e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com

ANALISIS PENERAPAN AUTO SCALING DAN LOAD BALANCING PADA WEB SERVER DI LINGKUNGAN OPENSTACK

Analysis of auto scaling and load balancing implementation on web servers in an openstack environment

Bagas Styawan[1*], Husain[2], Kurniadin Abdul Latif[3], I Putu Hariyadi[4], Khairan Marzuki[5]

Program Studi S1 Ilmu Komputer[1,4,5], Program Studi S1 Teknologi Informasi[2],
Program Studi S1 Rekayasa Perangkat Lunak[3]
Fakultas Teknik[1,2,3,4,5] Universitas Bumigora[1,2,3,4,5]

*Correspondent Author: bagastyawan00@gmail.com[1]

Author Email: husain@universitasbumigora.ac.id[2], kurniadin@universitasbumigora.ac.id[3] ,
putu.hariyadi@universitasbumigora.ac.id [4], khairan.marzuki@universitasbumigora.ac.id [5]

In Indonesian

Abstrak: Peningkatan jumlah pengguna dan permintaan layanan web menuntut sistem yang mampu menyesuaikan kapasitas sumber daya secara dinamis guna menjaga kinerja dan ketersediaan layanan. Salah satu solusi adalah penggunaan load balancing yang dikombinasikan dengan penskalaan otomatis (auto scaling) pada lingkungan Openstack. Penelitian ini menganalisis kinerja sistem load balancer dengan pendekatan auto scaling pada layanan web server Apache2 di OpenStack. Metodologi yang digunakan adalah Network Development Life Cycle (NDLC) meliputi tahapan Analysis, untuk mengidentifikasi kebutuhan sistem dan sumber daya. Tahapan Desain, mencakup perancangan topologi dan skenario pengujian. Dan tahapan Simulasi, meliputi instalasi dan konfigurasi OpenStack. Pengujian dilakukan menggunakan tiga skenario: beban CPU, permintaan HTTP, dan beban memori, masing-masing lima kali percobaan. Monitoring dilakukan menggunakan layanan Gnocchi. Hasil pengujian menunjukkan sistem mampu merespons peningkatan beban secara efektif dan menjaga stabilitas layanan web. Pengujian beban CPU menghasilkan scale up rata-rata 583 detik dan scale down 213 detik. Pengujian 1.000.000 request HTTP menghasilkan scale up 543 detik dan scale

In English

Abstract: *The increasing number of users and web service requests demands a system capable of dynamically adjusting resource capacity to maintain performance and service availability. One solution is the use of load balancing combined with automatic scaling (auto scaling) in the OpenStack environment. This study analyzes the performance of a load balancer system using an auto scaling approach on the Apache2 web server in OpenStack. The methodology employed is the Network Development Life Cycle (NDLC), which includes the Analysis phase to identify system requirements and resources. The Design phase covers the design of network topology and testing scenarios, while the Simulation phase involves the installation and configuration of OpenStack. Testing was conducted using three scenarios: CPU load, HTTP requests, and memory load, each performed five times. Monitoring was carried out using Gnocchi services. The results indicate that the system effectively responds to increasing load while maintaining web service stability. CPU load testing showed an average scale-up time of 583 seconds and scale-down time of 213 seconds. HTTP request testing with 1,000,000 requests resulted in an average scale-up of 543 seconds and scale-down of 297 seconds, while memory load testing produced a scale-up of 561 seconds and scale-down of 81 seconds. In*

e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com

down 297 detik, sedangkan pengujian beban memori menghasilkan scale up 561 detik dan scale down 81 detik. Kesimpulannya, penerapan load balancing dengan auto scaling dinilai efektif, meskipun metrik Gnocchi bersifat agregatif dan belum sepenuhnya real-time.

Kata kunci: OpenStack; Load Balancing; Auto Scaling; Apache2; Gnocchi

conclusion, the implementation of load balancing with auto scaling is considered effective, although Gnocchi metrics are aggregate and do not fully represent real-time conditions..

Keywords: *OpenStack; Load Balancing; Auto Scaling; Apache2; Gnocchi*

DOI: 10.52362/jisicom.v10i1.2302

Received: *2026-03-22.* ***Revised:*** *2026-04-20.* ***Accepted:*** *2026-05-15* ***Issue Period:*** *Vol.10 No.1 (2026), Pp. 117-126*

I. PENDAHULUAN

Meningkatnya jumlah pengguna internet turut memengaruhi kompleksitas desain, implementasi, dan manajemen jaringan pada berbagai layanan berbasis web. Infrastruktur jaringan dituntut untuk mampu menampung lonjakan permintaan yang tinggi serta memberikan layanan data yang optimal dan stabil. Apabila tidak diimbangi dengan manajemen sumber daya yang baik, permintaan yang berlebih terhadap web server dapat menyebabkan penurunan performa, overload, bahkan kegagalan layanan atau server down [1].

