# Perancangan Standarisasi Sistem *Dashboard* untuk Pengelolaan Sumber Daya *Server* Fakultas Rekayasa Industri Menggunakan New Relic

1st Muhammad Adam Zaidan
Departemen Sistem Informasi
Universitas Telkom
Bandung, Indonesia
adamzaidan@student.telkomuniversity.ac.id

2nd Mochamad Teguh Kurniawan
Departemen Sistem Informasi
Universitas Telkom
Bandung, Indonesia
teguhkurniawan@telkomuniversity.ac.id

3rd Umar Yunan Kurnia Septo Hediyanto
Departemen Sistem Informasi
Universitas Telkom
Bandung, Indonesia
umaryunan@telkomuniversity.ac.id

***Abstrak* — Perkembangan teknologi mengharuskan pengelolaan sumber daya *server* yang efisien, terutama di Fakultas Teknik Industri Universitas Telkom. Meningkatnya kompleksitas layanan TI akibat bertambahnya pengguna dan aplikasi menuntut sistem pemantauan yang efektif. Meskipun menggunakan New Relic, implementasinya belum optimal dalam analisis kinerja waktu nyata dan pengambilan keputusan. Penelitian ini bertujuan mengembangkan dan mengoptimalkan *dashboard* berbasis New Relic untuk manajemen *server* yang lebih efisien. Melalui metodologi PDCA *(Plan-Do-Check-Act*), penelitian ini mengidentifikasi tantangan alokasi sumber daya dan merekomendasikan solusi dengan memanfaatkan fitur New Relic seperti metrik khusus, peringatan, dan visualisasi. Data diperoleh melalui observasi, wawancara, dan kuesioner, kemudian dianalisis dengan menggunakan metode expert judgement dari laboran. Hasil yang diharapkan meliputi peningkatan kinerja *server*, efisiensi operasional, serta layanan TI yang lebih baik untuk keperluan akademik dan administrasi. Penelitian ini juga menawarkan kerangka kerja bagi institusi lain untuk mengadopsi alat pemantauan seperti New Relic.**

***Kata kunci*— Sistem *dashboard*, manajemen sumber daya, *server*, New Relic, Telkom University.**

## I. PENDAHULUAN

Seiring dengan pesatnya transformasi digital, institusi pendidikan menghadapi tantangan dalam memastikan pengelolaan sumber daya *server* yang efisien dan handal. Di Fakultas Rekayasa Industri Universitas Telkom, *server* memainkan peran krusial dalam mendukung berbagai aktivitas akademik dan administratif, seperti pendaftaran mata kuliah hingga pengelolaan data mahasiswa. Namun, meningkatnya jumlah pengguna serta kompleksitas layanan TI menyebabkan beban pada *server* semakin berat, yang jika tidak dikelola dengan baik dapat menimbulkan risiko downtime dan menghambat operasional institusi.

Sistem *dashboard* telah menjadi solusi populer dalam mengelola performa *server*. Penelitian terdahulu oleh (Vilar, 2010) menunjukkan bahwa *dashboard* tidak hanya digunakan untuk memantau kondisi sistem secara *real-time*, tetapi juga untuk mendukung pengambilan keputusan berbasis data. Selain itu, studi oleh (Timbul Sigiro, 2016) menawarkan konsep transparansi dalam pengelolaan sumber daya melalui sistem *dashboard*, yang dapat mengidentifikasi alokasi sumber daya secara terperinci. Di Fakultas Rekayasa Industri, New Relic telah digunakan sebagai platform pemantauan *server* dengan fitur-fitur seperti visualisasi metrik CPU, memori, dan jaringan. Namun, implementasinya masih terbatas pada pemantauan dasar, tanpa memanfaatkan fitur lanjutan seperti metrik khusus, alert, dan analisis yang lebih mendalam.

Keterbatasan ini menimbulkan sejumlah permasalahan utama, termasuk kurangnya standar *dashboard* yang memudahkan pengguna dalam memantau performa *server* secara konsisten dan responsif. Selain itu, tantangan lain adalah minimnya optimalisasi fitur-fitur New Relic untuk mendukung pengambilan keputusan yang cepat dan mitigasi risiko *server*. Hal ini berpotensi menurunkan efisiensi operasional *server* dan meningkatkan waktu respon terhadap insiden.

Penelitian ini bertujuan untuk merancang dan mengimplementasikan sistem *dashboard* berbasis New Relic yang terstandarisasi di Fakultas Rekayasa Industri. Dengan pendekatan PDCA (Plan-Do-Check-Act), penelitian ini diharapkan dapat meningkatkan efisiensi alokasi sumber daya *server*, mempercepat waktu respon terhadap insiden, serta menyajikan panduan sistematis bagi institusi lain yang menghadapi masalah serupa. Hasil akhir dari penelitian ini bertujuan untuk memberikan solusi sistematis dalam meningkatkan kinerja operasional *server* dan menyediakan standar *dashboard* untuk pengelolaan yang lebih efektif.

## II. KAJIAN TEORI

### A. Pengelolaan Sumber Daya *Server*

*Server* adalah sistem atau program yang menerima permintaan dari satu atau beberapa sistem atau program klien untuk melakukan aktivitas yang memungkinkan klien menyelesaikan tugas tertentu (MD Hanson, 2000). Sementara itu, Pengelolaan sumber daya *server* adalah proses perencanaan, implementasi, dan pengawasan terhadap penggunaan perangkat keras dan perangkat lunak pada *server*. Hal ini mencakup pemantauan kapasitas *server*, penggunaan CPU, RAM, penyimpanan, dan uptime untuk memastikan ketersediaan dan kinerja yang optimal. Teori yang mendasari pengelolaan sumber daya *server* meliputi

manajemen infrastruktur TI, pemantauan kinerja, serta konsep scalability dan load balancing.

