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

PENERAPAN *VERTICAL AUTOSCALING* SUMBER DAYA *VIRTUAL MACHINE* (VM) PADA *PROXMOX* MENGGUNAKAN *ANSIBLE* DAN *PROMETHEUS*

Nima Karunia [1*], I Putu Hariyadi[2], Khairan Marzuki[3]
[1,2,3]Ilmu Komputer, Teknik, Universitas Bumigora

***Abstract*— The advancement of virtualization technology drives the need for infrastructure systems capable of automatically and efficiently adjusting resource capacity. This study aims to implement vertical autoscaling on Virtual Machines (VMs) in the Proxmox platform by utilizing Ansible as an automation tool and Prometheus as a monitoring tool. The method employed follows the Network Development Life Cycle (NDLC), which consists of three main stages: analysis, design, and prototype simulation. The system was tested on two VMs, where Prometheus monitored CPU and memory usage metrics, while Alertmanager sent notifications to a webhook that executed Ansible playbooks to automatically increase or decrease VM resources. The experimental results show that the system can effectively adjust the number of CPU cores and memory capacity based on predefined load thresholds. The scale-up process occurs when CPU or memory usage exceeds 80% for one minute, while scale-down occurs when usage drops below 70% for five minutes. In conclusion, the integration of Proxmox, Prometheus, and Ansible for vertical autoscaling enhances resource utilization efficiency and service availability dynamically and automatically.**
Keyword— Vertical Autoscaling, Proxmox, Ansible, Prometheus, Virtual Machine, Network Development Life Cycle.

***Intisari*— Perkembangan teknologi virtualisasi mendorong kebutuhan akan sistem infrastruktur yang mampu menyesuaikan kapasitas sumber daya secara otomatis dan efisien. Penelitian ini bertujuan untuk menerapkan *vertical autoscaling* pada *Virtual Machine* (VM) di platform Proxmox dengan memanfaatkan Ansible sebagai alat otomasi dan Prometheus sebagai alat pemantauan. Metode yang digunakan mengacu pada *Network Development Life Cycle* (NDLC), yang mencakup tiga tahapan utama: analisis, perancangan, dan simulasi prototipe. Sistem diuji pada dua VM, di mana Prometheus memantau metrik penggunaan CPU dan memori, sementara Alertmanager mengirimkan notifikasi ke webhook yang menjalankan *playbook* Ansible untuk secara otomatis menambah atau mengurangi sumber daya VM. Hasil pengujian menunjukkan bahwa sistem mampu melakukan penyesuaian jumlah *CPU core* dan kapasitas memori secara efektif berdasarkan ambang batas beban yang telah ditentukan. Proses *scale up* dilakukan saat penggunaan CPU atau memori melebihi 80% selama satu menit, dan *scale down* dilakukan saat penggunaan turun di bawah 70% selama lima menit. Kesimpulannya, integrasi antara Proxmox, Prometheus, dan Ansible dalam penerapan *vertical autoscaling* mampu meningkatkan efisiensi pemanfaatan sumber daya serta ketersediaan layanan secara dinamis dan otomatis.**
***Kata Kunci*— *Vertical Autoscaling, Proxmox, Sensible, Prometheus, Virtual Machine,* Network Development life Cycle.**

I. PENDAHULUAN

Perkembangan teknologi informasi yang pesat telah membawa transformasi signifikan dalam pengelolaan infrastruktur system [1], terutama dengan hadirnya teknologi virtualisasi. Virtual machine (VM) menjadi solusi utama dalam pemanfaatan sumber daya komputasi secara fleksibel, skalabel, dan efisien [2]. Namun, dinamika beban kerja yang tidak menentu pada VM menjadi tantangan tersendiri dalam menjaga stabilitas dan performa sistem. Beban yang tiba-tiba meningkat dapat menyebabkan overload, sedangkan beban yang menurun tanpa diiringi penyesuaian alokasi sumber daya dapat menimbulkan pemborosan. Dalam konteks infrastruktur modern, efisiensi operasional menjadi krusial, sehingga diperlukan mekanisme penyesuaian sumber daya secara otomatis, adaptif, dan real-time agar sistem tetap optimal dalam berbagai kondisi [3] . oleh sebab itu permasalahan yang dihadapi dalma penelitian ini Adalah bagaimana mekanisme penyesuaian alokasi sumber daya yang otomatis, adaptif, dan real-time dapat diterapkan untuk mengatasi permasalahan overload dan pemborosan sumber daya pada virtual machine sehingga stabilitas dan performa sistem tetap terjaga secara optimal dalam menghadapi beban kerja yang dinamis dan tidak menentu.

