Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.2025129452
Vol. 12, No. 2, April 2025, hlm. 429-436 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 36/E/KPT/2019 e-ISSN: 2528-6579

IMPLEMENTASI LOAD BALANCING PADA CLUSTER SERVER BERBASIS RASPBERRY PI MEMANFAATKAN ALGORITMA LEAST CONNECTION

Fauzan Ghozi Mubarak[*1], Rakhmadhany Primananda[2], Agung Setia Budi[3]

[1,2,3]Universitas Brawijaya, Malang
Email: [1]fauzanghozi56@gmail.com, [2]rakhmadhany@ub.ac.id, [3]agungsetiabudi@ub.ac.id
[*]Penulis Korespondensi

(Naskah masuk: 27 November 2024, diterima untuk diterbitkan: 13 April 2025)

Abstrak

Keterbatasan sumber daya dalam implementasi server menjadi salah satu aspek yang perlu diperhatikan. Penggunaan Raspberry Pi, yang merupakan sebuah perangkat dengan ukuran kecil dan dapat digunakan sebagai server fisik dengan konsumsi sumber daya rendah, menjadi salah satu solusi. Namun, untuk menghindari overload pada server fisik Raspberry Pi, penggunaan cluster server sebagai pemerataan beban kerja dari satu server ke server lainnya menjadi relevan. Cluster server menggunakan *Docker swarm* memungkinkan pengaturan yang lebih mudah dan efisien dari banyak node dalam sebuah cluster. Pemerataan beban kerja menggunakan load balancing dengan algoritma Least Connection. Algoritma Least Connection berjalan berdasarkan antrian paling sedikit, ketika antrian di suatu node rendah, node tersebut akan menerima beban kerja baru, sehingga mencegah overload pada satu node sementara node lainnya masih memiliki kapasitas yang tersedia. Pendistribusian image ke setiap node dalam cluster dilakukan dengan menggunakan *Ansible* untuk memastikan pengelolaan konfigurasi yang konsisten dan otomatis di seluruh node. Secara umum, penelitian ini mengintegrasikan tiga teknologi utama, yaitu Docker untuk manajemen kontainer, *Ansible* untuk manajemen konfigurasi, dan load balancing dengan algoritma Least Connection. Perancangan dan implementasi penelitian difokuskan pada penerapan dan optimalisasi masing-masing teknologi ini. Hasil pengujian penelitian menunjukkan bahwa *ansible* efektif dalam mempermudah pengelolaan konfigurasi dan otomatisasi distribusi image di seluruh node dalam kluster. Sementara itu, load balancing dengan algoritma Least Connection berhasil membagi beban kerja secara optimal berdasarkan antrian atau koneksi terendah, yang secara signifikan meningkatkan responsivitas dan efisiensi sistem secara keseluruhan.

Kata kunci: *Load Balancing, Docker, Ansible, Cluster Server , Raspberry Pi, Docker swarm*

IMPLEMENTATION OF LOAD BALANCING ON A RASPBERRY PI-BASED SERVER CLUSTER UTILIZING THE LEAST CONNECTION ALGORITHM

Abstract

Limited resources in server implementation is one aspect that needs to be considered. The use of Raspberry Pi, which is a device with a small size and can be used as a physical server with low resource consumption, is one solution. However, to avoid overloading the Raspberry Pi physical server, the use of a server cluster as an equalization of workload from one server to another becomes relevant. Cluster servers using Docker swarm allow easier and more efficient organization of many nodes in a cluster. Workload equalization uses load balancing with the Least Connection algorithm. The Least Connection algorithm runs based on the least queue, when the queue at a node is low, the node will receive a new workload, thus preventing overload on one node while other nodes still have available capacity. Image distribution to each node in the cluster is done using Ansible to ensure consistent and automated configuration management across nodes. In general, this research integrates three main technologies, namely Docker for container management, Ansible for configuration management, and load balancing with the Least Connection algorithm. The design and implementation of the research focused on the application and optimization of each of these technologies. The test results show that Ansible is effective in simplifying configuration management and automating image distribution across all nodes in the cluster. Meanwhile, load balancing with the Least Connection algorithm successfully divides the workload optimally based on the lowest queue or connection, which significantly improves overall system responsiveness and efficiency.

Keywords: *Load Balancing, Docker, Ansible, Cluster Server , Raspberry Pi, Docker swarm*

1. PENDAHULUAN

Tantangan keterbatasan sumber daya menjadi penghalang utama dalam mengembangkan infrastruktur teknologi informasi. Kondisi seperti pasokan daya listrik yang tidak konsisten, akses internet yang terbatas, dan anggaran yang terbatas menyulitkan penerapan solusi modern. Dalam

mengatasi kendala ini, pendekatan untuk membangun sistem informasi dengan keterbatasan sumber daya yaitu menggunakan Raspberry Pi.

