

Laporan Projek

Resource Monitoring dengan Prometheus dan Grafana

Mata Kuliah Workshop Administrasi Jaringan



Dosen:

Dr. Ferry Astika Saputra, S.T, M.Sc

Oleh:

Fatimah Az-zahra' Mahros(3122600002)

Alga Vania Salsabillah(3122600010)

Amirotul Ummah(3122600017)

Kelas 2 D4 Teknik Informatika A

Departemen Teknik Informatika dan Komputer

Politeknik Elektronika Negeri Surabaya

1. Pendahuluan

Keberhasilan sebuah aplikasi web tidak hanya diukur dari kelancaran dan ketersediaannya bagi pengguna, tetapi juga dari performa yang optimal. Untuk mencapai hal tersebut, administrasi jaringan yang handal dan proaktif sangatlah penting. Salah satu aspek terpenting dalam administrasi jaringan adalah pemantauan sumber daya, yang memungkinkan dilakukannya identifikasi dan pencegahan potensi masalah pada aplikasi sebelum dapat berakibat fatal pada performa jaringan.

Seiring dengan meningkatnya penggunaan teknologi kontainer seperti Docker dalam arsitektur sistem informasi, dapat muncul kompleksitas baru dalam administrasi jaringan. Aplikasi web modern sering kali terbagi menjadi beberapa kontainer, dengan masing-masing kontainernya menjalankan komponen aplikasi yang berbeda, seperti front-end dan back-end terpisah. Kerumitan tersebut memerlukan pendekatan pemantauan yang lebih canggih dan terintegrasi untuk memastikan performa aplikasi yang optimal dan kehandalan jaringan secara keseluruhan.

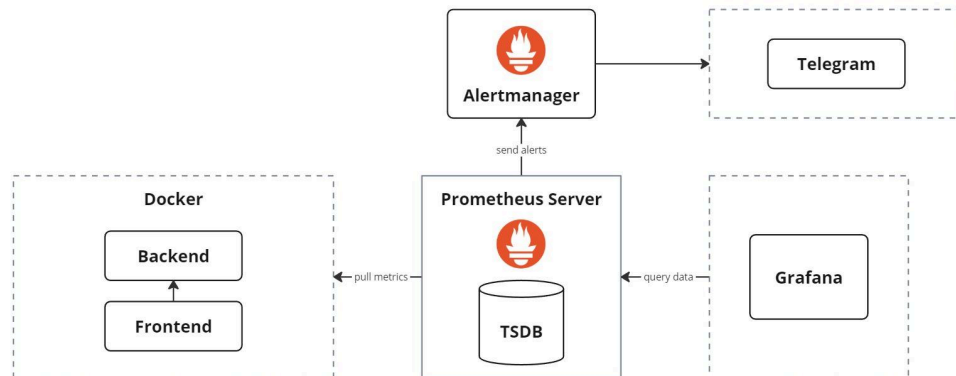
Untuk memenuhi kebutuhan tersebut, proyek ini bertujuan untuk menerapkan sistem pemantauan sumber daya yang efektif menggunakan Prometheus dan Grafana pada kontainer Docker yang menampung sebuah aplikasi web sederhana. Aplikasi yang kami pantau terdiri dari dua bagian, front-end dan back-end. Akan tetapi, pemantauan akan hanya kami lakukan terhadap sisi back-end dari aplikasi karena untuk monitoring di Prometheus harus menambahkan *end-point metrics*, sedangkan *front-end* tidak bisa langsung menambahkan *end-point* seperti pada back-end. Sistem ini akan memantau metrik CPU dan Request HTTP dari aplikasi menggunakan Prometheus, serta menyediakan visualisasi data yang mudah dipahami untuk administrator jaringan menggunakan Grafana. Selain itu, proyek ini juga akan mengimplementasikan *alert system* melalui Telegram untuk mendeteksi dan memberi tahu administrator jaringan apabila terjadi kelainan pada container aplikasi yang dipantau. Dengan demikian, proyek ini diharapkan dapat meningkatkan kemampuan memantau dan merespons terhadap masalah jaringan, sehingga memastikan kelancaran dan ketersediaan aplikasi web secara keseluruhan.

2. Ruang Lingkup

Ruang lingkup proyek ini mengarah pada pembuatan sistem pemantauan untuk Central Processing Unit (CPU) dan request HTTP dari back-end sebuah aplikasi dalam kontainer Docker, yang dijalankan diatas sebuah *Virtual Machine* Debian menggunakan Prometheus dan Grafana, dengan rincian berikut:

1. Penelitian ini dibatasi pada lingkungan container pada *virtual machine* yang digunakan dalam infrastruktur Teknologi Informasi.
2. Penelitian proyek ini mengimplementasikan implementasi alat monitoring Prometheus untuk pengumpulan data dan Grafana untuk visualisasi data.
3. Pemantauan kinerja CPU dan HTTP Request secara real-time menjadi fokus dari proyek ini dengan tujuan agar dapat mendeteksi masalah dan anomali sejak dini sehingga dapat mengelola sumber daya secara efisien.

