Analisa Implementasi Load Balancing untuk Menghindari Bottleneck

Diajukan untuk Memenuhi Tugas Mata Kuliah
”Workshop Administrasi Jaringan”
Dosen: Ferry Astika S.T M.Sc

Disusun Oleh :

Muhammad Fattachul Aziz	3122600018
Muhammad Amir Abdurrozaq	3122600023
Fangga Malik Alfian	3121600029

JURUSAN D4 TEKNIK INFORMATIKA
DEPARTEMEN TEKNIK INFORMATIKA DAN KOMPUTER
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA
2023/2024

Daftar Isi

Abstraksi

Dalam era digital yang semakin berkembang, efisiensi dan keandalan sistem TI menjadi faktor kunci bagi keberhasilan operasional banyak organisasi. Salah satu tantangan utama yang dihadapi adalah bottleneck dalam infrastruktur TI, yang dapat menghambat kinerja sistem dan mengurangi produktivitas. Penelitian ini bertujuan untuk meminimalisir bottleneck dengan menerapkan server monitoring menggunakan Prometheus dan Grafana.

Prometheus merupakan sistem monitoring dan alerting open-source yang dirancang untuk mengumpulkan dan menyimpan metrik dalam bentuk *timeseries*. Grafana, sebagai platform analitik open-source, digunakan untuk visualisasi data metrik yang dikumpulkan oleh Prometheus. Penelitian ini mengkaji efektivitas penggunaan Prometheus dan Grafana dalam mendeteksi, menganalisis, dan mengatasi bottleneck di server.

Metode penelitian melibatkan implementasi Prometheus dan Grafana pada server dalam lingkungan pengujian. Data metrik yang dikumpulkan oleh Prometheus dianalisis untuk mengidentifikasi pola dan anomali yang menunjukkan adanya bottleneck, yang kemudian akan dilakukan tindak lanjut dengan mengatur ulang replikasi dari tiap servicenya.

Pendahuluan

1. Latar belakang

Dalam era digital yang semakin berkembang, efisiensi dan keandalan sistem TI menjadi faktor kunci bagi keberhasilan operasional banyak organisasi. Salah satu tantangan utama yang dihadapi adalah bottleneck dalam infrastruktur TI, yang dapat menghambat kinerja sistem dan mengurangi produktivitas. Penelitian ini bertujuan untuk meminimalisir bottleneck dengan menerapkan server monitoring menggunakan Prometheus dan Grafana.

Prometheus merupakan sistem monitoring dan alerting open-source yang dirancang untuk mengumpulkan dan menyimpan metrik dalam bentuk timeseries. Grafana, sebagai platform analitik open-source, digunakan untuk visualisasi data metrik yang dikumpulkan oleh Prometheus. Penelitian ini mengkaji efektivitas penggunaan Prometheus dan Grafana dalam mendeteksi, menganalisis, dan mengatasi bottleneck di server.

Metode penelitian melibatkan implementasi Prometheus dan Grafana pada server dalam lingkungan pengujian. Data metrik yang dikumpulkan oleh Prometheus dianalisis untuk mengidentifikasi pola dan anomali yang menunjukkan adanya bottleneck, yang kemudian akan dilakukan tindak lanjut dengan mengatur ulang replikasi dari tiap servicenya.

Penelitian ini akan menentukan metrik penting yang diperlukan untuk mendeteksi kelangkaan sistem komputasi. Selain itu, akan dijelaskan bagaimana mengintegrasikan Prometheus dan Grafana secara efektif, serta bagaimana menganalisis data yang diperoleh untuk menemukan dan mengatasi kelangkaan. Oleh karena itu, dengan menggunakan metode pemantauan yang lebih efisien, penelitian ini diharapkan dapat memberikan kontribusi signifikan dalam peningkatan kinerja sistem komputasi.

Melalui studi kasus dan eksperimen yang dilakukan, penelitian ini juga akan mengevaluasi efektivitas penggunaan Prometheus dan Grafana dalam skenario nyata. Hasil dari penelitian ini diharapkan dapat memberikan panduan praktis bagi pengelola sistem untuk memanfaatkan alat-alat tersebut dalam meminimalisir bottleneck dan meningkatkan stabilitas serta efisiensi sistem komputasi mereka.

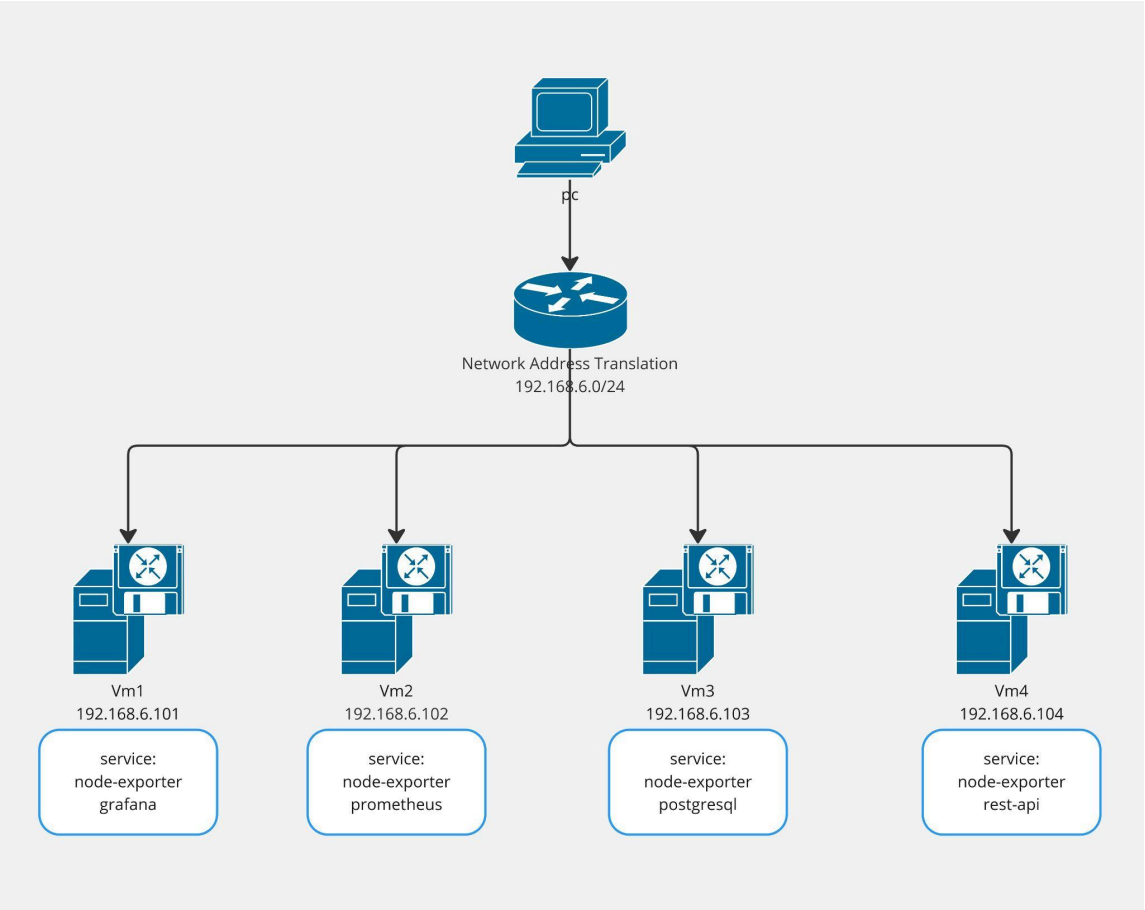
2. Tujuan Penelitian:

Penelitian ini bertujuan untuk Mengkaji efektivitas penggunaan Prometheus dan Grafana dalam mendeteksi, menganalisis, dan mengatasi bottleneck di server. Menyediakan solusi untuk meminimalisir bottleneck dengan menggunakan data yang dikumpulkan oleh Prometheus dan divisualisasikan oleh Grafana.

Lingkup Penelitian

1. Identifikasi Bottleneck:
    - Menganalisis berbagai jenis bottleneck yang umum terjadi dalam infrastruktur TI.
    - Menentukan metrik kunci yang dapat mengindikasikan adanya bottleneck pada server.
2. Implementasi Alat Monitoring:
    - Mengatur dan mengonfigurasi Prometheus untuk mengumpulkan metrik server.
    - Mengatur dan mengonfigurasi Grafana untuk visualisasi data metrik yang dikumpulkan oleh Prometheus.
3. Pengumpulan dan Analisis Data:
    - Mengumpulkan data metrik dari server selama periode pengujian.
    - Menganalisis data untuk mengidentifikasi pola dan anomali yang menunjukkan adanya bottleneck.
4. Tindak Lanjut:
    - Mengembangkan strategi untuk mengatasi bottleneck yang teridentifikasi.
    - Mengimplementasikan solusi berdasarkan hasil analisis (misalnya, pengaturan ulang replikasi dari tiap servis).
5. Evaluasi dan Validasi:
    - Mengevaluasi efektivitas solusi yang diterapkan dalam mengurangi atau menghilangkan bottleneck.
    - Melakukan uji coba berulang untuk memastikan stabilitas dan peningkatan kinerja sistem setelah penerapan solusi.
6. Batasan Penelitian:
    - Penelitian ini dibatasi pada penggunaan Prometheus dan Grafana dalam lingkungan server uji coba.
    - Tidak mencakup analisis kinerja aplikasi di luar lingkungan server yang diawasi oleh Prometheus.
    - Fokus pada pengukuran metrik yang relevan dengan bottleneck, tanpa mendalami aspek lain seperti keamanan data.
7. Manfaat Penelitian:
    - Meningkatkan pemahaman tentang penggunaan alat monitoring open-source dalam mengelola infrastruktur TI.
    - Memberikan panduan praktis bagi organisasi dalam mengidentifikasi dan mengatasi bottleneck pada server mereka.
    - Kontribusi terhadap peningkatan efisiensi dan keandalan sistem TI melalui monitoring yang lebih baik.

Design System



Rancangan infrastruktur sebelum optimasi melibatkan sebuah PC yang terhubung melalui NAT Network ke empat VM dalam subnet 192.168.6.0/24. VM1 (192.168.6.101) menjalankan layanan node-exporter dan Grafana, yang bertanggung jawab untuk mengumpulkan data metrik dari sistem operasi serta menyediakan platform visualisasi data. VM2 (192.168.6.102) menjalankan node-exporter dan Prometheus, di mana Prometheus bertugas mengumpulkan, menyimpan, dan memproses data metrik dari seluruh sistem yang dimonitor.

VM3 (192.168.6.103) menjalankan node-exporter dan PostgreSQL, di mana PostgreSQL berfungsi sebagai database untuk menyimpan data yang diperlukan oleh sistem. VM4 (192.168.6.104) menjalankan node-exporter dan REST API service yang di-deploy menggunakan Dockerfile. Setiap VM dilengkapi dengan node-exporter untuk memastikan bahwa metrik yang relevan dapat dikumpulkan dan dianalisis.

Tahapan Pelaksanaan

Persiapan Penelitian:

- **Pengkajian Literatur:**
    - Melakukan studi literatur untuk memahami konsep bottleneck, monitoring sistem, Prometheus, dan Grafana.
    - Mengidentifikasi studi kasus dan praktik terbaik yang relevan.
- **Perencanaan Penelitian:**
    - Menyusun rencana penelitian yang mencakup tujuan, metodologi, dan jadwal kegiatan.
    - Mengidentifikasi sumber daya yang dibutuhkan, termasuk perangkat keras, perangkat lunak, dan tenaga ahli.

Pengaturan Lingkungan Pengujian:

- **Penyediaan Infrastruktur:**
    - Menyiapkan server uji coba yang akan digunakan untuk penelitian.
    - Memastikan lingkungan pengujian yang representatif dan sesuai dengan kondisi operasional sebenarnya.
- **Instalasi dan Konfigurasi Menggunakan Docker:**
    - Menginstal Docker pada server yang akan digunakan untuk penelitian.
    - Mengonfigurasi Docker untuk memastikan lingkungan kontainer yang stabil dan aman.