Permintaan tinggi terhadap web server sering kali menyebabkan server tidak mampu melayani request secara optimal, yang berpotensi menimbulkan overload atau downtime [1]. Untuk mengatasi masalah ini, Load Balancing diterapkan untuk mendistribusikan beban kerja secara merata ke beberapa server, sehingga meningkatkan efisiensi dan stabilitas sistem [2]. Di lingkungan OpenStack, load balancing menjadi kunci untuk pemanfaatan sumber daya jaringan dan komputasi secara optimal. Meski demikian, kapasitas server virtual tetap terbatas, sehingga sistem kadang masih menghadapi kelebihan beban [3].

Dengan demikian, merujuk pada beberapa penelitian yang telah dilakukan, belum terdapat metode yang secara komprehensif mampu memanajemen web server ketika mengalami kelebihan kapasitas pada saat proses pembagian beban berlangsung. Ketika load balancer mendistribusikan beban ke setiap server yang memiliki kapasitas sumber daya terbatas, kinerja server dinilai belum optimal dalam melayani permintaan (request) yang masuk [4]. Oleh karena itu, diperlukan suatu pendekatan tambahan yang dapat mengoptimalkan kinerja load balancer dalam mendistribusikan beban kerja ke masing-masing server yang tersedia [5].

Berdasarkan permasalahan tersebut, penelitian ini melakukan analisis terhadap penerapan dan optimalisasi load balancing dengan pendekatan auto scaling pada web server menggunakan OpenStack. Pendekatan auto scaling diharapkan mampu menjadi solusi dalam memanajemen web server ketika kapasitas layanan mencapai batas maksimal. Auto scaling merupakan konsep pengelolaan sumber daya komputasi pada lingkungan cloud atau virtual yang memungkinkan penyesuaian kapasitas sumber daya secara otomatis sesuai dengan kebutuhan aplikasi dan variasi beban kerja [4].

Auto scaling terbagi menjadi dua jenis, yaitu vertical scaling dan horizontal scaling. Vertical scaling memungkinkan penambahan kapasitas sumber daya, seperti CPU, RAM, atau disk, pada satu instance server secara otomatis berdasarkan aturan konfigurasi tertentu [6]. Namun, dalam praktiknya, pendekatan ini memiliki keterbatasan akibat faktor kapasitas perangkat keras, biaya, serta batasan teknis lainnya. Oleh sebab itu, horizontal

e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com

scaling menjadi alternatif yang lebih efektif, karena mampu menambahkan instance server baru secara otomatis ke dalam infrastruktur jaringan tanpa perlu memodifikasi instance server yang sudah ada, sehingga meningkatkan skalabilitas sistem [6].

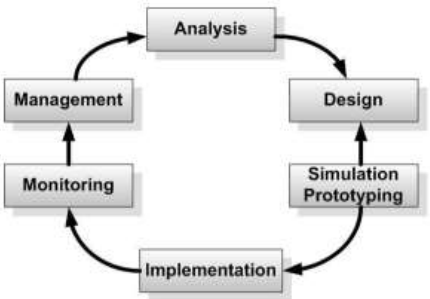
Pada lingkungan OpenStack, terdapat beberapa layanan load balancing yang dapat digunakan, di antaranya Load Balancing as a Service (LBaaS) dan Octavia, yang merupakan layanan load balancing bawaan OpenStack [7]. Load balancing Octavia dapat dikombinasikan dengan mekanisme auto scaling untuk meningkatkan kinerja sistem. Pendekatan auto scaling horizontal memungkinkan sistem menambahkan instance server baru secara otomatis ketika seluruh server mengalami kondisi overload, serta menghapus instance tersebut saat tingkat permintaan menurun. Proses ini dikendalikan oleh aturan yang telah dikonfigurasi sebelumnya menggunakan algoritma threshold-based scaling, seperti batas penggunaan CPU atau tingkat trafik yang masuk [8].