Autoscaling merupakan solusi yang telah banyak digunakan dalam sistem cloud computing untuk menyesuaikan sumber daya secara otomatis berdasarkan kondisi aktual penggunaan [4]–[8]. Secara umum, autoscaling terbagi menjadi dua pendekatan: horizontal dan vertikal. Horizontal autoscaling menambah atau mengurangi jumlah VM berdasarkan kebutuhan beban kerja, sementara vertical autoscaling menyesuaikan kapasitas CPU dan memori dari VM yang sudah berjalan tanpa membuat instansi baru. Meskipun pendekatan horizontal lebih umum diterapkan pada

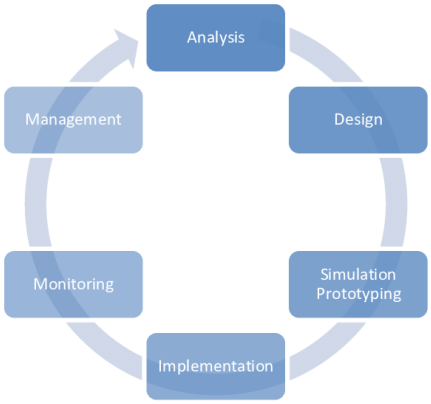
platform cloud komersial seperti AWS dan GCP, pendekatan vertikal memberikan keunggulan dalam efisiensi sumber daya, terutama di lingkungan yang terbatas secara fisik atau non-komersial. Penelitian ini memilih fokus pada vertical autoscaling karena pendekatan ini memungkinkan adaptasi sumber daya secara granular tanpa mengganggu instansi yang sedang berjalan [9].

Berbagai penelitian sebelumnya telah mengeksplorasi mekanisme autoscaling, namun sebagian besar berfokus pada pendekatan horizontal dan mengandalkan layanan cloud pihak ketiga. Misalnya, penelitian [10] mengkaji penerapan horizontal autoscaling untuk web server di AWS, [11] membahas optimasi reliabilitas melalui fitur autoscaling dan autohealing di Google Cloud Platform, dan [12] menerapkan elastic load balancing untuk distribusi beban di lingkungan AWS. Selain itu, pendekatan prediktif berbasis model Bi-LSTM juga telah digunakan untuk meningkatkan efisiensi proses autoscaling [3]. Meski demikian, sangat sedikit studi yang secara eksplisit mengeksplorasi integrasi sistem open-source seperti Proxmox dengan solusi otomasi seperti Ansible dan sistem pemantauan real-time seperti Prometheus untuk membangun sistem vertical autoscaling secara mandiri di luar ekosistem cloud komersial.

Penelitian ini menawarkan pendekatan alternatif dengan menerapkan vertical autoscaling sumber daya VM pada platform Proxmox, yang dikombinasikan dengan Prometheus sebagai sistem pemantauan dan Ansible sebagai alat otomasi infrastruktur. Proxmox, sebagai platform virtualisasi open-source, menyediakan fleksibilitas tinggi untuk pengelolaan VM, namun belum memiliki fitur autoscaling bawaan. Dengan mengintegrasikan pemantauan metrik real-time dari Prometheus dan otomatisasi eksekusi melalui Ansible, sistem ini diharapkan mampu menyesuaikan kapasitas CPU dan memori VM secara otomatis berdasarkan ambang batas penggunaan yang ditentukan. Tujuan dari penelitian ini adalah membangun prototipe sistem vertical autoscaling berbasis open-source yang tidak hanya efisien dan hemat biaya, tetapi juga mampu meningkatkan ketersediaan layanan dan mengurangi intervensi manual dalam pengelolaan sumber daya infrastruktur TI.

II. METODOLOGI PENELITIAN

Penelitian ini menggunakan metode **Network Development Life Cycle (NDLC)** yang terdiri dari enam tahapan: *analysis*, *design*, *simulation prototype*, *implementation*, *monitoring*, dan *managemen* [13]–[15]. Namun, penelitian ini hanya fokus pada tiga tahapan awal: analisis, desain, dan simulasi prototipe, untuk memastikan validitas sistem sebelum diimplementasikan secara luas.



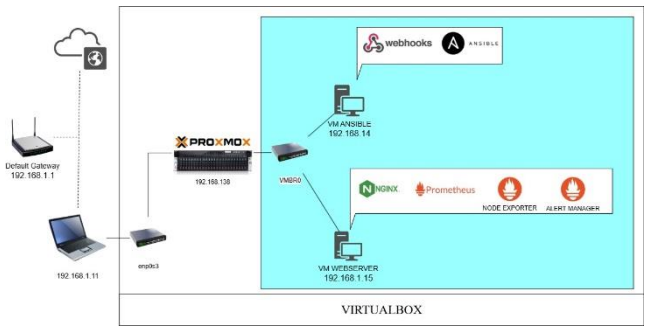
Gambar 1. NDLC

A. Analisis

Pada tahap ini, peneliti mengidentifikasi permasalahan yang berkaitan dengan tidak tersedianya fitur autoscaling vertikal otomatis di Proxmox. Analisis dilakukan melalui studi literatur dan eksperimen awal untuk memahami kebutuhan sistem. Sistem ditujukan untuk dapat menyesuaikan jumlah CPU dan memori VM secara otomatis berdasarkan ambang batas beban, tanpa intervensi manual.

