JIfoTech (Journal of Information Technology)
Vol. 05, No. 02, September 2025

Desain Dan Implementasi *Autoscaling Virtual Machine (VM)* Pada *Proxmox Terintegrasi Load Balancing* Menggunakan *Ansible*

Justin Irianto[1], I Putu Hariyadi[2], Ondi Asroni[3]
[1,2,3]Ilmu Komputer, Teknik, Universitas Bumigora
[1]justinirianto@gmail.com, [2]putu.hariyadi@universitasbumigora.ac.id, [3]ondi817@gmail.com

***Abstract*— The need for efficient and adaptive infrastructure is increasingly important as the complexity of virtualization-based services increases. This research aims to design and implement a horizontal autoscaling system integrated with load balancing on Virtual Machines (VMs) on the Proxmox platform, with the goal of improving resource utilization efficiency and automatically maintaining service availability. The system utilizes Prometheus for CPU metric monitoring, while Alertmanager sends notifications to webhooks that execute Ansible playbooks to add or remove VMs according to workload needs. This research uses the Network Development Life Cycle (NDLC) method which includes three main stages: analysis, design, and prototype simulation. Testing was conducted on three VMs, where one VM acts as a webserver running Nginx, Prometheus, Node Exporter, and Alertmanager, while the other VM is used for autoscaling and load balancing processes. The autoscaling mechanism is triggered when CPU utilization exceeds 80% for scale-up processes and falls below 60% for scale-down processes, with new VMs cloned from templates and automatically added or removed from the HAProxy backend. Test results show that the system is capable of automatically scaling VMs and dynamically updating load balancer configurations, with an average response time of 2 minutes 50 seconds for scale-up and 5 minutes 13 seconds for scale-down. This implementation has been proven to improve resource efficiency while maintaining continuous service performance in an on-premises virtualization environment.**
Keyword : Autoscaling, Proxmox, Ansible, Prometheus, Load Balancing, Virtual Machine.

***Intisari*— Kebutuhan akan infrastruktur yang efisien dan adaptif semakin penting seiring meningkatnya kompleksitas layanan berbasis virtualisasi. Penelitian ini bertujuan untuk merancang dan mengimplementasikan sistem horizontal autoscaling yang terintegrasi dengan load balancing pada Virtual Machine (VM) di platform Proxmox, dengan tujuan meningkatkan efisiensi pemanfaatan sumber daya dan menjaga ketersediaan layanan secara otomatis. Sistem memanfaatkan Prometheus untuk pemantauan metrik CPU, sementara Alertmanager mengirimkan notifikasi ke webhook yang mengeksekusi playbook Ansible untuk menambah atau menghapus VM sesuai kebutuhan beban kerja. Penelitian ini menggunakan metode Network Development Life Cycle (NDLC) yang mencakup tiga tahap utama: analisis, desain, dan simulasi prototipe. Pengujian dilakukan pada tiga VM, di mana satu VM berperan sebagai webserver yang menjalankan Nginx, Prometheus, Node Exporter, dan Alertmanager, sedangkan VM lainnya digunakan untuk proses autoscaling dan load balancing. Mekanisme autoscaling dipicu saat penggunaan CPU melebihi 80% untuk proses scale up dan turun di bawah 60% untuk scale down, dengan VM baru dikloning dari template dan otomatis ditambahkan atau dihapus dari backend HAProxy. Hasil pengujian menunjukkan bahwa sistem mampu melakukan penskalaan VM secara otomatis dan memperbarui konfigurasi load balancer secara dinamis, dengan rata-rata waktu respons 2 menit 50 detik untuk scale up dan 5 menit 13 detik untuk scale down. Implementasi ini terbukti mampu meningkatkan efisiensi sumber daya sekaligus menjaga performa layanan secara berkelanjutan pada lingkungan virtualisasi lokal.**

***Kata Kunci*— Autoscaling, Proxmox, Ansible, Prometheus, Load Balancing, Virtual Machine**

I. PENDAHULUAN

Perkembangan teknologi informasi yang semakin pesat mendorong kebutuhan akan infrastruktur virtualisasi yang adaptif dan efisien untuk mendukung berbagai layanan digital [1]–[3]. Sistem berbasis virtual machine (VM) kerap menghadapi fluktuasi beban kerja yang signifikan akibat dinamika trafik pengguna atau proses komputasi intensif [4], [5]. Beban kerja yang tiba-tiba meningkat dapat menyebabkan overload yang menurunkan kinerja dan ketersediaan layanan, sedangkan beban yang menurun drastis dapat menyebabkan pemborosan sumber daya jika kapasitas yang dialokasikan tidak dikelola secara otomatis. Dalam konteks ini, mekanisme pengelolaan sumber daya yang bersifat manual menjadi kurang efisien dan rawan keterlambatan respons. Salah satu solusi yang banyak digunakan untuk meningkatkan efisiensi dan ketersediaan layanan adalah dengan menerapkan *autoscaling* dan *load balancing* [6].