Monitoring Awal:

- Menjalankan kontainer Prometheus untuk mulai mengumpulkan data metrik dari server.
- Memastikan bahwa semua metrik yang relevan (CPU usage, memory usage, disk I/O, network traffic, dll.) dikumpulkan dengan benar.

Visualisasi Data:

- Membuat dasbor Grafana untuk visualisasi data metrik yang dikumpulkan oleh Prometheus.
- Mengatur dasbor untuk memudahkan identifikasi pola dan anomali.

Pengembangan dan Deployment REST API Service:

- Mengembangkan layanan REST API yang akan di-deploy menggunakan Dockerfile.

Analisis Data:

- Identifikasi Bottleneck:
    - Menganalisis data metrik untuk mengidentifikasi pola yang menunjukkan adanya bottleneck.
    - Menggunakan visualisasi Grafana untuk memudahkan interpretasi data.
- Penentuan Penyebab:

    - Menggali lebih dalam untuk menentukan penyebab utama bottleneck yang teridentifikasi.
    - Melakukan analisis lebih lanjut untuk mengonfirmasi temuan.

Pengembangan Solusi:

- Mengembangkan strategi untuk mengatasi bottleneck yang teridentifikasi, misalnya pengaturan ulang replikasi dari tiap servis.

Implementasi Solusi:

- Melakukan perubahan yang diperlukan pada konfigurasi sistem berdasarkan analisis data.
- Memantau efek perubahan untuk memastikan bahwa bottleneck teratasi.

Evaluasi dan Validasi:

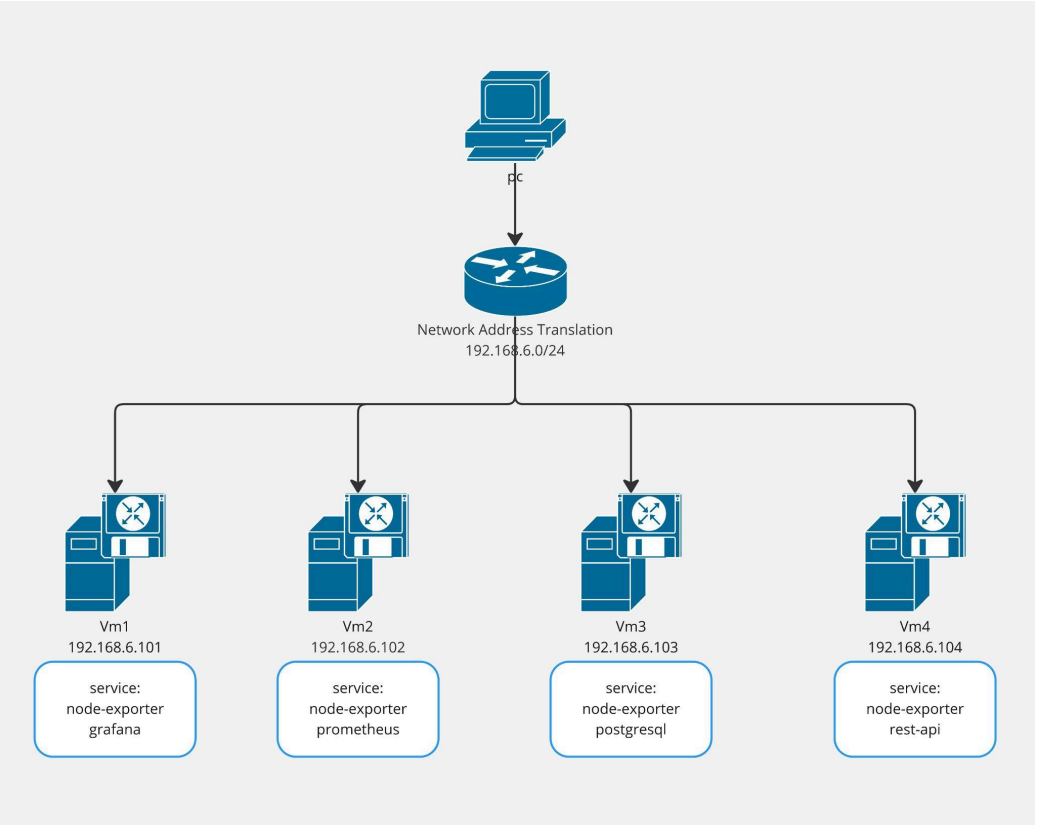
- Pengujian Ulang:
    - Mengulangi pengumpulan dan analisis data setelah implementasi solusi.
    - Membandingkan data sebelum dan sesudah implementasi untuk mengevaluasi efektivitas solusi.

Pelaporan dan Dokumentasi:

- Menyusun laporan akhir penelitian yang mencakup temuan, analisis, dan rekomendasi.
- Membuat panduan praktis untuk penggunaan Prometheus dan Grafana dalam mengidentifikasi dan mengatasi bottleneck.

Implementasi

1. Design Sistem Optimasi



Deskripsi:
Sebelum optimasi, sistem terdiri dari empat VM dengan layanan sebagai berikut:

- VM1 (192.168.6.101): node-exporter, grafana
- VM2 (192.168.6.102): node-exporter,, prometheus
- VM3 (192.168.6.103): node-exporter, postgresql
- VM4 (192.168.6.104): node-exporter, rest-api

2. Deploy Service

Berikut ini adalah docker compose dan config dari tiap service yang akan dideploy ke tiap virtul machinenya.

