automated through a CI/CD pipeline using GitHub Actions. The evaluation compares non-Blue-Green and Blue-Green deployment scenarios based on operational indicators such as downtime duration, service stability, and rollback capability in case of failures. The results show that the Blue-Green strategy successfully reduces downtime from an average of 80 seconds in the non-Blue-Green scenario to zero seconds, accelerates rollback processes, and improves system scalability. Although the configuration becomes more complex, this approach proves to be safe, reliable, and effective in supporting continuous deployment in modern DevOps practices that demand uninterrupted services.

Keywords : blue-green deployment; devops; docker; CI/CD; zero downtime.

ABSTRAK

Downtime saat proses *deployment* masih menjadi kendala utama yang dapat mengganggu ketersediaan layanan dan menurunkan kepercayaan pengguna *website*. Penelitian ini bertujuan untuk mengimplementasikan serta mengevaluasi efektivitas strategi *blue-green deployment* berbasis *containerization* dalam mencapai *zero downtime*. Implementasi dilakukan dengan memanfaatkan Docker dan Traefik pada infrastruktur berbasis *Virtual Private Server* yang dipisahkan untuk layanan *frontend* dan *backend*, serta diotomatisasi melalui CI/CD *pipeline* menggunakan GitHub Actions. Evaluasi dilakukan dengan membandingkan skenario *non-Blue-Green* dan *Blue-Green deployment* berdasarkan indikator operasional seperti durasi *downtime*, stabilitas layanan, serta kemampuan *rollback* saat terjadi kegagalan. Hasil penelitian menunjukkan bahwa strategi ini berhasil menurunkan *downtime* dari rata-rata 80 detik pada skenario *non-Blue-Green* menjadi 0 detik pada penerapan *Blue-Green*, mempercepat proses *rollback*, serta meningkatkan skalabilitas sistem. Meskipun konfigurasi menjadi lebih kompleks, pendekatan ini terbukti aman, andal, dan efektif dalam mendukung *continuous deployment* pada praktik DevOps modern yang menuntut layanan tanpa interupsi.

Kata kunci : *blue-green deployment*; devops; docker; CI/CD; *zero downtime*.

Diterima Redaksi: 10-08-2025 | Selesai Revisi: 23-08-2025 | Diterbitkan Online: 30-09-2025

1. PENDAHULUAN

Dalam pengembangan sistem berbasis digital, *availability* menjadi aspek krusial yang mempengaruhi pengalaman pengguna secara langsung. Berdasarkan standar ISO/IEC 25010, *availability* termasuk dalam kategori *reliability*, bersama dengan karakteristik seperti *faultlessness*, *fault tolerance*, dan *recoverability*, yang secara kolektif menjamin sistem tetap beroperasi dan dapat diakses saat dibutuhkan (Mulyawan et al., 2021). Dalam konteks proses *deployment*, idealnya pembaruan aplikasi tidak mengganggu layanan yang sedang berjalan. *Downtime* yang terjadi selama *deployment* dapat menimbulkan ketidakpuasan pengguna dan menurunkan kepercayaan terhadap layanan digital (Bogojeska et al., 2021).

Untuk menghindari hal tersebut, praktik *deployment* modern menuntut adanya metode yang mendukung pembaruan tanpa gangguan atau *zero downtime*. Salah satu pendekatan yang dirancang untuk menjawab tantangan ini adalah *blue-green deployment* (Chaitanya K. Rudrabhatla, 2020). Dengan metode ini, dua versi aplikasi (lama dan baru) dapat berjalan bersamaan, memungkinkan versi baru diuji secara langsung sebelum digunakan penuh, sementara versi lama tetap melayani pengguna hingga pergantian yang dilakukan secara bertahap (Chaitanya K. Rudrabhatla, 2020).

Penerapan *Continuous Integration/Continuous Deployment* (CI/CD) telah menjadi standar dalam otomatisasi *pipeline* pengembangan perangkat lunak modern. CI/CD memungkinkan proses *build, test,* dan *deployment* berjalan secara konsisten dan efisien (Zampetti et al., 2021). Namun, untuk mencapai *deployment* yang benar-benar bebas dari *downtime*, dibutuhkan strategi implementasi yang tepat.

