Analisis Sistem Monitoring dan Alerting Prometheus-Grafana Berbasis Docker pada Server VPS dalam Jaringan VPN

Azzuri Putra Mahendra
Teknik Multimedia dan Jaringan, Teknik Informatika dan Komputer, Politeknik Negeri Jakarta
Depok, Jawa Barat
azzuri.putra.mahendra.tik22@mhsw.pnj.ac.id

The issue of disrupted application availability and the risk of hardware damage due to server workload overload requires an accurate, real-time anomaly detection system. Therefore, this study aims to implement and analyze the performance of a Docker-based monitoring and alerting system using the integration of Prometheus, Grafana, and Alertmanager within a WireGuard VPN network. The research method employed is experimental, conducting workload simulations using stress-ng and fio on a VPS. During system verification, it was proven that the monitoring and alerting system could successfully run on top of the Docker architecture, utilizing the Docker Engine, Docker Image, and Docker Container on the server. When the workload simulation was performed, system performance was measured through the Anomaly Detection Time (ADT) and Resolved Detection Time (RDT) metrics. The test results indicated that the system is highly responsive, with the highest average latency on the monitoring server (Monserver) recorded at 25.4 ms during CPU workload testing. Meanwhile, on the target server (Testserver), the highest average latency reached 28 ms under the RAM workload scenario, with an anomaly detection time (ADT) of 27.2 ms during Disk testing. These results align with the API latency thresholds according to SigNoz's acceptable p95 standard. Furthermore, the application of a 10-minute evaluation time window referencing Site Reliability Engineering (SRE) standards proved effective in eliminating false positives during flapping spike testing. From a security standpoint, the integration of a cloud firewall and VPN successfully isolated all monitoring service ports from public access entirely.

Keywords: Docker, Grafana, Monitoring, Prometheus, WireGuard VPN

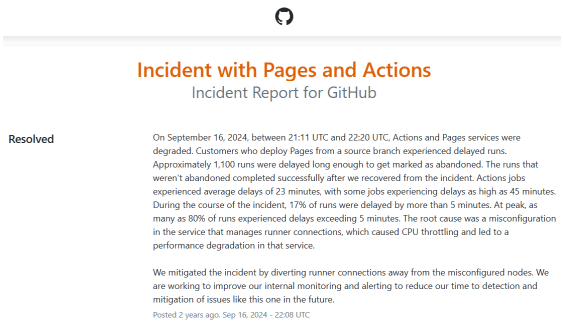
Masalah ketersediaan aplikasi yang terganggu dan risiko kerusakan perangkat keras akibat beban kerja berlebih pada server memerlukan sistem deteksi anomali secara real-time yang akurat. Oleh karenanya, penelitian ini bertujuan untuk mengimplementasikan dan menganalisis performa sistem monitoring serta alerting berbasis Docker menggunakan integrasi Prometheus, Grafana, dan Alertmanager di dalam jaringan VPN WireGuard. Metode penelitian yang digunakan adalah eksperimental dengan melakukan simulasi beban kerja menggunakan stress-ng dan fio pada VPS. Ketika verifikasi sistem dilakukan, terbukti bahwa sistem monitoring dan alerting mampu berjalan di atas arsitektur Docker dengan adanya Docker Engine, Docker Image, dan Docker Container pada server. Ketika simulasi beban kerja dilakukan, performa sistem diukur melalui metrik Anomaly Detection Time (ADT) dan Resolved Detection Time (RDT). Hasil pengujian menunjukkan bahwa sistem sangat responsif dengan rata-rata latensi tertinggi tercatat sebesar 25,4 ms pada pengujian beban kerja CPU. Sementara itu, pada server target (Testserver), rata-rata latensi tertinggi mencapai 28 ms pada skenario beban kerja RAM, dengan latensi deteksi anomali (ADT) sebesar 27,2 ms pada pengujian Disk. Hasil ini selaras dengan ambang batas latensi API sesuai standar acceptable p95 dari SigNoz. Penerapan jendela waktu evaluasi 10 menit yang merujuk pada standar Site Reliability Engineering (SRE) pun terbukti efektif mengeliminasi false positive pada pengujian flapping spike. Dari sisi keamanan, integrasi cloud firewall dan VPN berhasil mengisolasi seluruh port layanan monitoring dari akses publik secara total.