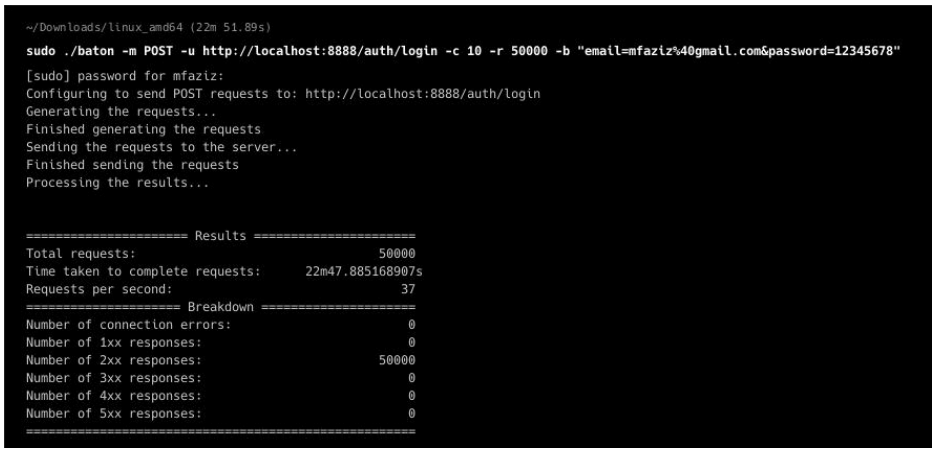
a. **Virtual Machine 1**
https://github.com/MasLazu/WorkshopAdministrasiJaringan/tree/main/project-uas/config%20sebelum%20optimasi/vm1

b. **Virtual Machine 2**
https://github.com/MasLazu/WorkshopAdministrasiJaringan/tree/main/project-uas/config%20sebelum%20optimasi/vm2

c. **Virtual Machine 3**
https://github.com/MasLazu/WorkshopAdministrasiJaringan/tree/main/project-uas/config%20sebelum%20optimasi/vm3

d. **Virtual Machine 4**
https://github.com/MasLazu/WorkshopAdministrasiJaringan/tree/main/project-uas/config%20sebelum%20optimasi/vm4

3. Hasil Perfomance Test



Hasil Performance Test Sebelum Optimasi:

Pada gambar yang diunggah, kita dapat melihat hasil dari tes performa yang dilakukan menggunakan alat pengujian baton. Berikut adalah detail dan penjelasan dari hasil tes tersebut:

Perintah yang digunakan:

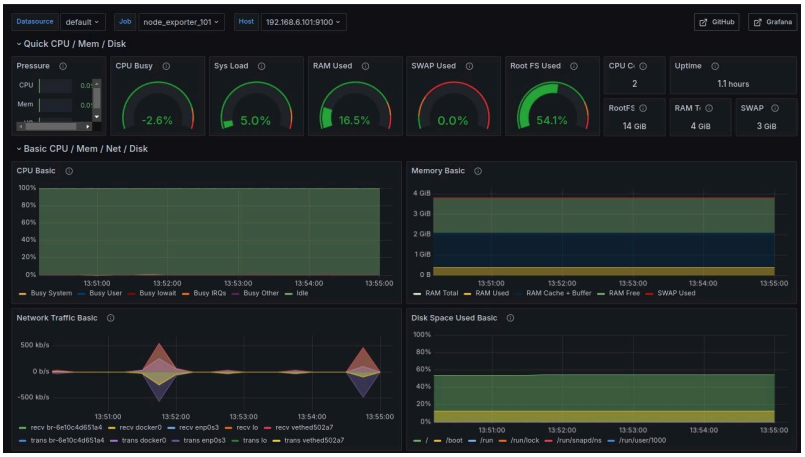
```
sudo ./baton -m POST -u http://localhost:8888/auth/login -c 10 -r 50000 -b "email=mfaziz%40gmail.com&password=12345678"
```

Parameter Perintah:

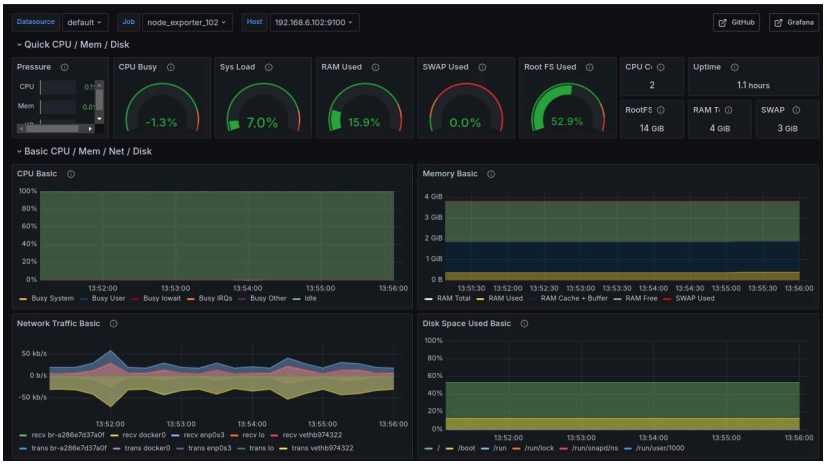
- **-m POST**: Metode HTTP yang digunakan adalah POST.
- **-u http://localhost:8888/auth/login**: URL endpoint yang diuji adalah http://localhost:8888/auth/login.
- **-c 10**: Jumlah koneksi yang digunakan secara bersamaan adalah 10.
- **-r 50000**: Jumlah total permintaan yang dikirimkan adalah 50.000.
- **-b "email=mfaziz%40gmail.com&password=12345678"**: Payload atau data yang dikirim dalam permintaan POST.

Hasil ini menunjukkan bahwa server mampu menangani semua permintaan dengan sukses tanpa adanya kesalahan koneksi atau kesalahan respon lainnya. Meskipun semua permintaan diproses dengan sukses, kecepatan pemrosesan (37 requests per second) menunjukkan ada ruang untuk peningkatan, terutama jika tujuan utamanya adalah untuk meningkatkan throughput.