Terdapat beberapa penelitian terdahulu terkait dengan *load balancing* dan *autoscaling*. Penelitian yang dilakukan oleh [7], menerapkan autoscaling berbasis Kubernetes untuk menjaga stabilitas server melalui Service Load Balancer.

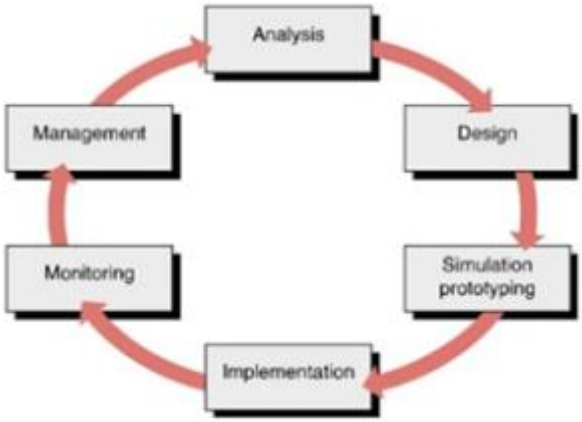
Sedangkan Penelitian yang dilakukan oleh [8], memanfaatkan fitur bawaan Google Cloud Platform (GCP) untuk load balancing dan autoscaling aplikasi web. Tak hanya itu, penelitian yang dilakukan oleh [9] menggunakan Terraform sebagai Infrastructure as Code (IaC) untuk membangun infrastruktur cloud secara otomatis. Penelitian yang dilakukan oleh [10] juga menerapkan autoscaling berbasis AWS menggunakan EC2, ALB, dan CloudWatch. Sementara itu, menurut [11] mengembangkan sistem autoscaling horizontal berbasis SDN untuk klaster VNF transparan menggunakan OpenFlow dan OpenStack.Penelitian terdahulu hanya berfokus pada *platform cloud* atau *Kubernetes* sehingga mendorong ketertarikkan peneliti untuk melakukan pendekatan yang lebih relevan untuk sistem infrastruktur lokal dengan integrasi alat yang berbeda yaitu dengan menggabungkan *autoscaling* berbasis *alert* dari *Prometheus* dan *Ansible* di lingkungan lokal menggunakan *Proxmox*.

Desain dan Implementasi *Autoscaling Virtual Machine(VM)* Pada *Proxmox* Terintegrasi *Load Balancing* menggunakan *Ansible* merupakan solusi untuk meningkatkan efisiensi dan kinerja sistem infrastruktur lokal berbasis virtualisasi. *Autoscaling* adalah proses otomatis untuk menambah atau mengurangi sumber daya VM berdasarkan kebutuhan beban kerja, sementara *load balancing* mendistribusikan trafik secara merata ke beberapa VM untuk menjaga performa. *Virtual Machine (VM)* adalah komputer virtual yang berjalan di atas server fisik, dan *Proxmox* adalah *platform* virtualisasi *open-source* yang memungkinkan pengelolaan VM. *Ansible* digunakan untuk otomatisasi konfigurasi infrastruktur melalui file *YAML* [12], [13]. Dalam sistem ini, *Prometheus* berfungsi memantau sumber daya VM webserver dan mengirimkan *alert* ke VM ansible yang akan diterima oleh *webhook* dan menjalankan file *Python* untuk memicu *Ansible playbook* dalam pembuatan dan penghapusan VM baru. Setelah VM baru dibuat, *load balancer Haproxy* menggunakan metode *Round Robin* untuk mendistribusikan trafik klien secara merata antara VM webserver dan VM baru, memastikan performa yang stabil dan efisien.

Dengan adanya sistem ini dapat meningkatkan efisiensi dan kinerja infrastruktur lokal dengan *autoscaling* otomatis dan *load balancing*, yang memastikan sumber daya cukup dan trafik terdistribusi merata. Penggunaan *Proxmox* sebagai *platform* virtualisasi *open-source* menjadikan solusi ini lebih hemat biaya, sementara *Ansible* memungkinkan otomatisasi pengelolaan infrastruktur, mengurangi risiko kesalahan dan mempercepat *deployment*.

II. METODOLOGI PENELITIAN

Penelitian ini menggunakan pendekatan Network Development Life Cycle (NDLC)[14], [15] yang terdiri dari tiga tahapan utama, yaitu analisis, perancangan, dan simulasi prototipe. Metode ini dipilih untuk memastikan pengembangan sistem autoscaling vertikal pada Virtual Machine (VM) dilakukan secara terstruktur, mulai dari identifikasi kebutuhan hingga pengujian kinerja sistem.