B. Perancangan

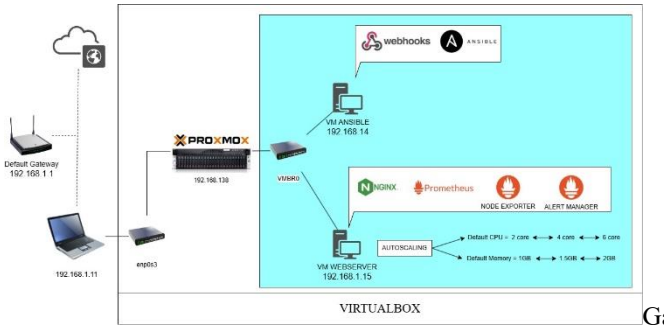
Desain sistem mencakup arsitektur jaringan, alur kerja autoscaling, dan pengaturan komunikasi antar komponen (Prometheus, Ansible, dan Alertmanager). Terdapat dua VM utama: satu VM bertindak sebagai webserver dan sistem monitoring (Prometheus, Node Exporter, Alertmanager, Nginx), dan satu lagi sebagai kontrol node (Ansible dan Webhook). Sistem dikembangkan dalam lingkungan virtual menggunakan Proxmox yang dijalankan pada VirtualBox.



Gambar 2. Rancangan Topologi Uji Coba Vertical Autoscaling di Proxmox

Di dalam Proxmox terdapat dua VM, yaitu VM Ansible dan VM Webserver. VM Ansible berfungsi sebagai pusat otomasi yang menjalankan Ansible untuk menambah atau mengurangi sumber daya (CPU dan memori) pada VM Webserver secara otomatis. VM ini juga menjalankan Webhook yang menerima notifikasi dari Alertmanager untuk memicu proses scaling. Sementara itu, VM Webserver berperan sebagai server web sekaligus sistem monitoring. VM ini menjalankan Nginx sebagai webserver, Prometheus untuk memantau performa sistem, Node Exporter untuk menyuplai

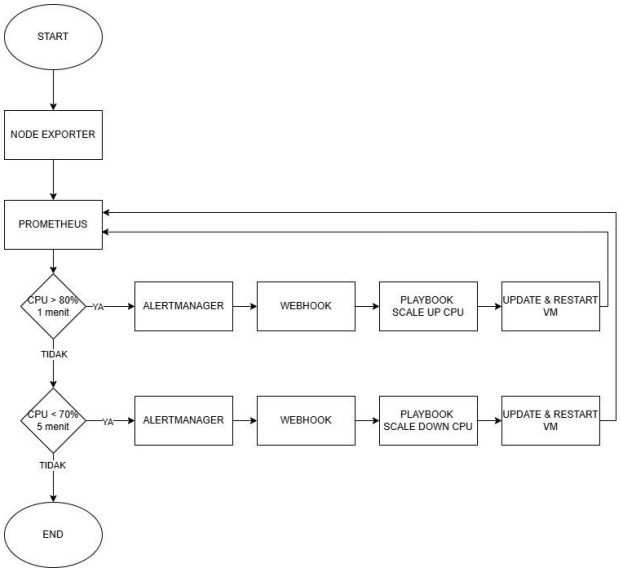
metrik penggunaan sumber daya, serta Alertmanager yang mengirimkan notifikasi ke Webhook saat terjadi lonjakan atau penurunan beban. Kedua VM saling terhubung melalui jaringan LAN virtual yang dikonfigurasi di dalam Proxmox sehingga komunikasi antar layanan dapat berjalan secara lokal.



Gambar 3. Rancangan topologi ujicoba Autoscaling

Berdasarkan Gambar 3 proses scale up dan scale down CPU maupun memori pada VM Webserver dikendalikan melalui kombinasi Prometheus, Alertmanager, Webhook, dan Ansible. Prometheus memantau metrik CPU dan RAM dari Node Exporter, kemudian Alertmanager mengirimkan notifikasi ke Webhook jika terjadi lonjakan (di atas 80% selama 1 menit) atau penurunan beban (di bawah 70% selama 5 menit). Webhook selanjutnya mengeksekusi playbook Ansible untuk menambah atau mengurangi alokasi CPU dan memori secara otomatis, sehingga penggunaan sumber daya menjadi optimal tanpa pemborosan kapasitas fisik.

Alur kerja sistem ditentukan dalam dua skenario utama: CPU dan memori. Prometheus memantau metrik melalui Node Exporter, dan saat ambang batas tercapai (CPU/memori >80% selama 1 menit), Alertmanager akan mengirimkan alert ke Webhook untuk menjalankan playbook Ansible yang melakukan scaling.