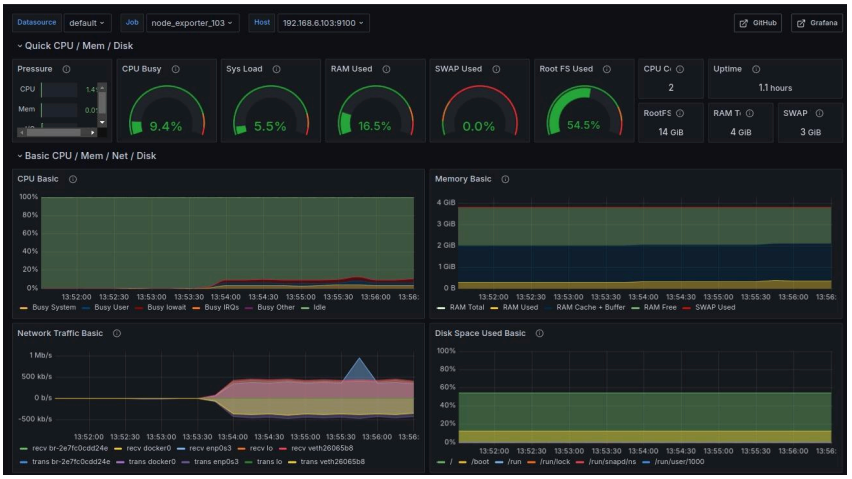
4. Hasil Monitor



monitor vm1 sebelum optimasi - grafana



monitor vm2 sebelum optimasi - Prometheus



monitor vm3 sebelum optimasi - postgres

monitor vm4 sebelum optimasi - rest api

Secara keseluruhan, VM 4 mengalami penggunaan CPU yang sangat tinggi, yang bisa menyebabkan bottleneck yang menjadikan kita harus melakukan optimalisasi pada design system yang kita buat.

5. Evaluasi dari Hasil Monitor

a. Total Requests:

Sebanyak 50.000 permintaan telah dikirimkan selama pengujian.

b. Time Taken to Complete Requests:

Waktu yang dibutuhkan untuk menyelesaikan semua permintaan adalah 22 menit dan 47.885 detik.

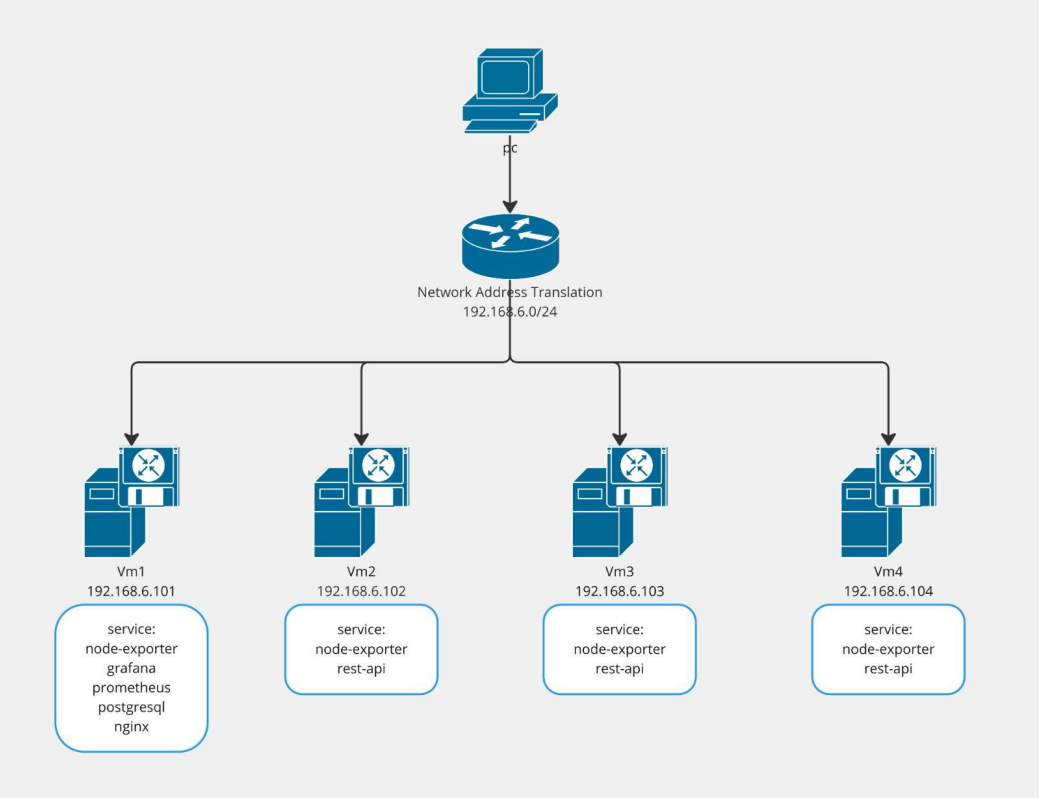
c. Requests per Second:

d. Rata-rata permintaan yang dapat diproses per detik adalah 37 permintaan.

Kesimpulan:

Hasil tes ini memberikan gambaran tentang kinerja server saat menangani sejumlah besar permintaan. Untuk lebih mengoptimalkan sistem dan mengurangi bottleneck, distribusi beban kerja di antara server perlu diperhatikan, seperti yang direncanakan dalam desain optimasi berikutnya.

6. Perancangan Design yang lebih Optimal dan Minim Bottleneck



Setiap server terhubung ke router, dan semua server berada di subnet 192.168.6.0/24. Server menjalankan berbagai layanan, termasuk node-exporter, grafana, prometheus, postgresql, dan nginx.

7. Deploy Service Setelah Dioptimasi

Berikut ini adalah docker compose dan config dari tiap service yang akan dideploy ke tiap virtul machinenya.

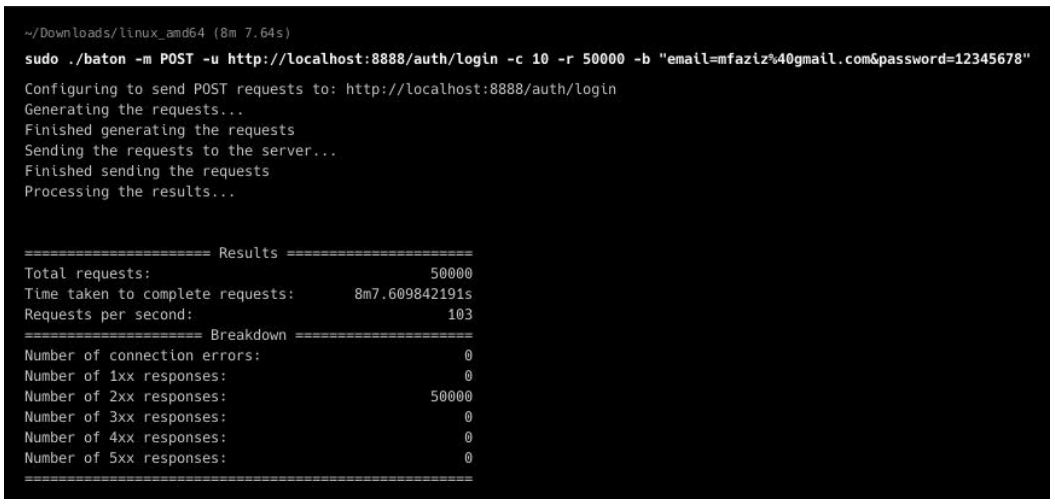
a. **Virtual Machine 1**
https://github.com/MasLazu/WorkshopAdministrasiJaringan/tree/main/project-uas/config%20setelah%20optimasi/vm1