3. Desain Sistem



4. Tim & Tugas

Nama	Tugas
Alga Vania Salsabillah	Menyiapkan dua aplikasi yang akan dijalankan di Container Docker. Menyiapkan Prometheus dan Grafana yang dijalankan di Virtual Machine.
Fatimah Az-zahra' Mahros	Menyiapkan Prometheus dan Grafana yang dijalankan di Virtual Machine. Menyambungkan Container ke Prometheus.
Amirotul Ummah	Menyambungkan data source dari Prometheus ke Grafana dan membuat visualisasi data di Grafana. Menyiapkan Alert Manager ke Telegram.

5. Tahapan Pelaksanaan

Tanggal	Kegiatan
---------	----------

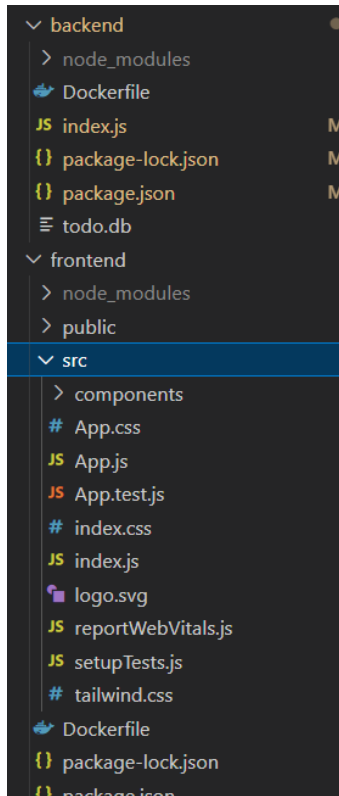
27/05/2024	Menyiapkan dua aplikasi yang akan dijalankan di Container Docker.
29/05/2024	Menyiapkan Prometheus dan Grafana yang dijalankan di Virtual Machine.
30/05/2024	Menyambungkan Container ke Prometheus.
31/05/2024	Menyambungkan data source dari Prometheus ke Grafana dan membuat visualisasi data di Grafana.
01/06/2024	Menyiapkan Alert Manager ke Telegram.

6. Implementasi

1. Pengaturan Lingkungan Pengembangan

Sebelum memulai implementasi, pastikan Docker telah terinstal terlebih dahulu. Docker akan digunakan untuk mengelola kontainer yang menjalankan aplikasi frontend dan backend. Selain itu, pastikan juga untuk memiliki Node.js dan npm terinstal untuk pengembangan aplikasi.

2. Pembuatan Aplikasi Frontend dan Backend



Aplikasi frontend dibangun menggunakan React, sementara aplikasi backend menggunakan Node.js dan Express framework. Aplikasi backend akan berkomunikasi dengan database SQLite untuk penyimpanan data. Pastikan untuk menginstal dependensi yang diperlukan menggunakan npm di kedua aplikasi. Di sini kami membuat aplikasi Todo List.

3. Docker Compose untuk Kontainerisasi

```

🔥 docker-compose.yml
1  version: '3.8'
2
3  services:
4    frontend:
5      build:
6        context: ./frontend
7        dockerfile: Dockerfile
8      ports:
9        - "4000:3000"
10     depends_on:
11       - backend
12
13     backend:
14       build:
15         context: ./backend
16         dockerfile: Dockerfile
17       ports:
18         - "5000:5000"
19

```

Definisikan Docker Compose file untuk menentukan konfigurasi kontainer Docker. Dalam file ini, spesifikasi setiap layanan, termasuk build dari Dockerfile yang sesuai dan pengaturan jaringan yang diperlukan. Jangan lupa untuk port forwarding service frontend agar portnya bukan 3000, karena port 3000 nanti akan digunakan oleh Grafana.

4. Integrasi dengan Prometheus dan Grafana

```

// Add the options to the prometheus middleware most option are for
http_request_duration_seconds histogram metric
const metricsMiddleware = promBundle({
  includeMethod: true,
  includePath: true,
  includeStatusCode: true,
  includeUp: true,
  customLabels: {project_name: 'backend', project_type: 'test_metrics_labels'},
  promClient: {
    collectDefaultMetrics: {
    }
  }
});

// add the prometheus middleware to all routes
app.use(metricsMiddleware)

```

Tambahkan Prometheus ke dalam project untuk memantau kesehatan dan kinerja aplikasi. Dalam aplikasi backend, tambahkan kode untuk mengumpulkan metrik yang relevan. Di sini kami menggunakan library prombundle. Setelah itu deploy dan run dengan cara **docker compose up --build**.