Gambar 4. Alur Kerja Sistem Autoscaling CPU

Gambar 4. menunjukkan alur kerja autoscaling vertikal CPU pada sistem berbasis Proxmox yang terintegrasi dengan Prometheus dan Ansible. Proses dimulai ketika Node Exporter mengirimkan metrik penggunaan CPU VM Webserver ke Prometheus untuk dianalisis secara berkala. Jika beban CPU melebihi 80% selama 1 menit, Prometheus memicu alert ke Alertmanager yang diteruskan ke Webhook, kemudian mengeksekusi playbook Ansible scale_up_cpu untuk menambah core CPU melalui API Proxmox dan melakukan update serta restart VM agar konfigurasi baru aktif. Sebaliknya, jika beban CPU turun di bawah 70% selama 5 menit, Prometheus memicu alert scale down yang memicu playbook scale_down_cpu untuk mengurangi alokasi CPU dan merestart VM guna menghemat sumber daya. Apabila tidak ada kondisi yang terpenuhi, sistem tetap dalam kondisi pemantauan tanpa melakukan perubahan konfigurasi. Flowchart ini memperlihatkan bahwa autoscaling CPU berjalan otomatis dari tahap pemantauan hingga eksekusi perubahan sumber daya sesuai kondisi beban kerja.

C. Simulasi Prototipe

Pengujian dilakukan melalui empat skenario, yaitu scale up CPU, scale down CPU, scale up memori, dan scale down memori, masing-masing diulang lima kali. Pada pengujian CPU, VM Webserver diawali dengan alokasi 2 core CPU dan diberi stress test bertahap hingga penggunaan CPU melebihi 80% selama 1 menit, memicu scale up ke 4 core dan kemudian 6 core. Beban kemudian diturunkan hingga penggunaan CPU di bawah 70% selama 5 menit, sehingga sistem melakukan scale down bertahap kembali ke 2 core. Sementara itu, pengujian memori dimulai dengan alokasi 1 GB RAM dan mengalami scale up ke 1,5 GB dan 2 GB saat penggunaan RAM melebihi 80% selama 1 menit, lalu scale down kembali ke 1 GB ketika penggunaan turun di bawah 70%. Pengujian memanfaatkan **stress test** untuk meningkatkan beban, dan sistem akan merespons secara otomatis berdasarkan threshold yang ditentukan.

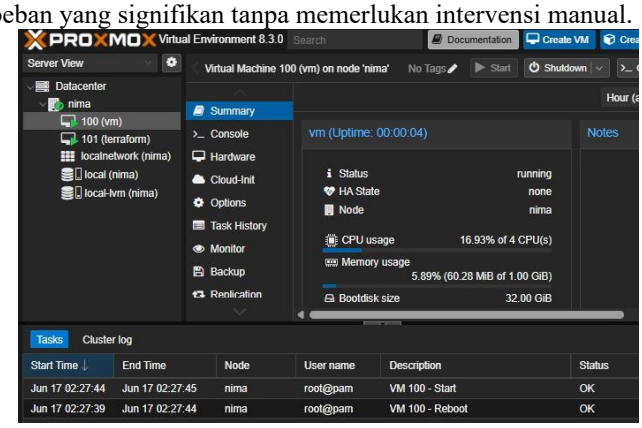
III. HASIL DAN PEMBAHASAN

Setelah seluruh komponen sistem autoscaling selesai dikonfigurasi dan diverifikasi, tahap selanjutnya adalah melakukan pengujian fungsional untuk menilai kinerja sistem dalam menyesuaikan sumber daya secara otomatis. Pengujian ini bertujuan untuk memastikan integrasi antara Proxmox, Prometheus, Alertmanager, Webhook, dan Ansible berjalan optimal dalam mendeteksi perubahan beban kerja, memicu alert, dan mengeksekusi proses penskalaan CPU maupun memori tanpa intervensi manual. Skenario pengujian dibagi menjadi empat kondisi utama, yaitu scale up CPU, scale down CPU, scale up memori, dan scale down memori, yang merepresentasikan situasi lonjakan dan penurunan beban sistem. Melalui keempat skenario ini, diperoleh gambaran

menyeluruh tentang kemampuan sistem autoscaling vertikal dalam mengelola sumber daya VM secara dinamis, termasuk respons terhadap beban kerja yang fluktuatif, stabilitas layanan webserver, dan efisiensi pemanfaatan sumber daya fisik yang tersedia.