Gambar 1. NDLC

Gambar 1 menunujukkan tahapan pada NDLC, mulai dari tahapan analisis hingga manajemen. Namun pada penelitian ini hanya dilakukan sampai tahap simulation prototyping. Adapun penjelasan setiap tahapan sebagai berikut.

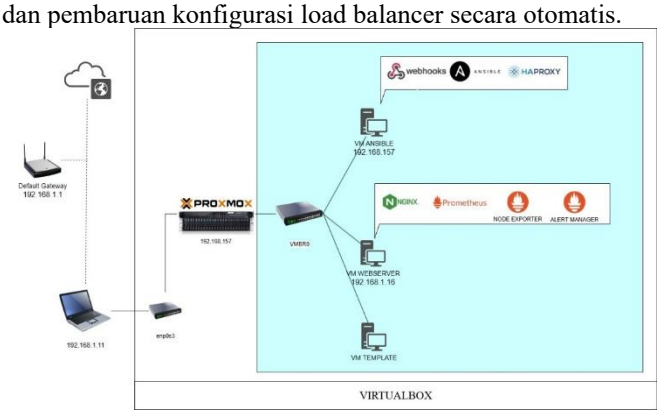
A. Tahap analisa

Pada tahap analisis, pengumpulan data dilakukan dengan metode studi literatur untuk memperoleh pemahaman yang komprehensif terkait penerapan Horizontal Autoscaling dan Load Balancing pada Virtual Machine (VM) Proxmox. Penulis menelaah artikel ilmiah, buku, e-book, paper konferensi, dan sumber daring yang membahas mekanisme autoscaling serta integrasinya dengan sistem load balancing pada lingkungan virtualisasi. Hasil penelusuran literatur menunjukkan terdapat beberapa penelitian relevan yang menjadi referensi utama dalam perancangan sistem ini, khususnya yang berkaitan dengan implementasi horizontal autoscaling pada Proxmox serta penerapan load balancing untuk mendistribusikan beban kerja VM secara efisien. Temuan-temuan tersebut menjadi dasar untuk menyusun rancangan sistem yang sesuai dengan tujuan penelitian dan kondisi lingkungan virtualisasi lokal.

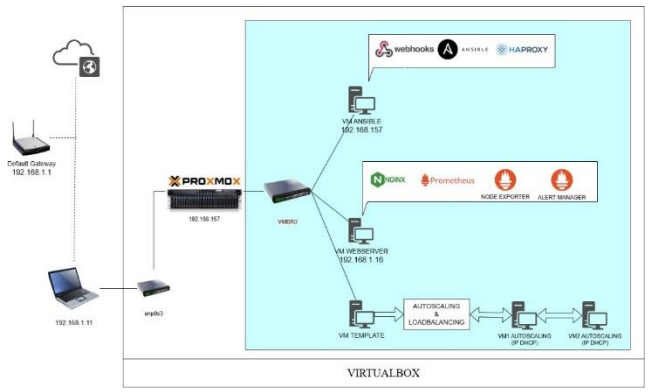
B. Tahap Perancangan

Tahap desain bertujuan untuk merancang topologi sistem dan alur kerja autoscaling horizontal yang terintegrasi dengan load balancing pada lingkungan virtualisasi Proxmox, serta mengidentifikasi kebutuhan perangkat keras dan perangkat lunak untuk mendukung proses pengujian. Rancangan jaringan uji coba dilakukan melalui virtualisasi menggunakan Proxmox yang dijalankan di atas VirtualBox pada satu komputer dengan sistem operasi Windows 11. Lingkungan pengujian terdiri dari tiga VM utama, yaitu VM Ansible, VM Webserver, dan VM Template. VM Ansible berperan sebagai pusat kontrol autoscaling yang menjalankan Ansible untuk mengeksekusi proses scale up dan scale down secara otomatis, menerima notifikasi dari Alertmanager melalui Webhook, serta menjalankan HAProxy sebagai load balancer untuk mendistribusikan trafik ke VM webserver secara merata. VM Webserver berfungsi menjalankan Nginx sebagai webserver sekaligus menjadi node pemantauan beban dengan

Prometheus dan Node Exporter, serta mengirim notifikasi melalui Alertmanager ke VM Ansible saat terjadi lonjakan atau penurunan beban CPU. Sementara itu, VM Template berperan sebagai basis pembuatan VM baru ketika proses scale up dilakukan, di mana VM ini telah dikonfigurasi dengan Nginx sehingga dapat langsung ditambahkan ke backend HAProxy untuk melayani trafik setelah dikloning oleh Ansible. Topologi jaringan uji coba ditunjukkan pada Gambar 3.1, yang menggambarkan keterhubungan antar-VM, proses pemantauan beban, pengiriman alert, eksekusi scaling, dan pembaruan konfigurasi load balancer secara otomatis.