Kata kunci: Docker, Grafana, Monitoring, Prometheus, VPN WireGuard

I. PENDAHULUAN

Transformasi infrastruktur teknologi informasi saat ini telah bergeser secara masif ke arah virtualisasi tingkat aplikasi menggunakan teknologi *container*. Docker telah menjadi standar dalam pengemasan aplikasi karena kemampuannya menjamin konsistensi *deployment* dan efisiensi sumber daya [1]. Karena pergeseran teknologi tersebut, sistem *monitoring* dan *alerting* pada server juga perlu diadaptasi dengan teknologi kontainerisasi. Utilisasi ekstrem yang tidak terpantau pada CPU, RAM, dan Disk server mampu meningkatkan suhu komponen secara signifikan

[2]. Suhu komponen yang terlalu tinggi tidak hanya mengancam ketersediaan layanan, tetapi juga mempercepat penuaan fisik perangkat keras dasar akibat beban kerja yang tidak terpantau [3]. Selain itu, performa layanan atau aplikasi juga dapat terganggu akibat penggunaan *resource* yang berlebihan seperti yang dialami oleh *Github Pages and Actions* [4] pada Gambar 1.



Gambar 1. Gangguan pada Github Pages and Actions

Beberapa penelitian telah membahas upaya pemantauan performa server. Putrayana [5] menerapkan sistem *monitoring* jaringan berbasis IP menggunakan Blackbox Exporter, namun belum mampu menjangkau metrik internal server secara spesifik. Husna & Rosyani [6] menggunakan Zabbix untuk *monitoring* ISP, namun implementasinya yang berbasis instalasi monolitik pada sistem operasi cenderung kurang fleksibel dalam mendukung arsitektur *microservice* dibandingkan sistem *containerized* berbasis Prometheus. Sementara itu, Saory dkk. [7] berhasil mengintegrasikan Prometheus dan Grafana untuk mempercepat deteksi anomali pada VM, namun pengujiannya masih terbatas pada lingkungan lokal tanpa mempertimbangkan aspek keamanan akses ketika layanan monitoring tersebut diekspos ke lingkungan publik. Ratih dkk. [8] memang mengusulkan strategi keamanan VPS berlapis, namun belum menyentuh spesifikasi pengamanan akses eksklusif untuk *pipeline monitoring*.

Meskipun pemenuhan fitur dan keamanan di atas merupakan aspek fungsionalitas yang krusial, efisiensi performa dalam alur pengiriman data notifikasi menjadi parameter krusial dalam keberhasilan sistem monitoring. Hal ini berkaitan erat dengan latensi API, yaitu waktu yang dibutuhkan untuk mengirimkan data dari satu komponen ke komponen lainnya melalui protokol HTTP [9]. Sebagai acuan standar industri, SigNoz (sebuah platform *observability* global) menetapkan ambang batas performa latensi API seperti pada Gambar 1 untuk memastikan sistem tetap responsif.