A. Scale Up CPU

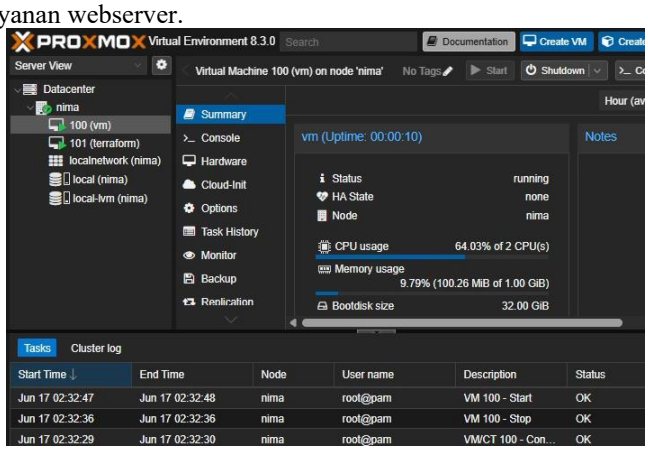
Dalam skenario scale up CPU, sistem diuji untuk menilai kemampuannya meningkatkan jumlah core CPU secara otomatis ketika terjadi lonjakan beban kerja. VM Webserver yang difungsikan sebagai server web sekaligus node monitoring memulai pengujian dengan alokasi awal 2 core CPU, sebagai kapasitas minimal untuk menjalankan layanan dasar. Untuk memicu autoscaling, dilakukan stress test CPU secara bertahap menggunakan tool penguji beban hingga penggunaan CPU mencapai lebih dari 80% selama 1 menit, sesuai ambang batas yang telah dikonfigurasi di Prometheus alert rules. Ketika kondisi ini terpenuhi, Prometheus mendeteksi lonjakan beban dan mengirimkan alert ke Alertmanager, yang kemudian diteruskan ke Webhook untuk mengeksekusi playbook Ansible scale_up_cpu. Playbook ini menambahkan jumlah core CPU pada VM Webserver melalui API Proxmox dan diakhiri dengan restart VM agar konfigurasi baru aktif. Proses ini menunjukkan bahwa sistem mampu meningkatkan kapasitas pemrosesan secara otomatis, menjaga ketersediaan layanan webserver meskipun terjadi peningkatan beban yang signifikan tanpa memerlukan intervensi manual.



Gambar 5. VM webserver setelah scale up CPU

B. Scale Down CPU

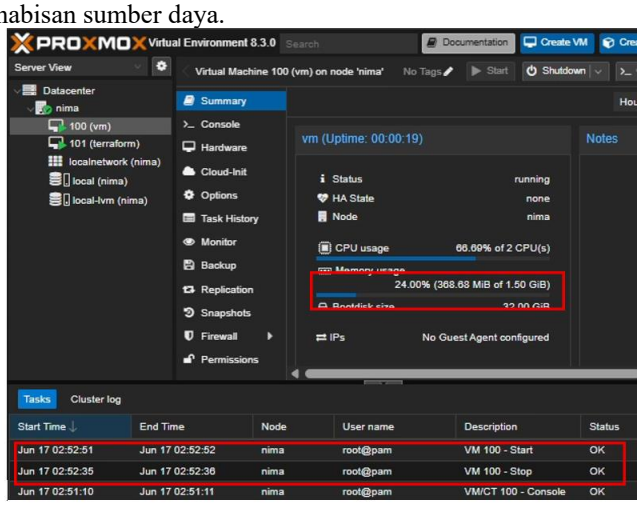
Skenario scale down CPU dirancang untuk menguji kemampuan sistem mengurangi jumlah core CPU secara otomatis ketika beban kerja rendah, sehingga pemanfaatan sumber daya menjadi lebih efisien. Kondisi awal VM Webserver pada pengujian ini memiliki 4 core CPU, yaitu konfigurasi yang dihasilkan dari proses scale up sebelumnya. Selanjutnya, beban CPU diturunkan secara bertahap dengan menghentikan stress test hingga pemakaian CPU turun di bawah 70% selama 5 menit, sesuai ambang batas penurunan yang telah ditetapkan. Ketika kondisi ini tercapai, Prometheus mengidentifikasi beban rendah dan memicu alert scale down CPU, yang dikirim ke Alertmanager dan diteruskan ke Webhook. Webhook kemudian mengeksekusi playbook Ansible scale_down_cpu, mengurangi jumlah core CPU melalui API Proxmox, dan melakukan restart VM untuk menerapkan konfigurasi baru. Skenario ini membuktikan bahwa sistem mampu melakukan efisiensi sumber daya secara otomatis, mengurangi risiko pemborosan kapasitas CPU ketika beban kerja tidak tinggi, tanpa mengganggu akses ke layanan webserver.



Gambar 6. VM vm webserver sebelum scale down cpu

C. Skenario Scale Up Memori

Skenario scale up memori menguji respons sistem dalam meningkatkan kapasitas RAM saat terjadi lonjakan penggunaan memori. Pada pengujian ini, VM Webserver diawali dengan alokasi memori 1 GB, sebagai kapasitas dasar untuk menjalankan layanan. Proses stress test memori dilakukan secara bertahap hingga konsumsi RAM melebihi 80% selama 1 menit, sesuai aturan yang dikonfigurasi pada Prometheus. Saat ambang batas ini tercapai, Prometheus memicu alert scale up memory yang diteruskan oleh Alertmanager ke Webhook. Webhook mengeksekusi playbook Ansible scale_up_memory, yang menambahkan 512 MB RAM melalui API Proxmox, kemudian VM di-restart agar alokasi baru aktif. Melalui skenario ini, terlihat bahwa sistem dapat merespons peningkatan beban memori secara real-time, memastikan layanan webserver tetap berjalan lancar tanpa kehabisan sumber daya.