b. **Virtual Machine 2**
https://github.com/MasLazu/WorkshopAdministrasiJaringan/tree/main/project-uas/config%20setelah%20optimasi/vm2

c. **Virtual Machine 3**
https://github.com/MasLazu/WorkshopAdministrasiJaringan/tree/main/project-uas/config%20setelah%20optimasi/vm3

d. **Virtual Machine 4**
https://github.com/MasLazu/WorkshopAdministrasiJaringan/tree/main/project-uas/config%20setelah%20optimasi/vm4

8. Hasil Performance test setelah Optimasi



Hasil Performance Test Setelah Optimasi:

Pada gambar yang diunggah, kita dapat melihat hasil dari tes performa yang dilakukan menggunakan alat pengujian baton. Berikut adalah detail dan penjelasan dari hasil tes tersebut:

Perintah yang digunakan:

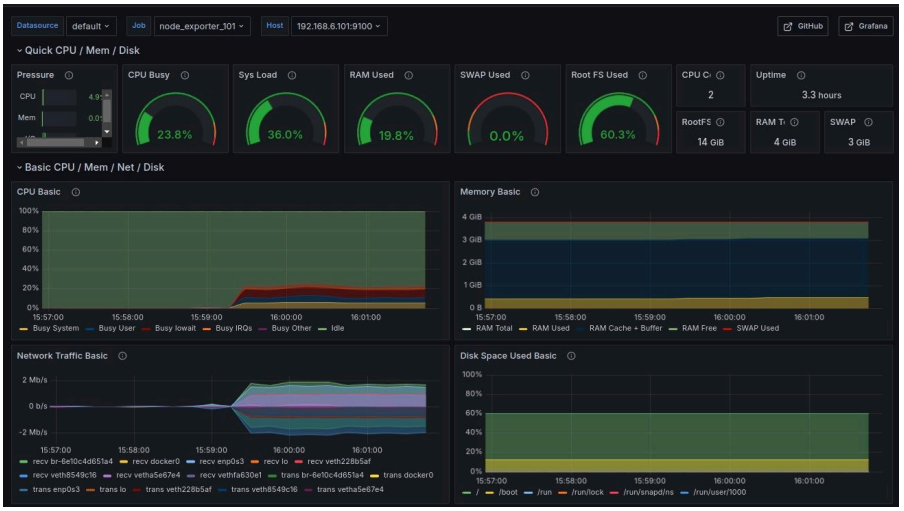
```
sudo ./baton -m POST -u http://localhost:8888/auth/login -c 10 -r 50000 -b "email=mfaziz%40gmail.com&password=12345678"
```

Parameter Perintah:

- **-m POST**: Metode HTTP yang digunakan adalah POST.
- **-u http://localhost:8888/auth/login**: URL endpoint yang diuji adalah http://localhost:8888/auth/login.
- **-c 10**: Jumlah koneksi yang digunakan secara bersamaan adalah 10.
- **-r 50000**: Jumlah total permintaan yang dikirimkan adalah 50.000.
- **-b "email=mfaziz%40gmail.com&password=12345678"**: Payload atau data yang dikirim dalam permintaan POST.

Hasil ini menunjukkan bahwa server mampu menangani semua permintaan dengan sukses tanpa adanya kesalahan koneksi atau kesalahan respon lainnya.
Sebelum dioptimasi terdapat bottleneck pada vm4 yang menjalankan service rest api, penggunaan cpu pada vm tersebut hampir 100%. untuk mengurangi bottleneck tersebut dilakukan horizontal scale untuk service tersebut dengan mengalokasikan tiga server untuk menjalankan rest api tersebut serta menjalankan nginx sebagai load balancer pada vm1, sisanya dijalankan di vm1 semua, kecepatan pemrosesan (103 requests per second), lebih baik dari sebelumnya.