API type	Good p50	Acceptable p95	Worth investigating at p99
Internal service-to-service (same region)	Under 10ms	Under 50ms	Over 100ms
User-facing REST API	Under 100ms	Under 300ms	Over 500ms
GraphQL (single query)	Under 150ms	Under 500ms	Over 1s
gRPC (unary call)	Under 50ms	Under 200ms	Over 500ms
External/third-party API	Under 200ms	Under 1s	Over 2s

Gambar 1. Ambang Batas Latensi API

Selain penentuan ambang batas performa, ketepatan konfigurasi waktu pada *pipeline monitoring* juga memegang peranan penting untuk mencegah notifikasi berlebih (*alert fatigue*). Berdasarkan prinsip Site Reliability Engineering (SRE) dari Google [10], penentuan parameter—seperti interval *scraping* data yang rapat sebesar 5 detik dan jendela waktu evaluasi (*sliding window*) selama 10 menit—sangat krusial untuk meredam fenomena *alert flapping* (kondisi di mana status peringatan terus berganti secara instan akibat lonjakan metrik sesaat). Integrasi standar SRE ini diperlukan agar sistem mampu mengeliminasi *false positive* dan menjamin bahwa setiap notifikasi yang diteruskan ke administrator merupakan anomali yang valid.

Di samping pemenuhan standar performa dan keandalan operasional tersebut, *scientific gap* yang ditemukan adalah minimnya kajian model integrasi *monitoring* berbasis Docker Container yang memiliki isolasi keamanan ketat; sebagian besar implementasi saat ini masih mengekspos *port* layanan ke jaringan publik. Penelitian ini mengisi celah tersebut dengan mengintegrasikan seluruh layanan *monitoring* (Prometheus, Grafana, Alertmanager, dan *middleware*) di dalam saluran terenkripsi VPN WireGuard dan *cloud firewall*. Adanya VPN WireGuard akan membuat layanan *monitoring* dan *alerting* tetap bisa diekspos ke internet namun dengan akses privat [11]. Kontribusi utamanya adalah penyediaan arsitektur *monitoring* yang *portable*, aman, dan responsif, didukung oleh analisis kuantitatif performa latensi.

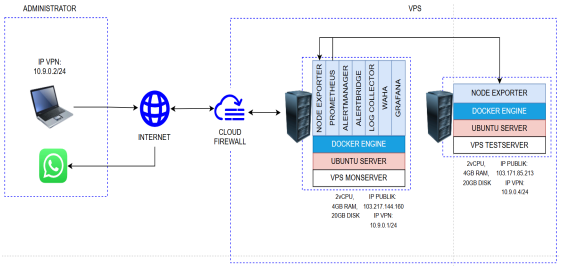
Berdasarkan hal tersebut, penelitian ini bertujuan membangun sistem pemantauan yang aman tanpa mengorbankan responsivitas.

Pertanyaan penelitian yang diajukan adalah: (1) Bagaimana konfigurasi isolasi keamanan Prometheus-Grafana berbasis Docker memanfaatkan VPN WireGuard dan Cloud Firewall? (2) Bagaimana pemenuhan standar latensi (p50 dan p95) pada alur pengiriman data di dalam jaringan terenkripsi tersebut? Dengan demikian, tujuan akhir penelitian ini adalah mengimplementasikan arsitektur tersebut sekaligus menganalisis dampak keamanannya terhadap performa latensi sistem.

II. METODOLOGI PENELITIAN

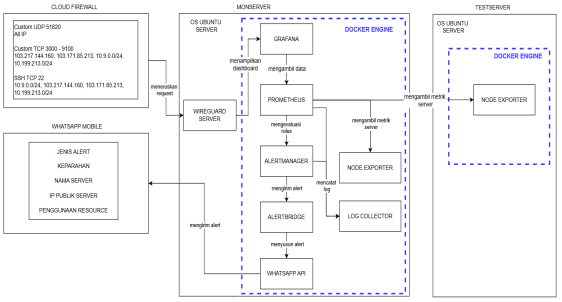
Penelitian ini menggunakan pendekatan eksperimental dengan fokus pada analisis sistem monitoring dan alerting. Objek penelitian adalah VPS dengan spesifikasi 2 vCPU, RAM 4 GB, dan penyimpanan SSD 20 GB yang menjalankan sistem operasi Ubuntu 24.04 LTS. Arsitektur sistem dibangun di atas ekosistem Docker Container untuk menjamin isolasi dan portabilitas layanan. Lingkungan pengujian dikonfigurasi dalam jaringan privat menggunakan protokol VPN WireGuard yang diintegrasikan dengan *cloud firewall* untuk membatasi akses pada *port-port* layanan (22, 3000, 9090, 9093, 9100).