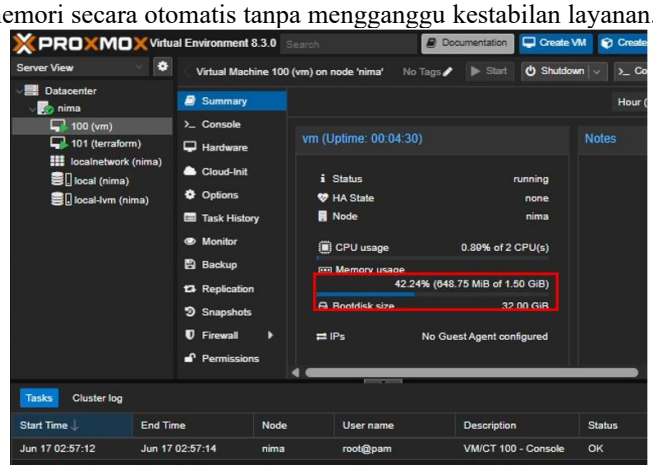


Gambar 7. VM webserver setelah scale up memori

D. Skenario Scale Down Memori

Skenario terakhir, scale down memori, dilakukan untuk memastikan sistem mampu mengurangi kapasitas RAM secara

otomatis saat beban memori rendah, sehingga pemakaian sumber daya fisik menjadi lebih hemat. Setelah skenario scale up, VM Webserver berada pada alokasi memori 1,5 GB. Beban memori kemudian dikurangi dengan menghentikan stress test hingga konsumsi RAM turun di bawah 70% selama 5 menit. Prometheus mendeteksi kondisi ini dan memicu alert scale down memory, yang diteruskan oleh Alertmanager ke Webhook. Selanjutnya, playbook Ansible scale_down_memory dijalankan untuk mengurangi 512 MB RAM, sehingga VM kembali ke konfigurasi awal 1 GB RAM, diikuti dengan restart VM agar konfigurasi baru aktif. Hasil pengujian menunjukkan bahwa sistem dapat mengoptimalkan pemanfaatan sumber daya dengan menurunkan alokasi memori secara otomatis tanpa mengganggu kestabilan layanan.



Gambar 8. VM webserver sebelum scale down memory

Berdasarkan uji coba yang telah dilakukan, diperoleh hasil analisis yang menunjukkan kinerja sistem dalam menjalankan proses autoscaling vertikal CPU pada VM Webserver. Pengujian ini bertujuan untuk menilai sejauh mana sistem mampu mendeteksi lonjakan beban kerja, memicu proses penskalaan otomatis, dan menyelesaikan perubahan konfigurasi CPU hingga VM siap digunakan kembali. Setiap siklus autoscaling mencakup tahapan deteksi beban oleh Prometheus, pengiriman alert melalui Alertmanager, eksekusi playbook Ansible melalui Webhook, hingga restart VM untuk menerapkan penambahan core CPU. Hasil pengujian ini memberikan gambaran mengenai kecepatan respons, stabilitas proses scaling, dan efisiensi pemanfaatan sumber daya dalam menghadapi kondisi beban yang dinamis.

TABEL I
HASIL ANALISA UJI COBA AUTOSCALING CPU

Percobaan	SCALE UP						SCALE DOWN			
	2 core ke 4 core			4 core ke 6 core			6 core turun ke 4 core		4 core turun ke 2 core	
	T1	T2	T3	T1	T2	T3	T2	T3	T2	T3
Pertama	6	60	49	7	60	62	300	40	300	39
Kedua	4	60	51	4	60	55	300	41	300	40
Ketiga	5	60	54	4	60	58	300	40	300	41
Keempat	4	60	50	5	60	60	300	39	300	30
Kelima	4	60	49	4	60	59	300	41	300	39
Terlama	6	60	54	7	60	62	300	41	300	41
Tercepat	4	60	49	4	60	55	300	39	300	30
Rata-rata	4,6	60	50,6	4,8	60	58,8	300	40,2	300	37,8

Pada tabel terlihat skenario autoscaling vertikal CPU, pengujian dilakukan melalui dua tahap scale up (2 core ke 4 core dan 4 core ke 6 core) serta dua tahap scale down (6 core ke 4 core dan 4 core ke 2 core). Pada tahap scale up pertama dari 2 core ke 4 core, rata-rata durasi T1 (waktu dari stress test hingga beban CPU terdeteksi) adalah 4,6 detik, T2 (hingga alert Prometheus berstatus firing) stabil di 60 detik, dan T3 (dari alert firing hingga webserver kembali aktif) rata-rata 50,6 detik, sehingga total rata-rata waktu scale up mencapai 115,2 detik atau sekitar 1,92 menit. Pada tahap scale up kedua dari 4 core ke 6 core, rata-rata T1 sedikit meningkat menjadi 4,8 detik, T2 tetap 60 detik, dan T3 menjadi 58,8 detik, menghasilkan total waktu rata-rata 123,6 detik atau sekitar 2,06 menit. Sementara itu, proses scale down CPU memerlukan waktu lebih lama karena sistem menunggu beban rendah selama 300 detik (5 menit) sebelum mengeksekusi pengurangan core. Pada tahap 6 core ke 4 core, rata-rata T3 adalah 40,2 detik, sedangkan pada tahap 4 core ke 2 core rata-rata 37,8 detik, sehingga total durasi scale down masing-masing sekitar 5,7–5,8 menit. Hasil ini menunjukkan bahwa proses scale up CPU berlangsung lebih cepat dan responsif, sedangkan scale down memerlukan durasi lebih lama demi menjaga stabilitas sistem. Secara keseluruhan, mekanisme autoscaling vertikal CPU berbasis Proxmox, Prometheus, dan Ansible mampu menyesuaikan alokasi core CPU secara otomatis, meskipun setiap proses scaling memerlukan restart VM yang menimbulkan jeda singkat pada layanan.

