JITEL (Jurnal Ilmiah Telekomunikasi, Elektronika, dan Listrik Tenaga) p-ISSN: 2774-7972
Vol. 4, No. 2 IRWNS 2024, pp. 167-176 e-ISSN: 2775-6696
DOI: https://doi.org/10.35313/jitel.v4.i2.2024.167-176

Performa *quality of service* (QoS) melalui implementasi *blue-green deployment* pada infrastruktur *multiple server*

Taufik Irfan[1], Farhan Mulya Wijaya[2*], Slameta[3]
[1,2,3]Jurusan Teknik Elektro, Politeknik Negeri Bandung
Jl. Gegerkalong Hilir Ds. Ciwaruga, Kabupaten Bandung Barat, Indonesia
[1]taufik.irfan@polban.ac.id, [2*]farhan.mulya.tkom420@polban.ac.id, [3]slameta polban.ac.id

ABSTRAK

Dalam pengembangan perangkat lunak, strategi *deployment* seperti *blue-green deployment* menjadi krusial untuk merilis aplikasi dengan efisien, minim risiko dan menghindari *downtime*. Dalam penelitian ini, *blue-green deployment* digunakan untuk memastikan pembaruan aplikasi berjalan lancar dan handal. Strategi ini melibatkan dua lingkungan terpisah, biru dan hijau, lingkungan biru menjalankan versi aplikasi saat ini, sedangkan lingkungan hijau menjalankan versi baru. Dengan demikian, analisis performa *quality of service* (QoS) dilakukan pada infrastruktur *multiple server* untuk memvalidasi efektivitas strategi ini. Pengujian dilakukan dengan mengukur tiga parameter QoS *throughput*, *response time*, dan *packet loss* menggunakan dua klien pada kondisi sebelum, selama, dan setelah *update deployment* yang melibatkan penambahan fitur pada aplikasi web. Pengujian QoS pada aplikasi web menunjukkan keterkaitan erat antara *throughput*, *response time*, dan *packet loss*, di mana perubahan pada satu parameter mempengaruhi parameter lainnya. Berdasarkan standar TIPHON dari ETSI, skenario 1 hingga 5 memiliki *response time* sangat baik dan *packet loss* sangat baik, sementara skenario 6 hingga 8 memiliki *response time* buruk dan *packet loss* yang berkisar dari baik hingga sedang.

Kata kunci: *blue-green deployment*, *multiple* server, *update deployment*, QoS

ABSTRACT

In software development, deployment strategies such as blue-green deployment are crucial to release applications efficiently, minimize risks, and avoid downtime. In this study, blue-green deployment is used to ensure application updates run smoothly and reliably. This strategy involves two separate environments, blue and green, the blue environment runs the current version of the application, while the green environment runs the new version. Thus, a quality of service (QoS) performance analysis is performed on a multiple server infrastructure to validate the effectiveness of this strategy. Testing is done by measuring three QoS parameters throughput, response time, and packet loss using two clients under conditions before, during, and after an update deployment involving the addition of features to a web application. QoS testing on a web application shows a close relationship between throughput, response time, and packet loss, where changes in one parameter affect the other parameters. Based on the TIPHON standard from ETSI, scenarios 1 to 5 have excellent response time and excellent packet loss, while scenarios 6 to 8 have poor response time and packet loss ranging from good to moderate.

Keywords: *blue-green deployment, update deployment, QoS, multiple server*

1. PENDAHULUAN

Pada kondisi saat ini banyak perusahaan, organisasi, atau tim pengembang perangkat lunak yang memiliki keinginan untuk merilis aplikasi mereka ke pasar dengan cara yang efisien, hemat waktu, dan minim risiko serta tanpa mengalami *downtime*. Hal ini juga diiringi dengan peningkatan frekuensi rilis yang lebih sering. Selain itu, dari segi arsitektur dan kode tim pengembang, ada kebutuhan untuk dapat menulis kode dan melakukan *deployment* secara paralel agar dapat dilakukan dalam waktu bersamaan. Dengan adanya peningkatan frekuensi rilis, terdapat potensi bahwa kode yang di-*deploy* dapat berdampak pada reliabilitas aplikasi atau pengalaman pengguna. Semua ini menjadi tantangan bagi tim *engineering* untuk melakukan pengiriman perangkat lunak kepada pengguna dengan risiko minimal terhadap produk dan pengguna [1].

Penerapan CI/CD dan implementasi *blue-green* deploymet pada infrastruktur *multiple server* telah cukup banyak dikaji pada penelitian sebelumnya. Pada penelitian [2] melakukan perbandingan