Raspberry Pi, dengan ukurannya yang kecil dan konsumsi daya yang rendah, menjadi pilihan yang sesuai dengan keterbatasan sumber daya. Raspberry Pi dapat dijadikan komputer yang berfungsi selayaknya server (Wai Zhao, Jegatheesan, & Chee Loon, 2015). Namun saat server melambat dikarenakan overload yang membuat user tidak nyaman dalam waktu akses layanan server, sehingga permintaan dari banyak user tidak dapat ditangani secara optimal. Di sinilah potensi clustering untuk pemerataan beban dari server satu ke server lainnya menggunakan load balancing (E Rohadi, A Amalia, A Prasetyo, M F Rahmat, A Setiawan, I Siradjuddin, 2020). Raspberry Pi Cluster sebagai platform komputasi paralel menawarkan potensi untuk meningkatkan daya tanggap dan kapasitas sistem. Namun, mengelola dan mengkonfigurasi cluster secara manual dapat menjadi tugas yang rumit dan memakan waktu. Untuk mengatasi tantangan ini, penerapan manajemen konfigurasi otomatis dengan *Ansible* menjadi penting. *Ansible* memungkinkan pengelolaan konsisten dan efisien terhadap konfigurasi di seluruh node dalam kluster, memastikan bahwa perubahan yang diperlukan dapat diterapkan secara cepat dan konsisten (Redhat 2023).

Selain itu, dalam upaya untuk mendistribusikan beban kerja secara merata dan menghindari single point of failure, penerapan metode Load Balancing menjadi relevan, Berdasarkan latar belakang tersebut, penelitian ini dilakukan untuk mengembangkan solusi yang mengintegrasikan Raspberry Pi sebagai server fisik yang tidak membutuhkan daya listrik besar. Raspberry Pi akan dikonfigurasi dalam cluster menggunakan *Docker swarm*, dengan manajemen konfigurasi otomatis yang dikelola oleh *Ansible*, serta distribusi beban kerja yang efisien menggunakan algoritma Load Balancing Least Connection.

2. LITERATUR TERKAIT

2.1. Landasan penelitian terkait

Landasan penelitian ini didasarkan pada beberapa aspek penting. Pertama, keterbatasan sumber daya pada perangkat server tradisional, yang umumnya membutuhkan perangkat keras mahal dan konsumsi energi tinggi, mendorong perlunya solusi yang lebih efisien. Raspberry Pi menjadi alternatif perangkat yang layak karena ukurannya yang kecil, biaya rendah, dan konsumsi daya yang hemat, sehingga cocok untuk aplikasi yang membutuhkan efisiensi biaya dan sumber daya. Kedua, pemanfaatan Raspberry Pi sebagai server fisik telah banyak diterapkan dalam penelitian untuk mendukung aplikasi skala kecil hingga menengah, menjadikannya pilihan yang tepat untuk mengimplementasikan sistem server yang lebih terjangkau dan hemat daya. Ketiga, penerapan teknologi *Docker swarm*, yang sering digunakan dalam pengelolaan cluster kontainer, memungkinkan pemanfaatan beberapa node dalam sistem terdistribusi secara efisien. *Docker swarm* menawarkan kemudahan dalam pengaturan dan manajemen cluster, serta meningkatkan skalabilitas sistem. Teknologi ini relevan dalam konteks pengelolaan beban kerja yang lebih terdistribusi, khususnya ketika digunakan bersama dengan Raspberry Pi sebagai infrastruktur server.

2.2. Cluster Server

Cluster server adalah gabungan beberapa server yang bertanggung jawab untuk mendistribusikan beban kerja di antara server. Server yang tergabung dalam cluster disebut anggota cluster. Cluster dapat terdiri dari node atau server individual. Node merupakan sistem komputer fisik dengan alamat IP host yang berbeda, yang menjalankan satu atau lebih server aplikasi (IBM 2023) .

Server clustering bekerja dengan menggabungkan beberapa node ke dalam satu cluster untuk meningkatkan keandalan, kinerja, dan skalabilitas.

2.3. Load Balancing

Load Balancing adalah teknik yang digunakan untuk mendistribusikan beban kerja secara merata di antara beberapa node atau server dalam sebuah kluster atau jaringan komputer. Tujuan utama Load Balancing adalah untuk meningkatkan kinerja sistem dengan memastikan bahwa setiap node dalam kluster berkontribusi dalam pengolahan permintaan dengan cara yang efisien.