Gambar 2. Rancangan Jaringan Ujicoba



Gambar 3. Rancangan Topologi Ujicoba

Berdasarkan Gambar 3, proses pengujian scale up dan load balancing dimulai ketika penggunaan CPU VM Webserver melebihi 80% selama satu menit. Kondisi ini dipantau oleh Prometheus melalui metrik yang dikirim oleh Node Exporter, dan ketika ambang batas terlampaui, Alertmanager mengirimkan notifikasi ke Webhook. Selanjutnya, Webhook memicu Ansible playbook untuk menjalankan proses scale up, yaitu mengkloning VM Template menjadi VM baru yang telah dikonfigurasi sebelumnya untuk menjalankan Nginx sebagai webserver tambahan. Setelah VM baru aktif dan memperoleh alamat IP, sistem secara otomatis menambahkan IP tersebut ke konfigurasi backend HAProxy, sehingga trafik pengguna dapat terdistribusi secara merata ke seluruh VM webserver melalui metode round-robin.

Sementara itu, proses scale down berlangsung ketika penggunaan CPU VM Webserver turun di bawah 60% selama lima menit. Prometheus mendeteksi kondisi ini dan Alertmanager kembali mengirimkan notifikasi ke Webhook, yang kemudian memicu Ansible playbook untuk mengeksekusi proses scale down. Pada tahap ini, sistem akan mematikan dan menghapus VM hasil autoscaling yang tidak lagi dibutuhkan, sekaligus menghapus alamat IP VM tersebut dari konfigurasi backend HAProxy. Dengan demikian, trafik hanya akan dialihkan ke VM webserver yang masih aktif. Mekanisme ini memungkinkan sistem mengelola sumber daya secara efisien dan adaptif sesuai beban layanan yang berubah secara dinamis, sekaligus menjaga ketersediaan layanan melalui integrasi autoscaling dan load balancing.

Pada saat scale up, playbook Ansible melakukan cloning VM baru dari template yang telah dikonfigurasi sebelumnya. Setelah VM baru aktif, alamat IP-nya secara otomatis ditambahkan ke backend HAProxy, sehingga trafik pengguna dapat terdistribusi merata ke seluruh VM webserver. Sebaliknya, ketika beban kerja turun di bawah ambang batas, sistem akan melakukan scale down dengan menghapus salah satu VM hasil autoscaling dan secara bersamaan memperbarui konfigurasi HAProxy untuk menghapus IP VM tersebut dari daftar backend. Seluruh proses ini berjalan otomatis dan berulang, sehingga jumlah VM selalu menyesuaikan kebutuhan beban kerja, menjaga efisiensi pemanfaatan sumber daya dan ketersediaan layanan secara dinamis.

C. Tahap Simulation Prototyping

Tahap simulation prototyping terdiri dari tiga bagian utama, yaitu instalasi dan konfigurasi perangkat pendukung, uji coba, dan analisis hasil. Uji coba sistem dilakukan melalui dua tahap pengujian, yaitu verifikasi konfigurasi dan pengujian autoscaling beserta load balancing untuk memastikan sistem dapat berjalan secara otomatis sesuai perancangan.

Proses instalasi dan konfigurasi dibagi menjadi empat tahap, mencakup Proxmox, VM Ansible, VM Webserver, dan VM Template. Pertama, Proxmox dikonfigurasi dengan melakukan instalasi pada VirtualBox, diikuti pengalokasian CPU, memori, dan penyimpanan sesuai kebutuhan pengujian. Kedua, VM Ansible disiapkan dengan instalasi paket-paket utama, meliputi Ansible sebagai alat otomasi, HAProxy sebagai load balancer, dan Webhook untuk menerima alert dari Alertmanager dan memicu playbook Ansible. Ketiga, VM Webserver dikonfigurasi sebagai node pemantauan dan penyedia layanan, yang menjalankan Prometheus untuk monitoring, Node Exporter untuk penyedia metrik sistem, Alertmanager untuk manajemen alert, dan Nginx sebagai webserver uji coba. Terakhir, VM Template disiapkan dengan Nginx agar dapat langsung dikloning oleh Ansible ketika proses scale up dilakukan.

Skenario pengujian autoscaling horizontal dan load balancing dilakukan dalam satu siklus percobaan, dengan kondisi awal hanya terdapat satu VM webserver aktif. Tahap