TABEL 2
HASIL ANALISA UJI COBA AUTOSCALING CPU

Percobaan	SCALE UP						SCALE DOWN			
	1 GB ke 1.50 GB			1.50 GB ke 2GB			2GB turun ke 1.5GB		1.5GB turun ke 1 GB	
	T1	T2	T3	T1	T2	T3	T2	T3	T2	T3
Pertama	4	60	45	5	60	48	300	40	300	39
Kedua	5	60	45	6	60	47	300	40	300	41
Ketiga	4	60	47	4	60	48	300	39	300	40
Keempat	4	60	46	5	60	46	300	41	300	41
Kelima	5	60	47	6	60	47	300	40	300	39
Terlama	5	60	47	6	60	48	300	41	300	41
Tercepat	4	60	45	4	60	46	300	39	300	39
Rata-rata	4,4	60	46	5,2	60	47,2	300	40	300	40

Dari tabel tersebut terlihat bahwa proses scale up memori terdiri dari dua tahap, yaitu dari 1 GB ke 1,5 GB dan dari 1,5 GB ke 2 GB. Durasi tercepat terjadi pada percobaan pertama dengan waktu 4 detik (T1) untuk memicu proses, 60 detik (T2) untuk menjalankan playbook dan proses scaling di Proxmox, serta 45 detik (T3) untuk VM kembali aktif setelah restart. Sedangkan waktu rata-rata untuk tahap ini adalah ±4,4 detik (T1) untuk trigger awal dan ±46 detik (T3) untuk proses booting VM setelah scaling. Proses scale up ke 2 GB menunjukkan hasil yang hampir serupa dengan rata-rata 5,2

detik (T1) untuk trigger awal dan ±47,2 detik (T3) untuk proses aktif kembali.

Untuk skenario scale down memori, baik dari 2 GB ke 1,5 GB maupun 1,5 GB ke 1 GB, durasi T1 secara konsisten memerlukan 300 detik (5 menit), sesuai dengan threshold yang dikonfigurasi pada Prometheus untuk mendeteksi beban rendah. Waktu T2 rata-rata tercatat ±40 detik, dan T3 rata-rata ±40 detik, yang merepresentasikan durasi eksekusi playbook Ansible serta proses restart VM. Hal ini menunjukkan bahwa proses scale down lebih lama pada tahap trigger awal, karena sistem harus menunggu kondisi idle selama lima menit sebelum melakukan pengurangan sumber daya, sedangkan proses eksekusi scaling relatif konsisten.

Secara keseluruhan, hasil ini menunjukkan bahwa proses scale up lebih cepat dalam merespons lonjakan beban, sedangkan scale down membutuhkan waktu lebih lama karena mempertimbangkan stabilitas beban rendah sebelum pengurangan sumber daya dilakukan. Dengan rata-rata durasi eksekusi (T2 + T3) sekitar 100 detik, sistem mampu melakukan penyesuaian kapasitas memori secara otomatis tanpa intervensi manual, meskipun setiap perubahan memerlukan restart VM yang bersifat sementara disruptif terhadap layanan.

IV. KESIMPULAN DAN SARAN

Berdasarkan hasil perancangan dan pengujian sistem, dapat disimpulkan bahwa mekanisme autoscaling vertikal berbasis Proxmox, Prometheus, dan Ansible mampu menyesuaikan alokasi sumber daya CPU dan memori secara otomatis sesuai kondisi beban kerja. Proses scale up menunjukkan respons yang cepat, dengan rata-rata durasi kurang dari 2 menit untuk menambah core CPU atau kapasitas memori, sehingga mampu menjaga ketersediaan dan stabilitas layanan webserver saat terjadi lonjakan beban. Sementara itu, scale down memerlukan waktu lebih lama, sekitar 5–6 menit, karena sistem menunggu periode beban rendah selama lima menit untuk mencegah gangguan pada layanan. Hasil pengujian juga menunjukkan bahwa proses scaling berjalan stabil dan konsisten, baik pada CPU maupun memori, dengan variasi waktu antarpercobaan yang relatif kecil.