Dengan mengombinasikan metode load balancing Octavia dan threshold-based scaling pada OpenStack, diharapkan dapat tercipta infrastruktur teknologi informasi yang terukur, adaptif, dan efisien sesuai dengan kebutuhan layanan. Load balancing berperan dalam mendistribusikan lalu lintas secara merata ke server yang tersedia sehingga dapat mengurangi beban kerja pada masing-masing server [9]. Sementara itu, auto scaling memungkinkan jumlah server disesuaikan secara otomatis dengan tingkat permintaan, sehingga sistem tetap mampu melayani lonjakan beban tinggi tanpa mengalami downtime [4]. Pada penelitian ini, analisis dilakukan terhadap performa auto scaling ketika web server menerima lonjakan beban yang berlebihan, dengan parameter pengujian meliputi penggunaan CPU (CPU utilization), jumlah permintaan HTTP, dan penggunaan memori (memory usage) [10]. Hasil penelitian ini diharapkan dapat memberikan rekomendasi bagi pengembangan ilmu pengetahuan dalam manajemen infrastruktur jaringan pada lingkungan virtual.

II. METODE DAN MATERI

Pada penelitian ini digunakan metode Network Development Life Cycle (NDLC) untuk menganalisis penerapan dan optimalisasi load balancing dengan pendekatan auto scaling pada lingkungan OpenStack., NDLC merupakan metode yang digunakan dalam pengembangan jaringan komputer secara sistematis dan terstruktur. Metode NDLC terdiri atas enam tahapan, yaitu analysis, design, simulation/prototyping, implementation, monitoring, dan management [11].

Dalam penelitian ini, tidak seluruh tahapan NDLC diterapkan. Penelitian hanya memanfaatkan tiga dari enam tahapan NDLC, yaitu analysis, design, dan simulation/prototyping, karena fokus penelitian diarahkan pada perancangan, pengujian, dan analisis performa sistem tanpa melibatkan tahap implementasi penuh dan operasional jangka panjang [10],[11]. Adapun tahapan yang digunakan dalam metode NDLC pada penelitian ini dijelaskan sebagai berikut:



Gambar 1. Model NDLC

1. Tahap Analisis

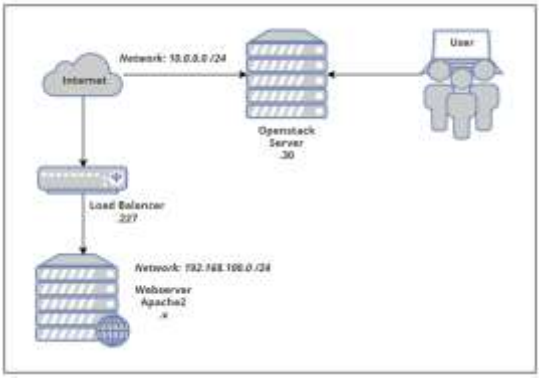
Pada tahap ini, penulis melakukan pengumpulan data dari berbagai sumber ilmiah, seperti internet, artikel ilmiah, e-book dan penelitian terdahulu yang berkaitan dengan sistem load balancer dan auto scaling pada lingkungan

e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com

OpenStack. Terdapat beberapa penelitian sejenis yang dijadikan perbandingan dengan topik yang akan diteliti, hasil analisis dari penelitian atau jurnal tersebut yang berkaitan dengan kekurangan implementasi dan penggunaan beberapa tools yang kemudian dijadikan bahan acuan dalam pengembangan penelitian ini

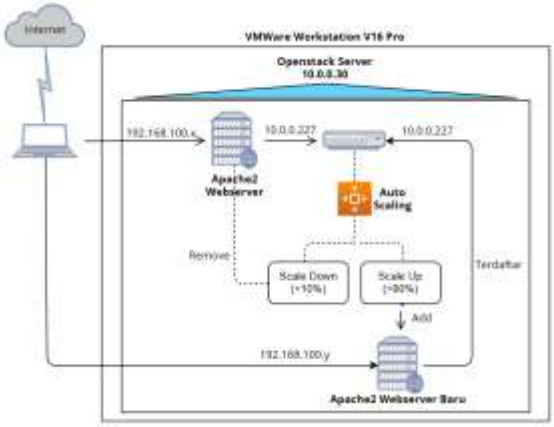
2. Tahap Desain dan Rancangan Arsitektur

Pada tahapan ini penulis akan membuat desain gambar topologi jaringan yang akan dibangun, membuat rancangan simulasi pengujian atau skenario uji yang mungkin bisa dilakukan, serta perancangan pengalamatan Ip.