B. Sistem *Dashboard*

Sistem *dashboard* adalah antarmuka visual yang menampilkan data secara ringkas dan terstruktur untuk memudahkan pengambilan keputusan. Dalam konteks pengelolaan sumber daya *server*, *dashboard* menyediakan informasi terkait status dan kinerja *server*, serta mempermudah analisis dan pemantauan *real-time*. Teori dasar yang terkait dengan *dashboard* mencakup konsep visualisasi data, representasi grafis, dan prinsip desain interaktif yang efisien.

C. New Relic

New Relic adalah *software* observabilitas memberikan wawasan mendalam tentang kinerja aplikasi, *server*, dan database. Teori yang terkait dengan New Relic mencakup pemantauan kinerja sistem, pengumpulan data secara *real-time*, serta analisis dan pelaporan metrik yang relevan untuk pengelolaan sumber daya TI. New Relic dapat mengurangi waktu pemulihan atau *Mean Time To Repair* (MTTR) dengan data yang dapat diambil tindakan secara *real-time*, menghubungkan perubahan performa host dengan perubahan konfigurasi. New Relic memastikan pemantauan yang dilakukan selalu sesuai dengan kondisi aktual dan membantu dalam mengoptimalkan kinerja aplikasi maupun *server*.

## III. METODE

Memberikan gambaran rancangan penelitian yang meliputi prosedur atau langkah-langkah penelitian, waktu penelitian, sumber data, cara perolehan data dan menjelaskan metode yang akan digunakan dalam penelitian [10 pts].

A. Rancangan Penelitian

Penelitian ini menggunakan metode Plan, Do, Check, Act (PDCA), yang merupakan siklus perbaikan berkelanjutan. Metode ini terdiri dari empat tahap utama, yaitu sebagai berikut.

1. *Plan* (Perencanaan)

Merupakan tahapan di mana peneliti akan merumuskan tujuan penelitian, melakukan analisis masalah, dan merencanakan rancangan sistem *dashboard* untuk pengelolaan sumber daya *server*. Ini termasuk identifikasi metrik yang akan dipantau dan perencanaan penggunaan New Relic sebagai alat pemantauan.

2. *Do* (Pelaksanaan)

Tahap ini melibatkan implementasi sistem *dashboard* yang telah dirancang. Peneliti akan mengumpulkan data sumber daya *server*, mengatur New Relic untuk memantau kinerja *server*, serta mengembangkan dan menguji *dashboard* sesuai dengan kebutuhan yang telah dianalisis.

3. *Check* (Pemeriksaan)

Peneliti akan melakukan evaluasi kinerja sistem. Evaluasi ini mencakup analisis data yang dikumpulkan, pemantauan kinerja *server* melalui *dashboard*, serta mendapatkan umpan balik dari pengguna untuk mengevaluasi sejauh mana sistem memenuhi tujuan yang telah ditetapkan.

4. *Act* (Tindakan Perbaikan)

Pada tahap ini peneliti akan melakukan perbaikan dan penyempurnaan sistem *dashboard* untuk meningkatkan efektivitas dan efisiensinya. Siklus PDCA akan diulang untuk terus memperbaiki sistem berdasarkan evaluasi yang diterima.

B. Cara Perolehan Data

Data dalam penelitian ini akan diperoleh melalui tiga metode utama, yaitu observasi lapangan, wawancara dengan stakeholder, dan penyebaran kuisioner, untuk mendapatkan informasi yang relevan mengenai pengelolaan sumber daya *server* dan kebutuhan pengguna terhadap sistem *dashboard*.

1. Observasi

Observasi dilakukan terhadap penggunaan sistem yang ada untuk mengidentifikasi masalah yang berkaitan dengan pengelolaan sumber daya *server*. Observasi juga akan dilakukan untuk melihat sejauh mana sistem *dashboard* yang ada dapat memenuhi kebutuhan pengguna.

2. Wawancara

Wawancara akan dilakukan dengan staf IT dan pengguna sistem untuk mengumpulkan informasi terkait kebutuhan dan kendala yang dihadapi dalam pengelolaan *server*. Wawancara ini juga akan memberikan wawasan tentang ekspektasi terhadap sistem *dashboard* yang akan dikembangkan.

3. Pengumpulan Data Sistem

engumpulan data terkait kinerja *server* dan sumber daya yang digunakan akan dilakukan melalui New Relic. Data ini akan meliputi penggunaan CPU, RAM, penyimpanan, dan status *server* dalam periode waktu tertentu yang akan dipantau melalui *dashboard*.

C. Metode Evaluasi

Evaluasi dalam penelitian ini dilakukan pada tahap Check dari siklus PDCA untuk mengukur sejauh mana sistem *dashboard* yang diimplementasikan memenuhi tujuan yang telah ditetapkan. Metode evaluasi yang digunakan mencakup pendekatan kualitatif dan kuantitatif, dengan fokus pada umpan balik pengguna dan analisis kinerja sistem.

1. Evaluasi Kinerja Sistem

Kinerja sistem *dashboard* akan dievaluasi berdasarkan data yang dikumpulkan dari New Relic, yang mencakup metrik seperti penggunaan CPU, memori, dan ketersediaan *server*. Analisis ini bertujuan untuk memastikan bahwa *dashboard* memberikan informasi yang akurat dan *real-time* tentang status sumber daya *server*.

2. Umpan Balik Pengguna dan Stakeholder

Umpan balik dari pengguna dan stakeholder dikumpulkan melalui wawancara dan kuisioner. Pengguna akan diminta untuk memberikan masukan terkait kemudahan penggunaan, fitur-fitur yang disediakan, serta seberapa efektif sistem dalam membantu pengelolaan sumber daya *server*. Umpan balik ini akan membantu untuk mengidentifikasi area yang perlu diperbaiki dalam sistem *dashboard*.