Penelitian oleh Taufik Irfan et al. (2024) menunjukkan efektivitas strategi *blue-green deployment* pada infrastruktur *multi-server* menggunakan layanan *cloud* seperti Google Cloud Platform (GCP), dengan HAProxy sebagai *load balancer* untuk menjaga ketersediaan layanan (Irfan et al., 2024). Pendekatan tersebut mengintegrasikan HAProxy dengan CI/CD *pipeline* GitLab dalam arsitektur yang kompleks dan paralel, serta mengevaluasi performa sistem secara langsung menggunakan metrik *Quality of Service* (QoS) seperti

DOI: https://doi.org/10.33020/saintekom.v15i2.926

throughput, response time, dan packet loss (Irfan et al., 2024).

Dalam penelitian ini, kompleksitas arsitektur tetap menjadi tantangan utama. Namun, pendekatan dilakukan dengan memanfaatkan layanan *Virtual Private Server* dari Contabo dan *Platform as a Service* milik Zeabur (Tencent Cloud VPS) yang relatif lebih terjangkau dibanding infrastruktur *cloud* besar seperti Google Cloud Platform. Integrasi Docker dan Traefik digunakan untuk mengimplementasikan strategi *blue-green deployment* dalam konteks *container orchestration*, di mana Docker memungkinkan pengemasan aplikasi secara *portable* dan Traefik bertindak sebagai *dynamic reverse proxy* sekaligus *load balancer* (Nuryasa & Suharjo, 2024). Konfigurasi layanan didefinisikan secara deklaratif dalam Docker Stack YAML dan diintegrasikan langsung dengan *pipeline* CI/CD, memungkinkan *deployment* otomatis antar versi aplikasi tanpa *downtime*, dengan orkestrasi layanan *multi-container* yang fleksibel namun tetap memerlukan perhatian teknis untuk sinkronisasi yang optimal (Muzumdar et al., 2024).

Penelitian ini difokuskan pada eksplorasi dan pengembangan CI/CD *pipeline* menggunakan pendekatan *blue-green deployment* berbasis *containerization*. Dengan memanfaatkan Docker dan Traefik, penelitian ini bertujuan untuk merancang strategi *deployment* yang tidak hanya memastikan *zero downtime*, tetapi juga menjaga stabilitas sistem selama proses berlangsung. Pendekatan ini diharapkan dapat memberikan solusi yang adaptif dan efisien bagi pengembangan sistem modern yang menuntut ketersediaan layanan secara kontinu (Deepak, 2024).

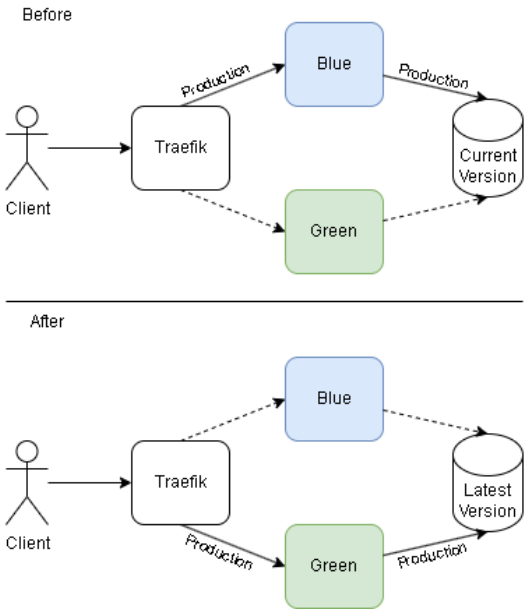
2. METODE

Penelitian ini bertujuan mengimplementasikan strategi *blue-green deployment* dengan *containerization*, serta mengevaluasi sejauh mana metode ini mampu mengurangi *downtime*, mempertahankan *uptime* sistem pasca *deployment*. Proses penelitian dilakukan dengan menelusuri konfigurasi, penerapan, serta konteks penggunaan Docker dan Traefik dalam infrastruktur mandiri berbasis *Virtual Private Server* (Contabo VPS) dan *Platform as a Service* (Tencent Cloud VPS) yang dirancang secara modular.