2.4. Algoritma Least Connection

Algoritma least connection bekerja dengan memilih server yang memiliki koneksi aktif paling sedikit untuk menangani permintaan baru. Dengan demikian, algoritma ini memastikan bahwa beban kerja tersebar secara merata di antara server, menghindari situasi di mana satu server menjadi kelebihan beban sementara yang lain masih memiliki kapasitas yang cukup. Metode ini sangat efektif dalam situasi di mana permintaan klien tidak dapat diprediksi dan bervariasi, karena secara dinamis mengalokasikan sumber daya berdasarkan keadaan nyata setiap server. Cara kerja algoritma least connection melibatkan pemeriksaan jumlah koneksi aktif pada setiap server sebelum mengarahkan permintaan baru. Ketika sebuah permintaan masuk, load balancer akan menghitung jumlah koneksi aktif yang saat ini ditangani oleh setiap server dalam cluster. Server dengan jumlah koneksi aktif paling sedikit akan dipilih untuk menangani permintaan tersebut.

2.5. *Ansible*

Ansible merupakan platform otomatisasi open-source yang digunakan untuk mengotomatiskan manajemen konfigurasi pada penelitian ini. *Ansible* dikembangkan oleh Red Hat Inc. *Ansible* menggunakan bahasa pemrograman YAML untuk menentukan tugas dan konfigurasi yang harus dijalankan pada server atau host lainnya, sehingga memungkinkan pengguna untuk melakukan *provisioning*, konfigurasi, dan *deployment* aplikasi secara cepat dan mudah. *Ansible* terdiri dari tiga komponen utama, yaitu node pengontrol (control node), node yang dikontrol (managed node), dan inventory berisi daftar node tersebut (Ansible, 2023).

2.6. Docker

Docker adalah sebuah platform software yang memungkinkan pengguna untuk membuat, menguji, dan mengimplementasikan aplikasi. Melalui Docker, perangkat lunak dikemas ke dalam unit standar yang disebut kontainer, yang mencakup semua komponen yang dibutuhkan untuk menjalankan perangkat lunak tersebut, seperti pustaka, alat sistem, kode, dan waktu proses setiap server.

2.6. *Docker swarm*

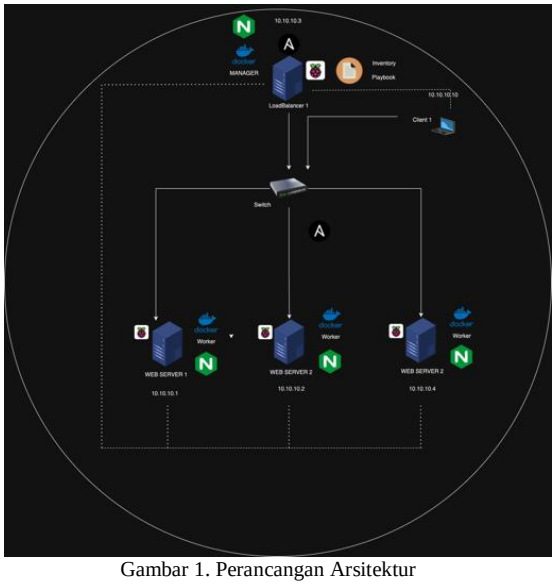
Docker swarm adalah alat orkestrasi yang memungkinkan pengelolaan dan penyebaran kontainer Docker dalam lingkungan yang terdistribusi (Turnbull, 2016). *Docker swarm* merupakan teknologi clustering untuk manajemen banyak docker host, host atau mesin mesin yang terinstall docker dan join ke dalam klaster yang sama disebut node. node sendiri terbagi menjadi dua yaitu node manager dan node worker. Node manager memiliki tanggung jawab untuk menjalankan tugas-tugas seperti orkestrasi, penyebaran layanan, dan pemeliharaan status cluster secara keseluruhan. Node worker dalam cluster bertugas menjalankan kontainer berdasarkan instruksi dari node manager. Node worker tidak memiliki hak untuk membuat keputusan terkait penjadwalan atau pengelolaan cluster, melainkan hanya mengikuti perintah yang diberikan oleh node manager. Tugas utama dari worker adalah memastikan bahwa layanan yang telah dijadwalkan oleh manajer berjalan sesuai dengan yang diharapkan.

Docker swarm bekerja dengan cara mengubah beberapa Docker host menjadi satu cluster terkoordinasi. Dalam konfigurasi ini, satu atau lebih node akan bertindak sebagai manager, sementara sisanya berperan sebagai worker. Node manager bertanggung jawab untuk mengatur dan mengelola cluster, melakukan penjadwalan layanan, dan memastikan status cluster tetap konsisten. Worker akan menjalankan tugas yang diterimanya dan melaporkan statusnya kembali ke manager.

3. METODE PENELITIAN

3.1. Perancangan Perangkat Keras

Perancangan perangkat keras menggunakan model *Server-based architecture*, di mana server berperan sebagai pusat pengelolaan dan penyedia layanan, sementara klien berperan sebagai pengguna layanan tersebut.