Naskah diterima tanggal 16 Juli 2024, dipresentasikan tanggal 24 Juli 2024
*E-mail korespondensi

tiga teknik untuk mencapai *zero-downtime* dalam *cloud infrastructure* dengan menggunakan *blue-green deployment*. Kemudian penelitian [3] menjelaskan bagaimana mengimplementasikan melakukan percobaan metode *canary deployment* dengan menggunakan *proxy* dan penyeimbangan beban dua tahap (*load balancing*). Dalam penelitian ini, dilakukan penerapan baru dari versi produksi yang ada menggunakan strategi *canary*, *load balancing*, dan *performance measurement*. Berikutnya, peneliti [4] melakukan percobaan metode strategi *blue-green deployment* pada layanan *cloud native*. Dalam penelitian ini menghasilkan empat solusi untuk pengimplementasian penerapan *blue-green*. Solusi berbasis DNS adalah pendekatan sederhana yang dapat digunakan di sebagian besar lingkungan dengan *server* DNS. Selanjutnya, penelitian [5] melakukan perbandingan strategi *deployment* yang dilakukan antara *canary deployment*, *blue-green deployment*, dan *rolling deployment*. Penelitian [6] dilakukan eksperimen pada strategi *deployment* dengan OpenStack sebagai *cloud* privat dan AWS sebagai *cloud* publik. Berikutnya, penelitian [7] mengimplementasikan strategi *deployment* di OpenStack dan sekaligus bereksperimen di OpenStack serta percobaan simulasi pada CloudSim. Berikutnya, penelitian [8] membahas tentang melakukan beberapa eksperimen pada implementasi strategi *deployment* untuk *software as a service* (SaaS) pada *platform cloud* dengan *response time* sebagai parameter tunggal dari QoS. Selain itu terdapat penerapan *blue-green* dan *canary* pada *hybrid software* untuk sistem industri yang kompleks [9]. Selanjutnya, [10] dalam penelitian ini diungkapkan strategi implementasi bertahap (*blue-green deployment*) untuk aplikasi konsumen menggunakan sistem Kafka. Kemudian [11] melakukan implementasi strategi *deployment* dengan *pipeline* CI/CD yang di mana fokus utamanya adalah integrasi *cloud*, dengan penekanan pada penggunaan sumber daya awan untuk menguji aplikasi sebelum diperkenalkan ke lingkungan produksi secara menyeluruh.

Apabila dilihat dari penelitian sebelumnya, secara tidak langsung penelitian ini berusaha mengembangkan strategi *blue-green deployment* pada infrastruktur *multiple server* dengan menambahkan QoS dalam pengujian. Di mana QoS ini merupakan aspek penting dalam mengevaluasi performa suatu sistem, termasuk tiga parameter yang akan diujikan pada sistem ini yaitu, *throughput*, *response time*, dan *packet loss*. Pengujian kinerja sistem dari sisi QoS menjadi sangat penting dalam konteks implementasi *blue-green deployment* menggunakan CI/CD *pipeline*. Hal ini bertujuan untuk memastikan bahwa sistem yang digunakan mampu memberikan layanan yang konsisten dan dapat diandalkan dalam menganalisis data pada saat proses *update deployment* pada sistem yang diujikan. Dengan menguji sistem dari perspektif QoS, kita dapat mengevaluasi apakah sistem tersebut mampu beroperasi dengan baik secara konsisten, memberikan informasi yang akurat dan tepat waktu. Selain dari itu, tujuan penelitian ini pun membuat perancangan dan implementasi *blue-green deployment* pada infrastruktur *multiple server* yang identik menggunakan HAProxy sebagai load balancing. Gitlab CI/CD *pipeline* digunakan sebagai sistem automation dalam *update deployment* yang berupa aplikasi web.

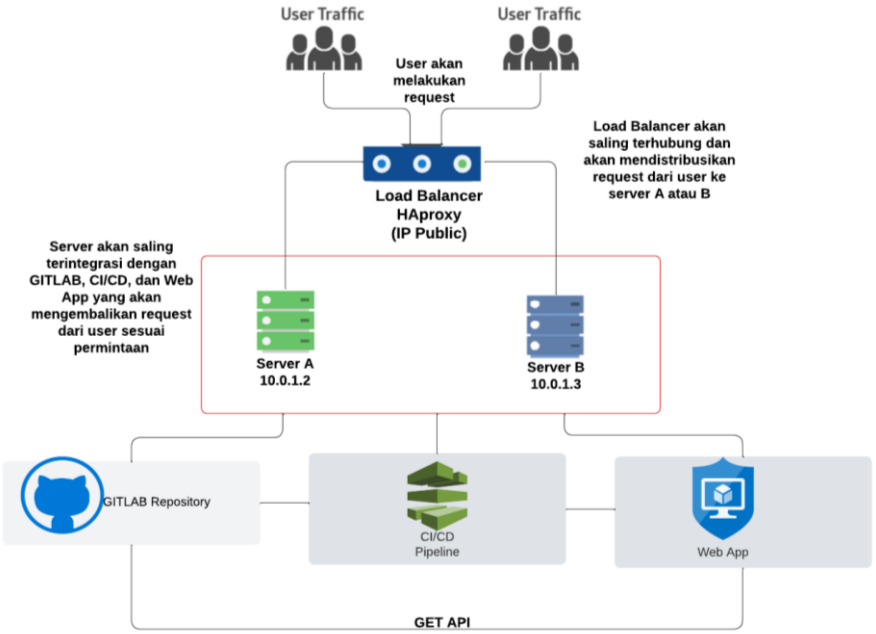
2. METODE PENELITIAN

2.1 Ilustrasi Sistem

Gambar 1 merupakan ilustrasi dari sistem yang dibuat. Dari Gambar 1 terlihat bahwa ilustrasi sistem dari pembuatan metode *blue-green deployment* pada infrastruktur *multiple server* dengan *website* sebagai *user interface*, dimulai dengan *client* akan melakukan *request* menuju *server* namun sebelumnya akan didistribusikan oleh *load balancer* HAProxy menggunakan algoritma/konsep *load balancer* yang ditentukan, yang dimana *server* akan saling terhubung dan berkomunikasi dengan CI/CD *pipeline*, Gitlab *repository*, dan aplikasi web.