DOI: https://doi.org/10.33020/saintekom.v15i2.926

2.1. *Blue-Green Deployment*

Blue-Green Deployment adalah strategi yang diterapkan untuk memastikan proses pembaruan sistem dapat dilakukan tanpa menyebabkan gangguan terhadap layanan yang sedang berjalan (Antara et al., 2024). Metode ini menggunakan dua lingkungan terpisah yang berjalan secara paralel, yaitu satu sebagai lingkungan *production* dan satu lagi sebagai lingkungan *development*. Lingkungan *production* tetap melayani pengguna selama versi baru disiapkan dan diuji terlebih dahulu pada lingkungan *development*. Setelah versi baru dinyatakan siap, lalu lintas dialihkan dari *environment production* sebelumnya ke versi yang telah diperbarui, memungkinkan transisi berlangsung tanpa *downtime*. Teknik ini menjadi solusi efektif untuk meminimalkan risiko pada saat *deployment* dan mempercepat *rollback* jika terjadi masalah. *Blue-Green Deployment* mendukung CI/CD dengan menyediakan dua lingkungan identik untuk pengujian realistis, sehingga meningkatkan keandalan sistem dan memperkuat kolaborasi DevOps. Konsep dari *Blue-Green Deployment* dapat dilihat pada Gambar 1.



Gambar 1. Konsep *Blue-Green Deployment*

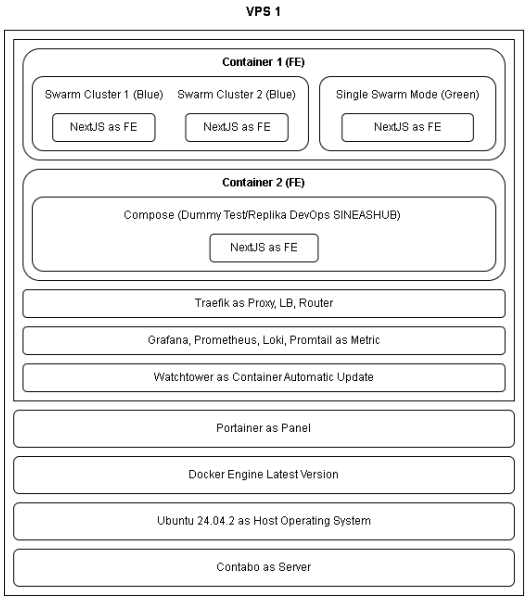
Dalam penerapannya, alur kerja dilakukan dengan memisahkan dua *branch* utama, yaitu *branch development* sebagai tempat pengembangan dan *branch production* sebagai basis *deployment* utama. Setiap kontribusi dari *developer* dilakukan melalui *pull request* ke *branch development* terlebih dahulu. Setelah melalui proses *review* dan validasi, kode akan di-*deploy* ke lingkungan *development* untuk dilakukan pengujian fungsionalitas. Jika hasil pengujian menunjukkan bahwa sistem berjalan stabil tanpa kendala, maka perubahan tersebut akan diteruskan melalui *pull request* kedua menuju *branch production* sebagai tahap akhir *deployment*. Masing-masing

DOI: https://doi.org/10.33020/saintekom.v15i2.926

branch memiliki CI/CD *pipeline* yang terpisah, sehingga proses *build* dan *deploy* untuk pengujian dan *production* dapat berjalan mandiri dan terkendali. Pendekatan ini memungkinkan tim untuk mengontrol kualitas kode dan menjamin keberlangsungan layanan tanpa interupsi, sejalan dengan prinsip *zero downtime deployment* (Manukonda, 2022).

2.2. Skema *Containerization*

Perancangan *containerization* pada sistem ini menggunakan dua buah *Virtual Private Server* (VPS) yang difungsikan secara penuh untuk lingkungan *production*, masing-masing memiliki peran tersendiri dalam pengelolaan *frontend* (FE) dan *backend* (BE). Seluruh layanan dijalankan dalam container terisolasi berbasis Docker untuk memastikan fleksibilitas, efisiensi sumber daya, serta mendukung prinsip *blue-green deployment*. Proses skalabilitas dan pemeliharaan sistem dapat dilakukan secara lebih terstruktur, sekaligus memudahkan penerapan mekanisme rollback apabila terjadi kegagalan pada saat pembaruan layanan. Skema *Containerization* VPS Contabo yang digunakan dalam penelitian ini ditunjukkan pada Gambar 2.