Sebelum masuk ke Prometheus dan Grafana, konfigurasi `/etc/prometheus/prometheus.yml` seperti ini:

```
GNU nano 7.2 /etc/prometheus/prometheus.yml
# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.

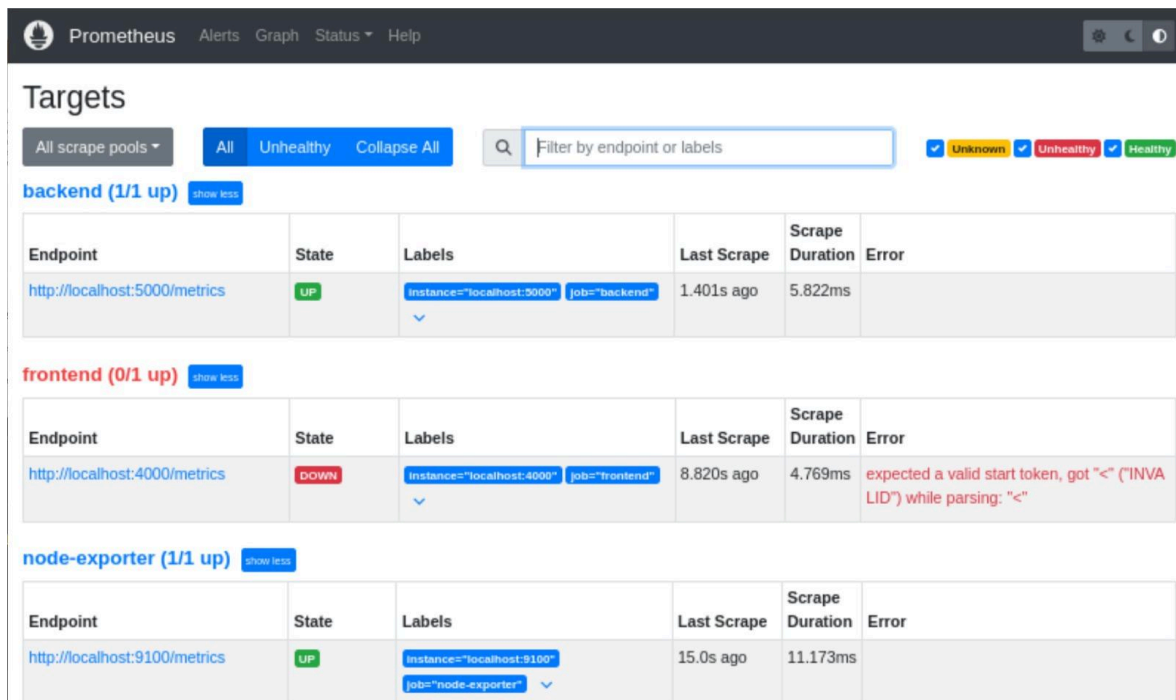
static_configs:
  - targets: ["localhost:9090"]

- job_name: "node-exporter"
  static_configs:
    - targets: ["localhost:9100"]

- job_name: 'frontend'
  static_configs:
    - targets: ['localhost:4000'] # Replace 'frontend' with your frontend service name

- job_name: 'backend'
  static_configs:
    - targets: ['localhost:5000'] # Replace 'backend' with your backend service name
```