Gambar 2. Topologi Jaringan Fisik

Pada gambar 2. terdapat server openstack yang terhubung ke internet dengan jaringan publik Ip address 10.0.0.30 dan layanan load balancer dengan ip address 10.0.0.227. Serta satu instance web server Apache2 dengan jaringan private ip address 192.168.100.10. Rancangan Load Balancer digunakan untuk mendistribusikan trafik ke beberapa instance backend, user dapat mengirimkan permintaan HTTP langsung ke Load Balancer. Load Balancer dikonfigurasi menggunakan Ip address 10.0.0.227 dengan ketentuan port untuk menerima request, yaitu port 80 atau 443. Load balancer mendistribusikan trafik ke web server Apache2 pada kondisi awal. Selain itu, penggunaaan monitoring Gnocchi yang diimplemetasikan dari layanan Openstack dapat membantu pengambilan keputusan auto scaling untuk menentukan penskalaan Horizontal. sistem monitoring Gnocchi akan memantau dan mengambil data kenaikan beban berdasarkan parameter CPU yang kemudian dikirimkan ke Auto Scaling Policy. Pada kondisi awal atau kondisi normal server tanpa beban, status alarm scale down menunjukkan “alarm” dan status alarm scale up yaitu “ok”. Ketentuan jumlah beban cpu yang diterima dapat mempengaruhi ambang bawah (scale down) sebesar <10% maupun ambang atas (scale up) sebesar >80%.



Gambar 3. Rancangan Alur Uji Coba Auto Scaling

Pada gambar 3. terdapat 1 (satu) buah perangkat laptop sebagai host yang terhubung ke internet. Dalam laptop di install sebuah aplikasi berbasis virtual, yaitu VMWare Workstation v16 sebagai penampung virtual mesin untuk menjalankan server Openstack. Openstack menggunakan interface br-ex sebagai jembatan ke internet. Di dalam

e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com

Openstack dikonfigurasi 1 (satu) instance web server Apache2 yang juga menggunakan layanan load balancer octavia dan auto scaling. Ketika auto scaling menerima beban melebihi 80%, maka Apache web server baru dibuat dengan mendaftarkan IP ke Load Balancer, yang kemudian membagi traffic yang masuk secara bergantian berdasarkan aturan algoritma Round-Robin. Dan pada saat beban turun ke batas sclae down <10%, instance Apache web server lama dihapus secara otomatis tanpa mempengaruhi layanan yang ada..

3. Simulasi/Prototyping

Pada tahap ini dilakukan instalasi dan konfigurasi sistem sesuai dengan kesiapan sumber daya yang tersedia. Proses ini mencakup instalasi OpenStack sebagai platform open-source cloud untuk menjalankan lingkungan virtualisasi, konfigurasi load balancing sebagai pengendali distribusi lalu lintas ke setiap instance, serta konfigurasi auto scaling untuk memastikan penambahan dan pengurangan instance dapat berjalan otomatis berdasarkan ambang batas beban tertentu. Selain itu, lingkungan monitoring disiapkan untuk mendukung proses pencatatan dan analisis pemanfaatan sumber daya sistem.

Instalasi diawali dengan konfigurasi sistem operasi Ubuntu Server 22.04 LTS pada server OpenStack, termasuk pengaturan pengalamatan IP pada setiap antarmuka jaringan guna memastikan komunikasi antar layanan OpenStack berjalan stabil. Selanjutnya dilakukan instalasi dan konfigurasi layanan inti OpenStack, seperti Neutron untuk manajemen jaringan, Nova untuk pengelolaan komputasi, Keystone untuk autentikasi dan otorisasi, serta Glance sebagai layanan manajemen image. Pada sisi instance, web server Apache2 diinstal pada sistem operasi Ubuntu Server 18.04 dengan konfigurasi sumber daya yang seragam agar proses pengujian performa dapat dilakukan secara objektif.

Konfigurasi load balancing dilakukan menggunakan layanan Octavia yang terintegrasi dengan Amphora sebagai load balancer engine untuk mendistribusikan beban ke instance web server secara merata. Selanjutnya, mekanisme auto scaling dikonfigurasikan dengan menetapkan ambang batas (threshold) beban yang memicu kondisi scale up dan scale down, berdasarkan parameter penggunaan CPU dan jumlah permintaan. Sebagai pendukung, layanan monitoring Gnocchi digunakan untuk mengumpulkan dan menganalisis metrik kinerja yang menjadi acuan dalam proses auto scaling.

Pengujian dilakukan pada lingkungan OpenStack yang telah dikonfigurasi dengan layanan load balancing dan auto scaling pada instance web server Apache2. Sebelum pengujian beban dilakukan, sistem dipastikan berada dalam kondisi normal dengan memverifikasi keterhubungan seluruh instance web server ke load balancer serta memastikan layanan dapat diakses melalui alamat IP load balancer. Monitoring awal dilakukan untuk memastikan status alarm auto scaling berada pada kondisi scale down dengan status “alarm” dan scale up dengan status “ok”, sehingga sistem siap untuk menjalankan skenario pengujian.