Pertama-tama, pada tahap pembuatan *server* yang terinstal pada Google Cloud Platform (GCP), dua *instance virtual* dibuat dengan konfigurasi yang sama dikarenakan kedua *server* dikonsep secara identik, termasuk pengaturan *firewall* untuk mengizinkan lalu lintas yang diperlukan. Selanjutnya, pada tahap pembuatan *load balancing* HAproxy yang terinstal pada GCP, *load balancer* menggunakan HAproxy yang diimplementasikan pada algoritma/konsep yang ditentukan sesuai kebetuhuan. *Load balancer* dirancang untuk dapat mendistribusikan *request* ke 2 *server* yang tersedia, dimana skenario dibuat bahwa *server*-a merupakan *server* utama dan *server*-b merupakan *server* cadangan. Jika dilakukan *update* pada *server*-a maka server tersebut tidak dapat diakses (*down*) oleh *client* dan *request* akan diarahkan menuju server lain yaitu *server* backup. Kemudian, aplikasi web dikembangkan menggunakan teknologi web HTML, CSS, dan *framework* Django berbasis Python di dalam *environment* pengembangan seperti Visual Studio Code sebelum melakukan *deployment*.

Terakhir, pembuatan CI/CD *pipeline* dilakukan dengan mengonfigurasi Git *repository* untuk menyimpan kode sumber dan *pipeline* yang memungkinkan otomatisasi proses *deployment*, termasuk langkah-langkah akses ke *server*, metode, dan logika *blue-green deployment* dan integrasi dengan *load balancer* di GCP.

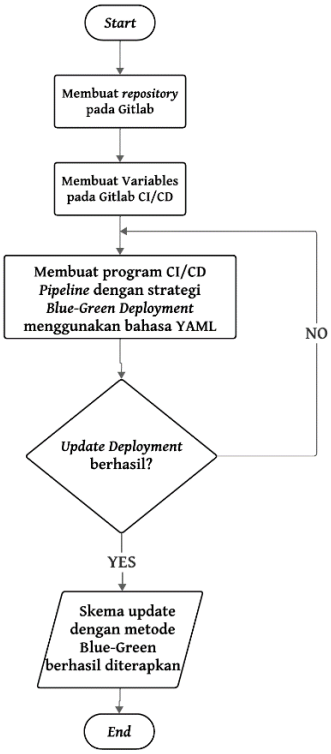


Gambar 1. Ilustrasi penggunaan sistem

Sistem kerja proyek yaitu ketika akan ada *update code* pada fitur aplikasi maka *system automation* CI/CD *pipeline* akan mengarahkan bertahap ke *server* A dan B yang dibantu oleh HAProxy sebagai load balancer. Ketika proses *update* berlangsung ke server A maka user yang mengakses otomatis akan dialihkan ke server B hal tersebut sangat menguntungkan bagi perusahaan ataupun tim perangkat lunak karena saat *update* berlangsung menghasilkan *zero down-time* pada aplikasi.