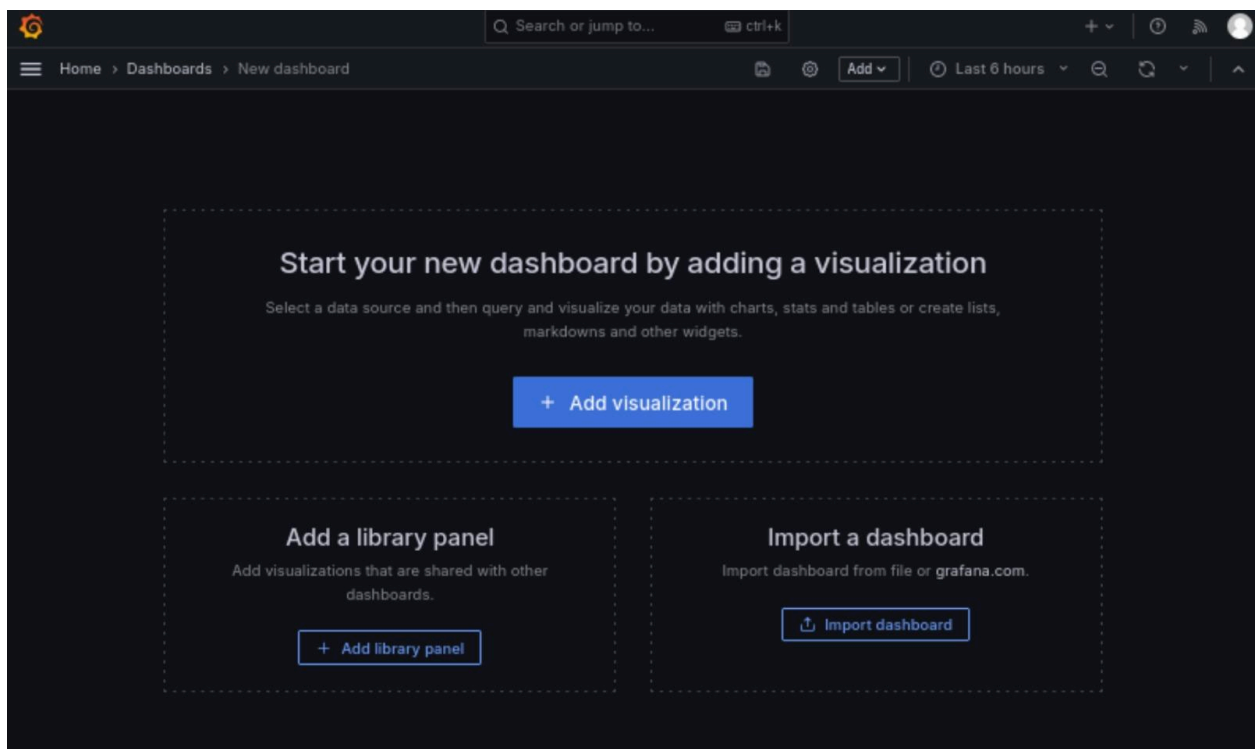
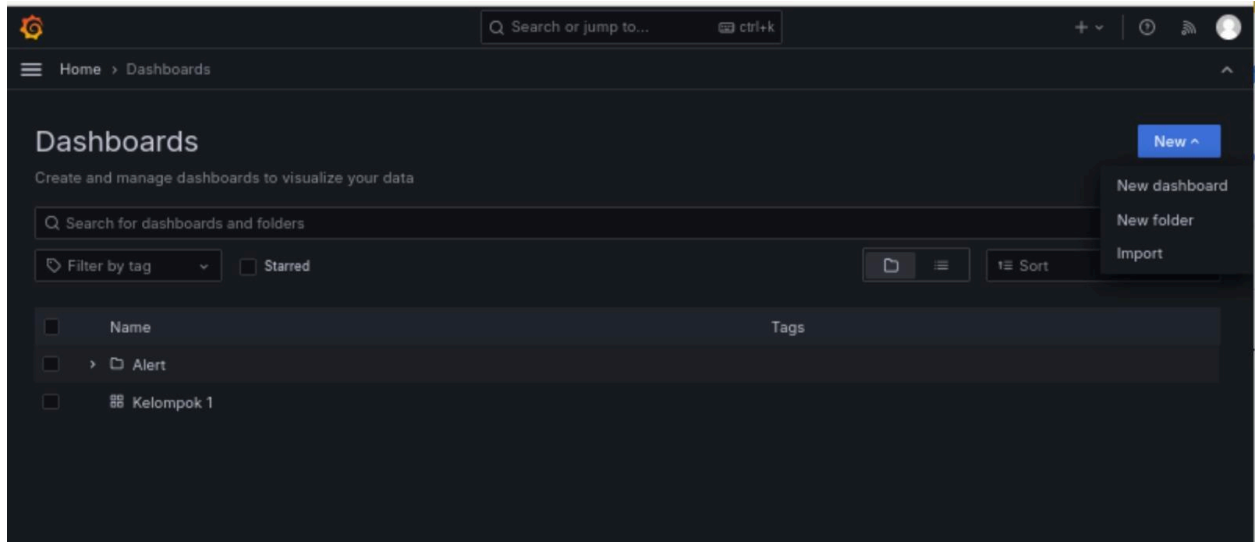
Lalu restart prometheusnya. Setelah itu di **Targets** Prometheus akan muncul Node Exporter, frontend, dan backend. Namun karena yang dikonfigurasi adalah node exporter dan backend, maka yang up hanya itu saja. Karena di sini kita tidak mengambil metrics dari frontendnya. Walaupun data yang up adalah node exporter dan backend, yang kita akan pakai adalah backend saja.



The screenshot shows the Prometheus web interface's 'Targets' page. It displays three scrape pools: 'backend' (1/1 up), 'frontend' (0/1 up), and 'node-exporter' (1/1 up). Each pool has a table with columns for Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:5000/metrics	UP	instance="localhost:5000" job="backend"	1.401s ago	5.822ms	
http://localhost:4000/metrics	DOWN	instance="localhost:4000" job="frontend"	8.820s ago	4.769ms	expected a valid start token, got "<" ("INVALID") while parsing: "<"
http://localhost:9100/metrics	UP	instance="localhost:9100" job="node-exporter"	15.0s ago	11.173ms	

Gunakan Grafana sebagai antarmuka visual untuk menganalisis dan memvisualisasikan data metrik dari Prometheus. Konfigurasi dashboard Grafana untuk menampilkan metrik yang penting bagi pengembangan aplikasi. Pertama kita menuju ke Grafana lalu klik “Dashboard”. Setelah itu tambahkan Dashboard baru seperti gambar di bawah ini:



Panel Title

No data

Query 1 Transform data 0 Alert 0

Data source prometheus MD = auto = 768 Interval = 30s Query inspector

A (prometheus)