Pengujian beban dilakukan dengan memberikan peningkatan penggunaan CPU, permintaan HTTP, dan penggunaan memori pada instance web server. Beban CPU diuji menggunakan tools stress dengan tingkat penggunaan 80–100% selama 600 detik yang dilakukan sebanyak lima kali. Pengujian permintaan HTTP dilakukan menggunakan ab (Apache Benchmark) dengan jumlah 1.000.000 request, yang juga diuji sebanyak lima kali. Selain itu, pengujian memori dilakukan menggunakan stress-ng dengan beban memori sebesar 20% selama 600 detik. Selama pengujian berlangsung, metrik kinerja dipantau untuk mengamati perubahan status alarm scale up dan scale down. Ketika ambang batas terlampaui, sistem secara otomatis menambahkan instance baru dan mendistribusikan trafik melalui load balancer tanpa menyebabkan downtime. Setelah beban dihentikan, instance lama dihapus secara otomatis sesuai konfigurasi instance replacement, yang ditandai dengan kembalinya status alarm scale up ke kondisi “ok” dan layanan tetap dapat diakses secara normal.

III. PEMBAHASAN DAN HASIL

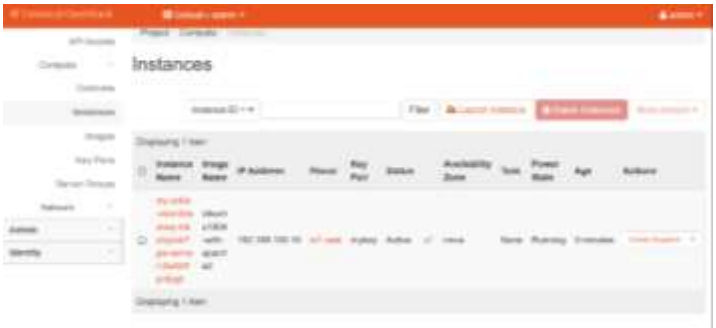
1. Persiapan instalasi dan konfigurasi

e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com

Dari metodologi yang dibahas sebelumnya, maka pada tahapan ini akan membahas simulation prototyping yang mana akan dibagi menjadi tiga bagian, diantaranya memaparkan hasil instalasi dan konfigurasi pada server Openstack untuk penerapan instance load balancing dan auto scaling, serta membuat lingkungan testing pada satu lingkungan kerja dalam openstack. Selain itu juga memuat penerapan uji coba atau testing system serta analisis hasil dari uji coba berdasarkan skenario uji coba yang dijelaskan pada bab sebelumnya.

Gambar 4. Verifikasi Instalasi Layanan Openstack

Layanan OpenStack terdiri dari beberapa komponen utama yang saling terintegrasi untuk membangun dan mengelola infrastruktur cloud, khususnya dalam konfigurasi instance, load balancing, dan auto scaling. Neutron berfungsi mengelola jaringan lokal dan publik, termasuk subnet, router, port, dan floating IP. Keystone berperan sebagai layanan autentikasi dan otorisasi untuk mengelola pengguna, proyek, peran, serta token akses pada seluruh layanan OpenStack. Nova bertanggung jawab atas manajemen komputasi, meliputi pembuatan, menjalankan, dan penghentian instance atau mesin virtual, sedangkan Glance digunakan untuk mengelola dan menyimpan image mesin virtual yang dijalankan oleh Nova. Placement digunakan oleh Nova untuk melacak dan mengalokasikan sumber daya komputasi seperti CPU, RAM, dan disk. Heat berfungsi sebagai layanan orkestrasi untuk mengotomatisasi penyediaan infrastruktur dan pengelolaan sumber daya menggunakan template. Octavia berperan sebagai layanan load balancing dalam mendistribusikan trafik ke beberapa instance. Ceilometer digunakan untuk mengumpulkan data penggunaan sumber daya, Aodh berfungsi dalam pembuatan alarm berdasarkan metrik, dan Gnocchi digunakan sebagai penyimpanan metrik performa yang mendukung proses monitoring dan pemicu alarm secara terintegrasi.

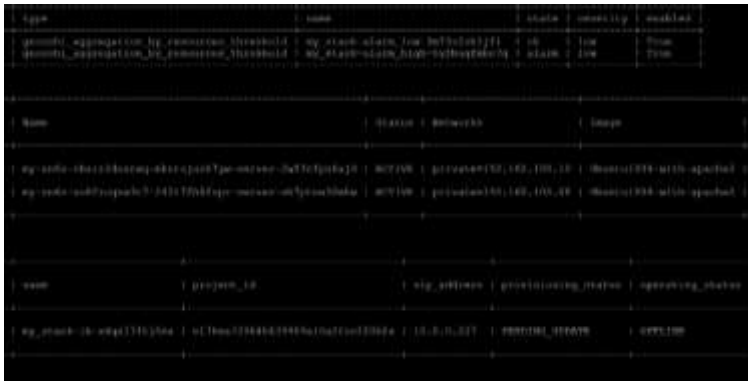


Gambar 5. Kondisi Awal Sebelum Scale Up

Pada gambar 5. terdapat instance yang berhasil dibuat menggunakan image ubuntu1804-with-apache2. Pada instance ini diinstall web server apache yang juga telah dikonfigurasi load balancing dan autoscaling, sehingga saat auto scaling aktif, maka instance baru terbentuk tanpa menghilangkan konfigurasi yang ada. Terlihat bahwa instance berhasil memperoleh alamat IP private yaitu 192.168.100.10.

