Analisis Perbandingan Performa NGINX dan Caddy sebagai Load Balancer Menggunakan Algoritma Round Robin pada Lingkungan Container

Rudy Sunardy[1], M. Ramaddan Julianti[2], Muhammad Ariq Brilian[3]
[1,2,3] Institut Teknologi dan Bisnis Bina Sarana Global, Tangerang, Indonesia 15113
Email: [1]rudysunardi@global.ac.id, [2]m.ramaddan.julianti@global.ac.id, [3]ariqmbrilian@gmail.com

Abstrak – Peningkatan penggunaan aplikasi berbasis web menuntut infrastruktur server yang mampu memberikan layanan secara cepat, stabil, dan andal meskipun terjadi lonjakan jumlah pengguna. Salah satu pendekatan yang umum diterapkan untuk mengatasi permasalahan tersebut adalah load balancing, yaitu mekanisme yang mendistribusikan permintaan pengguna ke beberapa backend server agar beban kerja dapat terbagi secara merata. Di antara berbagai perangkat lunak load balancer, NGINX dan Caddy menjadi dua solusi yang banyak digunakan karena kemudahan implementasi dan dukungan terhadap lingkungan berbasis kontainer. Namun, kajian yang membandingkan kinerja keduanya menggunakan algoritma Round Robin pada lingkungan containerized masih terbatas. Oleh karena itu, penelitian ini dilakukan untuk menganalisis dan membandingkan performa NGINX dan Caddy sebagai load balancer menggunakan algoritma Round Robin. Penelitian menerapkan metode eksperimen dengan membangun lingkungan pengujian berbasis Docker dan melakukan load testing menggunakan Azure Load Testing pada tiga skenario, yaitu kondisi normal, high availability, dan horizontal scalability. Evaluasi dilakukan berdasarkan response time, throughput, error rate, penggunaan CPU, dan penggunaan memori. Hasil penelitian menunjukkan bahwa kedua load balancer mampu mendistribusikan beban kerja dengan baik, namun NGINX secara umum memberikan throughput yang lebih tinggi, tingkat kegagalan yang lebih rendah, serta penggunaan memori yang lebih efisien dibandingkan Caddy. Temuan ini memberikan gambaran yang lebih komprehensif mengenai karakteristik kedua load balancer pada lingkungan berbasis kontainer dan dapat menjadi referensi dalam memilih solusi load balancing yang sesuai dengan kebutuhan implementasi sistem.

Kata Kunci - Load Balancing, Round Robin, NGINX, Caddy, High Availability

Abstract – Abstract—The growing demand for web-based applications has increased the need for server infrastructures that can deliver fast, reliable, and highly available services, even under heavy traffic conditions. Load balancing has become a widely adopted approach to address this challenge by distributing incoming requests evenly across multiple backend servers. Among the available load-balancing solutions, NGINX and Caddy have gained considerable attention due to their ease of deployment and compatibility with containerized environments. However, comparative studies evaluating their performance using the Round Robin algorithm in container-based deployments remain limited. This study aims to compare the performance of NGINX and Caddy as load balancers employing the Round Robin algorithm. An experimental approach was adopted by deploying both load balancers in a Docker-based environment and conducting performance evaluations using Azure Load Testing under three scenarios: normal operation, high availability, and horizontal scalability. The evaluation focused on response time, throughput, error rate, CPU utilization, and memory consumption. The experimental results demonstrate that both load balancers effectively distribute client requests across backend servers. Nevertheless, NGINX consistently achieved higher throughput, lower error rates, and more efficient memory utilization than Caddy across most testing scenarios. Although Caddy delivered competitive performance in several response time measurements, its reliability decreased under higher workloads. These findings provide a comprehensive performance evaluation of NGINX and Caddy in containerized environments and offer practical insights for selecting an appropriate load-balancing solution based on specific system requirements.

Keywords - Load Balancing, Round Robin, NGINX, Caddy, High Availability

I. PENDAHULUAN

Era digital menuntut keberlangsungan dan kinerja layanan internet yang prima untuk memberikan pengalaman terbaik bagi pengguna. Pengguna internet modern mengharapkan waktu aktif yang tinggi dan kinerja yang stabil dari layanan yang mereka gunakan. Namun, banyak sistem yang tidak mampu memenuhi harapan ini karena berbagai keterbatasan, salah satunya adalah kurangnya mekanisme load balancing. Load balancing adalah teknik yang memungkinkan pembagian beban kerja ke beberapa server atau salinan sistem yang sama, sehingga dapat meningkatkan kinerja dan keandalan sistem secara keseluruhan [1]. Tanpa load balancer, sistem seringkali menghadapi masalah seperti overload, downtime, dan kinerja yang tidak konsisten ketika jumlah pengguna meningkat drastis. Hal ini dapat menyebabkan pengalaman pengguna yang buruk dan potensi kerugian bagi penyedia layanan khususnya dalam penggunaan aplikasi berbasis *Application Programming Interface* (API) [2].

Penelitian sebelumnya telah menunjukkan pentingnya load balancing dalam meningkatkan performa dan keandalan sistem menemukan bahwa NGINX memiliki kemampuan yang lebih baik dalam menangani beban sedang hingga berat dibandingkan dengan HAProxy [3]. Penelitian lainnya juga menyoroti perlunya optimasi pada layanan API untuk menangani permintaan yang tinggi [4]. Penelitian lainnya, menunjukkan bahwa algoritma least-connection memiliki kinerja yang lebih baik dalam menangani requests yang tinggi dibandingkan dengan algoritma round robin [5]. Algoritma round robin menghasilkan throughput yang lebih baik dan stabilitas yang signifikan dibandingkan algoritma

least connection dan IP hash [6]. Penelitian lain juga menemukan bahwa algoritma NEW_HASH mengungguli IP hash dalam hal response time, tingkat kegagalan, dan throughput [7]. Penelitian-penelitian ini menegaskan bahwa pemilihan algoritma load balancing yang tepat dan optimasi sistem secara keseluruhan sangat penting untuk mencapai performa yang optimal.

Berdasarkan penelitian-penelitian sebelumnya, sebagian besar studi berfokus pada perbandingan NGINX dengan HAProxy atau evaluasi algoritma load balancing tertentu [3]. Penelitian yang secara khusus membandingkan NGINX dan Caddy menggunakan algoritma Round Robin pada lingkungan container berbasis Docker masih relatif terbatas. Selain itu, sebagian besar penelitian hanya mengevaluasi response time dan throughput, sementara analisis terhadap penggunaan CPU, memori, dan tingkat kegagalan pada beberapa skenario implementasi belum banyak dilakukan.