Rancangan sistem dalam penelitian ini terdiri dari server *monitoring* (Monserver) dan server target (Testserver) yang saling terhubung melalui jaringan VPN. Topologi yang menggambarkan konektivitas antar server, administrator, dan proteksi *cloud firewall* melalui jaringan internet diilustrasikan pada Gambar 1.



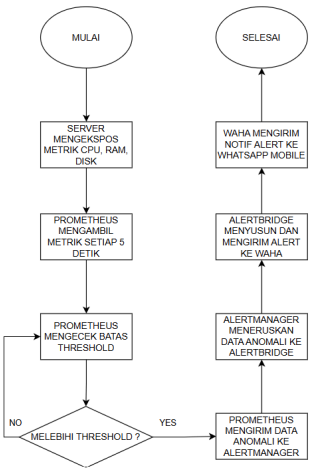
Gambar 2. Topologi Penelitian

Lingkungan Virtual Private Server (VPS) diisolasi secara ketat melalui aturan *cloud firewall* guna mencegah akses tidak sah dari jaringan publik [12]. Di dalam jaringan internal VPS, kontainer Node Exporter mengekspos metrik *resource* yang kemudian diambil oleh Prometheus melalui mekanisme *pull-based* dengan interval *scraping* 5 detik [13]. Prometheus sendiri akan menyimpan metrik-metrik tersebut sesuai perannya sebagai *time-series database* dan melakukan evaluasi sesuai *rules* yang diterapkan [14]. Interaksi fungsional antar komponen di dalam lingkungan Docker, digambarkan melalui arsitektur penelitian pada Gambar 3.



Gambar 3. Arsitektur Penelitian

Evaluasi kueri dilakukan menggunakan jendela waktu rata-rata 10 menit (*sliding window*) untuk menjaga stabilitas deteksi. Alur notifikasi dimulai dari Alertmanager yang meneruskan data anomali ke *middleware* Alertbridge untuk dikonversi menjadi format pesan WhatsApp, kemudian dikirimkan melalui WAHA API ke perangkat administrator. WAHA API sendiri merupakan kontainer yang bertindak sebagai gerbang pengiriman pesan ke aplikasi WhatsApp Mobile [15]. Alur notifikasi dapat dilihat seperti pada Gambar 4.



Gambar 4. Alur Notifikasi

Teknik pengumpulan data dilakukan melalui pengamatan langsung terhadap respon sistem saat diberikan perlakuan berupa *stress test* menggunakan alat bantu *stress-ng* dan *fio* (khusus Disk saja). Skenario pengujian mencakup beban tunggal (CPU, RAM, Disk), beban gabungan, serta kondisi *node down*. Penentuan ambang batas untuk mengkondisikan beban kerja kritis dalam skenario

eksperimen ini merujuk pada standar IBM Cloud Pak System Documentation [16], yang menetapkan tingkat keparahan kritis (*critical severity*) terjadi ketika persentase utilisasi pada seluruh komponen utama—CPU, RAM, maupun Disk—telah melampaui angka 90%.