2.2 Pembuatan Strategi *Blue-Green Deployment* menggunakan Gitlab CI/CD *Pipeline*

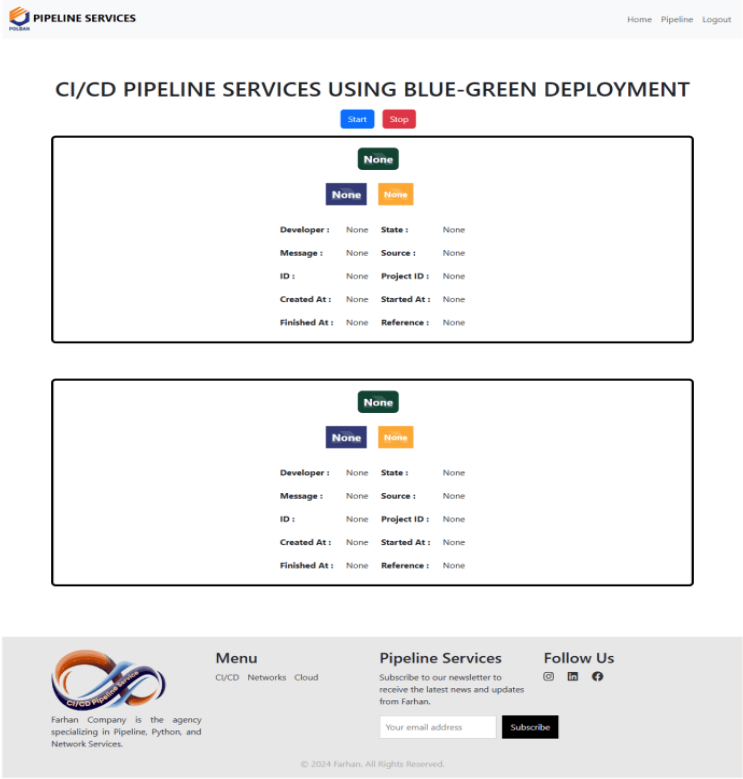
Gambar 2 merupakan arsitektur dan diagram alir sistem CI/CD *pipeline* menggunakan metode *blue-green deployment*, CI/CD *pipeline* merupakan alur kerja otomatis untuk menguji, membangun, dan menyebar perangkat lunak. Ini membantu tim mengotomatisasi pengujian, pembangunan, dan penyebaran perangkat lunak secara efisien. untuk merealisasikannya dibutuhkan *tools Software* yang telah terinstall, selanjutnya membuat codingan pada VS *Code* untuk CI/CD *pipeline* dengan metode *blue-green deployment* menggunakan Bahasa YAML. Ketika aplikasi di-*deploy* pada satu *server*, lalu *request* yang masuk kedalamnya akan dikontrol dan ditahan berdasarkan kebutuhan menggunakan *load balancer*. Selanjutnya pada *blue-green deployment*, menggunakan 2 *server* yang identik, yang dimana *blue environment* atau *environment* biru yang dimaksudkan merupakan primary *environment* yang berada sekarang di production (web V1), yaitu *environment* yang sedang dipakai *user*. Ketika tim pengembang akan melakukan *upgrade* web ke v2, maka dari tim infrastruktur menyiapkan satu *environment* identik untuk dilakukan perubahan versi web terbaru di *green environment*. Di dalam *green environment* tersebut kita akan men-*deploy* versi terbaru secara penuh (web v2).



Gambar 2. Diagram alir sistem CI/CD *pipeline* menggunakan *blue-green deployment*

2.3 Pembuatan Aplikasi Web

Gambar 3 merupakan sistem pendukung dari penelitian yaitu sebuah aplikasi web yang dikembangkan menggunakan teknologi web HTML, CSS, dan *framework* Django berbasis python di dalam *environment* pengembangan seperti Visual Studio Code sebelum melakukan *deployment*.



Gambar 3. Tampilan aplikasi web halaman CI/CD *pipeline*

2.4 Skenario Pengujian

Pengujian dilakukan pada server dengan melakukan 3 skenario percobaan yakni menggunakan *blue-green deployment* sebelum aplikasi *update*, sesedang aplikasi *update*, dan sesudah aplikasi *update* dengan menggunakan *httperf* sebagai *tools* untuk mengukur parameter QoS. Hal ini dilakukan untuk menguji, komparasi, dan melakukan analisis pada server yang berjalan. Skenario *update* aplikasi pada server yaitu menambahkan fitur-fitur pada aplikasi yang dapamemperbaiki dan membuat pr*o*fessional aplikasi yang dibuat.

Terdapat tiga parameter QoS yang akan diuji dalam sistem ini: *throughput*, *response time*, dan *packet loss* [12].

1. ***Throughput***: merupakan ukuran kecepatan transfer data dalam *bits per second* (bps). Berbeda dengan *bandwidth* yang menunjukkan kapasitas maksimum transfer data, *throughput* menunjukkan kecepatan transfer data yang sesungguhnya terjadi.
2. ***Response time***: parameter ini mengevaluasi seberapa cepat web *server* merespon permintaan pelanggan. *Response time* adalah jumlah dari *latency* dan waktu pemrosesan di sisi *server*.
3. ***Packet loss***: menunjukkan jumlah paket data yang hilang selama transmisi, yang bisa disebabkan oleh gangguan atau kepadatan jaringan. *Packet loss* dinyatakan sebagai rasio kegagalan *request* yang tidak mengembalikan respon ke *client*.

Adapun perhitungan ketiga parameter QoS tersebut dapat dilakukan berdasarkan persamaan sebagai berikut:

$$Throughput = \frac{\text{paket yang diterima}}{\text{lama pengamatan}} X\ 100\% \quad (1)$$

$$Response\ time = latency - processing\ time \quad (2)$$

$$Packet\ Loss = \frac{(\text{Paket dikirim} - \text{paket diterima})}{\text{Paket dikirim}} X\ 100\% \quad (3)$$

Skenario pengujian dilakukan dengan tiga kondisi yaitu pada saat sebelum *update* aplikasi, sesedang *update* aplikasi, dan setelah *update* aplikasi. Pada Tabel 1 ditunjukkan skenario pengujian yang dilakukan.