Berdasarkan kesenjangan tersebut, penelitian ini menawarkan evaluasi komprehensif terhadap dua web server modern, yaitu NGINX dan Caddy, menggunakan algoritma Round Robin pada lingkungan container berbasis Docker. Penelitian mengevaluasi empat metrik utama, yaitu response time, throughput, penggunaan CPU, dan penggunaan memori melalui tiga skenario pengujian yang merepresentasikan kondisi normal, high availability, dan horizontal scalability sehingga memberikan gambaran yang lebih komprehensif mengenai karakteristik masing-masing load balancer.

Penelitian ini bertujuan untuk mengevaluasi dampak implementasi load balancer pada sistem yang ada di PT. Andal Software menggunakan *stateful* website [8]. Penelitian ini akan menggunakan metode eksperimental dengan mengukur performa sistem sebelum dan sesudah implementasi load balancer. Performa sistem akan diukur berdasarkan beberapa metrik, seperti response time, throughput, dan tingkat kegagalan [9]. Metode eksperimental dipilih karena memungkinkan pengukuran langsung dampak implementasi load balancer pada sistem yang ada.

Dengan melakukan penelitian ini, diharapkan dapat diperoleh pemahaman yang lebih mendalam tentang manfaat implementasi load balancer pada sistem yang ada di PT. Andal Software. Hasil penelitian ini diharapkan dapat memberikan kontribusi bagi peningkatan performa, ketersediaan, dan skalabilitas sistem, sehingga dapat memberikan pengalaman pengguna yang lebih baik dan meningkatkan efisiensi operasional perusahaan.

II. METODE PENELITIAN

A. Desain Penelitian

Penelitian ini menggunakan metode eksperimen dengan pendekatan komparatif untuk membandingkan kinerja dua perangkat lunak *load balancer*, yaitu NGINX dan Caddy, yang sama-sama menerapkan algoritma *Round Robin*. Pendekatan eksperimen dipilih karena memungkinkan pengukuran secara langsung terhadap performa kedua *load balancer* pada lingkungan pengujian yang dikendalikan, sehingga hasil yang diperoleh dapat dibandingkan secara objektif.

Lingkungan pengujian dibangun menggunakan teknologi *containerization* berbasis Docker dengan memanfaatkan aplikasi web berbasis Flask sebagai *backend service*. Seluruh konfigurasi pengujian dibuat identik pada kedua *load balancer* agar perbedaan hasil yang diperoleh hanya dipengaruhi oleh karakteristik masing-masing perangkat lunak. Evaluasi kinerja dilakukan menggunakan Azure Load Testing melalui tiga skenario pengujian, yaitu kondisi normal, *high availability*, dan *horizontal scalability*. Parameter yang dianalisis meliputi *response time*, *throughput*, *error rate*, penggunaan CPU, dan penggunaan memori.

Secara umum, tahapan penelitian yang dilakukan terdiri atas:

1. Identifikasi masalah, yaitu mengidentifikasi permasalahan pada sistem yang belum menerapkan mekanisme *load balancing*, sehingga berpotensi mengalami penurunan performa ketika terjadi peningkatan jumlah pengguna.
2. Studi literatur, yaitu mengumpulkan referensi yang berkaitan dengan konsep *load balancing*, algoritma *Round Robin*, teknologi Docker, NGINX, Caddy, serta penelitian terdahulu yang relevan.
3. Analisis kebutuhan sistem, yaitu menentukan kebutuhan perangkat keras, perangkat lunak, topologi jaringan, serta parameter pengujian yang akan digunakan selama penelitian.
4. Perancangan lingkungan pengujian, yaitu membangun lingkungan eksperimen menggunakan *virtual machine*, Docker, aplikasi Flask sebagai *backend service*, serta melakukan konfigurasi NGINX dan Caddy sebagai *load balancer*.
5. Implementasi dan konfigurasi sistem, yaitu mengimplementasikan algoritma *Round Robin* pada masing-masing *load balancer* serta memastikan seluruh layanan berjalan sesuai dengan rancangan.
6. Pelaksanaan pengujian, yaitu melakukan *load testing* menggunakan Azure Load Testing pada tiga skenario pengujian untuk memperoleh data performa masing-masing *load balancer*.
7. Analisis hasil pengujian, yaitu membandingkan hasil pengukuran berdasarkan parameter *response time*, *throughput*, *error rate*, penggunaan CPU, dan penggunaan memori untuk menentukan karakteristik performa NGINX dan Caddy.
8. Penarikan kesimpulan, yaitu menyusun kesimpulan berdasarkan hasil analisis serta memberikan rekomendasi penggunaan *load balancer* sesuai dengan karakteristik kebutuhan sistem.

A. Metode Pengumpulan Data

Metode pengumpulan data yang dipakai dalam penelitian ini terbagi menjadi 2 macam yaitu studi pustaka dan studi lapangan berikut adalah penjelasannya:

1. Metode Observasi

   Peneliti melakukan pengamatan dengan mempelajari referensi yang terkait dengan container, Docker, load balancer, NGINX, dan Caddy. Penelitian ini juga menggunakan berbagai referensi yang berasal dari jurnal untuk mendukung penelitian, termasuk jurnal nasional, jurnal internasional, video tutorial YouTube, dan website resmi. Observasi dilakukan dengan membuat development environment dan melakukan uji beban (load test) pada aplikasi microservice yang telah dikonfigurasi dengan load balancer [10]. Uji beban ini bertujuan untuk mengamati platform load balancer NGINX dan Caddy serta kinerjanya terhadap sistem yang berjalan di Docker [11]. Observasi dilakukan dengan mengacu langsung pada sistem yang sedang berjalan, khususnya saat terjadi peningkatan traffic dari user dan adanya kesalahan pada sistem [12]. Tujuan dari observasi ini adalah untuk memahami bagaimana sistem beroperasi di bawah tekanan yang meningkat dan mengidentifikasi potensi masalah yang muncul akibat lonjakan pengguna.