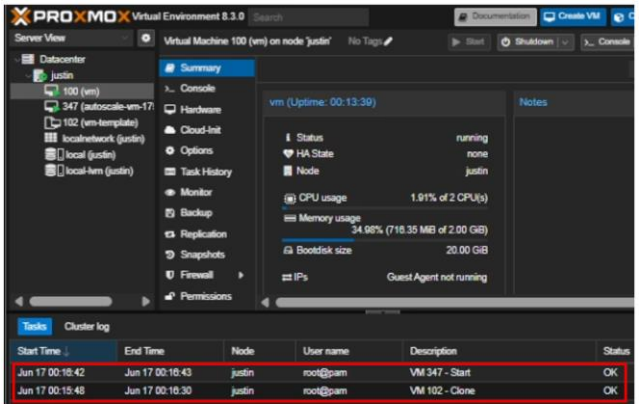
scale up dimulai dengan melakukan stress test CPU hingga penggunaan mencapai lebih dari 80% selama satu menit, memicu proses cloning VM kedua dari template yang telah disiapkan. VM baru tersebut otomatis ditambahkan ke backend HAProxy, sehingga permintaan pengguna mulai terbagi ke dua VM secara round-robin. Proses stress test dilanjutkan hingga sistem kembali melakukan scale up untuk menambahkan VM ketiga, menghasilkan tiga VM webserver aktif yang seluruhnya terdaftar dalam load balancer.

Tahap scale down dimulai ketika beban CPU turun di bawah 60% selama lima menit. Sistem secara otomatis menghapus satu VM hasil autoscaling dan memperbarui konfigurasi HAProxy untuk menghapus entri IP VM tersebut. Jika kondisi beban rendah berlanjut, sistem akan menonaktifkan VM kedua, meninggalkan satu VM aktif seperti kondisi awal. Seluruh proses scale up dan scale down ini berjalan secara otomatis berdasarkan metrik yang dikumpulkan oleh Node Exporter, dianalisis oleh Prometheus, dan dieksekusi melalui playbook Ansible. HAProxy memperbarui daftar backend server secara dinamis untuk memastikan distribusi trafik tetap seimbang tanpa mengganggu layanan. Satu siklus pengujian ini bertujuan untuk memverifikasi keandalan sistem dalam menyesuaikan jumlah VM terhadap beban kerja, sekaligus memastikan proses integrasi dan penghapusan VM dari load balancer berlangsung lancar tanpa downtime layanan.

III. HASIL DAN PEMBAHASAN

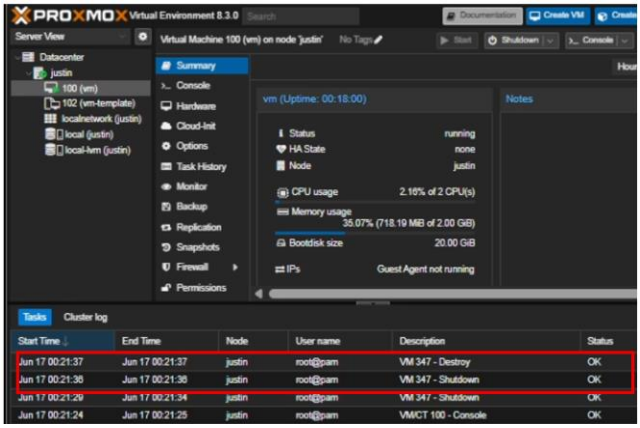
Setelah proses instalasi dan konfigurasi selesai, tahap selanjutnya adalah melakukan pengujian sistem. Pengujian dilakukan melalui dua pendekatan utama yaitu verifikasi konfigurasi masing-masing komponen, dan pengujian melalui skenario. Pengujian sistem dilakukan untuk mengevaluasi efektivitas mekanisme vertical autoscaling dalam menyesuaikan alokasi sumber daya VM secara otomatis berdasarkan beban kerja. Pengujian dibagi ke dalam dua skenario utama, yaitu skenario scale up dan load balancing, serta scale down. Masing-masing skenario dirancang untuk mengamati respons sistem terhadap perubahan kondisi beban serta efektivitas sistem dalam menjaga efisiensi sumber daya dan stabilitas layanan.

Pada skenario pertama, yaitu scale up dan load balancing, sistem diuji dengan menambahkan beban secara bertahap pada VM menggunakan uji stres. Tujuannya adalah untuk mensimulasikan lonjakan trafik atau penggunaan layanan yang tinggi, seperti yang umum terjadi pada aplikasi web atau sistem berbasis layanan daring. Ketika penggunaan CPU atau memori melebihi ambang batas yang telah ditentukan, yaitu 80% selama 1 menit, Prometheus mendeteksi lonjakan tersebut dan mengaktifkan alert melalui aturan yang telah dikonfigurasikan sebelumnya. Alertmanager kemudian meneruskan sinyal tersebut ke endpoint webhook yang akan memicu eksekusi playbook oleh Ansible. Berdasarkan playbook tersebut, sistem secara otomatis menambahkan jumlah core CPU atau kapasitas memori pada VM. Hasil pengujian menunjukkan bahwa sistem mampu meningkatkan CPU dari 2 core menjadi 4 core, lalu 6 core, sesuai dengan lonjakan beban. Sementara itu, memori juga meningkat dari 1 GB menjadi 2 GB dalam dua tahap. Waktu rata-rata respon sistem dari deteksi beban hingga perubahan aktif tercatat antara 10 hingga 15 detik. Meskipun tidak dilakukan penambahan VM seperti dalam horizontal scaling, peningkatan kapasitas VM ini berperan dalam mendistribusikan beban kerja secara lebih merata dalam satu instansi VM, yang dalam konteks ini dapat disebut sebagai bentuk load balancing internal.