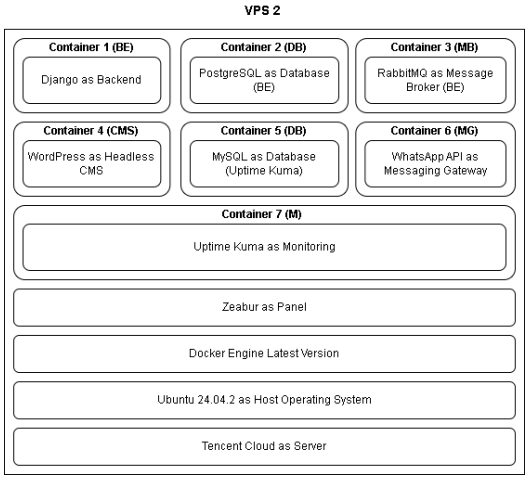


Gambar 2. Skema VPS Contabo

VPS 1, yang di*hosting* di penyedia layanan Contabo, dirancang khusus untuk menangani layanan *frontend* (FE) yang dikembangkan menggunakan *framework* Next.js. *Container* pertama menjalankan beberapa skenario *deployment* untuk kebutuhan *production* dan *development*, dengan konfigurasi Docker Swarm yang memungkinkan pengujian dan pembaruan sistem tanpa mengganggu layanan yang sedang berjalan. *Container* kedua digunakan sebagai *dummy environment* atau replika DevOps dari sistem internal perusahaan, yang digunakan untuk pengujian skenario *deployment* dan *monitoring* performa.

DOI: https://doi.org/10.33020/saintekom.v15i2.926

VPS ini juga dilengkapi dengan komponen-komponen pendukung seperti Traefik sebagai *reverse proxy*, *load balancer*, dan *router*; Grafana, Prometheus, Loki, serta Promtail untuk *monitoring* dan *log observability*, serta Watchtower untuk pembaruan *container* otomatis (Sai, 2024). Untuk pengelolaan visual, digunakan Portainer sebagai panel antarmuka, dan seluruh sistem dijalankan dalam Docker Engine versi terbaru pada sistem operasi Ubuntu 24.04.2 LTS. Skema *Containerization* VPS Tencent Cloud yang digunakan dalam penelitian ini ditunjukkan pada Gambar 3.



Gambar 3. Skema VPS Tencent Cloud

VPS 2, yang dijalankan menggunakan PaaS Zeabur dengan infrastruktur Tencent Cloud, juga merupakan bagian dari sistem *production* dan bertugas menangani seluruh layanan *backend* (BE) serta komponen pendukung lainnya. *Backend* sistem ini tidak dikembangkan dari nol, melainkan menggunakan aplikasi *open source* Taiga sebagai *platform* manajemen proyek berorientasi Scrum (Firera et al., 2023). *Container* pertama menjalankan layanan *backend* berbasis Django, disusul *container* lainnya seperti PostgreSQL untuk basis data utama *backend*, RabbitMQ sebagai *message broker*, serta WhatsApp API sebagai *messaging gateway*.

Selain itu, VPS 2 juga memuat WordPress sebagai *headless* CMS untuk konten eksternal, MySQL sebagai basis data layanan *monitoring*, serta Uptime Kuma untuk pemantauan ketersediaan layanan secara *real-time*. Semua *container* ini dikelola menggunakan *panel* Zeabur, dan dijalankan menggunakan Docker Engine terbaru pada sistem operasi Ubuntu 24.04.2 LTS.

Dengan skema ini, arsitektur sistem mampu memisahkan tanggung jawab antara *frontend* dan *backend* secara jelas, meningkatkan skalabilitas dan kemudahan manajemen layanan, serta mendukung proses *deployment* yang stabil, fleksibel, dan efisien. Fokus

DOI: https://doi.org/10.33020/saintekom.v15i2.926

utama pengembangan dilakukan pada sisi *frontend* karena dibangun khusus sesuai kebutuhan, sedangkan *backend* memanfaatkan solusi *open source* yang sudah teruji (Naga Murali Krishna Koneru, 2025).

3. HASIL DAN PEMBAHASAN

3.1. Implementasi *Blue-Green Deployment*

Pada implementasi *Blue-Green Deployment*, tujuan utamanya adalah memastikan proses *deployment* aplikasi berjalan lancar, minim *downtime*, dan mudah untuk melakukan *rollback* jika terjadi masalah di *production*. Pendekatan ini membagi *environment* menjadi dua, yaitu *blue (production)* dan *green (development)*, sehingga *update* aplikasi dapat diuji terlebih dahulu di *green* sebelum dipindahkan ke *blue*.

a. *Setup Environment*

Tahap awal adalah menyiapkan dua *environment* terpisah (*blue* dan *green*) pada Docker Swarm. Konfigurasi dilakukan melalui file docker-stack.yaml yang mendefinisikan *service* untuk *production* (web-prod) dan *development* (web-dev), serta pengaturan jaringan, volume, dan konfigurasi eksternal. Selain itu, CI/CD *pipeline* diatur menggunakan GitHub Actions untuk mengotomasi proses *build, scan, deploy,* dan *cleanup resource*.