3. Studi Kasus Implementasi

Studi kasus dilakukan untuk menganalisis implementasi sistem *dashboard* pada beberapa unit atau departemen yang telah menggunakan sistem. Dengan membandingkan kondisi sebelum dan setelah penggunaan *dashboard*, studi kasus ini

bertujuan untuk melihat dampak langsung dari penggunaan *dashboard* terhadap efisiensi pengelolaan *server*.

4. Penilaian ROI (*Return on Investment*)

Penilaian ROI dilakukan untuk mengukur apakah investasi dalam pengembangan dan implementasi sistem *dashboard* memberikan manfaat yang sesuai dengan biaya yang dikeluarkan. Penilaian ini melibatkan perhitungan biaya yang dikeluarkan untuk pengembangan dan implementasi sistem dibandingkan dengan peningkatan efisiensi dan pengelolaan sumber daya *server* yang tercapai setelah sistem diimplementasikan.

## IV. HASIL DAN PEMBAHASAN

Bab ini akan menjelaskan hasil dari penelitian serta pembahasan terkait desain dan implementasi sistem *dashboard* untuk pengelolaan sumber daya *server* menggunakan New Relic. Pembahasan ini mencakup proses perancangan, implementasi, pengujian sistem, serta *feedback* dari pengguna dan *expert judgement.*

### A. Desain dan Perancangan Sistem

Desain sistem *dashboard* yang dikembangkan berfokus pada pemantauan sumber daya *server* secara *real-time*. Sistem ini memanfaatkan New Relic untuk mengumpulkan data kinerja *server* dan menampilkan informasi tersebut dalam bentuk visualisasi yang mudah dipahami. *Dashboard* yang dirancang menampilkan metrik-metrik penting seperti penggunaan CPU, memori, ketersediaan *server*, dan lainnya, yang memungkinkan administrator untuk melakukan pemantauan secara lebih efisien dan cepat.

Pemantauan jaringan adalah aspek penting dalam manajemen jaringan, yang memungkinkan operator untuk mengidentifikasi perilaku jaringan dan status komponennya. Hal ini mendukung rekayasa lalu lintas, kualitas layanan, serta deteksi anomali untuk pengambilan keputusan yang lebih baik (Tsai et al., 2018). Standarisasi *dashboard* monitoring bertujuan untuk menciptakan antarmuka yang konsisten, efektif, dan mudah digunakan, sehingga meningkatkan visibilitas data, mempercepat respons terhadap peristiwa kritis, serta mendukung konsistensi dan kemudahan penggunaan.

Standarisasi ini melibatkan berbagai elemen, seperti desain tata letak yang menampilkan metrik penting, sistem peringatan berkode warna, dan navigasi intuitif. Selain itu, penerapan praktik terbaik industri, seperti ITIL dan NIST, memastikan *dashboard* sesuai standar keamanan dan regulasi. Dengan visualisasi data yang terstruktur dan pelaporan yang sederhana, pengguna dapat menganalisis kinerja jaringan dengan cepat dan efisien. Pendekatan ini tidak hanya meningkatkan efisiensi operasional, tetapi juga mendukung kebutuhan strategis pengawasan *server* dengan memberikan pengalaman yang andal dan mudah beradaptasi bagi semua pengguna.

Dalam sebuah *dashboard* pemantauan jaringan, metrik utama berperan penting sebagai indikator kinerja utama (*Key Performance Indicator*/KPI) yang memberikan gambaran langsung mengenai kondisi dan kesehatan jaringan. Berikut merupakan metrik yang dianggap esensial dalam memastikan performa jaringan.

TABEL 1
METRIK UTAMA PADA *DASHBOARD*

| Metrik | Deskripsi | Alasan | Rumus |
|---|---|---|---|
| Bandwidth Usage | Kapasitas data yang digunakan dalam periode waktu tertentu | Memastikan kapasitas jaringan tidak melebihi batas menghindari *bottleneck.* | $Bandwidth\ Usage = \frac{Jumlah\ data\ dikirim\ /\ diterima\ (bits)}{Waktu(detik)}$ (1) |
| Memory Usage | Persentase penggunaan memori *server* | Memantau apakah *server* memiliki cukup memori untuk menjalankan layanan tanpa terjadinya kelebihan beban. | $Memory\ Usage = \frac{Memori\ yang\ digunakan}{Total\ memori\ tersedia} \times 100\%$ (2) |
| CPU Utilization | Persentase penggunaan prosesor dalam memproses permintaan data | Mengidentifikasi kemungkinan *overload* yang dapat mempengaruhi performa aplikasi | $CPU\ Usage = \frac{Waktu\ CPU\ aktif}{Total\ CPU\ tersedia} \times 100\%$ (3) |
| RTT (Round Trip Time) | Waktu perjalanan paket data dari pengirim ke penerima dan kembali. | Mengukur *latency* yang dapat mempengaruhi performa aplikasi | $RTT = 2 \times Propagationdelay$ (4) Di mana: Propagation delay merupakan lama waktu yang dibutuhkan untuk sebuah *packet* untuk mencapai destinasinya |
| Packet Loss | Persentase paket data yang hilang selama transmisi | Mendeteksi gangguan jaringan yang dapat menyebabkan data tidak sampai ke tujuan | $P = \frac{(Packets\ Sent - Packets\ Received)}{Packets\ Sent} \times 100\%$ (5) Di mana: P adalah jumlah *packet loss* |
| Jitter | Variasi dalam waktu kedatangan paket data | Memastikan kestabilan koneksi, terutama untuk aplikasi *real-time* seperti VoIP | $J_i = \lvert D_i - D_{i-1} \rvert$ (6) Di mana: - $D_i$ adalah waktu antar kedatangan paket ke-*i*. - $D_{i-1}$ adalah waktu antar kedatangan paket sebelumnya |
| Network Throughput | Kecepatan transfer data melalui sebuah jaringan | Mengukur kecepatan transfer data, apakah sesuai dengan standar yang sudah ditetapkan. | $Throughput(Mbps) = \frac{Total\ Data\ Transferred}{TimeInterval(s)} \times 8$ (7) |

Di samping itu, agen jaringan seperti New Relic Infrastructure Agent harus diinstal pada *environment* yang berjalan untuk menampilkan data dan metrik di *dashboard*. Berikut metode pengambilan data untuk masing-masing metrik.