Kick start your query Explain Run queries Builder Code

Metric Label filters

Select metric Select label = Select value x +

Time series

Search options

All Overrides

Panel options

Title

Panel Title

Description

Transparent background

Panel links

Repeat options

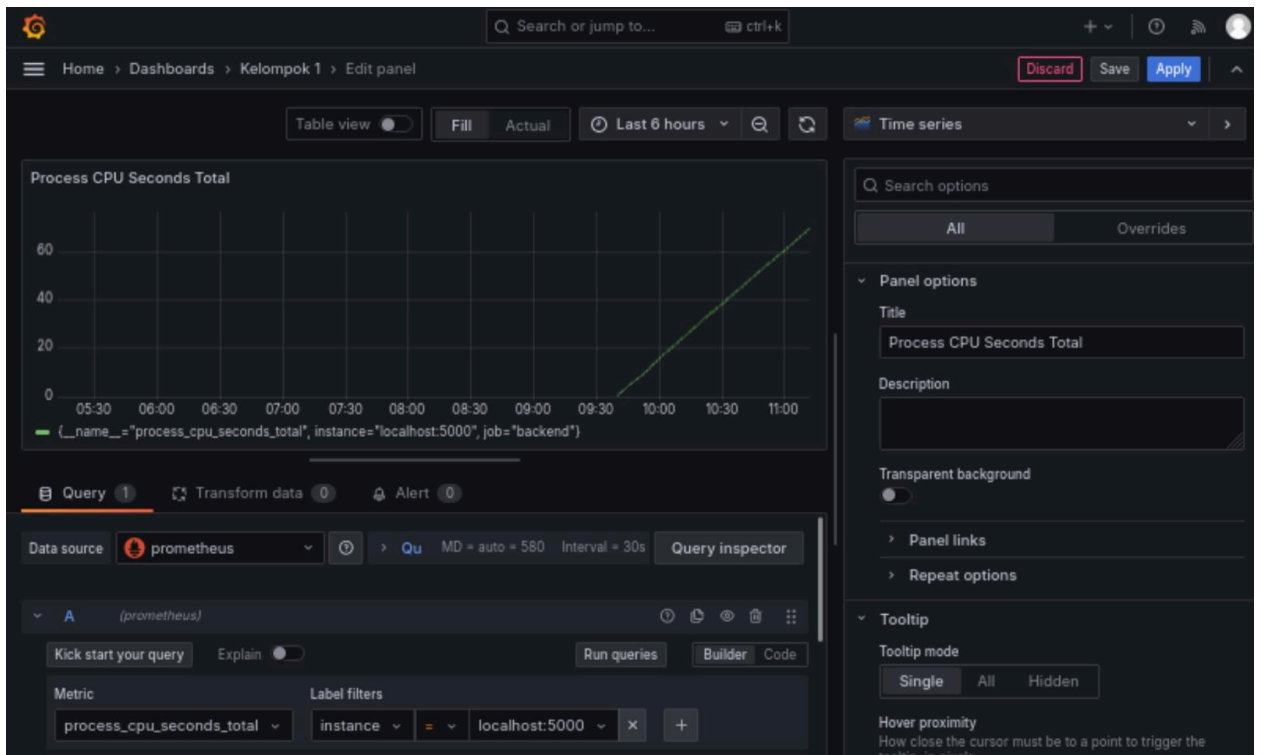
Tooltip

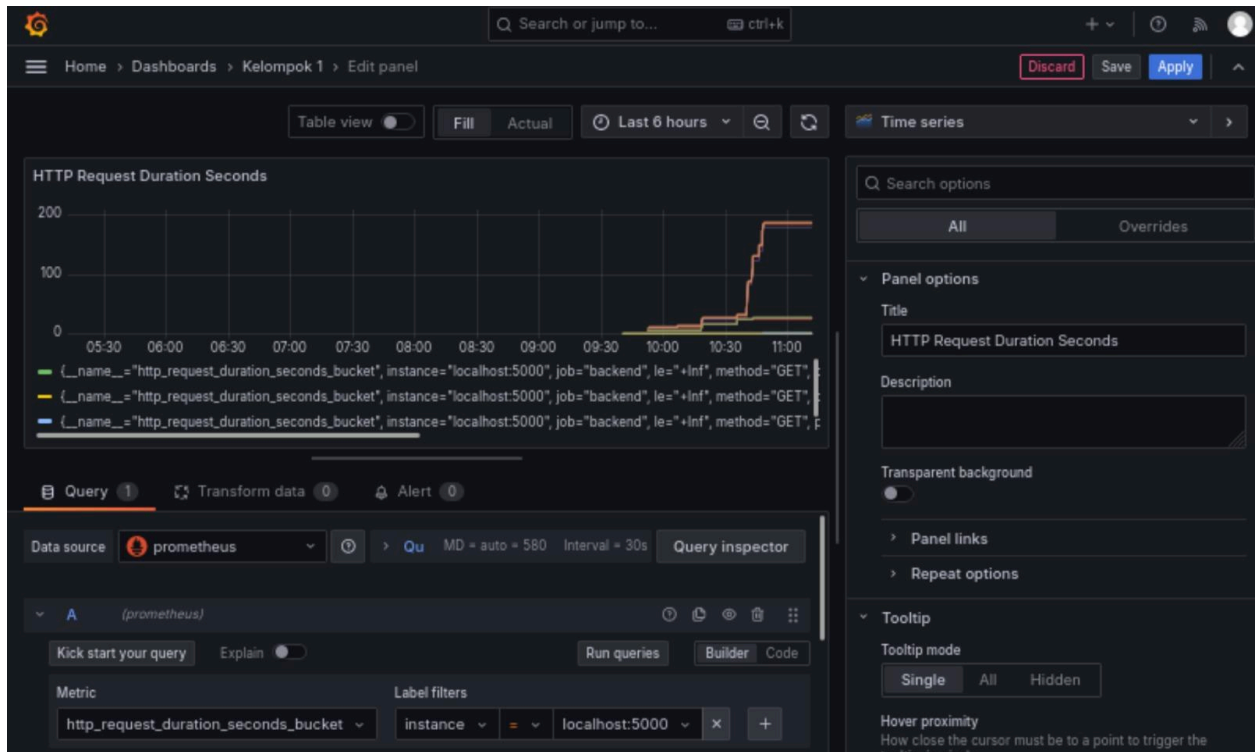
Tooltip mode

Single All Hidden

Hover proximity

How close the cursor must be to a point to trigger the tooltip in pixels





Di situ kita menambahkan Dashboard dengan data source dari Prometheus. Lalu kita bisa mendefinisikan metric mana yang akan dipantau, tambahkan label instance localhost:5000 untuk memantau hanya server backend dari container tadi saja.

Pada CPU Usage, yang dimonitor hanyalah CPU yang digunakan di service backendnya saja, bukan keseluruhan dari satu containernya. Ini adalah perbedaan utama menggunakan PromBundle dengan container monitoring lain seperti cAdvisor.

5. Konfigurasi Alertmanager

Tambahkan Alertmanager untuk memberikan notifikasi jika terjadi masalah dengan aplikasi. Konfigurasikan Alertmanager untuk memantau metrik dari Prometheus dan mengirimkan notifikasi melalui Telegram sesuai dengan rules tertentu.

Pertama kita klik **Home -> Alerting -> Alert Rules -> Contact Points**. Di sini kita mendefinisikan ke mana alert tersebut akan dikirim. Kita akan mengirimkan ke Telegram dengan konfigurasi seperti ini:

Contact points

Alertmanager Grafana

Choose how to notify your contact points when an alert instance fires

Update contact point

Name *

telegram-alert

Integration

Telegram

Test Duplicate Delete

BOT API Token

Configured Clear

Chat ID

Integer Telegram Chat Identifier

519307777

Optional Telegram settings

Notification settings

Lalu definisikan rules sesuai yang diinginkan. Di sini kita akan mendefinisikan jika metrics **CPU Usage** dan **HTTP Request Duration** melebihi *threshold* yang sudah ditentukan. Di sini *threshold*nya ditetapkan angka yang agak kecil untuk kebutuhan testing.

Edit rule

1. Enter alert rule name

Enter a name to identify your alert rule.

Name

