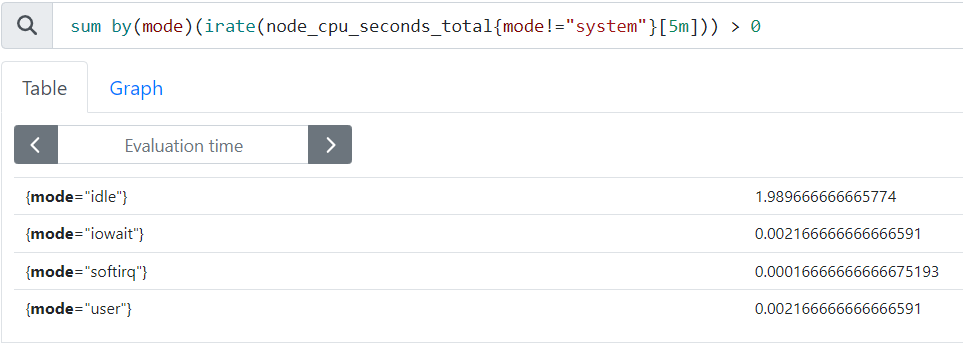
# Hàm

irate(node\_cpu\_seconds\_total[5m])

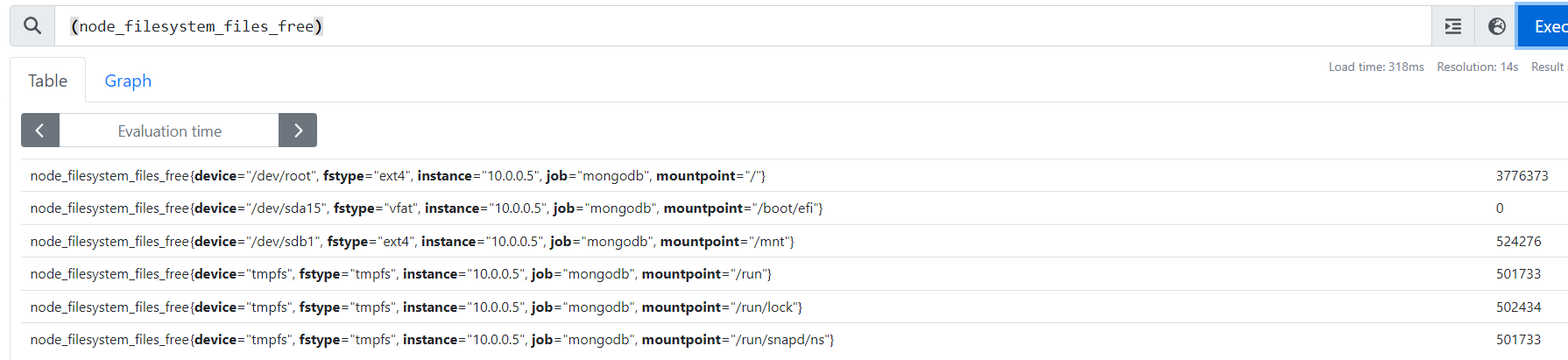
irate tính toán tốc độ tăng trưởng trung bình của các giá trị đo lường trong một khoảng thời gian cố định (tức là cứ 5phut 1 tăng dần lên)

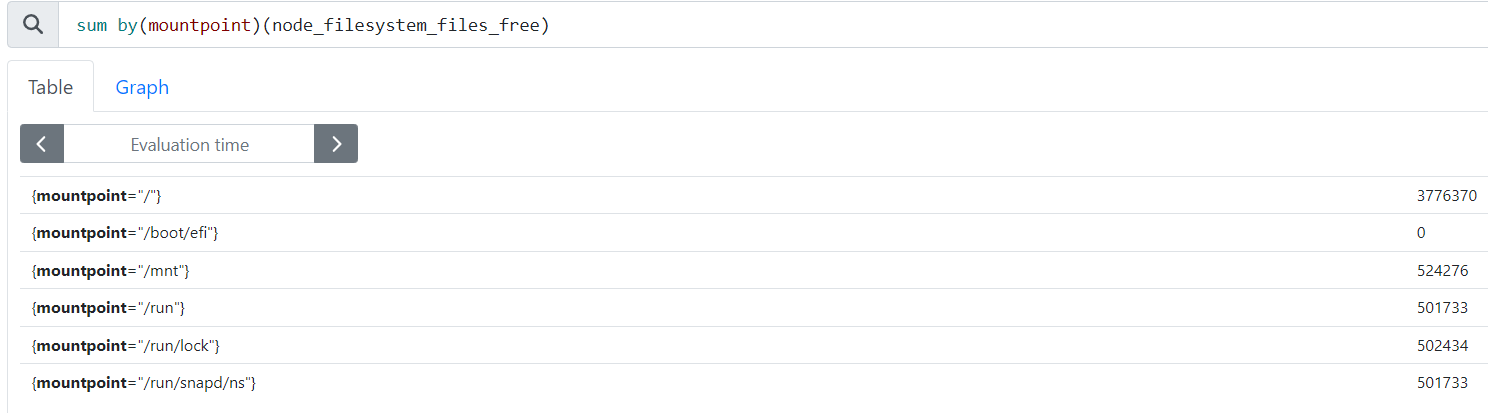
sum by(mode)(irate(node\_cpu\_seconds\_total{mode!="system"}[5m])) > 0

tổng irate của từng mode



Tổng của tất cả các mode cùng tên lại





# What is Monitoring

monitoring là cách ta sử dụng các công cụ để giám sát hệ thống và ứng dụng của ta. Monitoring sẽ cung cấp cho ta các giá trị về hiệu suất, trình trạng của hệ thống và ứng dụng, để ta có thể nhanh chóng phát hiện các vấn đề xảy ra với hệ thống.