Tabel 1. Skenario pengujian

Skenario	Jumlah *request* per detik pada *client* 1	Jumlah *request* per detik pada *client* 2	Jumlah total *request*
1	100	100	5000
2	150	150	5000
3	200	200	5000
4	250	250	5000
5	300	300	5000
6	350	350	5000
7	400	400	5000
8	450	450	5000

Pada pelaksanaan pengujian dengan kondisi sebelum *update* aplikasi dan setelah *update* aplikasi sesuai kode skenario akan dilakukan selama lima kali percobaan. Sedangkan, pada pelaksanaan pengujian dengan skenario sedang proses *update* aplikasi sesuai kode skenario akan dilakukan selama tiga kali percobaan, dikarenakan pada proses *update* aplikasi terbatasi. Proses pengujian terdiri dari 8 skenario untuk *client* 1 dan 8 skenario pula untuk *client* 2 yang dilakukan pada sebelum *update* aplikasi, sedang *update* aplikasi, dan setelah *update* aplikasi maka jumlah seluruh pengujian *client* 1 yaitu 120 percobaan dan *client* 2 pun sama yaitu 120 percobaan Hal ini dilakukan guna mendapatkan data hasil uji yang baik, yang kemudian nilai dari lima kali percobaan ini akan di hitung rata-ratanya. Nilai dari hasil pengujian disajikan dalam bentuk grafik.

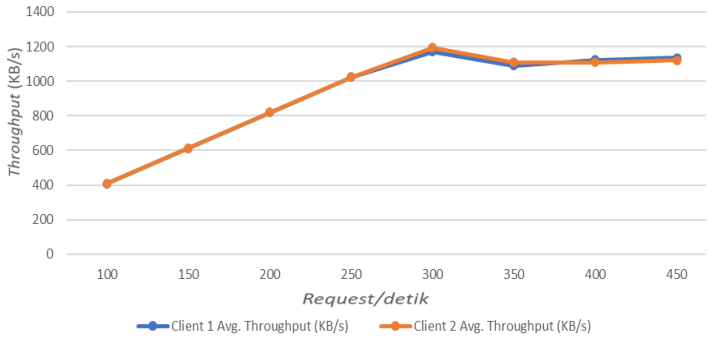
Delapan skenario untuk *client* 1 dan delapan skenario untuk *client* 2 untuk mengembalikan respon dari web fungsional atau aplikasi web. seperti yang telah disebutkan sebelumnya, untuk mendapatkan nilai rata-rata dari setiap parameter pengujian, yaitu nilai *throughput*, *response time*, dan *packet loss*, setiap kode skenario pengujian akan dijalankan sebanyak lima kali pada kondisi sebelum

update aplikasi dan setelah *update* aplikasi. Sedangkan, pada kondisi sedang proses *update* aplikasi dilakukan pengujian sebanyak tiga kali pada setiap skenario. Selain itu, nilai rata-rata yang dihasilkan oleh kedua *client* akan digunakan pada setiap percobaan untuk menghitung nilai parameter uji.

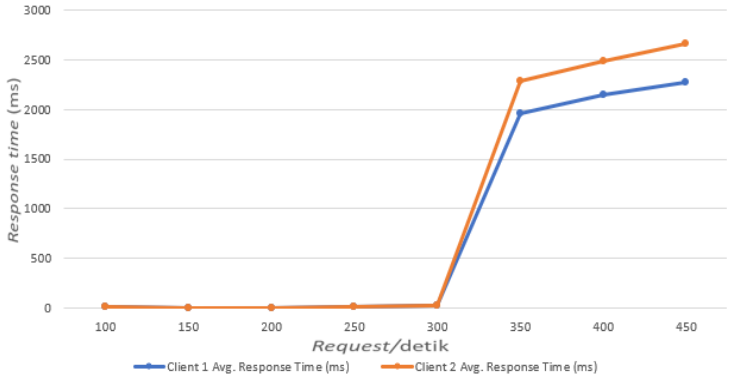
Dua klien, dua *server*, dan satu HAPoxy yang bertindak sebagai *load balancer* akan saling terhubung dengan aplikasi web. Nilai dari hasil pengujian direpresentasikan dalam bentuk grafik. Untuk menyederhanakan proses analisis dan kesimpulan, grafik dari data pengujian harus dibuat setelah data dari semua pengujian dikumpulkan. Analisa yang akan dilakukan adalah melakukan komparasi antara pengujian yang dilakukan oleh *client* 1 dan *client* 2.

3. HASIL DAN PEMBAHASAN

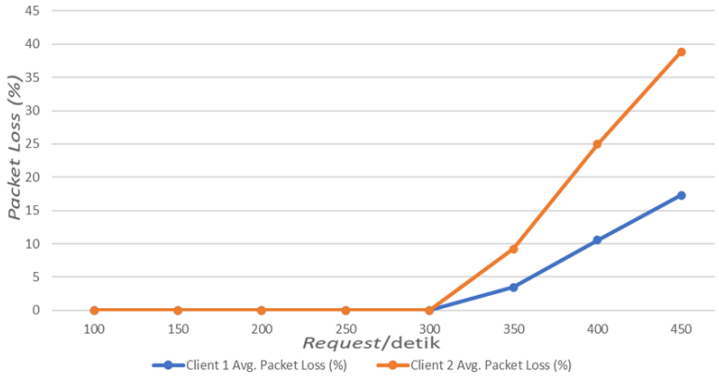
Pada bagian ini didapatkan nilai parameter QoS dari pengujian *blue-green deployment* pada infrastruktur *multiple server* sesuai dengan perancangan skenario yang sudah dibuat. Berikut grafik hasil pengujian QoS dapat dilihat pada Gambar 4 sampai dengan Gambar 12.