Dalam penelitian ini, performa monitoring dan *alerting* direpresentasikan oleh nilai latensi API yang terjadi di setiap lompatan (*hop*) komunikasi HTTP antar-komponen sistem (Prometheus, Alertmanager, Alertbridge, dan WAHA API). Sebagai bentuk operasionalisasi dari variabel latensi API tersebut, peneliti menetapkan dua parameter turunan, yaitu Anomaly Detection Time (ADT) dan Resolved Detection Time (RDT). Perhitungan dilakukan menggunakan rumus (1) dan (2) sebagai berikut:

$$ADT = T_{notifReceived} - T_{startsAt} \quad (1)$$

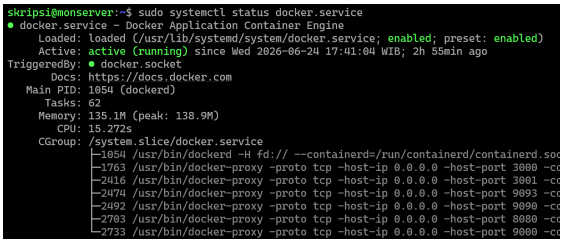
$$RDT = T_{notifReceived} - T_{endsAt} \quad (2)$$

Dalam persamaan di atas, $T_{notifReceived}$ merepresentasikan waktu saat notifikasi diterima oleh kontainer Log Collector, sedangkan $T_{startsAt}$ dan T_{endsAt}adalah *timestamp* validasi anomali dan pemulihan yang dihasilkan oleh mesin evaluasi Prometheus. Data numerik yang diperoleh dari 10 kali pengulangan setiap skenario kemudian diolah menggunakan teknik analisis deskriptif untuk menentukan nilai rata-rata, minimum, dan maksimum. Hasil analisis ini digunakan untuk mengevaluasi efektivitas arsitektur *monitoring* dalam jaringan VPN terhadap beban kerja kritis.

III. HASIL DAN PEMBAHASAN

A. Verifikasi Arsitektur Berbasis Kontainer

Pengujian tahap awal dilakukan untuk memastikan seluruh komponen sistem berjalan secara terisolasi di atas platform Docker. Berdasarkan hasil verifikasi melalui Docker Engine, Docker Image, dan Docker Container, sistem berhasil men-*deploy* tujuh kontainer pada Monserver (Prometheus, Alertmanager, Alertbridge, WAHA, Log Collector, Node Exporter, dan Grafana) serta satu kontainer Node Exporter pada Testserver. Bukti verifikasi dapat dilihat pada Gambar 5, Gambar 6, dan Gambar 7.

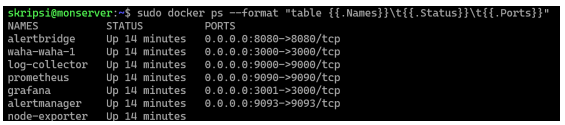


Gambar 5. Verifikasi Docker Engine

Gambar 5 menunjukkan berjalannya Docker Engine sebagai fondasi arsitektur Docker, sedangkan Gambar 6 menunjukkan citra Docker yang digunakan untuk menjalankan aplikasi sistem *monitoring* dan *alerting*.

Gambar 6. Verifikasi Docker Image

Gambar 7 menunjukkan aplikasi sudah berjalan sebagai Docker Container dan mengekspos beberapa *port* layanan.



Gambar 7. Verifikasi Docker Container

Penggunaan arsitektur kontainer ini terbukti memberikan isolasi sumber daya yang baik. Setiap layanan beroperasi menggunakan Docker Image yang konsisten, memastikan bahwa kendala pada satu layanan tidak mengganggu operasional layanan lainnya. Hal ini memverifikasi bahwa rancangan sistem telah memenuhi aspek fungsionalitas distribusi layanan yang portable dan mudah dikelola.

B. Analisis Responsivitas dan Performa Sistem (ADT & RDT)

Inti dari pengujian performa ini adalah mengukur Alert Detection Time (ADT) dan Resolved Detection Time (RDT) saat kedua server (Monserver dan Testserver) berada dalam kondisi beban kerja kritis (*critical load*). Hal ini dilakukan untuk mensimulasikan kondisi riil di mana sistem *monitoring* juga harus mampu memonitor dirinya sendiri dan server lain. Pengujian peforma dilakukan pada komponen CPU, RAM, dan Disk. Hasil pengujian performa dapat dilihat pada Tabel I dan Tabel II.