TABEL 2
METODE PENGAMBILAN DATA

| No | Metrik | Proses Pengambilan |
|---|---|---|
| 1. | Bandwidth Usage | 1. Agen membaca *counter* data dari NIC melalui SNMP/log OS<br>2. Menghitung perubahan nilai *counter* dalam interval waktu tertentu |
| 2. | Memory Usage | 1. Agen mengakses statistik memori dari OS API<br>2. Agen membaca total dan penggunaan memori<br>3. Menghitungnya menggunakan formula. |
| 3. | CPU Utilization | 1. Agen mengambil data waktu aktif dan idle CPU dari kernel<br>2. Menghitung persentase penggunaannya. |
| 4. | RTT | 1. Agen mengirim paket ICMP ke *server* tujuan dan mencatat waktu respons untuk menghitung perbedaan waktu perjalanan. |
| 5. | Packet Loss | 1. Agen mencatat total paket terkirim/diterima melalui ICMP atau SNMP<br>2. Menghitung persentase paket yang hilang. |
| 6. | Jitter | 1. Agen mencatat perbedaan waktu kedatangan paket menggunakan ICMP atau UDP<br>2. Menghitung variasi terhadap waktu rata-rata. |

B. Spesifikasi Hardware dan Software

Penelitian ini menggunakan New Relic sebagai platform utama untuk mengembangkan dan meningkatkan *dashboard* pemantauan *server* di Fakultas Rekayasa Industri Telkom University. Spesifikasi perangkat keras dan lunak dirancang untuk mendukung performa optimal dalam pengumpulan, pengolahan, dan analisis data secara *real-time*, dengan visualisasi yang mudah dipahami oleh berbagai jenis pengguna, baik pemula maupun berpengalaman. Pengembangan dan pengujian sistem *dashboard* ini dilakukan pada lingkungan Amazon AWS menggunakan *instance* EC2, Linux Debian, serta perangkat pribadi penulis sebagai infrastruktur utama.

Spesifikasi perangkat keras atau *hardware* yang digunakan dalam pengujian dan perancangan sistem monitoring ini adalah laptop Asus TUF Dash F15 FX516PE, yang berfungsi sebagai perangkat utama untuk pengujian dan host *server* monitoring. Laptop ini memiliki spesifikasi sebagai berikut.

TABEL 3
SPESIFIKASI HARDWARE

| No | Spesifikasi | Deskripsi |
|---|---|---|
| 1. | *Platform* | Intel H *Series* |
| 2. | *Processor* | Intel Core i7 11370H @ 3.30 GHz |
| 3. | *Memory* | 8GB x 2 DDR4 @ 3200 MHz |
| 4. | *Storage* | 512GB x 2 SSD NVMe |
| 5. | *WLAN* | Intel(R) Wi-Fi 6 AX201 160MHz |
| 6. | LAN Interface | Realtek RTL8168 Ethernet Controller |
| 7. | OS | Windows 11 23H2 Insider Build 64-bit |

Selain itu, diperlukan pula spesifikasi *software* dalam pengujian ini. Perangkat lunak yang digunakan adalah sebagai berikut.

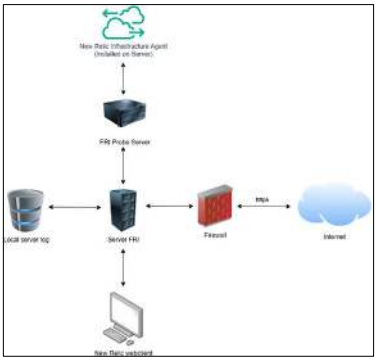
TABEL 3
SPESIFIKASI SOFTWARE

| NO | Komponen | Detail |
|---|---|---|
| **Platform Monitoring** | | |
| 1. | New Relic One | Platform utama untuk memonitor *server*, aplikasi, dan traffic jaringan |
| 2. | New Relic Infrastructure Agent | Untuk mengumpulkan metric *server* seperti CPU, memori, dan metrik jaringan seperti *bandwidth, throughput, RTT,* dsb. |
| 3. | New Relic APM Agent | Untuk memonitor performa aplikasi *backend* (apabila diperlukan). |
| 4. | New Relic Browser *Monitoring* | Untuk memantau kinerja *frontend* aplikasi (apabila diperlukan). |
| 5. | Site24x7 | Sebagai salah satu alat *monitoring* yang dijadikan pembanding terhadap New Relic |
| 6. | Grafana | Sebagai salah satu alat *monitoring* yang dijadikan pembanding terhadap New Relic |
| **Tools Pendukung** | | |
| 7. | Wireshark | Untuk analisis lalu lintas data jaringan |
| 8. | Postman | Untuk pengujian endpoint API |
| 9. | New Relic Insights | Untuk analisis data dan pembuatan query metrik menggunakan NRQL |
| **Standar dan Protokol** | | |
| 10. | NRQL (New ReliC Query Language) | Untuk menulis query data metrik di New Relic |
| 11. | TLS | Mengamankan komunikasi data antara *server* dan New Relic |
| 12. | HTTP/HTTPS | Protokol komunikasi antar sistem |

C. Diagram Topologi Dasar

Topologi ini menggambarkan arsitektur integrasi New Relic untuk memantau *server* melalui New Relic *Infrastructure Agent* yang mengumpulkan data dari FRI Probe *Server* dan *Server* FRI. *Server* ini terhubung dengan log lokal untuk menyimpan data sementara dan berkomunikasi dengan internet melalui *firewall* yang mengamankan transmisi data menggunakan HTTPS. Data yang dikumpulkan kemudian dianalisis dan ditampilkan melalui New Relic Webclient untuk pemantauan kinerja secara *real-time*.