Gambar 4. VM Baru Terbuat

Skenario kedua adalah scale down, yang bertujuan untuk menguji kemampuan sistem dalam mengurangi alokasi sumber daya saat beban kerja menurun. Setelah proses uji stres dihentikan dan beban kerja mulai stabil di bawah 70% selama periode waktu minimal 5 menit, sistem secara otomatis mendeteksi penurunan beban tersebut. Prometheus kembali mengaktifkan alert berdasarkan ambang batas yang telah ditetapkan, dan Alertmanager mengirimkan sinyal ke webhook. Ansible kemudian menjalankan playbook untuk mengurangi alokasi CPU dan memori secara bertahap. Berdasarkan pengujian, sistem mampu menurunkan CPU dari 6 core ke 4 core, dan akhirnya kembali ke konfigurasi awal yaitu 2 core. Memori juga diturunkan dari 2 GB ke 1.5 GB, lalu ke 1 GB. Tidak ditemukan gejala overreaction atau pengurangan sumber daya yang terlalu cepat, berkat penerapan delay selama 5 menit sebelum eksekusi scale down, yang memastikan bahwa penurunan beban memang stabil dan bukan bersifat sementara.



Gambar 5. VM Scale Up Terhapus

Kedua skenario ini membuktikan bahwa integrasi Proxmox, Prometheus, dan Ansible dapat berjalan sinergis dalam menerapkan mekanisme vertical autoscaling yang adaptif, responsif, dan efisien. Sistem ini mampu secara otomatis menyesuaikan alokasi sumber daya tanpa perlu campur tangan pengguna, sekaligus menjaga ketersediaan dan performa layanan dalam kondisi beban tinggi maupun rendah. Dari sisi efisiensi, pendekatan ini terbukti dapat mengurangi pemborosan sumber daya (overprovisioning) karena VM hanya menggunakan kapasitas yang dibutuhkan sesuai beban kerja yang sedang berjalan. Hasil pengujian juga menunjukkan bahwa waktu respon sistem cukup cepat, dengan rata-rata waktu eksekusi scaling di bawah 15 detik, baik untuk proses penambahan maupun pengurangan resource. Dengan demikian, sistem yang dibangun mampu menjadi solusi alternatif yang hemat biaya dan dapat diandalkan untuk pengelolaan VM secara otomatis di lingkungan virtualisasi berbasis open-source seperti Proxmox.

Berdasarkan uji coba yang telah dilakukan, diperoleh hasil analisis terkait waktu yang dibutuhkan oleh VM dalam menjalankan proses *autoscaling*. Pengujian dilakukan sebanyak lima kali untuk skenario *autoscaling* pada CPU. Data hasil pengujian dicatat dalam satuan detik dan disajikan pada Tabel 1. berikut:

TABEL I
HASIL ANALISA UJI COBA AUTOSCALING SCALE UP

Percobaan	**SCALE UP**					
	JUMLAH VM					
	1 ke 2			**2 ke 3**		
	T1	**T2**	**T3**	**T1**	**T2**	**T3**
Ke- 1	6	60	103	6	60	104
Ke- 2	5	60	105	4	60	106
Ke- 3	6	60	107	6	60	103
Ke- 4	4	60	105	5	60	105
Ke- 5	6	60	104	5	60	105
…….	…..	…..	…..	….		
Ke- 27	5	60	107	4	60	106
Ke- 28	6	60	106	6	60	104
Ke- 29	4	60	106	5	60	106
K3- 30	6	60	103	5	60	103
Terlama	6	60	107	6	60	106
tercepat	4	60	103	4	60	103
Rata-rata	5,4	60	104,8	5,2	60	104,6

Dari tabel di atas dapat dilihat bahwa waktu rata-rata sistem untuk memproses penambahan dari 1 ke 2 VM adalah 104,8 detik, dan untuk penambahan dari 2 ke 3 VM adalah 104,6 detik. Nilai ini mengindikasikan bahwa sistem memiliki waktu respon yang relatif stabil dalam memproses setiap tahap autoscaling. Waktu tercepat tercatat pada percobaan keempat, dengan T1 hanya 4 detik dan T3 selesai pada detik ke-103. Sedangkan waktu terlama terjadi pada percobaan ketiga, dengan T3 mencapai 107 detik. Meskipun terdapat sedikit variasi antar percobaan, sistem secara konsisten mampu menyesuaikan kapasitas layanan dalam rentang waktu sekitar 100 detik sejak beban mencapai ambang batas.