2. Pengujian

e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com



Gambar 6. Kondisi Monitoring Gnocchi Saat Terjadi Scale Up

Monitoring auto scaling dilakukan untuk memastikan alarm mampu mengambil data trafik saat permintaan (request) dikirim ke instance server serta untuk memantau kinerja load balancing. Pada tampilan monitoring terlihat aktivitas auto scaling yang berjalan pada satu project, di mana tabel alarm list menampilkan dua parameter alarm, yaitu my_stack_alarm_low untuk penghapusan instance dan my_stack_alarm_high untuk penambahan instance. Status "ok" menunjukkan bahwa alarm dapat mengambil data dengan normal, sedangkan status "alarm" menandakan bahwa trafik telah mencapai ambang batas yang ditetapkan. Selain itu, tabel server list digunakan untuk memantau jumlah instance yang sedang aktif guna memastikan terjadinya penambahan atau pengurangan server sesuai kondisi beban, sementara tabel load balancer list menunjukkan bahwa load balancer berfungsi dengan baik yang ditandai dengan status server dalam kondisi aktif.

Pada gambar 6 menunjukan bahwa auto scaling berhasil dipicu saat beban trafik berhasil dijalankan. Alarm scale up menunjukan perubahan status dari "ok" menjadi "alarm", yang kemudian membuat instance web server baru dengan alamat IP 192.168.100.88. kedua instance ini kemudian terdaftar pada Load Balancer, yang berfungsi untuk membagi beban kerja secara bergantian berdasarkan aturan algoritma Round-robin yang telah ditentukan. Hasil dari masing-masing tahap pengujian dapat dilihat pada tabel berikut.

3. Analisis hasil pengujian

Pada tahapan ini membahas hasil analisis pengujian auto scaling dan load balancing pada instance web server Apache2. Analisis dilakukan terhadap tiga skenario pengujian, yaitu pengujian beban CPU (CPU Stress), pengujian permintaan HTTP (HTTP request), dan pengujian beban memori. Setiap pengujian dianalisis berdasarkan waktu yang dibutuhkan untuk memicu alarm scale up dan alarm scale down. Setiap pengujian diambil data dari periode waktu instance menerima beban (T1) dan waktu yang dibutuhkan untuk memicu alarm scale up (T2) dalam konversi detik (s).

Table 1. Hasil Pengujian Scale Up

Percobaan	Alarm Scale Up					
	Stress CPU 600s		Request HTTP 1.000.000		Memori 20%	
	T1	T2	T1	T2	T1	T2
Ke-1	600s	560s	483s	516s	600s	486s
Ke-2	600s	574s	493s	474s	600s	491s
Ke-3	600s	504s	498s	530s	600s	471s
Ke-4	600s	621s	473s	612s	600s	693s
Ke-5	600s	659s	591s	586s	600s	668s
Waktu tercepat	600s	504s	473s	474s	600s	471s
Waktu terlama	600s	659s	591s	612s	600s	693s
Rata-rata	600s	583s	507s	543s	600s	561s

e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com

Pada tabel 1 dapat dilihat bahwa waktu rata-rata yang dibutuhkan sistem untuk memicu alarm scale up pada pengujian stress CPU adalah 583 detik, dan untuk pengujian request HTTP adalah 543 detik, serta pada pengujian memori 20% adalah 561 detik. Selain itu, sistem juga mendeteksi bahwa alarm scale up dapat dipicu setelah durasi pengiriman beban selesai dilakukan, seperti yang terlihat pada percobaan ke-4 dan ke-5 untuk stress CPU, percobaan ke-1, ke-3, dan ke-4 pada uji HTTP request, dan percobaan ke-4 dan ke-5 pada uji memori 20%. Waktu tercepat, tercatat pada percobaan ke-3 pada pengujian beban memori 20% selama 471 detik. Sedangkan waktu terlama, tercatat pada percobaan ke-4 pada pengujian yang sama selama 693 detik. Meskipun terdapat sedikit perbedaan waktu, sistem masih mampu menyesuaikan kebutuhan layanan dalam rentang waktu 600 detik semenjak beban mencapai ambang batas yang ditetapkan.