Pada tahap ini, seluruh konfigurasi dan *workflow deployment* telah disusun, namun belum dijalankan secara penuh di *production*. Semua proses masih dalam tahap persiapan dan validasi di *environment development (green)*.

b. *Deploy & Switch*

Setelah *environment* siap, CI/CD *pipeline* dijalankan setiap ada perubahan pada *branch development* (untuk *green*) dan *master* (untuk *blue/production*). Proses *build image*, *push* ke *registry*, dan *deploy* ke *server* dilakukan secara otomatis. Setelah aplikasi di *environment green* diverifikasi berjalan dengan baik, *deployment* dapat dipindahkan ke *blue environment* dengan melakukan *update routing* pada Traefik.

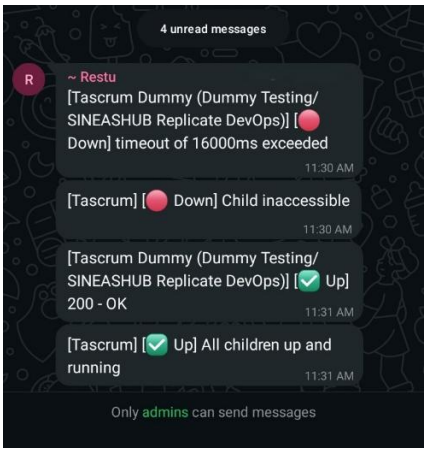
Pada tahap ini, *deployment* ke *production* dilakukan dengan strategi *rolling update* dan *auto-rollback* jika terjadi *error* (Kolawole, 2025). Jika aplikasi di *green* sudah stabil, *switch* ke *blue* dilakukan tanpa *downtime*. Semua *service* tetap berjalan, dan *rollback* dapat

DOI: https://doi.org/10.33020/saintekom.v15i2.926

dilakukan dengan cepat jika ditemukan masalah.

c. *Maintenance*

Setelah *deployment*, *workflow cleanup* dijalankan secara terjadwal untuk membersihkan *resource* yang tidak terpakai, dan Dependabot digunakan untuk mengelola *update dependency* secara otomatis guna menjaga keamanan aplikasi. Selain itu, sistem juga dilengkapi dengan mekanisme notifikasi melalui WhatsApp yang dikirim secara otomatis apabila terjadi kegagalan *deployment*. Fitur ini memastikan tim *Developer* dapat segera merespons insiden dan melakukan perbaikan tanpa menunggu laporan keluhan dari pengguna. *Push Notification* ke *Platform* WhatsApp ketika *website down* dalam penelitian ini ditunjukkan pada Gambar 4.



Gambar 4. *Push Notification* dengan *Uptime Kuma* dan WAHA

Dengan diterapkannya strategi *Blue-Green Deployment*, proses pengelolaan aplikasi menjadi lebih terstruktur, andal, dan efisien. Tahapan *deployment* dapat dilakukan secara bertahap tanpa mengganggu layanan yang sedang berjalan, sekaligus memberikan kemudahan dalam melakukan *rollback* apabila terjadi kegagalan. Pendekatan ini tidak hanya meminimalkan risiko pada saat rilis versi baru, tetapi juga meningkatkan keandalan sistem secara keseluruhan, dalam DevOps. Pada penelitian ini, perbandingan dilakukan antara Replika SINEASHUB DevOps (Dummy *Branch*) dan TASCRUM DevOps (Master *Branch*). Dummy *Branch* dibuat sebagai replika karena sistem DevOps internal sudah berjalan stabil dan tidak dapat diubah langsung pada *production*, sehingga digunakan sebagai media uji *deployment*. Sementara itu, Master *Branch* merupakan *environment production* dengan penerapan penuh *Blue-Green Deployment*. Perbandingan ini menegaskan perbedaan antara skenario *non-Blue-Green* pada Dummy *Branch* dan *Blue-Green* pada Master *Branch*, terutama terkait *downtime*, kompleksitas konfigurasi, dan

DOI: https://doi.org/10.33020/saintekom.v15i2.926

ketersediaan layanan. *Benchmark* antara dua pendekatan *deployment* menunjukkan bahwa strategi *Blue-Green* memberikan keunggulan yang signifikan dalam performa operasional dan pengurangan risiko pada lingkungan *production*. Perbandingan *non-Blue-Green Deployment* dan *Blue-Green Deployment* dalam penelitian ini ditunjukkan pada pada Tabel 1.