2. Studi Komparasi

   Dalam penelitian ini, studi komparasi dilakukan untuk mengevaluasi kinerja dan keandalan dua solusi load balancing, yaitu NGINX dan Caddy, menggunakan algoritma *round robin*. Penelitian ini melibatkan beberapa langkah penting yang dirancang untuk memastikan perbandingan yang menyeluruh dan objektif. Pertama, lingkungan pengujian disiapkan dengan menggunakan server yang memiliki spesifikasi identik, menjalankan sistem operasi Ubuntu, dan terhubung melalui jaringan lokal yang stabil. Hal ini dilakukan untuk mengeliminasi variabel luar yang dapat mempengaruhi hasil penelitian. Selanjutnya, eksperimen dilakukan dengan membuat beberapa skenario yang bervariasi. Untuk mensimulasikan traffic pengguna dan mengukur performa, digunakan alat pengujian Azure Load Test [13]. Metrik kinerja yang diukur meliputi latensi, throughput, penggunaan CPU, memori, serta tingkat kesalahan. Data yang diperoleh dari eksperimen ini dianalisis menggunakan statistik deskriptif dan inferensial untuk menentukan apakah terdapat perbedaan yang signifikan antara NGINX dan Caddy. Implementasi dan konfigurasi kedua load balancer dilakukan dengan mengikuti dokumentasi resmi dari NGINX dan Caddy. NGINX dikonfigurasi sebagai load balancer dengan algoritma *round robin*, dan hal yang sama dilakukan untuk Caddy. Setelah konfigurasi selesai, pengujian dilakukan di lingkungan yang telah disiapkan, dan data kinerja dikumpulkan dan dianalisis. Studi komparasi ini memberikan wawasan yang mendalam tentang performa dan keandalan NGINX dan Caddy sebagai solusi load balancing dengan algoritma *round robin*. Hasil penelitian diharapkan dapat membantu pengembang sistem dan penyedia layanan dalam memilih load balancer yang paling sesuai dengan kebutuhan spesifik aplikasi mereka.

Jenis penelitian yang digunakan merupakan penelitian kualitatif, dimana data diperoleh melalui observasi serta didukung dengan penggunaan studi pustaka. Data-data yang diperoleh, nantinya akan digunakan dalam pengembangan sistem kedepan. Pengembangan sistem berarti menyusun sistem yang baru untuk menggantikan sistem yang lama secara keseluruhan atau memperbaiki sistem yang sudah ada dengan harapan sistem berjalan dengan baik sesuai dengan kebutuhan PT Andal Software.

B. Metode Analisis dan Rancangan

Metode yang digunakan dalam pengembangan penelitian ini dibagi menjadi dua yaitu:

1. Metode Analisis

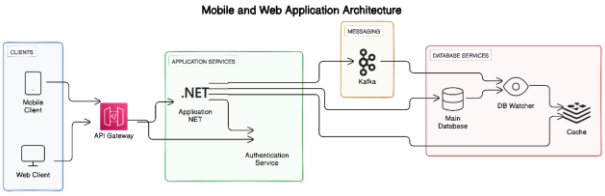
   Analisis dimulai dengan mengidentifikasi kebutuhan spesifik sistem, termasuk jumlah pengguna simultan, jenis beban kerja, serta metrik kinerja seperti waktu respons dan throughput [14]. Selanjutnya, dilakukan evaluasi terhadap berbagai algoritma load balancing, seperti *round robin*, Least Connection, dan IP Hash, untuk menentukan algoritma yang paling sesuai dengan karakteristik beban kerja sistem. Proses ini melibatkan simulasi dan pengujian menggunakan alat seperti JMeter untuk mengukur kinerja sistem dengan dan tanpa load balancer, guna menilai efektivitas setiap algoritma dalam menangani beban yang berbeda.

2. Metode Rancangan

   Rancangan load balancer melibatkan penyusunan arsitektur yang efektif, dimana load balancer ditempatkan di antara klien dan server aplikasi, baik sebagai titik masuk tunggal atau terdistribusi. Implementasi dilakukan dengan menggunakan perangkat lunak seperti NGINX atau HAProxy, yang dikonfigurasikan sesuai algoritma yang dipilih dan dilengkapi dengan mekanisme failover untuk memastikan keandalan sistem. Setelah implementasi, sistem load balancing dipantau menggunakan alat docker stats, dengan penyesuaian dilakukan berdasarkan data pemantauan untuk mengoptimalkan kinerja dan mencegah bottleneck, memastikan bahwa sistem dapat menangani peningkatan traffic dengan efisien.

III. HASIL DAN PEMBAHASAN

A. Topologi Jaringan yang sedang Berjalan



Gambar 1. *Topologi Jaringan yang sedang Berjalan*

Berikut adalah penjelasan dari topologi sistem yang berjalan berdasarkan Gambar 1:

1. Analisis dilakukan dalam upaya untuk mengetahui kelemahan yang ada pada sistem keamanan.

2. Rangkaian sistem jaringan yang berjalan pada gambar tersebut terdapat 3 lokasi dimana masing-masing *client* dan *server* terhubung dengan jaringan lokal dan jaringan internet untuk akses keluar.
3. Jaringan internet pada sistem jaringan yang berjalan diatas masih menggunakan model *client/server* tetapi jaringan lokal perusahaan ini tidak ada *dedicated server*.
4. Pada sistem yang berjalan diatas *client* tidak selalu berfungsi sebagai *client* tetapi bisa juga menjadi server, jadi pada topologi ini *client* bisa saja dijadikan sebagai server untuk penyedia layanan *client* lainnya.

B. *Masalah Yang Dihadapi*

Berdasarkan hasil observasi dan wawancara yang dilakukan, terdapat beberapa kendala utama yang dihadapi sistem saat ini, terutama ketika jumlah pengguna meningkat:

1. Sistem hanya memiliki satu *server*, yang tidak mampu menangani permintaan dari banyak pengguna secara bersamaan. Hal ini menyebabkan *server* mengalami *down* atau *overload*, sehingga pengguna tidak dapat mengakses layanan.
2. Kapasitas *server* yang terbatas juga berakibat pada performa yang lambat. Pengguna mengalami respons yang lama saat mengakses layanan, yang dapat mengganggu dan menurunkan kepuasan pengguna.

C. *Alternatif Pemecahan Masalah*

Berdasarkan hasil penelitian dan analisis, berikut adalah beberapa solusi yang direkomendasikan untuk mengatasi masalah *server* yang dihadapi saat ini:

1. Mengimplementasikan arsitektur high availability dengan menggabungkan beberapa server virtual. Hal ini akan memastikan bahwa layanan tetap tersedia bahkan jika satu server mengalami kegagalan.
2. Memasang load balancer untuk mendistribusikan beban traffic secara merata ke beberapa server back-end. Load balancer akan memilih server yang paling optimal untuk menangani setiap permintaan, sehingga meningkatkan performa dan stabilitas sistem.
3. Ketika jumlah pengguna mengalami lonjakan, memperbanyak server back-end untuk meningkatkan kapasitas pemrosesan dan penyimpanan data. Hal ini akan memungkinkan sistem untuk menangani lebih banyak permintaan secara efisien dan mencegah *server* menjadi *overload*.