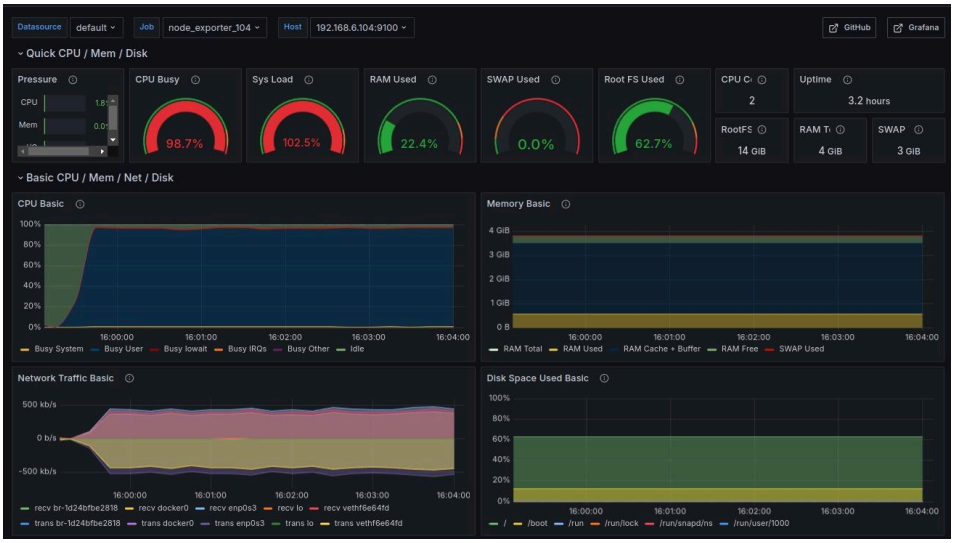
9. Hasil Monitoring Setelah Optimasi



monitor vm1 setelah optimasi - grafana, prometheus, postgres, nginx

monitor vm2 setelah optimasi - rest api

monitor vm3 setelah optimasi - rest api



monitor vm4 setelah optimasi - rest api

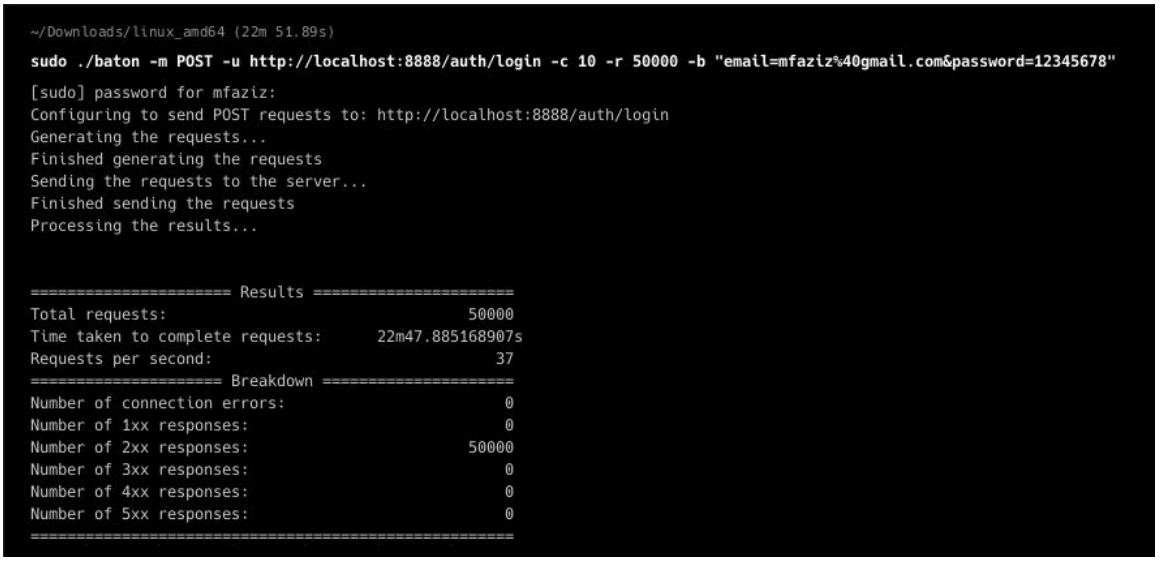
Secara keseluruhan, VM dioptimalkan, tetapi VM 1 sampai 3 mengalami penggunaan CPU yang sangat tinggi, yang menunjukkan bahwa mereka mungkin mendapat manfaat dari pengoptimalan lebih lanjut atau penskalaan untuk menangani beban dengan lebih efisien.

Tim dan Tugas

Eksekutor	**Task**
Aziz, Amir	Setup Prometheus
Aziz, Amir	Setup Node Exporter
Aziz, Fangga	Integrasi Node Exporter ke Prometheus
Aziz, Amir	Setup Grafana
Aziz, Amir	Menambahkan Dashboard Resource Monitoring di Grafana
Aziz	Studi Literatur Bagaimana Meminimalisir Bottleneck dengan Resource Monitoring Menggunakan Prometheus dan Grafana
Aziz	Membuat Rancangan Design yang Optimal dan Minimimalisir Bottleneck dengan Resource Monitoring
Aziz	Percobaan Antara Sebelum dan Sesudah Pengoptimalan Design System
Amir, Aziz	Membuat Laporan Hasil dan Kesimpulan

Sistem Testing

1. Sebelum Optimalisasi



Hasil Performance Test Sebelum Optimasi:

Pada gambar yang diunggah, kita dapat melihat hasil dari tes performa yang dilakukan menggunakan alat pengujian baton. Berikut adalah detail dan penjelasan dari hasil tes tersebut:

Perintah yang digunakan:

```
sudo ./baton -m POST -u http://localhost:8888/auth/login -c 10 -r 50000 -b "email=mfaziz%40gmail.com&password=12345678"
```

Parameter Perintah:

- **-m POST**: Metode HTTP yang digunakan adalah POST.

- **-u http://localhost:8888/auth/login**: URL endpoint yang diuji adalah http://localhost:8888/auth/login.
- **-c 10**: Jumlah koneksi yang digunakan secara bersamaan adalah 10.
- **-r 50000**: Jumlah total permintaan yang dikirimkan adalah 50.000.
- **-b "email=mfaziz%40gmail.com&password=12345678"**: Payload atau data yang dikirim dalam permintaan POST.

Hasil Test:

Total requests: 50000
Time taken to complete requests: 22m47.885168907s
Requests per second: 37

Breakdown:
Number of connection errors: 0
Number of 1xx responses: 0
Number of 2xx responses: 50000
Number of 3xx responses: 0
Number of 4xx responses: 0
Number of 5xx responses: 0
Penjelasan Hasil:

Total Requests:
Sebanyak 50.000 permintaan telah dikirimkan selama pengujian.

Time Taken to Complete Requests:
Waktu yang dibutuhkan untuk menyelesaikan semua permintaan adalah 22 menit dan 47.885 detik.
Requests per Second:
Rata-rata permintaan yang dapat diproses per detik adalah 37 permintaan.