Gambar 1. Perancangan Arsitektur

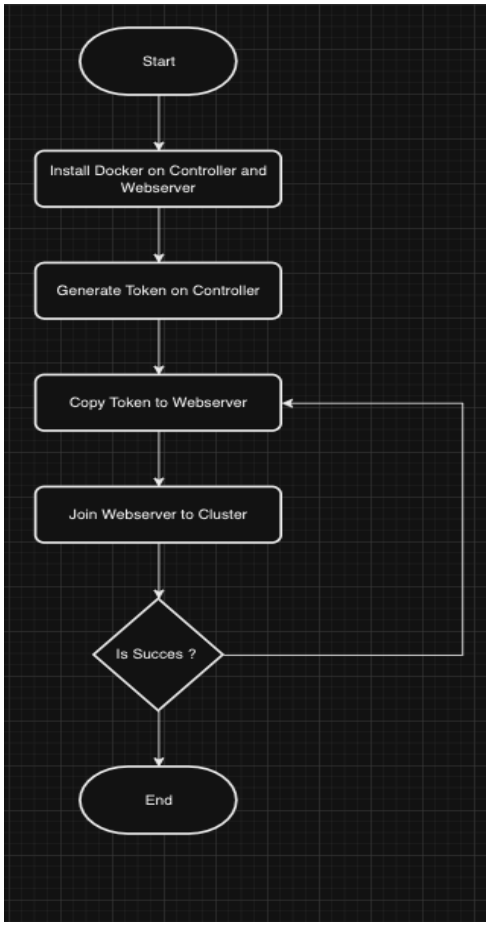
Infrastruktur yang tergambarkan diatas terdiri dari satu loadbalancer dan tiga web server yang terhubung dalam satu jaringan. Jaringan tersebut memiliki empat buah node dimana load balancer akan berperan sebagai manajer dan tiga web server berperan sebagai worker. Raspberry Pi yang bertindak sebagai web server berfungsi sebagai pusat penyedia layanan dan menerima permintaan dari klien dan memberikan respon berupa konten web. Sementara itu, Raspberry Pi yang bertindak sebagai Load Balancer berfungsi untuk mendistribusikan lalu lintas permintaan dari klien kepada tiga web server yang tersedia dengan menggunakan algoritma least connection, akan memilih server dengan koneksi aktif paling sedikit.

3.2. Perancangan Perangkat Lunak

Perancangan perangkat lunak terbagi menjadi tiga. Perancangan pertama adalah perancangan *docker swarm*. Perancangan kedua adalah perancangan *ansible*. Perancangan terakhir adalah perancangan loadbalancer

3.2.1 Perancangan *Docker swarm*

Perancangan *docker swarm* digunakan untuk untuk mengelola dan mengatur container di beberapa node.