Tabel 1. *Non-Blue-Green Deployment* dan *Blue-Green Deployment*

Parameter	Replika SINEASHUB DevOps / Dummy *Branch*	Tascrum DevOps / Master *Branch*
Github SHA	4ef29e7a4cbbbaede153e9b15aff02b49b2764d4	e9f487650c91b0be8b452563e76b7230a2123e51
Runtime	node:20.16.0-alpine3.20	oven/bun:latest
Banyak *Job*	1	3
Estimasi waktu Pemasangan Dependensi	14.72s	4.00s
Estimasi waktu *Build*	65.9s	72.4s
Estimasi waktu *Workflow*	3m 14s	4m 20s
Lamanya *Downtime*	80s	-
Menggunakan klaster?	Tidak	Ya
Menggunakan *Blue-Green*?	Tidak	Ya
Baris Konfigurasi Dockerfile?	8	18
Baris Konfigurasi Docker Stack?	26	244
Baris Konfigurasi CI/CD *Pipeline*?	53	154
.env terkekspos?	Ya	Tidak
Menggunakan Dependabot?	Tidak	Ya
Menggunakan Metrics?	Tidak	Ya

Dari tabel tersebut, terlihat bahwa strategi *Blue-Green Deployment* menghilangkan *downtime* sepenuhnya (dari 80 detik menjadi 0 detik), memungkinkan proses *switching* antar *environment* dilakukan secara terkontrol. Selain itu, meskipun *pipeline* yang diterapkan pada *master branch* memiliki konfigurasi yang lebih kompleks terlihat dari jumlah baris Dockerfile, Docker Stack, dan CI/CD *Pipeline* yang meningkat, hal ini memberikan dampak positif terhadap kestabilan sistem dan keamanan. Misalnya, *environment file* tidak lagi terekspos, dan proses dijalankan dalam *mode cluster* untuk meningkatkan skalabilitas (Idowu & Barnty, 2022). Secara keseluruhan, hasil ini menunjukkan bahwa implementasi *Blue-Green Deployment* berhasil mencapai target utamanya yaitu mengurangi resiko kegagalan,

DOI: https://doi.org/10.33020/saintekom.v15i2.926

mempercepat *rollback*, dan menjaga ketersediaan layanan secara maksimal selama proses *deployment*.

4. KESIMPULAN

Berdasarkan hasil penelitian, strategi *Blue-Green Deployment* berbasis *containerization* dengan Docker dan Traefik terbukti efektif dalam mengurangi *downtime* secara signifikan. Pengujian menunjukkan adanya perbedaan yang jelas antara skenario *non-Blue-Green* pada replika sistem dengan *downtime* rata-rata sekitar 80 detik, dibandingkan penerapan *Blue-Green* yang mampu menekan *downtime* hingga nol. Indikator ini menjadi ukuran utama efektivitas strategi dalam menjaga ketersediaan layanan. Karakteristik utama dari pendekatan ini adalah kemampuannya melakukan transisi antar versi aplikasi secara mulus, menyediakan *rollback* cepat, dan mempertahankan layanan tetap aktif selama proses *deployment*. Meskipun demikian, penelitian ini juga menemukan adanya kelemahan berupa peningkatan kompleksitas konfigurasi pada Docker Stack, dan CI/CD *pipeline*, serta kebutuhan sumber daya yang lebih besar karena menjalankan dua lingkungan secara paralel. Hal ini menunjukkan bahwa meskipun strategi ini andal dalam mencapai *zero downtime*, penerapannya tetap membutuhkan keterampilan teknis dan pengelolaan sumber daya yang matang. Penelitian selanjutnya dapat diarahkan pada upaya penyederhanaan konfigurasi, pengujian penerapan pada skala infrastruktur yang lebih besar, serta eksplorasi kombinasi dengan strategi *deployment* lain untuk mengurangi beban sumber daya sekaligus menjaga fleksibilitas sistem.