Secara keseluruhan, implementasi autoscaling vertikal ini efektif dalam mengoptimalkan pemanfaatan sumber daya fisik, mengurangi risiko overprovisioning, dan meminimalkan intervensi manual dalam pengelolaan VM. Namun, kelemahan sistem ini adalah setiap perubahan alokasi CPU dan memori memerlukan proses restart VM, yang menimbulkan jeda singkat pada layanan. Ke depan, pengembangan lebih lanjut dapat difokuskan pada mekanisme live scaling tanpa reboot atau integrasi autoscaling horizontal untuk meningkatkan fleksibilitas dan ketersediaan layanan.

Saran penelitian selanjutnya dapat difokuskan pada pengembangan autoscaling vertikal tanpa memerlukan reboot VM, misalnya melalui fitur hot-plug CPU dan memori atau live migration untuk meminimalkan downtime. Selain itu, integrasi dengan autoscaling horizontal dan algoritma deteksi beban berbasis prediksi dapat meningkatkan efisiensi dan responsivitas sistem terhadap lonjakan trafik. Pengujian pada lingkungan produksi skala besar atau integrasi dengan container dan orkestrasi seperti Kubernetes juga menjadi arah penelitian yang potensial untuk menghadirkan sistem autoscaling yang lebih andal, fleksibel, dan modern.

REFERENSI

[1] E. Barus, K. M. Pardede, and J. A. Putri Br. Manjorang, “Transformasi Digital: Teknologi Cloud Computing dalam Efisiensi Akuntansi,” *J. Sains dan Teknol.*, vol. 5, no. 3, 2024.

[2] A. B. Permadi, N. T. Khair, and M. R. Kurniawan, “IMPLEMENTASI VIRTUALISASI UNTUK PENGELOLAAN SERVER MENGGUNAKAN PROXMOX VE,” 2023.

[3] J. Park and J. Jeong, “An Autoscaling System Based on Predicting the Demand for Resources and Responding to Failure in Forecasting,” *Sensors*, vol. 23, no. 23, 2023.

[4] R. W. Z. King and P. H. Trisnawan, “Perbandingan Metode Autoscaling Vertical Pod Autoscaler dan Horizontal Pod Autoscaler Kubernetes Pada Google Cloud Platform,” *J. Pengemb. Teknol. Inf. Dan Ilmu Komput.*, vol. 8, no. 7, 2024.

[5] O. Pramadika and D. W. Chandra, “Provisioning Google Kubernetes Engine (GKE) Cluster dengan Menggunakan Terraform dan Jenkins pada Dua Environment,” *JIPI (Jurnal Ilm. Penelit. dan Pembelajaran Inform.*, vol. 8, no. 2, 2023.

[6] J. Entrialgo, M. García, J. García, J. M. López, and J. L. Díaz, “Joint Autoscaling of Containers and Virtual Machines for Cost Optimization in Container Clusters,” *J. Grid Comput.*, vol. 22, no. 1, 2024.

[7] 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.

[8] T. Tournaire, H. Castel-Taleb, and E. Hyon, “Efficient Computation of Optimal Thresholds in Cloud Auto-scaling Systems,” *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 8, no. 4, 2023.

[9] M. ZargarAzad and M. Ashtiani, “An Auto-Scaling Approach for Microservices in Cloud Computing Environments,” *J. Grid Comput.*, vol. 21, no. 4, pp. 0–39, 2023.

[10] M. Z. Asiari, “Analisis Kinerja Sistem Auto Scaling Pada Sistem Web Server Berbasis Clustering Menggunakan Sistem Virtual,” Universitas Hasanuddin, 2021.

[11] N. Ramsari and A. Ginanjar, “Implementasi Infrastruktur Server Berbasis Cloud Computing Untuk Web Service Berbasis Teknologi Google Cloud Platform,” *Conf. Senat. STT Adisutjipto Yogyakarta*, vol. 7, no. March 2022, 2022.

[12] S. F. Wandira and T. Y. Hadiwandra, “Desain Skalabel Website Menggunakan Elastic Load Balancing pada Amazone Virtual Private Cloud

(VPC)," *J. Teknol. Inform. dan Komput.*, vol. 9, no. 2, pp. 1460–1475, 2023.

[13] F. Naim, R. R. Saedudin, and U. Y. K. S. Hediyanto, "ANALYSIS OF WIRELESS AND CABLE NETWORK QUALITY-OF-SERVICE PERFORMANCE AT TELKOM UNIVERSITY LANDMARK TOWER USING NETWORK DEVELOPMENT LIFE CYCLE (NDLC) METHOD," *JIPI (Jurnal Ilm. Penelit. dan Pembelajaran Inform.*, vol. 7, no. 4, 2022.

[14] P. I. D. Candra Wulan, D. P. Perdana, and A. A. Kurniawan, "Performance analysis and development of OPD interconnection network using NDLC method in Boven Digoel diskominfo papua province," *Compiler*, vol. 11, no. 1, 2022.

[15] D. Haryanto and R. Kipran, "Design of wireless access point configuration network using packet trace r 6.2 at smp negeri 5 prabumulih with development method network development life cycle (ndlc)," *Int. J. Cist.*, vol. 2, no. 1, 2023.