Hasil ini menunjukkan bahwa mekanisme autoscaling berjalan dengan baik dan konsisten dalam setiap siklus pengujian. Penambahan jumlah VM sebagai respons terhadap lonjakan beban dapat terjadi dalam waktu yang masih dapat diterima secara operasional, terutama dalam konteks kebutuhan sistem layanan yang bersifat dinamis. Dengan proses otomatis dan terukur ini, sistem dapat menjaga ketersediaan layanan tanpa perlu intervensi manual yang memakan waktu.

Selain pengujian scale up, sistem juga diuji untuk skenario scale down, yaitu penurunan jumlah VM saat beban kerja menurun. Skenario ini mencerminkan kondisi di mana layanan tidak lagi membutuhkan kapasitas tinggi, sehingga sistem perlu secara otomatis mengurangi alokasi sumber daya guna menghindari pemborosan. Pengujian dilakukan dalam dua tahapan: dari 3 ke 2 VM, dan dari 2 ke 1 VM. Pada setiap tahap, waktu dievaluasi dalam dua titik: T2 (masa stabilisasi beban idle selama 5 menit/300 detik) dan T3 (waktu eksekusi scaling selesai setelah threshold terpenuhi).

TABEL 2
HASIL ANALISA UJI COBA AUTOSCALING SCALE DOWN

Percobaan	SCALE DOWN			
	JUMLAH VM			
	3 ke 2		2 ke 1	
	T2	T3	T2	T3
Ke- 1	300	13	300	12
Ke- 2	300	12	300	14
Ke- 3	300	13	300	13
Ke- 4	300	14	300	13
Ke- 5	300	13	300	13
…….	…..	…..	…..	….
Ke- 27	300	13	300	13
Ke- 28	300	12	300	12
Ke- 29	300	13	300	14
Ke- 30	300	14	300	13
Terlama	300	14	300	14
Tercepat	300	12	300	12
Rata-rata	300	13	300	13

Dari hasil pengujian, terlihat bahwa waktu eksekusi scaling dari 3 ke 2 VM berkisar antara 12 hingga 14 detik, dengan rata-rata 13 detik setelah masa idle (T2) tercapai. Begitu juga untuk skenario 2 ke 1 VM, sistem menunjukkan waktu eksekusi yang sama, dengan rata-rata 13 detik. Hal ini menunjukkan bahwa sistem memiliki responsivitas yang konsisten dan stabil dalam proses scale down, setelah beban rendah terdeteksi secara berkelanjutan selama 300 detik (5 menit).

Konsistensi waktu eksekusi dalam skenario scale down menunjukkan bahwa sistem mampu mempertahankan efisiensi operasional dengan mengurangi instansi VM secara tepat waktu, tanpa keterlambatan yang signifikan. Selain itu, implementasi waktu tunda selama 5 menit sebelum eksekusi scaling dilakukan juga terbukti efektif dalam mencegah false scaling yang dapat terjadi akibat fluktuasi beban jangka pendek. Dengan demikian, mekanisme scale down ini mendukung efisiensi sumber daya dan stabilitas layanan dalam jangka panjang, terutama pada sistem dengan kebutuhan beban kerja yang dinamis.

Secara keseluruhan, sistem vertical autoscaling yang dikembangkan dengan integrasi Proxmox, Prometheus, dan Ansible mampu merespons dinamika beban kerja secara adaptif dan efisien. Pada skenario scale up, sistem dapat meningkatkan sumber daya secara otomatis saat beban meningkat, sedangkan pada skenario scale down, sistem dapat menurunkan alokasi sumber daya saat beban menurun, dengan waktu respon yang stabil dan konsisten. Rata-rata waktu eksekusi scaling berkisar antara 11 hingga 14 detik, baik untuk proses peningkatan maupun pengurangan VM. Hasil ini menunjukkan bahwa sistem yang dibangun telah berhasil mencapai tujuan utama penelitian, yaitu meningkatkan efisiensi penggunaan sumber daya dan menjaga ketersediaan layanan secara otomatis dalam lingkungan virtualisasi.

IV. KESIMPULAN DAN SARAN

Berdasarkan hasil implementasi dan pengujian yang dilakukan, dapat disimpulkan bahwa sistem horizontal autoscaling dan load balancing berhasil diterapkan secara efektif pada platform virtualisasi Proxmox dengan memanfaatkan komponen-komponen open-source seperti Ansible untuk otomasi, Prometheus untuk pemantauan metrik, Alertmanager sebagai pengelola notifikasi, Node Exporter untuk monitoring sumber daya VM, webhook sebagai pemicu eksekusi playbook, serta HAProxy sebagai load balancer. Sistem ini mampu melakukan penskalaan otomatis VM berdasarkan beban CPU secara real-time, dengan pembuatan instansi baru saat penggunaan CPU melewati 80% lebih dari satu menit, dan penghapusan VM saat penggunaan turun di bawah 60% selama lima menit. Dalam pengujian menggunakan layanan webserver berbasis Linux sebagai target, sistem menunjukkan kemampuan adaptif dalam mengalihkan trafik dan mengelola beban kerja secara otomatis, dengan waktu penskalaan ke atas rata-rata 2 menit 50 detik dan penskalaan ke bawah rata-rata 5 menit 13 detik. Meskipun demikian, terdapat keterbatasan pada proses provisioning VM yang masih bergantung pada kecepatan cloning template, yang dapat menyebabkan jeda sebelum layanan siap digunakan. Oleh karena itu, optimalisasi pada proses cloning serta integrasi metode caching atau VM pre-provisioned dapat menjadi solusi untuk meningkatkan kecepatan respons sistem secara keseluruhan.