D. *User Requirement (Elisitasi)*

Tabel 1. Final Draft Elisitasi

Functional	
Analisis Kebutuhan	
No.	**Keterangan**
1.	Melakukan konfigurasi di Azure
2.	Melakukan konfigurasi Docker
3.	Melakukan konfigurasi Load Balancer
4.	Melakukan instalasi Aplikasi
Non-functional	
Analisis Kebutuhan	
No.	**Saya Ingin Sistem Dapat:**
1.	Tetap berjalan ketika ada kendala
2.	Memiliki penggunaan *resource* yang efisien

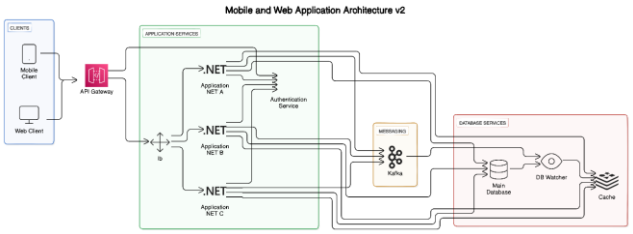
E. *Usulan Prosedur yang Baru*

Berdasarkan hasil analisa yang dilakukan terhadap sistem yang berjalan di PT Andal Software menunjukan bahwa terdapat beberapa masalah yang terjadi seperti tidak adanya mekanisme pembagian beban. Oleh karena itu dibutuhkan suatu sistem yang dapat memperbaiki masalah tersebut. Adapun sistem yang diusulkan berupa pemasangan load balancer.

F. *Rancangan Sistem*

Berikut adalah rancangan topologi jaringan yang diusulkan:

1. Tampilan Rancangan Sistem



Gambar 2. Tampilan Rancanan Sistem

Dengan *Load Balancer*

Merujuk pada Gambar 2, langkah pertama yang dilakukan adalah mengonfigurasi infrastruktur yang diperlukan. Peneliti menggunakan *virtual machine* (VM) yang disediakan oleh layanan *cloud* Azure, yang kemudian dikonfigurasi sesuai dengan spesifikasi yang dibutuhkan, mencakup *CPU*, *RAM*, dan penyimpanan yang memadai untuk menjalankan aplikasi dan *web server*. Selain itu, *firewall* juga diatur untuk memberikan akses ke Azure Load Testing, sementara koneksi *SSH* digunakan untuk akses dan konfigurasi lebih lanjut dari *VM*. Selanjutnya, dilakukan instalasi Docker untuk menyederhanakan proses pengembangan, deployment, dan pengujian aplikasi. Proses instalasi Docker mencakup pembaruan sistem, instalasi paket pendukung, penambahan kunci GPG Docker, dan repository Docker ke dalam sistem, diikuti dengan instalasi Docker itu sendiri. Setelah instalasi selesai, status Docker diverifikasi untuk memastikan bahwa semuanya berjalan dengan baik.

Aplikasi yang digunakan dalam penelitian ini adalah aplikasi Python sederhana berbasis Flask [15]. Aplikasi ini dikemas dalam sebuah Docker image melalui Dockerfile yang mendefinisikan lingkungan kerja, instalasi dependensi,

dan perintah untuk menjalankan aplikasi. Aplikasi Flask ini bertindak sebagai back-end yang akan diakses melalui NGINX dan Caddy. Sebagai penyimpanan data *caching* maka memerlukan redis [16]. Untuk pengetesan, peneliti membuat Docker Compose file yang mengatur layanan aplikasi Flask, NGINX, dan Caddy dalam satu isolated environment. Konfigurasi NGINX dan Caddy dibuat untuk mengarahkan permintaan ke aplikasi Flask, dan Docker Compose digunakan untuk membangun serta menjalankan semua layanan secara bersamaan, memastikan aplikasi siap diuji di bawah kedua web server.

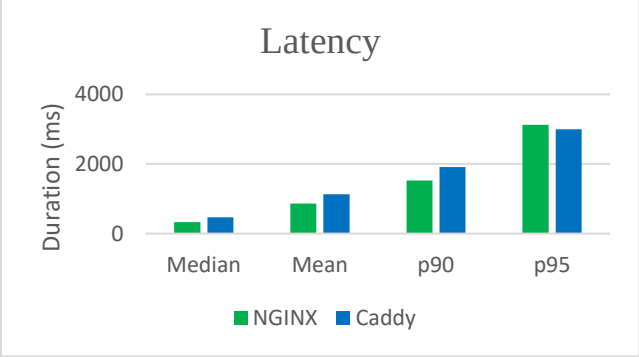
Tahap terakhir dalam penelitian ini adalah analisis performa, yang dilakukan menggunakan Azure Load Testing. Alat ini digunakan untuk mengukur berbagai metrik performa, termasuk response time rata-rata, requests per second (RPS), dan tingkat kegagalan (error rate). Pengujian dilakukan dengan mengirimkan sejumlah permintaan tertentu ke load balancer yang menjalankan NGINX dan Caddy, dan hasilnya dicatat untuk analisis lebih lanjut.

G. *Pengetesan*

Berikut adalah perbandingan hasil pengetesan antara NGINX dan Caddy dengan algoritma *round robin*:

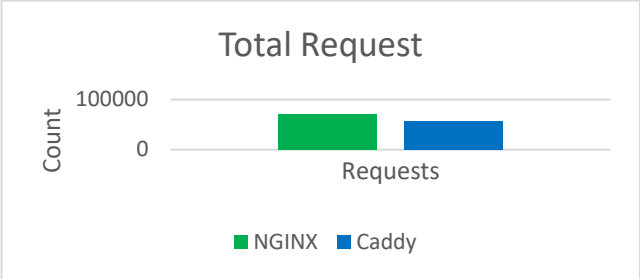
1. Skenario 1

Pada *environment* NGINX dan Caddy masing-masing terdapat satu *load balancer* dan tiga *backend container* yang semuanya berjalan.