Berikut merupakan desain topologi jaringan yang dirancang untuk proses monitoring *server*, khususnya sistem yang menggunakan New Relic. Diagram ini mencakup komponen-komponen utama, seperti perangkat pengguna (client), agent monitoring New Relic, *server cloud* FRI.



GAMBAR 1
Diagram Topologi Infrastruktur Jaringan FRI

Alur kerja network monitoring agent yang terpasang pada *server* dimulai saat agent New Relic yang sudah terpasang pada *Server* FRI mengumpulkan data performa. Data tersebut kemudian dikirimkan ke *server cloud* New Relic melalui koneksi internet yang aman (HTTPS) dan firewall. Selanjutnya, *server cloud* memproses data yang diterima untuk keperluan analisis dan visualisasi. Hasil analisis ini kemudian ditampilkan melalui *dashboard* pada New Relic Web Client, yang dapat diakses oleh admin atau user menggunakan PC.

D. Implementasi Sistem

Implementasi dimulai dengan instalasi New Relic Infrastructure Agent pada *server* Fakultas Rekayasa Industri. Mengingat alasan keamanan dan keberlanjutan operasional, eksperimen tidak dilakukan langsung pada *server* FRI. Sebagai alternatif, dibuatlah *server* simulasi dengan spesifikasi sebagai berikut, berdasarkan wawancara dengan laboran pengurus *server* FRI:

a. RAM: 4 GB
b. CPU: 1 Virtual CPU (vCPU)
c. Storage: 2 TB

*Server* ini dirancang untuk mereplikasi karakteristik dasar *server* FRI, sehingga pengujian dapat dilakukan tanpa mengganggu operasional *server* FRI.

1. Proses Instalasi New Relic Agent

Langkah-langkah instalasi New Relic Infrastructure Agent pada *server* simulasi di representasikan dengan flowchart berikut:


GAMBAR 2
*Flowchart* Instalasi *Monitoring Agent* New Relic

Pada *flowchart* di atas, alur instalasi New Relic Infrastructure Agent dimulai dengan persiapan lingkungan *server* yang digunakan. Proses ini meliputi pembaruan sistem operasi Linux (Ubuntu 20.04) dan memastikan koneksi internet stabil untuk mengunduh paket agen. Setelah itu, *repository* New Relic ditambahkan, diikuti dengan pengunduhan dan instalasi paket agen menggunakan perintah terminal yang sesuai.

Setelah instalasi, konfigurasi dilakukan dengan menambahkan API Key pada file konfigurasi 'newrelic-infra.yml', kemudian layanan agen dijalankan dan diatur agar otomatis berjalan setelah restart. Verifikasi instalasi dapat dilakukan dengan memeriksa status agen dan log koneksi, serta memastikan data terkirim ke *dashboard* New Relic. Terakhir, di *dashboard* New Relic, *server* simulasi yang telah terinstalasi dapat diverifikasi melalui menu Infrastructure dan All Entities.

2. Konfigurasi *Dashboard* New Relic

Setelah instalasi dan konfigurasi New Relic Infrastructure Agent pada *server* simulasi, langkah berikutnya adalah menyiapkan *dashboard* di New Relic untuk memantau metrik yang relevan. *Dashboard* ini dirancang untuk visualisasi data *real-time* dan historis terkait kesehatan *server*, memudahkan pengawasan performa. Tahapan konfigurasi *dashboard* dimulai dengan melakukan avigasi ke menu Infrastructure untuk mengakses data monitoring *server*.

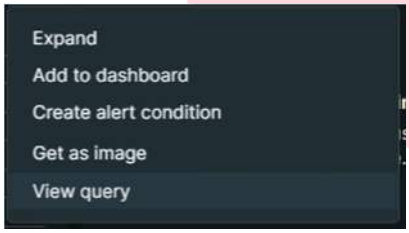
Lalu, dilakukan konfigurasi *widget dashboard* untuk menampilkan metrik-metrik penting yang relevan dengan kebutuhan monitoring *server* Fakultas Rekayasa Industri. Adapun metrik yang dikonfigurasikan sebagai berikut:

TABEL 4
KONFIGURASI *DASHBOARD* NEW RELIC

| Metrik | Deskripsi Widget | Tujuan |
|---|---|---|
| *CPU Usage* | Grafik garis (line chart) yang menunjukkan persentase penggunaan CPU secara *real-time*. | Memantau beban kerja CPU untuk mengidentifikasi potensi *overload*. |
| *Memory Usage* | Widget berisi total, penggunaan, dan sisa kapasitas RAM serta swap memory dalam bentuk grafik batang (bar chart). | Mengidentifikasi potensi *bottleneck* akibat penggunaan memori yang tinggi. |
| *Disk Usage* | Visualisasi dalam bentuk donut chart untuk memantau penggunaan disk berdasarkan partisi. | Memastikan ketersediaan ruang penyimpanan untuk aktivitas *server*. |
| *Network Latency* | Grafik yang menampilkan latensi jaringan dalam milidetik (ms). | Mengukur kualitas koneksi jaringan *server* dengan klien atau aplikasi eksternal. |
| *Alerts & Thresholds* | Tabel peringatan yang memuat status aktif dari semua ambang batas (*threshold*s) yang telah diatur. | Memberikan peringatan dini kepada administrator terhadap kondisi abnormal *server*. |
| *CPU Usage* | Grafik garis (line chart) yang menunjukkan persentase penggunaan CPU secara *real-time*. | Memantau beban kerja CPU untuk mengidentifikasi potensi *overload*. |

| Metrik | Deskripsi Widget | Tujuan |
|---|---|---|
| *Memory Usage* | Widget berisi total, penggunaan, dan sisa kapasitas RAM serta swap memory dalam bentuk grafik batang (bar chart). | Mengidentifikasi potensi *bottleneck* akibat penggunaan memori yang tinggi. |
| *Disk Usage* | Visualisasi dalam bentuk donut chart untuk memantau penggunaan disk berdasarkan partisi. | Memastikan ketersediaan ruang penyimpanan untuk aktivitas *server*. |