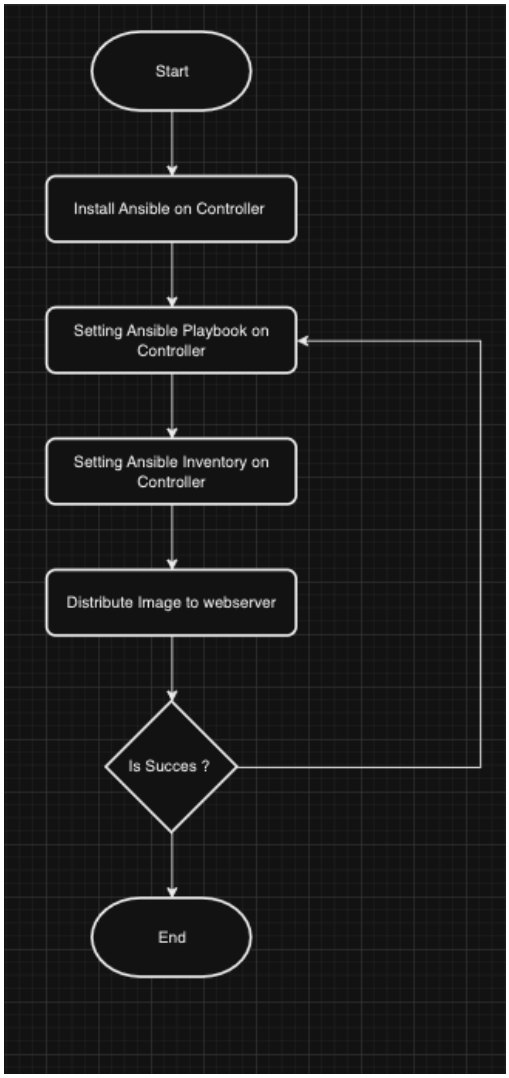


Gambar 2. Diagram Alur *Docker swarm*

Gambar 2 adalah representasi diagram alur dari alur perancangan *Docker swarm*. Proses dimulai dari instalasi Docker pada controller dan webserver. Langkah pertama adalah memastikan Docker terinstal di kedua perangkat tersebut. Setelah itu, pada controller dilakukan proses inisialisasi *Docker swarm* yang menghasilkan token khusus. Token ini kemudian disalin ke webserver yang berfungsi sebagai node slave. Webserver menggunakan token tersebut untuk bergabung dengan cluster yang dipimpin oleh controller. Setelah webserver berhasil bergabung dengan cluster, proses konfigurasi *Docker swarm* diakhiri dengan pengecekan apakah proses tersebut berhasil. Jika berhasil, maka proses konfigurasi selesai. Jika tidak, perlu dilakukan pengecekan ulang pada langkah-langkah sebelumnya untuk memastikan tidak ada kesalahan.

3.2.2 Perancangan *Ansible*

Perancangan *ansible* digunakan untuk mendistribusikan image yang sudah dibuild, image tersebut didistribusikan kepada worker atau web server.



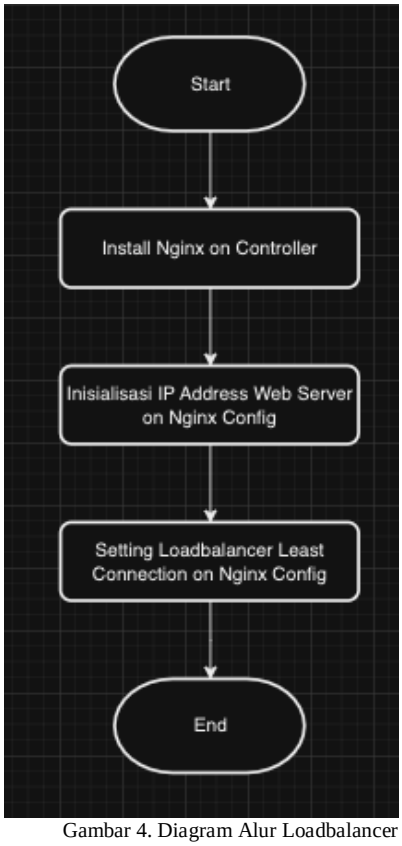
Gambar 3. Diagram Alur *Ansible*

Gambar 3 adalah representasi diagram alur dari perancangan *Ansible*. Langkah pertama yang harus dilakukan dalam menggunakan *Ansible* adalah melakukan instalasi software *ansible* itu sendiri. Setelah berhasil diinstal, langkah berikutnya adalah melakukan konfigurasi playbook. Playbook adalah sebuah file yang berisi serangkaian instruksi atau tugas yang akan dieksekusi oleh *ansible*. Playbook ini berfungsi untuk mendefinisikan langkah-langkah yang akan dijalankan oleh penerima playbook, yaitu node webserver. Selanjutnya, dalam proses perancangan *ansible*, dilakukan juga konfigurasi inventory. Inventory adalah sebuah file yang berisi daftar host atau server yang akan dikelola oleh *ansible*. Setelah semua pengaturan playbook dan inventory selesai dilakukan, proses distribusi image ke web server yang sudah diinisialisasi dalam inventory.

3.2.3 Perancangan Loadbalancer

Perancangan loadbalancer digunakan untuk mengelola permintaan client atau user. Ketika ada permintaan masuk, loadbalancer dengan algoritma

least connection akan melakukan pengecekan server mana yang memiliki jumlah koneksi terendahdan mengatur container di beberapa node.



Gambar 4. Diagram Alur Loadbalancer

Gambar 4 representasi diagram alur dari perancangan loadbalancer. Tahapan awal dalam penerapan Load Balancer adalah instalasi Nginx pada controller. Controller ini bertanggung jawab untuk mengelola dan mendistribusikan lalu lintas jaringan ke beberapa web server. Instalasi Nginx pada controller dilakukan dengan menjalankan perintah "sudo apt-get install nginx". Setelah instalasi selesai langkah selanjutnya adalah inisialisasi alamat IP dari web server yang akan menerima lalu lintas yang didistribusikan oleh load balancer. Ini dilakukan dengan mengedit file konfigurasi Nginx untuk memasukkan alamat IP dari web server. Kemudian, konfigurasi load balancer dengan algoritma "least connection" dilakukan pada file konfigurasi Nginx yang sama. Algoritma ini memastikan bahwa permintaan baru akan diarahkan ke server dengan koneksi paling sedikit.

3.3. Implementasi

Dalam implementasinya, menggunakan model Server-based architecture, di mana server berperan sebagai pusat pengelolaan dan penyedia layanan, sementara klien berperan sebagai pengguna layanan tersebut .