DAFTAR PUSTAKA

Antara, E. R. F. N. U., J-, I. R., Pocket, J., & Vihar, S. (2024). Optimizing Data Processing for Financial Services Platforms. *International Research Journal of Modernization in Engineering Technology and Science*, *2*(5), 296–317. https://doi.org/10.56726/IRJMETS60903

Bogojeska, J., Giurgiu, I., Stark, G., & Wiesmann, D. (2021). IBM predictive analytics reduces server downtime. *INFORMS Journal on Applied Analytics*, *51*(1), 63–75. https://doi.org/10.1287/INTE.2020.1064

Chaitanya K. Rudrabhatla. (2020). *Comparison of zero downtime based deployment techniques in public cloud infrastructure*. *June*. https://doi.org/10.13140/RG.2.2.14933.84969

Deepak. (2024). *Zero Downtime Deployments : SRE Strategies for*

DOI: https://doi.org/10.33020/saintekom.v15i2.926

Continuous Delivery. *1*(2), 17–29. https://doi.org/10.71141/30485037/V1I2P102

Firera, Al Musadieq, M., Solimun, & Hutahayan, B. (2023). *A Systematic Literature Review*. *14*(4), 282–307. https://doi.org/10.2991/978-2-38476-090-9_24

Idowu, M., & Barnty, B. (2022). *A Deep Dive into Blue-Green and Canary Deployments: Benefits, Challenges, and Best Practices*. *March*. https://www.researchgate.net/publication/390108683

Irfan, T., Wijaya, F. M., & Slameta, S. (2024). Performa quality of service (QoS) melalui implementasi blue-green deployment pada infrastruktur multiple server. *JITEL (Jurnal Ilmiah Telekomunikasi, Elektronika, Dan Listrik Tenaga)*, *4*(2), 167–176. https://doi.org/10.35313/jitel.v4.i2.2024.167-176

Kolawole, I. (2025). Improving Software Development with Continuous Integration and Deployment for Agile DevOps in Engineering Practices. *International Journal of Computer Applications Technology and Research*, *14*(01), 25–39. https://doi.org/10.7753/ijcatr1401.1002

Manukonda, A. K. (2022). Implementing Blue-Green Deployment Strategies to Minimize Downtime during Updates. *International Journal For Multidisciplinary Research*, *4*(6), 1–9. https://doi.org/10.36948/ijfmr.2022.v04i06.42907

Mulyawan, M. D., Kumara, I. N. S., Swamardika, I. B. A., & Saputra, K. O. (2021). Kualitas Sistem Informasi Berdasarkan ISO/IEC 25010: Literature Review. *Majalah Ilmiah Teknologi Elektro*, *20*(1), 15. https://doi.org/10.24843/mite.2021.v20i01.p02

Muzumdar, P., Bhosale, A., Basyal, G. P., & Kurian, G. (2024). Navigating the Docker Ecosystem: A Comprehensive Taxonomy and Survey. *Asian Journal of Research in Computer Science*, *17*(1), 42–61. https://doi.org/10.9734/ajrcos/2024/v17i1411

Naga Murali Krishna Koneru. (2025). Containerization Best Practices-Using Docker and Kubernetes for Enterprise Applications. *Journal of Information Systems Engineering and Management*, *10*(45s), 724–746. https://doi.org/10.52783/jisem.v10i45s.8905

Nuryasa, A., & Suharjo, I. (2024). Implementasi Traefik sebagai Reverse Proxy dengan Prinsip Zero Trust. *Jutisi : Jurnal Ilmiah Teknik Informatika Dan Sistem Informasi*, *13*(1), 107. https://doi.org/10.35889/jutisi.v13i1.1704

Sai, K. (2024). Enhanced Visibility for Real-time Monitoring and Alerting in Kubernetes by Integrating Prometheus,Prometheus, Grafana, Loki, and Alerta. *Interantional Journal of Scientific Research in Engineering and Management*, *08*(06), 1–5. https://doi.org/10.55041/ijsrem35639

Zampetti, F., Geremia, S., Bavota, G., & Di Penta, M. (2021). CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study. *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 471–482. https://doi.org/10.1109/ICSME52107.2021.00048

DOI: https://doi.org/10.33020/saintekom.v15i2.926