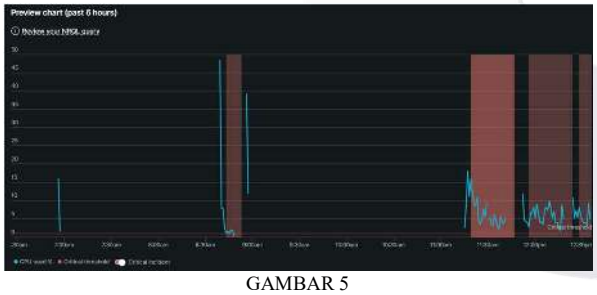
Setiap metrik pada *dashboard* dilengkapi dengan pengaturan ambang batas (*thresholds*) untuk memberi peringatan otomatis saat parameter mencapai nilai kritis. Pengaturan ini dilakukan dengan mengklik tiga titik di bagian kanan atas metrik yang diinginkan.




GAMBAR 3
*Pop-up* menu *dashboard* New Relic

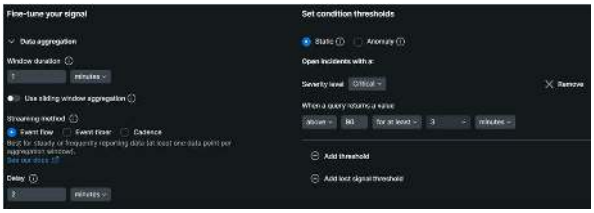
Alert dibuat menggunakan NRQL (New Relic Query Language), yang penulisannya mirip dengan SQL. Sebagai contoh, untuk membuat alert saat penggunaan CPU melebihi 90%, query berikut dipakai untuk memanggil dan menampilkan rata-rata penggunaan CPU dalam bentuk grafik.


GAMBAR 4
Query untuk alerting berdasarkan CPU Usage



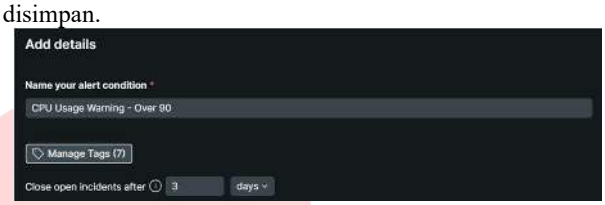

GAMBAR 5
Grafik *CPU Usage* 6 jam terakhir

Setelah itu, atur *threshold* untuk alert. Berdasarkan wawancara dengan laboran, *threshold* ideal untuk *server* FRI adalah jika penggunaan CPU melebihi 90% selama 3 menit. Ketika kondisi tersebut tercapai, sistem akan memberikan notifikasi ke email yang terhubung.




GAMBAR 6
Konfigurasi *alert*

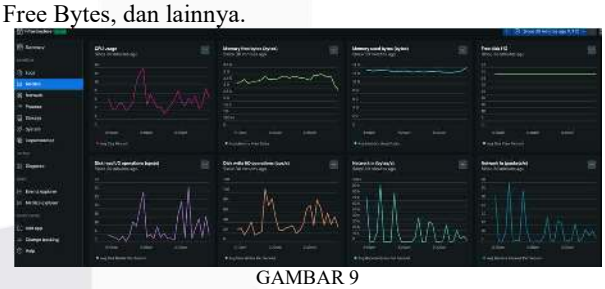
Setelah menentukan *threshold* dan parameter, beri nama untuk alert yang telah dibuat, misalnya ‘CPU Usage Warning – Over 90’. Jika notifikasi muncul, berarti alert telah berhasil disimpan.




GAMBAR 7
Konfigurasi nama dan durasi alert


GAMBAR 8
Notifikasi alert berhasil dibuat

Langkah terakhir adalah memastikan semua metrik tampil sesuai konfigurasi, yaitu Data *Real-time* (seperti CPU Usage dan Network Latency) dan Data Historis (grafik metrik sesuai rentang waktu). *Dashboard* yang diberi nama “Monitoring *Server* Simulasi FRI” kemudian disimpan untuk pengujian lebih lanjut, mencakup metrik seperti CPU Usage, Memory Free Bytes, dan lainnya.



GAMBAR 9
Contoh *dashboard* yang berhasil dibuat

3. Kendala dan Solusi Saat Implementasi

Dalam implementasi sistem monitoring menggunakan New Relic, beberapa kendala muncul yang memengaruhi kelancaran instalasi, konfigurasi, dan pengujian. Untuk mengatasi hal tersebut, beberapa solusi diterapkan agar sistem berfungsi optimal. Berikut adalah kendala-kendala yang dihadapi beserta solusi yang diterapkan:

a. Pengujian tidak dilakukan langsung pada *server* eksisting FRI, karena alasan keamanan dan operasional. Solusi yang diterapkan adalah dengan membuat *server* simulasi yang mereplikasi spesifikasi *server* FRI.
b. Beberapa metrik, seperti latensi jaringan (*network latency*) dan *swap memory*, sulit diinterpretasikan tanpa konteks tambahan. Solusi yang diberikan adalah dengan menyediakan deskripsi setiap metrik dalam laporan

implementasi, termasuk rumus, tujuan, serta dampaknya dalam mengelola performa *server*. Selain itu, penjelasan singkat diberikan kepada admin *server* mengenai cara membaca dan menganalisis data di New Relic.