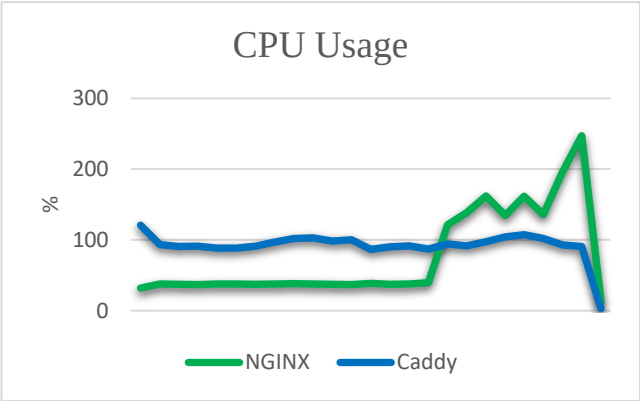
Gambar 3. Grafik *latency* skenario 1

Pada Gambar 3 menunjukkan NGINX memiliki waktu respons rata-rata yang lebih cepat dan throughput yang lebih tinggi dibandingkan dengan Caddy. Meskipun Caddy memiliki waktu respons minimum yang sedikit lebih cepat, waktu respons maksimum Caddy jauh lebih lama dibandingkan NGINX. Dengan tingkat kesalahan 0% pada NGINX dan 8% pada Caddy, penelitian ini menyimpulkan bahwa NGINX lebih andal dan efisien dalam menangani beban kerja server.



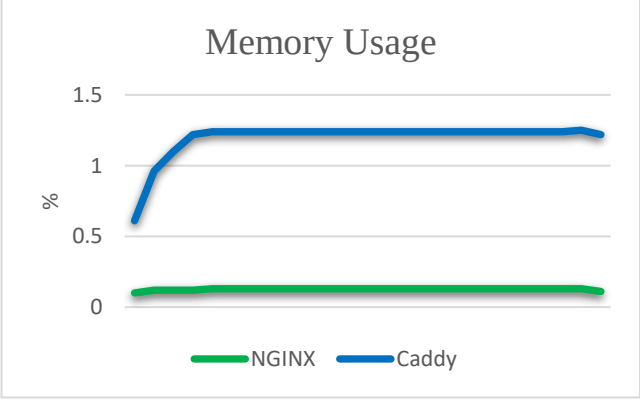
Gambar 4. Grafik total *request* skenario 1

Gambar 4 menunjukkan hasil penelitian yang membandingkan performa server NGINX dan Caddy, ditemukan bahwa NGINX secara keseluruhan memiliki kinerja yang lebih unggul. NGINX mampu menerima lebih banyak permintaan dan memproses semuanya tanpa gagal, sementara Caddy mengalami sejumlah kegagalan dalam pemrosesan permintaan.



Gambar 5. Grafik penggunaan *cpu* skenario 1

Gambar 5 menunjukkan puncak penggunaan CPU yang lebih tinggi tetapi juga memiliki nilai minimum yang lebih rendah dibandingkan Caddy. Rata-rata penggunaan CPU pada NGINX lebih rendah, menunjukkan bahwa puncak tinggi tersebut adalah nilai yang tidak sering terjadi. Penggunaan CPU pada Caddy lebih konsisten dengan rata-rata dan median yang lebih tinggi.

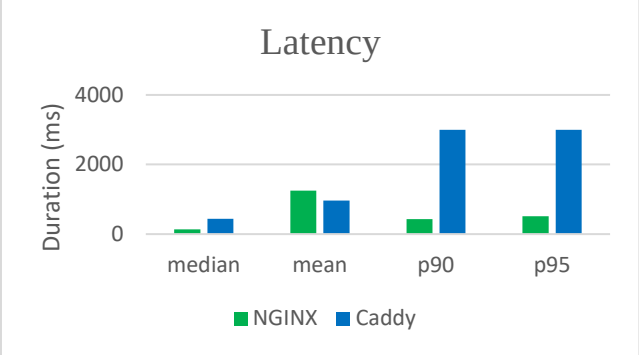


Gambar 6. Grafik penggunaan *memory* skenario 1

Gambar 6 menampilkan penggunaan memori yang jauh lebih rendah dan stabil. Caddy memiliki penggunaan memori yang lebih tinggi dan konsisten.

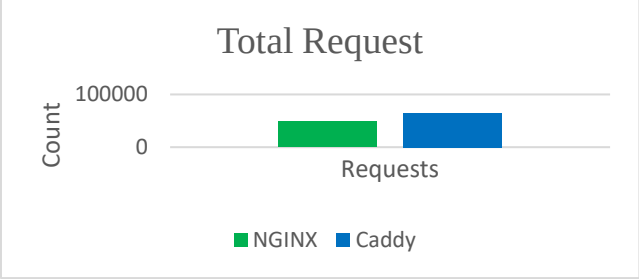
2. Skenario 2

Pada *environment* NGINX dan Caddy masing-masing terdapat satu *load balancer* dan dua *backend* yang mensimulasikan *high availability*.



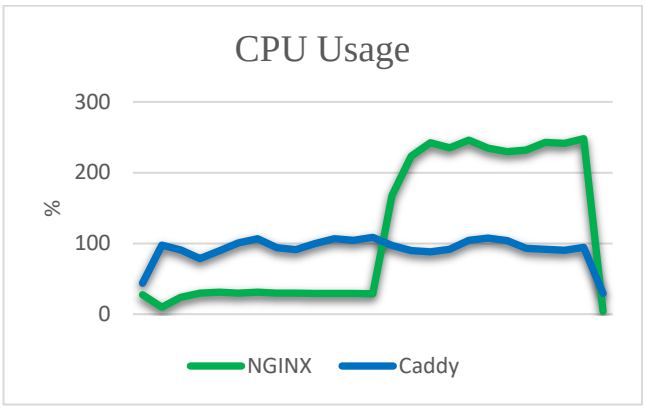
Gambar 7. Grafik *latency* skenario 2

Merujuk pada Gambar 7, meskipun NGINX memiliki waktu respons rata-rata yang lebih lambat (1.246 ms) dibandingkan dengan Caddy (963 ms), serta throughput yang lebih rendah (696,31/s berbanding 1.036,44/s), NGINX menunjukkan keandalan yang lebih baik dengan tingkat kesalahan 0% dibandingkan dengan Caddy yang memiliki tingkat kesalahan sebesar 43%. Selain itu, NGINX juga memiliki waktu respons maksimum yang lebih rendah (61,47 detik) dibandingkan Caddy (108,12 detik). Secara keseluruhan, meskipun Caddy mampu menangani lebih banyak permintaan per detik, NGINX menunjukkan stabilitas dan keandalan yang lebih baik dalam menangani permintaan tanpa kegagalan.