telegram-alert-CPU-Usage

2. Define query and alert condition

Define query and alert condition [Need help?](#)

A prometheus Options 10 minutes, MD = 43200, Min. Interval = 1s Set as alert condition

Kick start your query Explain

Run queries Builder Code

Metric Label filters

process_cpu_seconds_total instance = localhost:5000

+ Operations hint: add rate

```
process_cpu_seconds_total {instance="localhost:5000"}
```

Options Legend: Auto Format: Time series Step: auto Type: Instant

Add query

Rule type

Select where the alert rule will be managed. ⓘ [Need help?](#)

Grafana-managed

Data source-managed

The alert rule type cannot be changed for an existing rule.

Expressions

Manipulate data returned from queries with math and other operations.

B Reduce

Set as alert condition ⓘ

Takes one or more time series returned from a query or an expression and turns each series into a single number.

Input A ▾

Function Last ▾ Mode Strict ▾

C Threshold

✓ Alert condition ⓘ

Takes one or more time series returned from a query or an expression and checks if any of the series match the threshold condition.


Input B ▾

IS ABOVE ▾ 20

Custom recovery threshold ☐

Add expression ▾

Preview



Search or jump to... ctrl+k

+ v 🔔 🔊 👤

Home > Alerting > Alert rules > Edit rule

Save rule Save rule and exit Cancel Delete

3. Set evaluation behavior

Define how the alert rule is evaluated. [Need help?](#)

Folder
Select a folder to store your rule.

Alert or + New folder

Evaluation group
Rules within the same group are evaluated concurrently over the same time interval.

Default or + New evaluation group


All rules in the selected group are evaluated every 10s. [🔗](#)

Pending period
Period in which an alert rule can be in breach of the condition until the alert rule fires.

1m

☐ Pause evaluation [🔔](#)

> Configure no data and error handling



Search or jump to... ctrl+k

+ v 🔔 🔊 👤


Home > Alerting > Alert rules > Edit rule

Save rule Save rule and exit Cancel Delete

4. Configure labels and notifications

Select who should receive a notification when an alert rule fires.