E. Hasil Pengujian Sistem

Pengujian dilakukan dengan membandingkan metrik yang dikumpulkan New Relic pada *server* simulasi, dengan hasil dari Site24x7 dan Grafana. Hasil pengujian menunjukkan sebagai berikut:

TABEL 5
PERBANDINGAN HASIL PENGUJIAN

| Parameter | New Relic | Site 24x7 | Grafana (dengan Node Exporter & Prometheus) |
|---|---|---|---|
| *CPU Usage (min)* | *1-20% (Varies on certain scenarios)* | *1-10% (Varies depending on scenario)* | *1%-5% for basic setups, but complex queries and visualizations can increase CPU load.* |
| *Memory Requirements* | ~1GB RAM | ~150MB RAM | ~512MB RAM |
| *Deployment Type* | *Cloud-Based* | *Cloud-Based* | *On Premise or Cloud-Based* |
| *Visual Clarity* | *Very Good* | *Very Good* | *Good* |
| *CPU Usage (min)* | *1-20% (Varies on certain scenarios)* | *1-10% (Varies depending on scenario)* | *1%-5% for basic setups, but complex queries and visualizations can increase CPU load.* |

Berdasarkan hasil pengujian, pemilihan software monitoring harus disesuaikan dengan kebutuhan dan kapasitas organisasi. New Relic cocok untuk perusahaan besar dengan infrastruktur kompleks dan integrasi *cloud*, menawarkan kemudahan penggunaan dan visualisasi *real-time*, meskipun biayanya tinggi. Site24x7 merupakan alternatif hemat untuk perusahaan kecil hingga menengah, dengan fitur lebih sedikit namun harga lebih terjangkau. Grafana dengan Prometheus adalah solusi fleksibel dan hemat biaya, cocok untuk tim teknis berpengalaman, meskipun membutuhkan waktu implementasi lebih lama. Secara keseluruhan, New Relic lebih cocok untuk perusahaan besar, sementara Site24x7 dan Grafana sesuai untuk skala dan kebutuhan lebih spesifik.

F. Feedback Pengguna dan Validasi

Dari hasil wawancara, diperoleh informasi terkait spesifikasi virtual *server* yang digunakan di Fakultas Rekayasa Industri, ambang batas (*threshold*) yang digunakan untuk peringatan, kondisi eksisting *server*, waktu puncak penggunaan *server* (peak times), serta metrik yang diperlukan untuk ditampilkan pada *dashboard*.

Validasi dilakukan menggunakan metode *expert judgement* terhadap *dashboard* New Relic yang telah dirancang dan disajikan kepada laboran sebelumnya. Evaluasi ini bertujuan untuk mengukur tingkat kecocokan *dashboard* dengan kebutuhan *monitoring* kinerja *server* di Laboratorium Fakultas Rekayasa Industri. Penilaian dilakukan berdasarkan 14 kriteria yang telah ditetapkan.

TABEL 6
KUESIONER EXPERT JUDGEMENT

| No | Kriteria | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 1 | *Monitoring* performa CPU secara *real-time* | | | | V | |
| 2 | Kemampuan untuk memberikan alert jika CPU bermasalah | V | | | | |
| 3 | *Monitoring* penggunaan memori secara *real-time* | | | | | V |
| 4 | Kemampuan untuk memberikan alert jika memori bermasalah | | | V | | |
| 5 | *Monitoring* kapasitas storage | | | | | V |
| 6 | Pemantauan kondisi kesehatan storage secara keseluruhan | | | | V | |
| 7 | *Monitoring* aktivitas jaringan | | | | V | |
| 8 | Visualisasi status sistem dan aplikasi *container* | | | | V | |
| 9 | Kemudahan dalam mengidentifikasi masalah pada aplikasi *container* | | | V | | |
| 10 | Kemampuan untuk mengirim notifikasi via email/WA ketika sistem bermasalah | | | | V | |
| 11 | Kemampuan untuk menyimpan dan menampilkan data historis | | | | V | |
| 12 | Alat analisis untuk menilai pola performa selama waktu tertentu (e.g., saat heavy traffic) | | | | V | |
| 13 | Kemudahan dalam mengatur dan menggunakan *dashboard* | | V | | | |
| 14 | Efektivitas *dashboard* dalam memantau dan meningkatkan keamanan dan performa *server* | | V | | | |

G. Standarisasi Final *Dashboard* Monitoring

Dari hasil penelitian dan wawancara, sebuah *dashboard* monitoring dirancang yang berisikan komponen-komponen utama untuk mendukung pengawasan *server* secara *real-time* dan analisis jangka panjang. Berikut adalah tabel yang merangkum komponen yang direkomendasikan sebagai standar sebuah *dashboard* monitoring pada *server* FRI.