Gambar 5. Arsitektur Server

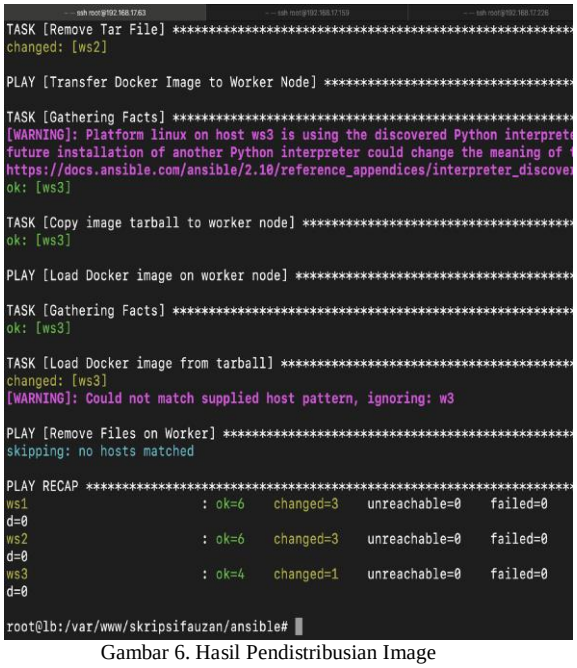
Pada Gambar 5 infrastruktur yang tergambarkan terdiri dari satu loadbalancer dan tiga web server yang terhubung dalam satu jaringan. Jaringan tersebut memiiki empat buah node dimana loadbalancer akan berperan sebagai manager dan tiga web server berperan sebagai worker. Raspberry Pi yang bertindak sebagai web server berfungsi sebagai pusat penyedia layanan dan menerima permintaan dari klien dan memberikan respon berupa konten web. Sementara itu, Raspberry Pi yang bertindak sebagai Loadbalancer berfungsi untuk mendistribusikan lalu lintas permintaan dari klien kepada tiga web server yang tersedia dengan menggunakan algoritma least connection, akan memilih server dengan koneksi aktif paling sedikit.

4. PENGUJIAN DAN ANALISIS

4.1. Pengujian Pendistribusian Image

Pengujian pendistribusian image yang dilakukan menggunakan *ansible*. Tujuan dari pengujian ini adalah untuk mengetahui apakah sistem yang dikembangkan dapat melaksanakan fungsi otomasi pendistribusian image melalui 1 node yaitu swarm manager dan melakukan load image dari file yang sudah didistribusikan.

Hasil pengujian distribusi image menunjukkan bahwa controller bertanggung jawab untuk mendistribusikan image yang telah dibuat oleh swarm manager kepada para worker dalam lingkungan swarm. Waktu yang dibutuhkan untuk proses ini bergantung pada ukuran file yang didistribusikan. *Ansible* berhasil menyalin file image berformat .tar dan memuatnya sehingga siap digunakan.



Gambar 6. Hasil Pendistribusian Image

4.2. Pengujian Clustering *Docker swarm*

Pengujian kedua dalah pengujian clustering dengan *docker swarm*. Tujuan dari pengujian ini adalah untuk mengetahui apakah *Docker swarm* dapat membuat dan mengatur cluster dari beberapa node.

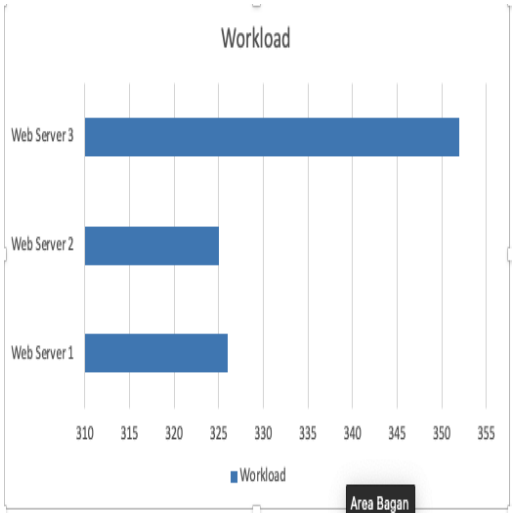
Gambar 7. Daftar Node Pada Swarm Manager

Hasil clustering dengan *docker swarm* menunjukkan bahwa dapat membuat cluster dengan controller menjadi master dan webserver menjadi slavenya.

4.3. Pengujian Pembagian Beban Kerja Least Connection

Bagian pertama dari pengujian load balancer dengan algoritma least connection adalah pengujian pembagian beban kerja. Tujuan pengujian ini adalah untuk memastikan apakah beban kerja telah terbagi sesuai dengan konfigurasi.