Labels
Add labels to your rule for searching, silencing, or routing to a notification policy. [Need help?](#)


Choose key = Choose value 

+ Add label


Notifications
Select who should receive a notification when an alert rule fires.


Select contact point Use notification policy

Notifications for firing alerts are routed to a selected contact point. [Need help?](#)

Alert manager:  grafana

Contact point

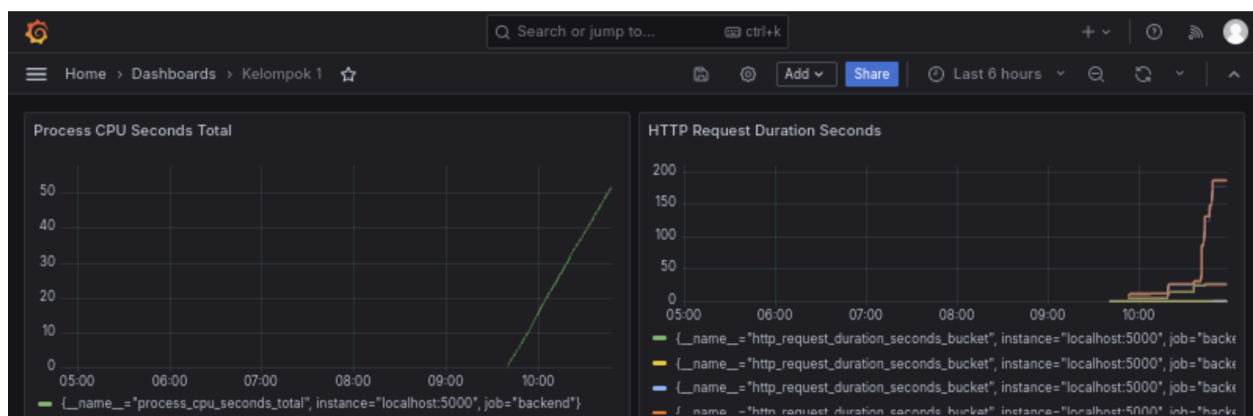
telegram-alert  [View or create contact points](#) [🔗](#)

Muting, grouping and timings (optional) 

The screenshot shows the Grafana Alert Rule configuration interface. At the top, there's a search bar and navigation links. The main section is titled '5. Add annotations' and includes a 'Summary (optional)' field with a text area containing the message: 'There are too many process that happened in the CPU, you may need to check your server.' Below this is a 'Description (optional)' field with a text area labeled 'Enter a description...'. At the bottom, there's a 'Runbook URL (optional)' field with a text area containing 'https://'. There are also buttons for '+ Add custom annotation' and 'Link dashboard and panel'.

7. System Testing

Proses system testing dilakukan untuk memastikan sistem pemantauan yang dibuat sudah dapat berjalan sesuai dengan yang diharapkan. System testing dilakukan terhadap pemantauan metriks CPU dan jangka waktu request HTTP yang telah dibuat pada Grafana seperti gambar di bawah ini.



Dari grafik yang diberikan Grafana, dapat dilihat bahwa penggunaan CPU dan request HTTP per detiknya ada yang meningkat secara drastis pada detik-detik tertentu. Hal ini dapat menandakan adanya ketidaknormalan saat aplikasi berjalan. Oleh karena itu, alert system

dapat kita pasang untuk mendeteksi anomali-anomali yang terjadi. Alert System akan mengirimkan sebuah pesan alert apabila telah memenuhi satu set alert rule yang kita buat. Pesan yang dikirimkan akan mengalami *pending* minimal selama satu menit setelah *alert* menyala

3. Set evaluation behavior
Define how the alert rule is evaluated. [Need help?](#)

Folder
Select a folder to store your rule.
Alert or + New folder

Evaluation group
Rules within the same group are evaluated concurrently over the same time interval.
Default or + New evaluation group

All rules in the selected group are evaluated every 1m. [🔗](#)

Pending period
Period in which an alert rule can be in breach of the condition until the alert rule fires.
1m

☐ Pause evaluation [🔗](#)

Threshold adalah batas minimum nilai yang ditetapkan untuk memicu *alerts* dari *metrics* yang diinginkan. Jika nilai *metrics* melebihi *threshold*, maka *alerts* “Firing” akan dikirimkan melalui *contact point* yang telah ditetapkan. Pada proyek ini, kami menggunakan Telegram sebagai *contact point*-nya. Jika nilai *metrics* mulai menurun dan mencapai kurang dari *threshold* yang ditetapkan, maka *alerts* “Resolved” akan terkirim.

Alert Rule CPU Usage

Rule type
Select where the alert rule will be managed. [Need help?](#)
Grafana-managed Data source-managed

The alert rule type cannot be changed for an existing rule.

Expressions
Manipulate data returned from queries with math and other operations.

B Reduce [Set as alert condition](#)

Takes one or more time series returned from a query or an expression and turns each series into a single number.

Input A

Function Last Mode Strict

C Threshold [Alert condition](#)

Takes one or more time series returned from a query or an expression and checks if any of the series match the threshold condition.