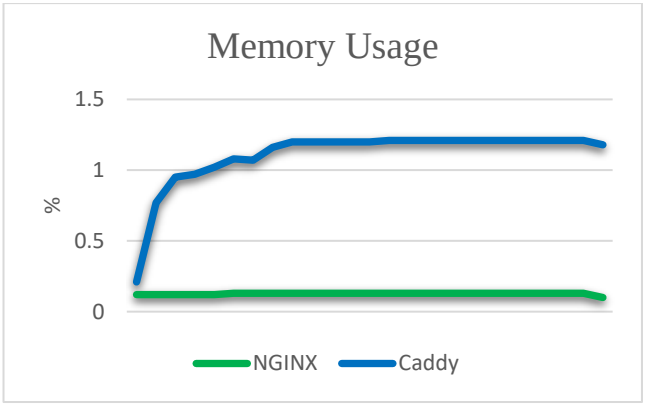
Gambar 8. Grafik total *request* skenario 2

Dalam sebuah penelitian yang membandingkan performa server NGINX dan Caddy, ditemukan bahwa kedua server memiliki kelebihan dan kekurangan masing-masing. NGINX menerima total 48.742 permintaan selama satu menit dan berhasil memproses semuanya tanpa mengalami kegagalan, sedangkan Caddy menerima lebih banyak permintaan, yaitu 64.259, namun hanya berhasil memproses 36.314 permintaan, dengan 27.945 permintaan mengalami kegagalan seperti pada Gambar 8.



Gambar 9. Grafik penggunaan *cpu* skenario 2

Grafik pada Gambar 9 menunjukkan puncak penggunaan CPU yang lebih tinggi tetapi juga memiliki nilai minimum yang lebih rendah dibandingkan Caddy. Rata-rata penggunaan CPU pada NGINX lebih rendah, menunjukkan bahwa puncak tinggi tersebut adalah nilai yang tidak sering terjadi. Penggunaan CPU pada Caddy lebih konsisten dengan rata-rata dan median yang lebih tinggi.

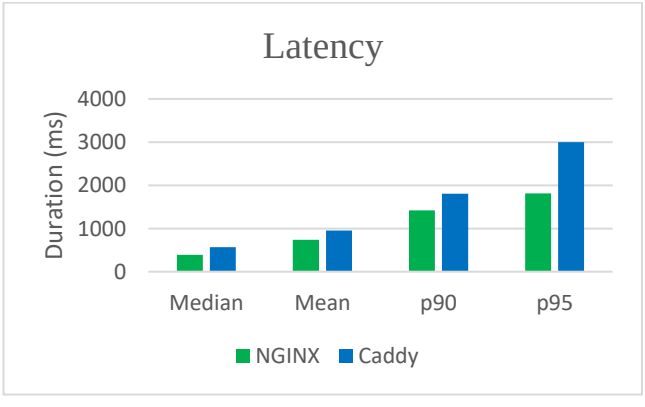


Gambar 10. Grafik penggunaan *memory* skenario 2

Gambar 10 menunjukkan penggunaan memori yang jauh lebih rendah dan stabil. Caddy memiliki penggunaan memori yang lebih tinggi dan konsisten.

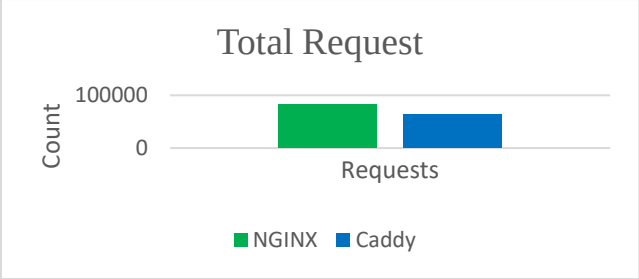
3. Skenario 3

Pada *environment* NGINX dan Caddy masing-masing terdapat satu *load balancer* dan empat *backend container* untuk mensimulasikan *scalability*, *horizontal autoscaling*.



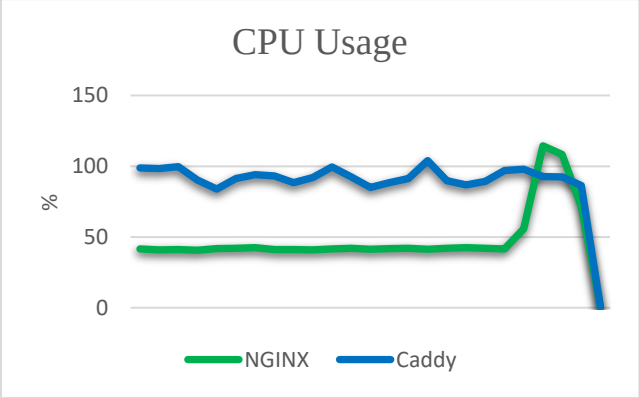
Gambar 11. Grafik *latency* skenario 3

Gambar 11 menggambarkan meskipun waktu respons rata-rata NGINX lebih cepat (740 ms) dibandingkan dengan Caddy (953 ms), Caddy memiliki waktu respons maksimum yang jauh lebih rendah (14,9 detik) dibandingkan dengan NGINX yang mencapai 85,18 detik. Namun, dalam hal throughput, NGINX memproses lebih banyak permintaan per detik (1.219,70/s) dibandingkan Caddy (956,00/s). Selain itu, tingkat kesalahan NGINX tetap 0%, menunjukkan keandalan yang lebih tinggi dibandingkan Caddy yang memiliki tingkat kesalahan sebesar 5%. Hasil ini mengindikasikan bahwa meskipun Caddy memiliki kecepatan maksimum yang lebih stabil, NGINX lebih unggul dalam hal kecepatan rata-rata, throughput, dan keandalan.