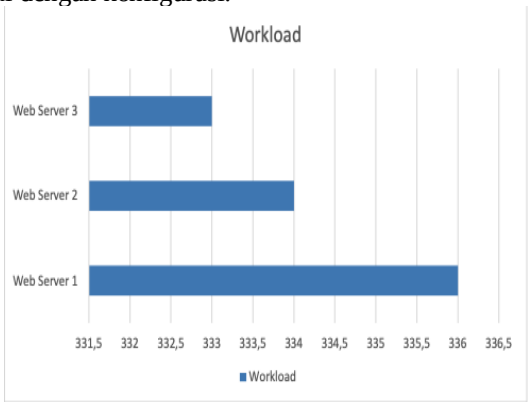
Pada ujicoba dilakukan 1000 request dengan sekali pengujian 100 secara serentak menggunakan apache benchmark. Hasil setelah melakukan apache benchmark. Presentase pembagian dari 1000 pada web server satu sekitar 32.60%, web server tiga sekitar 32.50% dan web server tiga sekitar 35.20%.



Gambar 8. Pembagian Beban Kerja

4.3. Pengujian Pembagian Beban Kerja Round Robin

Bagian pertama dari pengujian load balancer dengan algoritma least connection adalah pengujian pembagian beban kerja. Tujuan pengujian ini adalah untuk memastikan apakah beban kerja telah terbagi sesuai dengan konfigurasi.

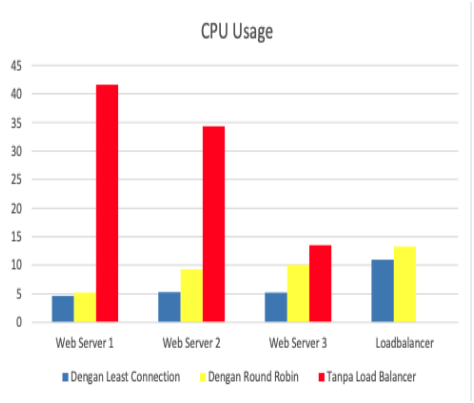


Gambar 9. Pembagian Beban Kerja Round Robin

Pada ujicoba dilakukan 1000 request dengan sekali pengujian 100 secara serentak menggunakan apache benchmark. Hasil setelah melakukan apache benchmark. Presentase pembagian dari 1000 pada web server satu sekitar 32.60%, web server tiga sekitar 32.50% dan web server tiga sekitar 35.20%.

4.4. Pengujian Benchmark Pada Node

Bagian kedua dari pengujian load balancer adalah pengujian benchmark pada node. Tujuan pengujian ini adalah untuk mengetahui perbedaan resource antar node ketika menggunakan loadbalancer dan Ketika tidak menggunakan loadbalancer.



Gambar 10. Pembagian Beban Kerja

Gambar 10 mendeskripsikan perbedaan CPU Usage node server saat menggunakan loadbalancer least connection dan round robin serta saat dengan mengakses server secara langsung tanpa menggunakan loadbalancer.

Hasil dari apache benchmark menunjukkan bahwa ketika mengakses server menggunakan loadbalancer dengan algoritma least connection, waktu respon dan throughput lebih lambat dibandingkan dengan mengakses server secara langsung. Hal ini terjadi karena penggunaan load balancer menambahkan lapisan tambahan dalam proses pengiriman permintaan ke server. Setiap permintaan harus melewati load balancer terlebih dahulu, yang kemudian memutuskan server mana yang akan menerima permintaan berdasarkan jumlah koneksi aktif.

Tabel 2. Hasil QOS

Quality of Service	Jumlah Request	Waktu Respon (mean)	Request Loss (%)	Throughput
Loadbalancer Least Connection	1000	261 ms	0%	200.89 Kbytes/sec
Loadbalancer Round Robin	1000	179.9 ms	0%	291.48 Kbytes/sec
Web Server 1	1000	101 ms	0%	519.19 Kbytes/sec
Web Server 2	1000	172.4 ms	0%	304.19 Kbytes/sec
Web Server 3	1000	156.9 ms	0%	334.23 Kbytes/sec

Tabel 2 mendeskripsikan perbedaan pengujian quality of service menggunakan loadbalancer dengan mengakses server secara langsung

5. KESIMPULAN DAN SARAN

Hasil pengujian menunjukkan bahwa mekanisme Least Connection dapat mendistribusikan beban kerja dengan efektif di antara node-node dalam kluster. Penggunaan algoritma Least Connection membuat penggunaan CPU lebih ringan. Hal ini terbukti saat pengujian benchmark, di mana penggunaan CPU algoritma Least Connection lebih rendah dibandingkan dengan algoritma Round Robin. Dalam pengujian benchmark dengan 1000 request, penggunaan CPU pada node load balancer mencapai 11%, sedangkan pada webserver satu tercatat sebesar 4.6%, webserver dua sebesar 5.3%, dan webserver tiga sebesar 5.2%. Sebaliknya, pada algoritma Round Robin, penggunaan CPU pada node load balancer mencapai 13.3%, sedangkan pada webserver satu 5.2%, webserver dua 9.3%, dan webserver tiga 10.1%. Namun, dalam hal response time, algoritma Least Connection memerlukan waktu lebih lama karena harus membaca koneksi aktif terendah. Di sisi lain, jika tidak menggunakan load balancing, terdapat lonjakan tinggi dalam penggunaan CPU. Hal ini terbukti dari hasil pengujian menggunakan Apache Benchmark.