TABEL I. RATA-RATA LATENSI ADT DAN RDT PADA MONSERVER

Komponen	Rata-rata ADT (ms)	Rata-rata RDT (ms)	Rentang SigNoz
CPU	10	25,4	Acceptable p95
RAM	8,6	21,4	Acceptable p95
Disk	9,4	18,8	Acceptable p95

Tabel I menunjukkan rata-rata latensi ADT dan RDT pada Monserver ketika beban kerja kritis diterapkan. Latensi paling tinggi berada pada angka 25,4ms di komponen CPU. Namun demikian, angka tersebut masih tergolong *acceptable* menurut rekomendasi SigNoz terkait latensi API. Selain pada Monserver, beban kerja kritis juga diterapkan pada Testserver untuk melihat performa latensi ADT dan RDT. Hasil pengujian tercantum pada Tabel II.

TABEL II. RATA-RATA LATENSI ADT DAN RDT PADA TESTSERVER

Komponen	Rata-rata ADT (ms)	Rata-rata RDT (ms)	Rentang SigNoz
CPU	10,2	24,2	Acceptable p95
RAM	7,8	28	Acceptable p95
Disk	27,2	25	Acceptable p95

Pada Tabel II, dapat dilihat bahwa rata-rata latensi tertinggi ada di angka 28ms. Angka ini tidak terpaut jauh dari rata-rata tertinggi pada Tabel I. Hasil ini membuktikan bahwa meskipun beban kerja pada Monserver dalam keadaan kritis, performa deteksi pada server target masih tergolong efektif. Apabila rata-rata pada Tabel I dan Tabel II dirata-ratakan kembali, hasilnya tercantum seperti pada Tabel III.

TABEL III. RATA-RATA LATENSI ADT DAN RDT PADA MONSERVER DAN TESTSERVER

Komponen	Rata-rata ADT (ms)	Rata-rata RDT (ms)	Rentang SigNoz
CPU	10,1	24,8	Acceptable p95
RAM	8,2	24,7	Acceptable p95
Disk	18,3	24,8	Acceptable p95

Hasil pada Tabel III menunjukkan bahwa sekalipun Monserver dan Testserver diberikan beban kerja kritis, sistem *monitoring* dan *alerting* masih mampu menjaga performa dengan rendahnya latensi API. Rata-rata ADT pada skenario beban kerja RAM (8,2 ms) dan CPU (10,1 ms) berhasil mendekati atau memenuhi standar ideal p50. Sementara itu, dalam kondisi beban kerja ekstrem, rata-rata RDT tertinggi yang tercatat pada sistem (24,8 ms) masih berada jauh di bawah ambang batas atas standar p95 (<50 ms).

C. Evaluasi Keamanan Akses Jaringan

Pengujian keamanan dilakukan dengan membandingkan akses *port* layanan melalui internet menggunakan VPN WireGuard dan tidak menggunakannya. Setiap akses yang dilakukan nantinya akan melewati *cloud firewall* untuk difilter berdasarkan *rules* yang diterapkan. Pada pengujian ini, tentunya alamat IP dari VPN diberi akses (*whitelist*) terhadap *port* layanan. Layanan yang diuji meliputi port 22, 3000, 3001, 9090, 9093, dan 9100. Hasil pengujian keamanan akses dapat dilihat pada Tabel IV.