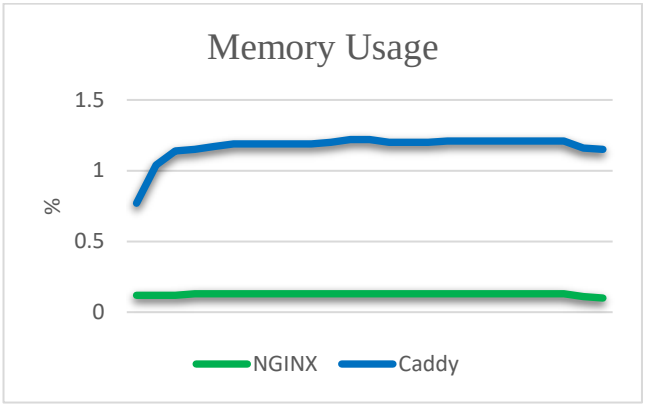
Gambar 12. Grafik total *request* skenario 3

Dalam penelitian yang membandingkan performa server NGINX dan Caddy, NGINX menunjukkan keunggulan dalam beberapa aspek penting. NGINX menerima dan berhasil memproses 81.720 permintaan dalam satu menit tanpa ada satu pun yang gagal, sementara Caddy menerima 63.096 permintaan dan memproses 59.661 di antaranya, dengan 3.435 permintaan mengalami kegagalan seperti pada Gambar 12.



Gambar 13. Grafik penggunaan *cpu* skenario 3

Grafik pada Gambar 13 menunjukkan puncak penggunaan CPU yang lebih tinggi tetapi juga memiliki nilai minimum yang lebih rendah dibandingkan Caddy. Rata-rata penggunaan CPU pada NGINX lebih rendah, menunjukkan bahwa puncak tinggi tersebut adalah nilai yang tidak sering terjadi. Penggunaan CPU pada Caddy lebih konsisten dengan rata-rata dan median yang lebih tinggi.



Gambar 14. Grafik penggunaan *memory* skenario 3

Gambar 14 menampilkan penggunaan memori yang jauh lebih rendah dan stabil. Caddy memiliki penggunaan memori yang lebih tinggi dan konsisten.

H. Pembahasan

Hasil penelitian menunjukkan bahwa baik NGINX maupun Caddy mampu mengimplementasikan algoritma *Round Robin* untuk mendistribusikan permintaan pengguna ke beberapa *backend server*. Namun, hasil pengujian pada tiga skenario menunjukkan adanya perbedaan karakteristik performa yang cukup signifikan antara kedua *load balancer*. Perbedaan tersebut mengindikasikan bahwa efektivitas implementasi *load balancing* tidak hanya ditentukan oleh algoritma distribusi beban yang digunakan, tetapi juga dipengaruhi oleh mekanisme internal masing-masing perangkat lunak dalam mengelola koneksi, memanfaatkan sumber daya sistem, serta menangani peningkatan beban kerja.

Pada skenario normal, NGINX menunjukkan performa yang lebih baik dibandingkan Caddy melalui *throughput* yang lebih tinggi, tingkat kegagalan (*error rate*) sebesar 0%, serta penggunaan memori yang lebih rendah. Kondisi tersebut menunjukkan bahwa NGINX mampu memanfaatkan sumber daya sistem secara lebih efisien ketika seluruh *backend server* berada dalam kondisi aktif. Hasil ini sejalan dengan penelitian Mulyana dkk. [8] yang melaporkan bahwa NGINX memiliki performa yang lebih stabil dibandingkan HAProxy pada lingkungan berbasis kontainer. Meskipun objek pembanding pada penelitian tersebut berbeda, kedua penelitian sama-sama menunjukkan bahwa efisiensi pengelolaan koneksi dan optimasi sumber daya merupakan faktor penting dalam meningkatkan kualitas layanan pada sistem *load balancing*.

Pada skenario *high availability*, ketika salah satu *backend server* tidak beroperasi, NGINX tetap mampu mempertahankan tingkat keberhasilan permintaan tanpa menghasilkan *error*, sedangkan Caddy mengalami peningkatan *error rate*. Temuan ini menunjukkan bahwa implementasi *Round Robin* pada NGINX lebih adaptif dalam mempertahankan kontinuitas layanan ketika terjadi gangguan pada salah satu server. Hasil tersebut mendukung penelitian Ikramsyah dkk. [17] yang menyatakan bahwa algoritma *Round Robin* mampu memberikan distribusi beban yang stabil pada lingkungan dengan spesifikasi *backend server* yang homogen. Namun, penelitian ini

memperlihatkan bahwa keberhasilan implementasi algoritma tersebut juga dipengaruhi oleh kemampuan perangkat lunak *load balancer* dalam melakukan pengelolaan koneksi ketika terjadi perubahan kondisi layanan.

Pada skenario *horizontal scalability*, penambahan jumlah *backend server* memberikan peningkatan kapasitas layanan pada kedua *load balancer*. Akan tetapi, NGINX tetap menunjukkan performa yang lebih konsisten melalui peningkatan *throughput*, penggunaan memori yang lebih efisien, dan *error rate* yang tetap berada pada angka 0%. Sebaliknya, Caddy masih mengalami kegagalan pada sebagian permintaan meskipun jumlah *backend server* telah ditingkatkan. Kondisi ini mengindikasikan bahwa kemampuan *load balancer* dalam memanfaatkan penambahan sumber daya tidak hanya bergantung pada mekanisme distribusi beban, tetapi juga pada efisiensi implementasi perangkat lunak dalam mengelola komunikasi antara *client* dan *backend server*.

Secara keseluruhan, hasil penelitian ini menunjukkan bahwa penggunaan algoritma *Round Robin* pada kedua *load balancer* mampu mendistribusikan beban kerja secara merata, namun karakteristik performa yang dihasilkan tidak selalu sama. Temuan tersebut memperkuat hasil penelitian sebelumnya yang menyatakan bahwa algoritma *Round Robin* merupakan algoritma yang sederhana dan efektif untuk lingkungan dengan spesifikasi server yang seragam [17]. Akan tetapi, penelitian ini memberikan perspektif baru bahwa efektivitas *load balancing* tidak hanya dipengaruhi oleh algoritma yang digunakan, tetapi juga oleh arsitektur dan mekanisme internal perangkat lunak *load balancer*. Dengan demikian, pemilihan *load balancer* pada lingkungan *containerized* sebaiknya mempertimbangkan tidak hanya jenis algoritma distribusi beban, tetapi juga kebutuhan performa, tingkat keandalan layanan, dan efisiensi penggunaan sumber daya.