Selain pengujian beban untuk memicu scale up. Pengujian ini juga memantau seberapa lama periode waktu yang dibutuhkan sistem untuk kembali ke keadaan normal atau sistem tanpa beban. Data yang ditampilkan berdasarkan tiga pengujian beban yang masing-masing dilakukan sebanyak 5 (lima) kali percobaan dengan konversi waktu dalam detik.

Table 2. Hasil Pengujian Scale Down

Percobaan	Alarm Scale Down		
	Stress CPU 600s	Request HTTP 1.000.000	Memori 20%
Ke-1	121s	290s	19s
Ke-2	270s	174s	22s
Ke-3	134s	190s	40s
Ke-4	322s	466s	177s
Ke-5	220s	365s	150s
Waktu tercepat	121s	174s	19s
Waktu terlama	322s	466s	177s
Rata-rata	213s	297s	81s

Pada tabel 2 dapat dilihat bahwa waktu rata-rata scale down aktif setelah periode pemberian beban selesai dilakukan adalah 213 detik untuk pengujian stress CPU, pada pengujian request HTTP adalah 297 detik, serta pengujian memori 20% adalah 81 detik. Selain itu, tercatat waktu scale down tercepat terjadi pada percobaan ke-1 selama 121 detik pada pengujian stress CPU, percobaan ke-2 selama 174 detik pada pengujian request HTTP, dan percobaan ke-1 selama 19 detik pada pengujian beban memori. Sedangkan waktu terlama tercatat pada percobaan ke-4 dari ketiga pengujian yang ada, masing-masing selama 322 detik pada pengujian stress CPU, selama 466 detik pada pengujian request HTTP, dan selama 177 detik pada pengujian beban memori.

Hasil dari pengujian beban scale up dan scale down, menunjukan sistem mampu secara efektif menambah dan menghapus instance server yang tidak diperlukan. Namun, variasi waktu yang dihasilkan dapat mengidentifikasikan adanya pengaruh kondisi sistem dan mekanisme evaluasi yang dihasilkan oleh sistem monitoring gnocchi, berupa interval pengumpulan data, metode agregasi (rata-rata) dan jenis beban yang diberikan untuk memicu auto scaling sebelum memutuskan penambahan maupun pengurangan resource. Dalam hal ini, pengujian beban memori cenderung lebih cepat meningkat dan menurun sehingga sistem lebih responsif, sebaliknya pengujian request HTTP membutuhkan waktu yang lebih lama untuk memproses scale down dikarenakan sistem mempertimbangkan stabilitas trafik yang berjalan.

IV. KESIMPULAN

Penelitian ini berhasil merancang dan mengimplementasikan sistem load balancing dengan pendekatan auto scaling pada lingkungan OpenStack. Sistem yang dikembangkan mengintegrasikan layanan load balancer, auto scaling, serta monitoring menggunakan Ceilometer dan Gnocchi pada web server Apache2 untuk menyesuaikan jumlah instance secara otomatis berdasarkan kondisi beban kerja. Mekanisme auto scaling berbasis instance replacement mampu berjalan sesuai dengan perancangan, di mana instance baru ditambahkan ketika alarm scale

e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com

up aktif pada kondisi beban CPU di atas 80% dengan rata-rata waktu 562 detik, sedangkan instance lama dihapus secara otomatis saat beban CPU turun di bawah 10% dengan rata-rata waktu 197 detik. Seluruh proses penskalaan tersebut berlangsung secara stabil tanpa menimbulkan gangguan terhadap layanan web yang diakses oleh klien.

Hasil pengujian menunjukkan bahwa sistem mampu merespons berbagai skenario beban secara efektif, meliputi pengujian stres CPU, permintaan HTTP dalam jumlah besar, dan beban memori. Pada pengujian stres CPU, penggunaan prosesor meningkat hingga 99% sehingga memicu proses scale up dengan rata-rata waktu 583 detik dan kembali ke kondisi normal untuk memicu scale down dalam waktu rata-rata 213 detik. Pengujian HTTP request sebanyak 1.000.000 permintaan menghasilkan peningkatan penggunaan prosesor hingga kisaran 86–89% dengan durasi scale up rata-rata 543 detik dan waktu pemulihan rata-rata 297 detik, sedangkan pada pengujian beban memori penggunaan prosesor meningkat hingga 98% dengan durasi scale up rata-rata 561 detik dan waktu scale down rata-rata 81 detik. Variasi waktu yang terjadi pada setiap pengujian menunjukkan bahwa pengambilan data metrik oleh layanan monitoring Gnocchi bersifat agregatif dengan interval tertentu, sehingga belum sepenuhnya merepresentasikan kondisi beban secara real-time. Dalam hal ini, pengujian beban memori cenderung lebih cepat meningkat dan menurun sehingga sistem lebih responsif, sebaliknya pengujian request HTTP membutuhkan waktu yang lebih lama untuk memproses beban, dikarenakan sistem mempertimbangkan stabilitas trafik yang berjalan. Oleh karena itu, pengembangan sistem selanjutnya disarankan untuk mengintegrasikan monitoring visual berbasis dashboard, seperti Grafana, guna mempermudah analisis performa sistem secara real-time dan historis serta meningkatkan akurasi pengambilan keputusan auto scaling..