Gambar 4. Grafik perbandingan *throughput* pada kondisi sebelum *update* aplikasi



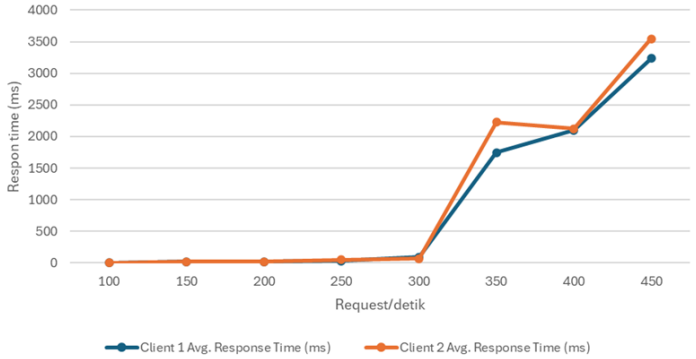
Gambar 5. Grafik perbandingan *response time* pada kondisi sebelum *update* aplikasi



Gambar 6. Grafik perbandingan *packet loss* pada kondisi sebelum *update* aplikasi

Dari pengujian QoS pada *server* yang mengembalikan aplikasi web fungsional sebelum *update* menunjukkan keterkaitan erat antara parameter *throughput*, *response time*, dan *packet loss*, serta adanya hubungan sebab-akibat di antara ketiganya. Pada skenario 1 hingga 4, *throughput* dan *response time* stabil tanpa adanya *packet loss*, mencerminkan kinerja optimal karena beban permintaan yang masih dalam batas kemampuan *server*. Namun, mulai skenario 5 dengan 300 request per detik, *throughput* mulai mengalami fluktuasi dan *packet loss* mulai muncul, mencapai 3,44% hingga 21,568% pada skenario 6 hingga 8. Hal ini menyebabkan *response time* meningkat drastis, dari 9,22 ms pada skenario 1 menjadi 752,82 ms pada skenario 5 dan lebih dari 1000 ms pada skenario 6 hingga 8. Pengujian menunjukkan bahwa skenario 1 hingga 4 memiliki *response time* sangat baik (indeks 1, <150 ms) menurut standar TIPHON. Namun, skenario 5 hingga 8 mengalami lonjakan *response time* yang buruk (indeks 4, >450 ms), menunjukkan penurunan kualitas saat beban tinggi. Kemudian, nilai *packet loss* pada skenario 1 hingga 5 memiliki predikat sangat baik (indeks 1, 0-3%). Skenario 6 dan 7 mendapat predikat baik (4-15%), sementara skenario 8 mendapat predikat sedang (>15%) karena nilai *packet loss* yang cukup besar. Kemacetan jaringan akibat peningkatan jumlah request menyebabkan *packet loss*, yang kemudian menurunkan *throughput* dan meningkatkan *response time*. Secara keseluruhan, kenaikan jumlah request per detik memicu kemacetan, yang mengakibatkan *packet loss* dan mempengaruhi performa *throughput* serta *response time*, menunjukkan bahwa ketiga parameter QoS ini saling berkaitan dan mempengaruhi satu sama lain.

Gambar 7. Grafik perbandingan *throughput* pada kondisi sedang *update* aplikasi

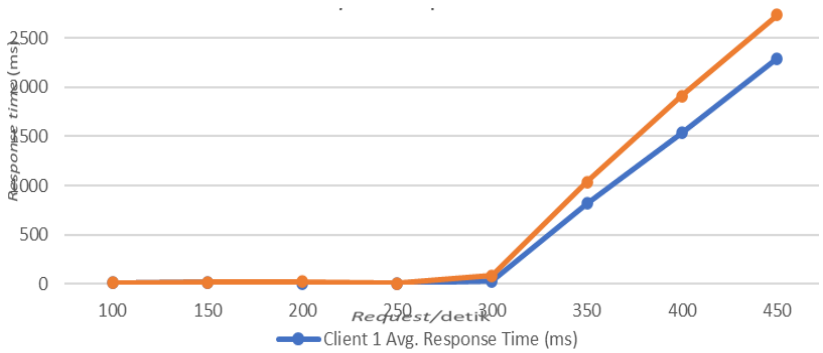


Gambar 8. Grafik perbandingan *response time* pada kondisi sedang *update* aplikasi

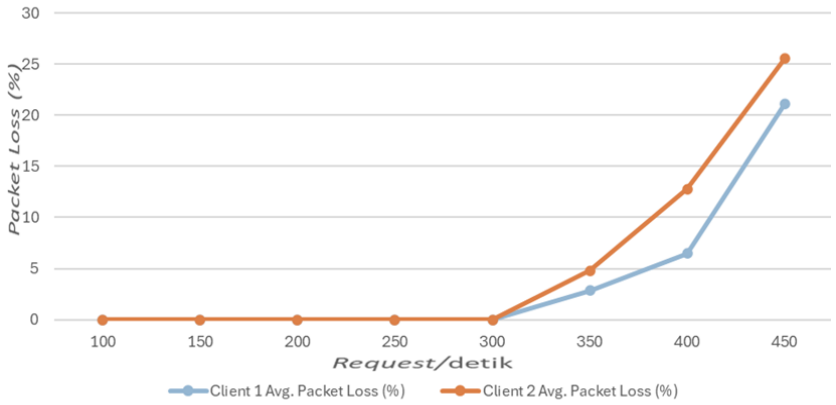
Gambar 9. Grafik perbandingan *packet loss* pada kondisi sedang *update* aplikasi