Penelitian ini memberikan implikasi terhadap pengembangan implementasi *load balancing* pada lingkungan berbasis kontainer. Hasil penelitian mendukung temuan Mulyana dkk. [8] yang menunjukkan bahwa NGINX merupakan salah satu *load balancer* yang mampu memberikan performa tinggi pada lingkungan *containerized*. Pada penelitian ini, keunggulan tersebut kembali terlihat melalui kemampuan NGINX dalam mempertahankan *throughput* yang lebih tinggi, *error rate* yang lebih rendah, serta penggunaan memori yang lebih efisien dibandingkan Caddy pada seluruh skenario pengujian. Temuan ini memperkuat bukti empiris bahwa karakteristik implementasi perangkat lunak memiliki pengaruh terhadap kualitas layanan yang dihasilkan, meskipun algoritma distribusi beban yang digunakan sama.

Di sisi lain, hasil penelitian ini juga melengkapi penelitian Ikramsyah dkk. [17] yang berfokus pada perbandingan algoritma *Round Robin* dan *Least Connection*. Apabila penelitian sebelumnya menitikberatkan pada pengaruh algoritma terhadap kualitas layanan, penelitian ini menunjukkan bahwa performa sistem juga dipengaruhi oleh perangkat lunak *load balancer* yang mengimplementasikan algoritma tersebut. Dengan demikian, penelitian ini memperluas ruang lingkup evaluasi *load balancing* dari aspek algoritma menuju aspek implementasi perangkat lunak.

Kontribusi utama penelitian ini terletak pada penyajian evaluasi performa NGINX dan Caddy menggunakan lima parameter pengujian, yaitu *response time*, *throughput*, *error rate*, penggunaan CPU, dan penggunaan memori pada tiga skenario implementasi yang berbeda, yaitu kondisi normal, *high availability*, dan *horizontal scalability*. Dibandingkan penelitian sebelumnya yang umumnya hanya mengevaluasi satu atau dua indikator performa, pendekatan yang digunakan dalam penelitian ini memberikan gambaran yang lebih komprehensif mengenai karakteristik kedua *load balancer*. Oleh karena itu, hasil penelitian ini dapat menjadi referensi bagi organisasi maupun praktisi dalam memilih solusi *load balancing* yang sesuai dengan kebutuhan sistem berbasis kontainer, khususnya pada implementasi yang mengutamakan stabilitas layanan, efisiensi sumber daya, dan skalabilitas sistem.

IV. KESIMPULAN DAN SARAN

Penelitian ini berhasil membandingkan performa NGINX dan Caddy sebagai load balancer menggunakan algoritma Round Robin pada lingkungan microservice berbasis Docker. Berdasarkan tiga skenario pengujian, NGINX menunjukkan performa yang lebih unggul dalam aspek throughput, tingkat keberhasilan pemrosesan permintaan, efisiensi penggunaan CPU dan memori, serta stabilitas sistem dibandingkan Caddy. Temuan ini menunjukkan bahwa NGINX lebih sesuai digunakan pada lingkungan dengan beban tinggi yang memerlukan keandalan tinggi. Meskipun demikian, penelitian ini masih terbatas pada penggunaan algoritma Round Robin dan satu lingkungan cloud sehingga hasilnya belum dapat digeneralisasi pada seluruh konfigurasi infrastruktur. Penelitian selanjutnya disarankan mengevaluasi algoritma load balancing lainnya, seperti Least Connection dan IP Hash, serta melakukan pengujian pada berbagai platform cloud dan skala infrastruktur yang lebih besar.

DAFTAR PUSTAKA

[1] Kumar, R., & Singh, A. (2024). *System design on AWS: Building scalable and reliable systems*. O'Reilly Media.

[2] Jin, Y. (2018). *Designing RESTful APIs: Principles and best practices*. Packt Publishing.

[3] Gibeon Mulyana, J., Sudiarto Raharjo, W., & Indriyanta, G. (2021). Studi komparasi performa NGINX dan HAPROXY sebagai load balancer di cloud menggunakan teknologi kontainer. *Jurnal Teknologi Informasi*, *9*(2), 123-130.

[4] Hendayun, M., Ginanjar, A., & Ihsan, Y. (2023). Analysis of application performance testing using load testing and stress testing methods in *API* service. *International Journal of Computer Science and Network Security*, *23*(5), 456-462.

[5] Harjanti, T. W., Setiyani, H., & Trianto, J. (2022). Load balancing analysis using round-robin and least-connection algorithms for server service

response time. *Jurnal Sistem Informasi*, *18*(3), 301-310.

[6] Rafli, M., Fitri, I., & Andrianingsih, A. (2022). Pengujian kinerja load balancing web server menggunakan NGINX reverse proxy berbasis OS Centos 7. *Jurnal Teknologi dan Informasi*, *12*(4), 215-222.

[7] Ma, C., & Chi, Y. (2022). Evaluation test and improvement of load balancing algorithms of NGINX. *Journal of Network and Computer Applications*, *124*, 45-52.

[8] Shrivastava, R., & Srivastav, P. (2024). *Programming concepts and state management in distributed systems*. Springer.

[9] Andresson, S. (2023). *Cloud-native infrastructure: Building modern systems with containerization and microservices*. Addison-Wesley Professional.

[10] Newman, S. (2021). *Building microservices: Designing fine-grained systems*. O'Reilly Media.

[11] Matthias, K., & Kane, S. P. (2021). *Docker: Up & running*. O'Reilly Media.

[12] A. Sidik, T. Afiansyah, Z. Hakim, and D. Sofia, "Applying the SCRUM Framework in Managing the Development of Candidate Profiling Project (A Case Study of Bagidata)," *Jurnal Sisfotek Global*, vol. 13, no. 1, pp. 35-40, Mar. 2023. DOI: http://dx.doi.org/10.38101/sisfotek.v13i1.3503.

[13] Microsoft. (2024, May 21). *Azure Load Testing documentation*. https://docs.microsoft.com/azure/load-testing/

[14] R. Tullah, M. F. Helmi, and R. Setiyanto, "Comparative Analysis of Brand Equity of Internet Service Providers between Indihome and Firstmedia," *Jurnal Sisfotek Global*, vol. 13, no. 1, pp. 15-19, Mar. 2023. DOI: http://dx.doi.org/10.38101/sisfotek.v13i1.3502.

[15] Python Software Foundation. (n.d.). *Welcome to Python.org*. https://www.python.org/

[16] Redis Labs. (n.d.). *Redis documentation*. https://redis.io/

[17] M. A. Ikramsyah, H. B. Seta, I. N. Isnainiyah, and Theresiawati, "Evaluating Service Quality in NGINX Load Balancing: A Comparative Study of Round Robin and Least Connection Algorithms," *Informatik: Jurnal Ilmu Komputer*, vol. 21, no. 3, 2025.