REFERENASI

[1] B. Setyaji, M. Pranata, and I. Kresna, “Performance Analysis of Load Balancing Learning Management System Moodle on Docker Container,” *Media Inform.*, vol. 22, no. 1, pp. 35–43, 2023, doi: 10.37595/mediainfo.v22i1.151.

[2] A. Husein, T. B. Utomo, and T. Irfan, “Implementasi Least Bandwidth Utilization Ratio Sebagai Metode Load Balancing Pada Arsitektur Software Defined Network,” *Pros. 13th Ind. Res. Work. Natl. Semin.* , vol. 13, no. 1, Aug. 2022.

[3] B. Arifwidodo, V. Metayasha, and S. Ikhwan, “Analisis Kinerja Load Balancing pada Server Web Menggunakan Algoritma Weighted Round Robin pada Proxmox VE,” *J. Telekomun. dan Komput.*, vol. 11, no. 3, p. 210, Dec. 2021, doi: 10.22441/incomtech.v11i3.11775.

[4] A. R. Nasution, F. Dewanta, and B. Aditya, “AUTO SCALING DATABASE SERVICE WITH MICRO KUBERNETES CLUSTER,” *J. Tek. Inform.*, vol. 3, no. 4, pp. 923–927, Aug. 2022, doi: 10.20884/1.jutif.2022.3.4.484.

[5] F. M. Nurzaman, F. Chahyadi, and M. R. Rathnomi, “Analisis Perbandingan Performa Load Balancer Nginx Dan Haproxy Pada Docker,” *J. Sustain. J. Has. Penelit. dan Ind. Terap.*, vol. 11, no. 01, pp. 16–25, Jul. 2022.

[6] S. Alharthi, A. Alshamsi, A. Alseiari, and A. Alwarafy, “Auto-Scaling Techniques in Cloud Computing: Issues and Research Directions,” *Sensors*, vol. 24, no. 17, 2024, doi: 10.3390/s24175551.

[7] I. Hidayah, R. Munadi, and I. D. Irawati, “IMPLEMENTASI HIGH-AVAILABILITY WEB SERVER MENGGUNAKAN LOAD BALANCING AS A SERVICE PADA OPENSTACK CLOUD,” *e-Proceeding Eng.* , vol. 6, no. 3, p. 10278, Dec. 2019.

[8] R. Subhi, I. Ruslianto, and U. Ristian, “Implementasi Teknik Scaling Pada Sistem Manajemen Balancing Server Berbasis Website,” *Coding J. Komput. dan Apl.*, vol. 9, no. 02, p. 316, 2021, doi: 10.26418/coding.v9i02.49659.

[9] R. Pratama, A. Lubis, and S. Wahyuni, “RANCANG BANGUN SISTEM LOAD BALANCER DENGAN LAYANAN CLOUD AMAZONE WEB SERVICES,” *J. Inf. Technol. Comput. Sci.*, vol. 5, no. 2, Dec. 2022.

[10] S. Prahara, M. Martanto, and I. Ali, “Optimalisasi Jaringan Internet Dengan Optimalisasi Load Balancing

e-ISSN : 2597-3673 (Online) , p-ISSN : 2579-5201 (Printed)
Vol.10 No.1 (June 2026)
Journal of Information System, Informatics and Computing
Website/URL: http://journal.stmikjayakarta.ac.id/index.php/jisicom
Email: jisicom@stmikjayakarta.ac.id , jisicom2017@gmail.com

Menggunakan Parameter Qos," *JATI (Jurnal Mhs. Tek. Inform.*, vol. 7, no. 1, pp. 211–217, 2023, doi: 10.36040/jati.v7i1.6256.

[11] F. H. Prasetyo, E. Infitharina, and M. Febriyansyah, "Penerapan Metode Network Development Life Cycle (NDLC) dalam Pengembangan Jaringan Komputer," vol. 26861089, pp. 1–8, 2025.