REFERENSI

[1] Sudianto and Sutopo, “Optimalisasi Implementasi Sistem Informasi Manajemen Berbasis Cloud Untuk Meningkatan Efisinsi Operasional Di Sektor Industri : Studi Literatur,” *J. Rekayasa Sist. Inf. dan Teknol.*, vol. 2, no. 4, 2025.

[2] D. Julianti, “Strategi Kebijakan Penguatan Pelayanan Publik Dan Pengawasan Perizinan Berusaha Dengan Aplikasi Berbasis Teknologi Informasi,” *KYBERNOLOGY J. Ilmu Pemerintah. dan Adm. Publik*, vol. 2, no. 2, 2024.

[3] F. P. E. Putra, A. M. U. Solichin, M. N. W. Hakim, and M. T. Ramadhan, “Pemanfaatan Teknologi Wireless dan Mobile Network Berbasis 5G Untuk Pemerataan Akses Jaringan di Indonesia,” *Infotek J. Inform. dan Teknol.*, vol. 8, no. 2, 2025.

[4] M. Azzahari, I. Khaldun, M. Mahmud, and R. Rozaliana, “Server Worker Power Optimization with Virtual Machine Live Migration Technique Using Fuzzy Mamdani,” *J. Artif. Intell. Softw. Eng.*, vol. 5, no. 2, 2025.

[5] A. F. Zahir, H. Wijaya, M. Sanwasih, and Arisantoso, “Analisis Efektivitas Metode Round-Robin dan Least-Connection dalam Load Balancing Terhadap Throughput Server Web,” *J. Ilm. Inform. dan Ilmu Komput.*, vol. 4, no. 1, 2025.

[6] K. Senjab, S. Abbas, N. Ahmed, and A. ur R. Khan, “A survey of Kubernetes scheduling algorithms,” *J. Cloud Comput.*, vol. 12, no. 1, pp. 1–26, 2023.

[7] R. Hananta, “Implementasi Load Balancer Server menggunakan Metode Autoscaling Berbasis Orcestrarionsystem,” Universitas Sains Mataram, 2023.

[8] S. Parulian, “Analisis dan Implementasi Infrastruktur Komputasi Awan Berbasis Web dengan Pemanfaatan Load Balancing dan Auto-Scaling pada Google Cloud Platform,” Universitas Pendidikan Indonesia, 2024.

[9] Y. Hidayat and B. Arifwidodo, “Implementasi Web Server Menggunakan Infrastructure As Code Terraform Berbasis Cloud Computing,” *Format J. Ilm. Tek. Inform.*, vol. 10, no. 2, p. 192, 2021.

[10] Sharvani et al., “An Auto-Scaling Approach to Load

Balance Dynamic Workloads for Cloud Systems," *Turkish J. Comput. Math. Educ.*, vol. 12, no. 11, pp. 515–531, 2021.

[11] Llorens et al., "An SDN-based solution for horizontal auto-scaling and load balancing of transparent VNF clusters," *Sensors*, vol. 21, no. 24, pp. 1–23, 2021.

[12] N. M. A. Yalestia Chandrawaty and I. P. Hariyadi, "Implementasi Ansible Playbook Untuk Mengotomatisasi Manajemen Konfigurasi VLAN Berbasis VTP Dan Layanan DHCP," *J. Bumigora Inf. Technol.*, vol. 3, no. 2, pp. 107–122, 2021.

[13] A. Zulfikar and Y. Akbar, "Otomasi Backup Konfigurasi Settingan Router Mikrotik Menggunakan Ansible dengan Metode Network DevOps," *Indones. J. Mach. Learn. Comput. Sci.*, vol. 5, no. 1, 2025.

[14] M. H. Prayitno and M. Yasir, "Peran Metode Network Development Life Cycle (NDLC) pada Implementasi Failover Base Transceiver Station," *Innov. J. Soc. Sci. Res.*, vol. 5, no. 2, 2025.

[15] F. H. P. Prasetyo, Erna, and Febriyansyah, "Penerapan Metode Network Development Life Cycle (NDLC) dalam Pengembangan Jaringan Komputer," *J. Informatics Commun. Technol.*, vol. 7, no. 1, 2025.