TABEL 7
REKOMENDASI KOMPONEN PADA *DASHBOARD*

| Komponen | Deskripsi | Alasan Penting |
|---|---|---|
| *CPU Metrics* | Menampilkan penggunaan CPU dalam bentuk grafik *real-time* dan rata-rata. Memberikan informasi tentang waktu *idle*, *load average*, dan persentase penggunaan *core* CPU. | Mengidentifikasi *overload server* sehingga dapat dilakukan mitigasi lebih awal. |
| *Memory Metrics* | Monitoring penggunaan RAM dan *swap memory*, termasuk *free*, *cached*, dan *buffer* memory. | Memastikan ketersediaan memori untuk mencegah crash akibat kehabisan memori. |
| *Disk Metrics* | Menampilkan kapasitas total, kapasitas terpakai, kapasitas tersisa, serta IOPS (*Input/Output Operations Per Second*). | Membantu memantau kesehatan disk dan mencegah masalah akibat disk penuh. |

| Komponen | Deskripsi | Alasan Penting |
|---|---|---|
| *Network Metrics* | Menampilkan metrik latency (RTT), *throughput*, *jitter*, dan *packet loss*. | Memantau kualitas konektivitas jaringan dan mendeteksi gangguan atau serangan seperti *packet flooding*. |
| *Process Monitoring* | Menampilkan daftar proses yang berjalan, termasuk penggunaan CPU, memori, dan I/O oleh setiap proses. | Mengidentifikasi proses yang memakan sumber daya berlebihan sehingga dapat dioptimalkan. |
| *Alert System* | Sistem peringatan otomatis yang memantau ambang batas, seperti CPU > 80% atau *disk usage* > 90%. | Memberikan notifikasi dini kepada administrator untuk mencegah *downtime*. |
| *Customizable Widgets* | Memungkinkan pengguna menyesuaikan tampilan *dashboard* sesuai dengan kebutuhan spesifik, seperti fokus pada keamanan atau performa tertentu. | Meningkatkan fleksibilitas dan relevansi *dashboard* untuk berbagai jenis pengguna. |
| *Historical Data* | Data performa *server* selama periode waktu tertentu dalam format grafik dan laporan. | Mendukung analisis tren performa dan perencanaan kapasitas *server* di masa depan. |
| *CPU Metrics* | Menampilkan penggunaan CPU dalam bentuk grafik *real-time* dan rata-rata. Memberikan informasi tentang *waktu idle*, *load average*, dan persentase penggunaan core CPU. | Mengidentifikasi *overload server* sehingga dapat dilakukan mitigasi lebih awal. |

Dengan adanya tabel standarisasi ini, bertujuan untuk memberikan guideline untuk konten dari alat monitoring yang digunakan agar dapat disesuaikan dengan kebutuhan *server* Fakultas Rekayasa Industri. Selama terdapat komponen ini dalam alat *monitoring server* yang dijadikan pilihan penanggung jawab *server*, diharapkan dapat mengurangi masalah yang seringkali dihadapi seperti downtime, dan mengurangi metrik seperti MTTR atau *Mean Time to Repair*.

## V. KESIMPULAN

### A. Kesimpulan

Penelitian ini menghasilkan beberapa kesimpulan terkait implementasi *dashboard* monitoring menggunakan New Relic untuk *server* Fakultas Rekayasa Industri:

1. Sebelum implementasi New Relic, pengelolaan *server* dilakukan secara manual dengan pemantauan terbatas pada metrik dasar seperti penggunaan CPU dan kapasitas penyimpanan. Sistem monitoring yang ada belum terintegrasi, menyulitkan pemantauan performa *server* secara menyeluruh dan *real-time*, yang menyebabkan keterlambatan dalam mendeteksi masalah operasional.
2. Penelitian mengidentifikasi bahwa kebutuhan utama adalah kemampuan memantau performa *server* secara *real-time*, menganalisis data historis, dan memberikan notifikasi dini terhadap anomali. Tantangan yang dihadapi termasuk ketidakmampuan sistem sebelumnya dalam memvisualisasikan data dengan jelas, ketergantungan pada pemantauan manual, dan minimnya integrasi dengan sistem keamanan serta jaringan.
3. Implementasi *dashboard* berbasis New Relic terbukti efektif dalam meningkatkan pengelolaan sumber daya *server*, dengan memberikan informasi *real-time* tentang penggunaan CPU, memori, disk, dan jaringan, serta menyediakan sistem peringatan otomatis untuk deteksi masalah lebih awal.

### B. Saran

Berdasarkan hasil penelitian dan batasan yang ditemukan, berikut adalah beberapa saran untuk penelitian dan implementasi di masa depan:

1. Untuk memaksimalkan penggunaan sebuah *software monitoring*, administrator *server* atau laboran perlu mendapatkan pelatihan mengenai cara membaca dan menganalisis data yang ditampilkan. Dokumentasi legkap mengenai penggunaan New Relic dan konfigurasi *dashboard* perlu disusun sebagai referensi teknis/acuan.
2. Penelitian ini membuka peluang untuk eksplorasi lebih lanjut, seperti pengembangan *dashboard* berbasis AI untuk prediksi performa *server* untuk mendukung migrasi ke skala *server* yang lebih besar seperti *server* yang menggunakan infrastruktur hybrid (*cloud* dan on-premises)
3. Diharapkan pada penelitian selanjutnya, ujicoba dapat dilakukan langsung pada objek penelitian yang diamati yaitu *server* FRI, agar implementasi bisa dilakukan langsung, serta data yang didapatkan bisa lebih akurat, serta mengurangi jumlah asumsi yang diambil selama penelitian berlangsung.

## REFERENSI

[1] MD Hanson. (2000). *Server Management* (G. Held, Ed.; 1st Edition). Auerbach Publications.

[2] Timbul Sigiro, M. M. (2016). DS-DASHBOARD: DISTRIBUTED SERVER'S RESOURCE MANAGEMENT, CASE STUDY INSTITUT TEKNOLOGI DEL. *Jurnal Teknologi*, *78*(6–3). https://doi.org/10.11113/jt.v78.8940

[3] Tsai, P.-W., Tsai, C.-W., Hsu, C.-W., & Yang, C.-S. (2018). Network Monitoring in Software-Defined Networking: A Review. *IEEE Systems Journal*, *12*(4), 3958–3969. https://doi.org/10.1109/JSYST.2018.2798060

[4] Vilar, P. (2010). Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th edition). *Journal of the American Society for Information Science and Technology*, *61*(5), 1073–1074. https://doi.org/10.1002/asi.21215