TABEL IV. HASIL PENGUJIAN KEAMANAN AKSES

Skenario	Port	Tanpa VPN	Dengan VPN
Akses SSH	22	Gagal	Berhasil
Akses kontainer WAHA	3000	Gagal	Berhasil
Akses *dashboard* Grafana	3001	Gagal	Berhasil
Akses kontainer Prometheus	9090	Gagal	Berhasil
Akses kontainer Alertmanager	9093	Gagal	Berhasil
Akses metrik Node Exporter	9100	Gagal	Berhasil

Hasil pada Tabel IV menunjukkan bahwa *cloud firewall* secara konsisten memblokir seluruh permintaan akses dari IP publik dengan status "Gagal". Akses ke antarmuka pemantauan dan metrik hanya dapat dilakukan setelah perangkat administrator terhubung ke jaringan VPN. Implementasi ini berhasil mengamankan data metrik server dari upaya *scraping* ilegal maupun serangan langsung pada *port* layanan. Dengan demikian, sistem tidak hanya unggul dari sisi performa deteksi, tetapi juga memiliki lapisan pertahanan yang solid untuk menjaga integritas data pemantauan.

IV. KESIMPULAN DAN SARAN

Penelitian ini sukses mengimplementasikan arsitektur monitoring Prometheus-Grafana berbasis Docker yang terisolasi aman menggunakan VPN

WireGuard dan *cloud firewall* tanpa mengorbankan performa sistem. Berdasarkan hasil eksperimen, integrasi lapisan keamanan berhasil menutup seluruh akses *port* kritis dari jaringan publik secara total. Pengujian kuantitatif menunjukkan sistem sangat responsif dengan akumulasi latensi API (metrik turunan ADT dan RDT) yang secara konsisten berada di bawah 50 ms pada seluruh skenario beban kerja kritis, sehingga memenuhi standar industri *Acceptable p95* dari SigNoz. Selain itu, konfigurasi parameter waktu berbasis prinsip Site Reliability Engineering (SRE) terbukti efektif meredam *alert flapping* dan mengeliminasi *false positive*. Keterbatasan penelitian ini terletak pada ruang lingkup pemantauan yang masih terbatas pada metrik dasar server dan diuji pada server *monitoring* tunggal.

Sebagai saran pengembangan, cakupan objek observabilitas ke depan dapat diperluas secara lebih granular, seperti menggunakan cAdvisor untuk memantau konsumsi *resource* spesifik di dalam setiap kontainer Docker dan Blackbox Exporter untuk analisis performa jaringan (*network monitoring*) antar-layanan. Selanjutnya, pengujian kinerja arsitektur terisolasi ini perlu dikembangkan pada lingkungan server *multi-node* atau orkestrasi seperti Kubernetes guna mengevaluasi pengaruh *overhead* enkripsi pada skala jaringan yang lebih masif. Terakhir, otomatisasi rotasi kunci (key rotation) pada VPN WireGuard serta penerapan penentuan ambang batas dinamis (*dynamic thresholding*) menjadi masalah yang sangat potensial untuk menghasilkan ekosistem pemantauan yang lebih aman dan adaptif.

V. REFERENSI

[1] Docker, “What is Docker?,” Docker Documentation, 2026. https://docs.docker.com/get-started/docker-overview/ (accessed June 23, 2026).

[2] L. Ladge and Y. S. Rao, “Analysis of Core Temperature Dynamics in Multi-Core Processors,” Journal of Low Power Electronics and Applications, vol. 15, no. 4, p. 68, Dec. 2025, doi: 10.3390/jlpea15040068.

[3] F. Gabbay and A. Mendelson, “Asymmetric aging effect on modern microprocessors,” Microelectronics Reliability, vol. 119, p. 114090, Apr. 2021, doi: 10.1016/j.microrel.2021.114090. Q2 SCOPUS

[4] “Incident with Pages and Actions,” Githubstatus, Sept. 16, 2024. https://www.githubstatus.com/incidents/69sb0f8lydp4 (accessed May 10, 2026).

[5] R. A. Putrayana, I. Akbar, and Wasis Haryono, “Implementasi Sistem Monitoring Jaringan Rumah Sakit Menggunakan Blackbox Exporter, Prometheus, Grafana, dan Alertmanager,” Sultra Jurnal Pengabdian Masyarakat, vol. 2, no. 1, pp. 114–122, 2025, doi: 10.54297/sjpm.v2i1.1219.