Pendistribusian image dengan *ansible* mempermudah controller mendistribusikan image ke worker atau web server. *Ansible* mengurangi waktu dan tenaga yang dibutuhkan untuk konfigurasi manual.

Penelitian ini menggunakan *Docker swarm* untuk cluster server karena cluster yang digunakan relatif kecil. Jika ingin mengimplementasikan cluster yang lebih kompleks dan berskala besar, peneliti menyarankan penggunaan Kubernetes.

DAFTAR PUSTAKA

ALANKAR, B., SHARMA, G., KAUR, H., VALVERDE, R. & CHANG, V., 2020. Experimental setup for investigating the efficient load balancing algorithms on virtual cloud. Sensors, 2024, p.7342. Available at: https://doi.org/10.3390/s20247342 [Diakses 25 September 2023]

ROHADI, E., AMALIA, A., PRASETYO, A., RAHMAT, M.F., SETIAWAN, A. & SIRADJUDDIN, I., 2020. Cluster implementation on mini Raspberry Pi computers using Round Robin Algorithm. *Journal of Physics*, 1450. Available at: https://doi.org/10.1088/1742-6596/1450/1/012068 [Diakses 25 September 2023].

ROHADI, E., PRASETYO, A. & RAHMAT, M.F., 2019. Implementasi klaster komputer mini RaspberryPi metode load balancing menggunakan algoritma Round Robin. *Jurnal Informatika Polinema*, 5(3), p.132. Available at: https://doi.org/10.33795/jip.v5i3.248 [Diakses 25 September 2023].

SUJANA, A.P., 2019. Implementasi cluster server pada Raspberry Pi dengan menggunakan metode load balancing. *Komputika : Jurnal Sistem Komputer*, 8(1), pp.37–43. Available at: https://doi.org/10.34010/komputika.v8i1.1623 [Diakses 25 September 2023].

HOWARD. "A Complete Guide to Server Cluster | FS Community." Knowledge, 22 Mar. 2024, community.fs.com/article/a-complete-guide-to-server-clusters.html.

LEE, RICH, & BINGCHIANG JENG. "Load-Balancing Tactics in Cloud." IEEE Xplore, 1 Oct. 2011, ieeexplore.ieee.org/document/6079471. [Accessed 6 Jan. 2023].

IBM, 2024. Introduction: Clusters. [online] Available at: https://www.ibm.com/docs/en/was-nd/8.5.5?topic=servers-introduction-clusters [Diakses 1 Februari 2024].

CITRIX 2024. Least Connection Method | NetScaler 14.1. [online] docs.netscaler.com. Available at: https://docs.netscaler.com/en-us/citrix-adc/current-release/load-balancing/load-balancing-customizing-algorithms/leastconnection-method.html [Diakses 10 Maret 2024].

AWS 2023. Apa Itu Docker? | AWS. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/id/docker/.

DOCKER INC., 2023. Docker Overview | Docker Documentation. [Online] Available at: https://docs.docker.com/get-started/overview/ [Accessed 25 Maret 2024].

ANSIBLE, 2023. Getting started with *Ansible* - *Ansible* Documentation. [Online] Available at: https://docs.*ansible*.com/*ansible*/latest/ [Accessed 10 Juni 2024

DOCKER, 2019. *Swarm mode overview*. [online] Docker Documentation. Available at: https://docs.docker.com/engine/swarm/ [Accessed 10 Juni 2024].

NAIK, N. 2016. *Building a Virtual System of Systems Using Docker swarm in Multiple Clouds.* [online] IEEE Xplore. doi:https://doi.org/10.1109/SysEng.2016.7753148.

SINGH, N.K., THAKUR, S., CHAURASIYA, H. AND NAGDEV, H. 2015. *Automated Provisioning of Application in IAAS Cloud Using Ansible Configuration Management*. [online] IEEE Xplore. doi:https://doi.org/10.1109/NGCT.2015.7375087.

MA, C. AND CHI, Y. 2022. Evaluation Test and Improvement of Load Balancing Algorithms of Nginx. *IEEE Access*, 10, pp.14311–14324. doi:https://doi.org/10.1109/access.2022.3146422.

AHMED, A. AND PIERRE, G. 2020. Docker-pi: Docker Container Deployment in Fog Computing Infrastructures. *International Journal of Cloud Computing*, 9(1), p.6. doi:https://doi.org/10.1504/ijcc.2020.105885.