Dalam pengujian menggunakan server yang mengembalikan aplikasi web selama *update*, dengan parameter *throughput*, *response time*, dan *packet loss* menunjukkan adanya keterkaitan erat di antara ketiga parameter tersebut, yang saling mempengaruhi kinerja sistem. Pada skenario 1 hingga 5, *throughput* stabil dengan nilai rata-rata 45,8 KB/s hingga 90,53 KB/s dan *response time* yang baik, di bawah 150 ms, tanpa adanya *packet loss*. Hal ini mencerminkan kondisi ideal di mana server dapat menangani beban permintaan dengan efisien. Namun, mulai skenario 6 dengan 350 request per detik, terjadi peningkatan signifikan pada *packet loss* dan *response time* yang melonjak drastis. *Packet loss* yang meningkat dari 8,78% hingga 35,24% pada skenario 6 hingga 8 menyebabkan penurunan *throughput* dan peningkatan *response time* yang mencapai lebih dari 2000 ms. Menurut standar TIPHON untuk tingkat latency jaringan, nilai *response time* dari skenario 1 hingga 5 mendapat indeks 1, menunjukkan kategori sangat baik (<150 ms). Namun, skenario 6 hingga 8 mendapat indeks 4, menunjukkan kategori buruk (>450 ms). Lonjakan nilai *response time* pada skenario terakhir ini menunjukkan bahwa kualitas tidak memenuhi standar yang diinginkan. Sedangkan, nilai *packet loss* dari kedua client menunjukkan bahwa skenario 1 hingga 5 memiliki predikat sangat baik (indeks 1, 0-3%). Skenario 6 mendapat predikat baik (indeks 2, 4-15%), sementara skenario 7 dan 8 memiliki *packet loss* >15%, sehingga mendapat predikat sedang. Kenaikan *packet loss* mengakibatkan kemacetan jaringan, yang memperlambat transmisi data dan menghambat respons server terhadap permintaan client. Secara keseluruhan, peningkatan jumlah request per detik mengakibatkan beban berlebih pada server, yang memicu *packet loss*, penurunan *throughput*, dan peningkatan *response time*, menunjukkan adanya hubungan sebab-akibat antara ketiga parameter QoS ini.

Gambar 10. Grafik perbandingan *throughput* pada kondisi setelah *update* aplikasi



Gambar 11. Grafik perbandingan *response time* pada kondisi setelah *update* aplikasi



Gambar 12. Grafik perbandingan *packet loss* pada kondisi setelah *update* aplikasi

Pengujian QoS pada aplikasi web menunjukkan adanya keterkaitan yang erat antara *throughput*, *response time*, dan *packet loss*, di mana perubahan pada satu parameter mempengaruhi parameter lainnya. *Throughput* tetap stabil dan optimal pada skenario 1 hingga 8, dengan nilai rata-rata *throughput* dari 30,54 KB/s hingga 60 KB/s, menandakan server mampu menangani beban dengan baik pada tingkat permintaan yang berbeda. Namun, seiring meningkatnya jumlah *request*, *response time* mengalami peningkatan signifikan terutama dari skenario 5 hingga 8, di mana terjadi lonjakan *response time* yang signifikan dengan rata-rata *response time* mencapai 81,52 ms untuk *Client* 2 pada skenario 5 dan terus meningkat pada skenario 6 hingga 8. Lonjakan *response time* ini disebabkan oleh meningkatnya latency jaringan dan waktu pemrosesan. Peningkatan jumlah request juga menyebabkan peningkatan *packet loss*, yang mulai terlihat pada skenario 6 dengan 350 request per detik dan terus memburuk pada skenario 7 dan 8, mengindikasikan kemacetan jaringan yang semakin parah. Berdasarkan standar TIPHON, nilai *response time* dari skenario 1-5 memiliki kategori sangat baik (indeks 1, <150 ms), tetapi skenario 6-8 memiliki kategori buruk (indeks 4, >450 ms). Sedangkan, nilai *packet loss* dari kedua *client* menunjukkan bahwa skenario 1 hingga 5 serta skenario 6 untuk client 1 memiliki predikat sangat baik (indeks 1, 0-3%). Skenario 6 untuk *client* 2 dan skenario 7 untuk kedua client mendapat predikat baik (4-15%). Skenario 8 memiliki *packet loss* >15%, sehingga mendapat predikat sedang. Lonjakan *response time* pada skenario terakhir tidak memenuhi standar kualitas yang diinginkan. Secara keseluruhan, pengujian ini menunjukkan bahwa meskipun *throughput* server stabil, peningkatan jumlah permintaan melebihi kapasitas jaringan dan pemrosesan server menyebabkan peningkatan *response time* dan *packet loss*, yang berdampak negatif pada kualitas layanan secara keseluruhan.