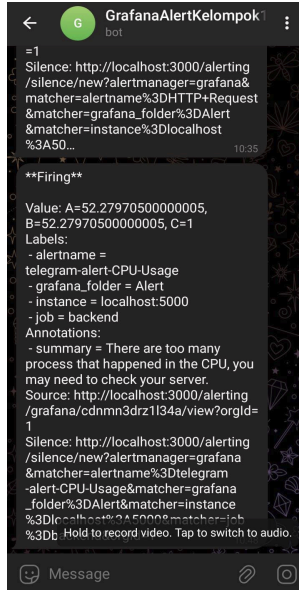
Input B

IS ABOVE 100

Custom recovery threshold ☐

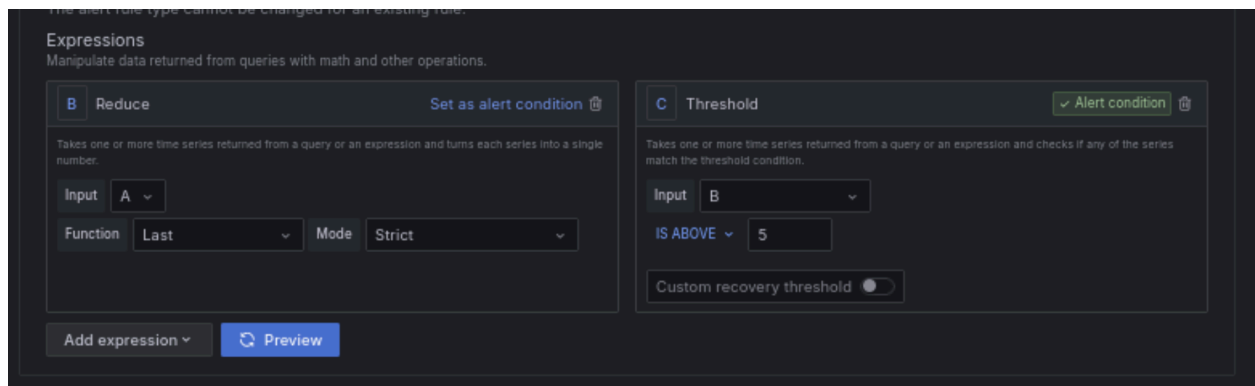
[Add expression](#) [Preview](#)

Setelah *alert rule* telah kita setup, dilakukan testing tesrhadap CPU usage dari container dengan melakukan *refresh* berkali-kali dengan frekuensi dan kecepatan tinggi terhadap aplikasi yang dipantau. Hal ini dapat membuat banyak process yang berjalan pada CPU, sehingga melebihi *threshold* yang telah ditentukan pada saat pembuatan alert rule, dan memicu terkirimnya alert pada *touchpoint* telegram yang telah ditentukan.

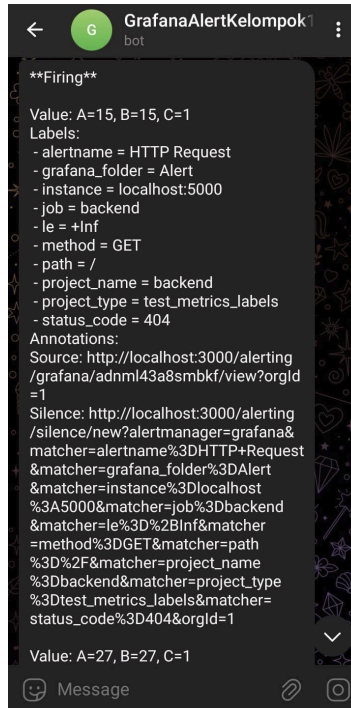


Pesan tersebut menampilkan *alerts* “Firing” karena pada CPU Usage mencapai 57% sedangkan Threshold yang ditentukan adalah 20.

Alert Rule HTTP Request



Setelah *alert rule* HTTP Request telah kita konfigurasi, kita bisa melakukan testing dengan melakukan *refresh* berkali-kali untuk mengirimkan banyak HTTP request terhadap aplikasi yang dipantau secara terus menerus. Dari situ, banyak HTTP request yang dikirimkan sehingga melebihi *threshold* yang telah ditentukan pada saat pembuatan alert rule, dan memicu terkirimnya alert pada *touchpoint* telegram yang telah ditentukan.



Pesan tersebut menampilkan *alerts* “Firing” karena HTTP Request mencapai 15 sedangkan Threshold yang ditentukan adalah 5.

Alert “Firing” akan berubah menjadi “Resolved” jika nilai *Metrics* tidak melebihi *Threshold* yang ditentukan. Saat melakukan *testing*, kami meningkatkan nilai Threshold secara signifikan di luar rentang *Metrics*. Nilai *Metric* akan tetap dinamis, namun kemungkinan besar tetap berada dalam rentang tersebut. Penyesuaian *Threshold* ini dilakukan secara drastis untuk memastikan bahwa nilai metrik tidak akan mencapai *Threshold* sehingga akan muncul *Alert* “Resolved”.