[6] M. A. Husna and P. Rosyani, “Implementasi Sistem Monitoring Jaringan dan Server Menggunakan Zabbix yang Terintegrasi dengan Grafana dan Telegram,” JURIKOM (Jurnal Riset Komputer), vol. 8, no. 6, pp. 247–255, Dec. 2021, doi: 10.30865/jurikom.v8i6.3631.

[7] [18]F. Saory, J. H. Jaman, and C. Author, “SISTEM MONITORING SERVER MENGGUNAKAN PROMETHEUS DAN GRAFANA DENGAN INTEGRASI SLACK DI PT. ATLAS LINTAS INDONESIA,” Jurnal Informatika dan Teknik Elektro Terapan, vol. 13, no. 3S1, Oct. 2025, doi: 10.23960/jitet.v13i3s1.8024.

[8] Ratih, N. Muniroh, and F. Mahardika, “Strategi Keamanan VPS Menggunakan Pendekatan Berlapis: Studi Kasus Integrasi Cloudflare, 2FA, dan Monitoring,” Blend Sains Jurnal Teknik, vol. 4, no. 2, pp. 449–460, Nov. 2025, doi: https://doi.org/10.56211/blendsains.v4i2.1315 .

[9] M. Kaur, “API Latency Explained - How to Measure, Diagnose, and Reduce It,” SigNoz, Mar. 09, 2026. https://signoz.io/guides/what-is-api-latency/ (accessed June 23, 2026).

[10] B. Beyer, C. Jones, J. Petoff, and Niall Richard Murphy, Site reliability engineering : how Google runs production systems. Sebastopol, Ca: Oreilly, 2016, p. 118. Accessed: Apr. 26, 2026. [Online]. Available: http://repo.darmajaya.ac.id/4636/1/Site%20Reliability%20Engineering_%20How%20Google%20Runs%20Production%20Systems%20%28%20PDFDrive%20%29.pdf

[11] M. F. Yuce et al., “WireGuard-AES: Hardware based encryption to WireGuard for VPN gateways,” SoftwareX, vol. 31, p. 102314, Sept. 2025, doi: 10.1016/j.softx.2025.102314.

[12] S. A. Imawan and S. Dwiasnati, “IMPLEMENTASI PEMBLOKAN SITUS DENGAN FIREWALL LAYER 7 PROTOCOL MENGGUNAKAN METODE NDLC PADA ROUTER MIKROTIK,” MULTINETICS, vol. 11, no. 1, pp. 22–34, May 2025, doi: 10.32722/multinetics.v11i1.6844.

[13] T. Leppänen, “Data visualization and monitoring with Grafana and Prometheus,” Bachelor’s Thesis, Turku University of Applied Sciences, 2021. Accessed: Feb. 06, 2026. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/512860/Turkuamk_Bachelors_Thesis_Leppanen_Tiia.pdf

[14] L. Giamattei et al., “Monitoring tools for DevOps and microservices: A systematic grey literature review,” Journal of Systems and Software, vol. 208, p. 111906, Feb. 2024, doi: 10.1016/j.jss.2023.111906.

[15] F. T. Zany, A. Sujud, and J. Nurgaza, “Pengembangan Sistem Manajemen Whatsapp API (Application Programming Interface) dengan Menggunakan Whatsapp-web.js,” Journal of Informatics and Computer Science, vol. 10, no. 2, pp. 140–153, Oct. 2024, doi: https://doi.org/10.33143/jics.v10i2.4761.

[16] IBM Corp., “IBM Cloud Pak System W4600 Documentation,” IBM, United States, 2024. Accessed: Apr. 04, 2026. [Online]. Available: https://www.ibm.com/docs/en/SSDLT6_2.3.3/pdf/cloud-pak-system-w4600-2.3.3-documentation.pdf