4. KESIMPULAN

Berdasarkan perancangan, implementasi, hasil pengujian dan analisis dari pengukuran performa QoS melalui implementasi *blue-green deployment* pada infrastruktur *multiple server* dapat diketahui bahwa penelitian ini berhasil menerapkan *blue-green deployment* pada infrastruktur *multiple server* dengan menggunakan HAProxy sebagai *load balancer* dan CI/CD *pipeline* untuk otomatisasi *update* aplikasi. Evaluasi QoS menggunakan tiga parameter (*throughput*, *response time*, *packet loss*) dan tiga kondisi (sebelum, saat, dan setelah *update*) dengan dua klien telah dilakukan. Hasil analisis menunjukkan adanya hubungan erat antara ketiga parameter QoS, di mana perubahan pada satu parameter mempengaruhi parameter lainnya, baik sebelum, selama, maupun setelah *update* aplikasi. Pada skenario 1 hingga 5, *throughput* dan *response time tetap* stabil tanpa adanya *packet loss*, menunjukkan kondisi optimal. Namun, pada skenario 6 hingga 8, *throughput* menjadi tidak stabil dan *packet loss* meningkat yang menyebabkan *response time* meningkat. Peningkatan jumlah *request* per detik memicu kemacetan jaringan, yang mengakibatkan peningkatan *packet loss*, penurunan *throughput*, dan peningkatan *response time* menunjukkan adanya hubungan sebab-akibat antara ketiga parameter QoS ini. Selain itu, tidak ada perbedaan signifikan dari tiga kondisi (sebelum, saat, dan setelah *update*) menunjukkan *blue-green deployment* berhasil dijalankan khususnya pada kondisi saat *update deployment*. *Blue-green deployment* pada infrastruktur *multiple server* memberikan keuntungan

zero-downtime deployment, *update* yang andal, dan minim risiko. Namun, tantangannya yaitu kompleksitas konfigurasi, ketergantungan pada autentikasi, dan biaya infrastruktur tambahan bagi perusahaan profesional.

REFERENSI

[1] I. Syarifudin, "Berkenalan dengan Deployment Strategies: Canary vs Blue Green," Medium.com, 16 Januari 2023. [Online]. Available: https://irwansyarifudin16.medium.com/kenalan-dengan-Deployment-strategies-Canary-vs-Blue-Green-26d4d0d2b602. [Accessed 24 Januari 26].

[2] C. K. Rudrabhatla, "Comparison of zero downtime based Deployment technique in pulic cloud Infrastructure," *I-SMAC (IoT in Social, Mobile,Analytics and Cloud) (I-SMAC)*, vol. 978-1-7281-5464-0, pp. 1082-1086, 2020.

[3] D. Ernst, A. Becker, and S. Tai, "Rapid Canary Through Proxying and Two-Stage Load Balancing," *IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 116-122, 2019.

[4] B. Yang, A. Sailer, S. Jain, and A. E. Tomala-Reyes, "Service Discovery Based Blue-Green Deployment Technique in Cloud Native Environment," *IEEE International Conference on Services Computing*, pp. 185-192, 2018.

[5] A. Singh and V. Mansotra, "A Comparison on Continuous Integration and Continuous Deployment (CI/CD) on Cloud Based on Various Deployment and Testing Strategies," *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, vol. 9, no. VI, pp. 4968-4977, 2021.

[6] H. W. Joaquin and S. Rodriguez, "A Policy Based Application Deployment Method in Hybrid Cloud Environment," *International Conference on Future Internet of Things and Cloud*, pp. 93-99, 2018.

[7] Y. Qiu, Q. Shen, Y. luo, C. Li, and Z. Wi, "A Secure Virtual Machine Deployment Strategy to Reduce Co-Residency in Cloud," *IEEE Trustcom/BigDataSE/ICESS*, pp. 347-354, 2017.

[8] K. Huang, B. Shen, "Service Deployment strategies for efficient execution of composite SaaS applications on cloud platform," *The Journal of Systems and Software*, vol. 0164-1212, 2015.

[9] P. Rajkovi´c, D. Aleksi´c, A. Djordjevi´c, and D. Jankovi´c, "Hybrid Software Deployment Strategy for Complex Industrial Systems," *MDPI*, vol. 11, pp. 1-28, 2022.

[10] R. Seth, P. Priyadarshini, R. Sharma, V. Rastogi, and Yash, "BLUE GREEN DEPL UE GREEN DEPLOYMENT STRATEGY FOR APPLICA TEGY FOR APPLICATIONS," *Technical Disclosure Commons*, pp. 1-17, 2023.

[11] J. J. Carroll, P.Anand, and D. Guo, "Preproduction Deploys: Cloud-Native Integration Testing," *IEEE*, 2021.

[12] A. A. Sukmandhani, "QoS (Quality of Services)," onlinelearning.binus.ac.id, 15 June 2020. [Online]. Available: https://onlinelearning.binus.ac.id/computer-science/post/qos-Quality-of-Services. [Accessed 26 Januari 2024].