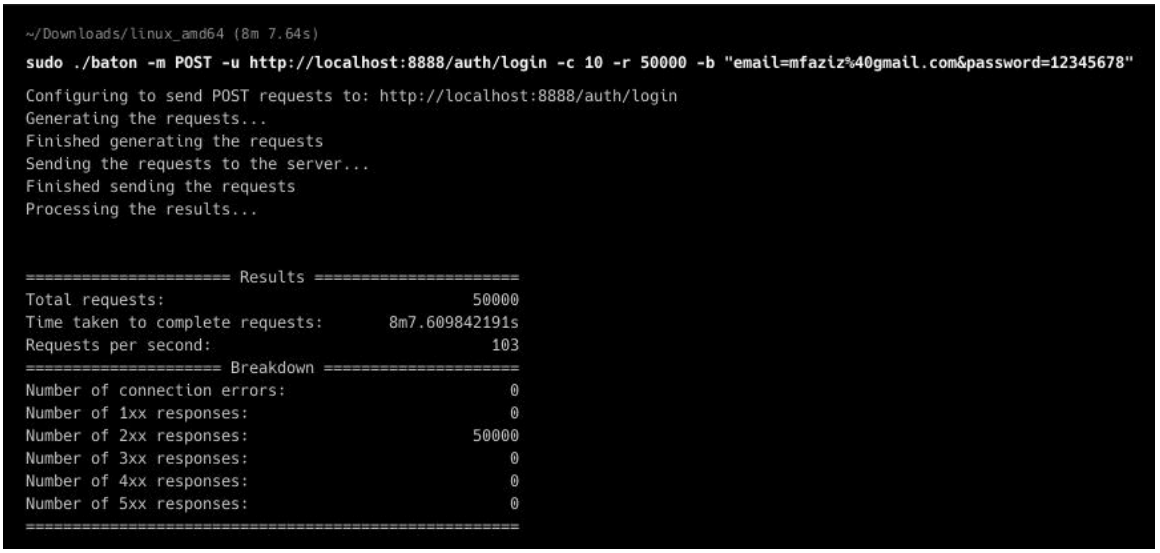
Breakdown (Rincian Respon):
Number of Connection Errors: Tidak ada kesalahan koneksi yang terjadi.
Number of 1xx Responses: Tidak ada respon informasi (1xx).
Number of 2xx Responses: Semua 50.000 permintaan mendapatkan respon sukses (2xx), yang menunjukkan bahwa semua permintaan berhasil diproses oleh server.
Number of 3xx Responses: Tidak ada respon redireksi (3xx).
Number of 4xx Responses: Tidak ada respon kesalahan klien (4xx).
Number of 5xx Responses: Tidak ada respon kesalahan server (5xx).

Analisis:
Hasil ini menunjukkan bahwa server mampu menangani semua permintaan dengan sukses tanpa adanya kesalahan koneksi atau kesalahan respon lainnya.

Meskipun semua permintaan diproses dengan sukses, kecepatan pemrosesan (37 requests per second) menunjukkan ada ruang untuk peningkatan, terutama jika tujuan utamanya adalah untuk meningkatkan throughput.

Kesimpulan:
Hasil tes ini memberikan gambaran tentang kinerja server saat menangani sejumlah besar permintaan. Untuk lebih mengoptimalkan sistem dan mengurangi bottleneck, distribusi beban kerja di antara server perlu diperhatikan, seperti yang direncanakan dalam desain optimasi berikutnya.

2. Setelah Optimalisasi



Hasil Performance Test Sebelum Optimasi:

Pada gambar yang diunggah, kita dapat melihat hasil dari tes performa yang dilakukan menggunakan alat pengujian baton. Berikut adalah detail dan penjelasan dari hasil tes tersebut:

Perintah yang digunakan:

```
sudo ./baton -m POST -u http://localhost:8888/auth/login -c 10 -r 50000 -b "email=mfaziz%40gmail.com&password=12345678"
```

Parameter Perintah:

- **-m POST**: Metode HTTP yang digunakan adalah POST.

- **-u http://localhost:8888/auth/login**: URL endpoint yang diuji adalah http://localhost:8888/auth/login.
- **-c 10**: Jumlah koneksi yang digunakan secara bersamaan adalah 10.
- **-r 50000**: Jumlah total permintaan yang dikirimkan adalah 50.000.
- **-b "email=mfaziz%40gmail.com&password=12345678"**: Payload atau data yang dikirim dalam permintaan POST.

Hasil Test:

Total requests: 50000
Time taken to complete requests: 8m7.609842191s
Requests per second: 103

Breakdown:
Number of connection errors: 0
Number of 1xx responses: 0
Number of 2xx responses: 50000
Number of 3xx responses: 0
Number of 4xx responses: 0
Number of 5xx responses: 0
Penjelasan Hasil:

Total Requests:
Sebanyak 50.000 permintaan telah dikirimkan selama pengujian.

Time Taken to Complete Requests:
Waktu yang dibutuhkan untuk menyelesaikan semua permintaan adalah 22 menit dan 47.885 detik.
Requests per Second:
Rata-rata permintaan yang dapat diproses per detik adalah 37 permintaan.

Breakdown (Rincian Respon):
Number of Connection Errors: Tidak ada kesalahan koneksi yang terjadi.
Number of 1xx Responses: Tidak ada respon informasi (1xx).
Number of 2xx Responses: Semua 50.000 permintaan mendapatkan respon sukses (2xx), yang menunjukkan bahwa semua permintaan berhasil diproses oleh server.
Number of 3xx Responses: Tidak ada respon redireksi (3xx).
Number of 4xx Responses: Tidak ada respon kesalahan klien (4xx).
Number of 5xx Responses: Tidak ada respon kesalahan server (5xx).

Perbandingan:

- **Waktu Penyelesaian:** Setelah optimalisasi, waktu yang diperlukan untuk menyelesaikan 50,000 permintaan berkurang dari 22 menit 47 detik menjadi 8 menit 7 detik.
- **Kecepatan Permintaan:** Setelah optimalisasi, kecepatan permintaan meningkat dari 37 permintaan per detik menjadi 103 permintaan per detik.
- **Kesalahan:** Tidak ada kesalahan koneksi atau kesalahan respons pada kedua pengujian, yang menunjukkan stabilitas server dalam menangani permintaan.

Kesimpulan: Optimalisasi pencegahan bottleneck telah berhasil meningkatkan performa server secara signifikan, mempercepat waktu penyelesaian permintaan dan meningkatkan throughput permintaan per detik.