Nhưng monitoring không dừng lại ở việc chỉ giám sát các giá trị về hệ thống như CPU hoặc memory, mà còn bao gồm các giá trị có thể giúp ích về mặt kinh doanh như số lượng khách hàng truy cập trang web, thời gian khách hàng ở lại trang web, số lượng khách hàng bấm vào một sản phẩm nào đó, ...

Nên đối với monitoring ta sẽ có hai đối tượng là:

* Hệ thống (the system).
* Kinh doanh (the business).

## System Monitoring

Infrastructure monitoring: CPU, Memory, disk

Application monitoring: monitor service, thời gian phản hồi của ứng dụng, tình trạng của ứng dụng (chạy hay chết), CPU và memory ứng dụng, số lượng yêu cầu (request) vào ứng dụng trong 1 giây, ...

Network monitoring: tầng mạng - các giá trị như số lượng *byte* ra và vào, tình trạng của firewall, ...

## Business

Để monitoring được tương tác của người dùng thì ta cần phải thực hiện ở tầng code front-end hoặc backend, ví dụ ở tầng front-end để theo dõi được hành vi của người dùng ta sẽ dùng Google Analytics.

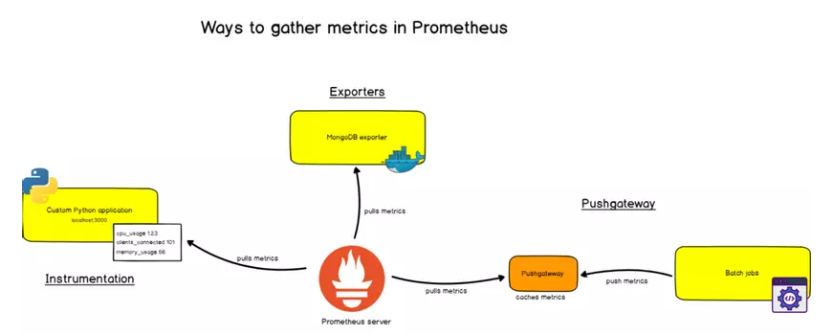
# Prometheus

Dữ liệu mà nó lưu trữ có thể là các metrics (thông số) về tình trạng server như lượng RAM, CPU

Prometheus sẽ chủ động pull (kéo) các metrics về qua HTTP. Bản thân các service thì thường không thể tự export được các metrics cho Prometheus mà cần đến các Instructmentation/Exporter

Install exporter để monitor các service

* Exporter là những app được viết cho mấy cái thông dụng như Database, Server. Chúng ta chỉ cần chạy nó và nó sẽ export các metrics thu thập được cho mình.



Một số exporter

* Node Exporter: export các metrics một con node (hiểu là một server) như CPU, RAM của node, dung lượng ổ đĩa, số lượng request tới node đấy, .etc.
* Postgres Exporter, giúp đọc dữ liệu từ các bảng trong Postgres và export ra cho Prometheus
* HAProxy Exporter

# Prometheus – grafana

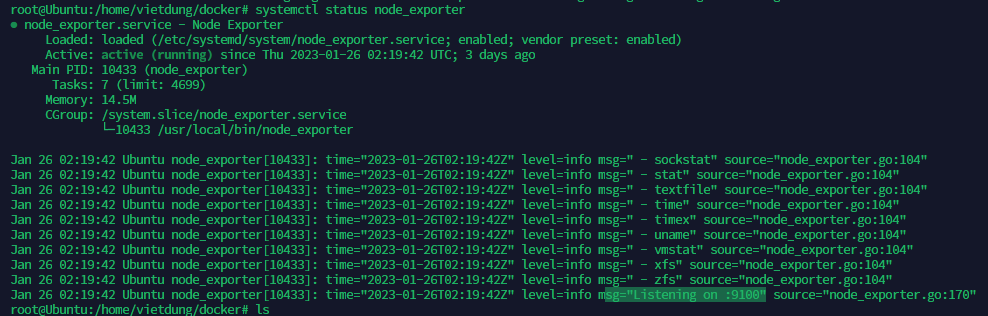
Grafana cung cấp giao diện để monitor,

Grafana kết hợp các plugin để monitor, bao gồm cả prometheus

Prometheus là công cụ để monitor metric của server

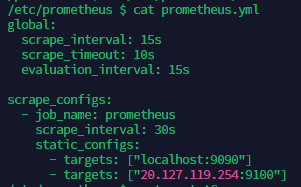
Prometheus cũng là 1 builtin exporter để monitor server port 9090

Nhưng n ko monitor được 1 số thông số của server, ta cần node\_exporter port 9100



Listen port 9100

* Ta có file config của prometheus



* Prometheus monitor cả nó và node\_expoter

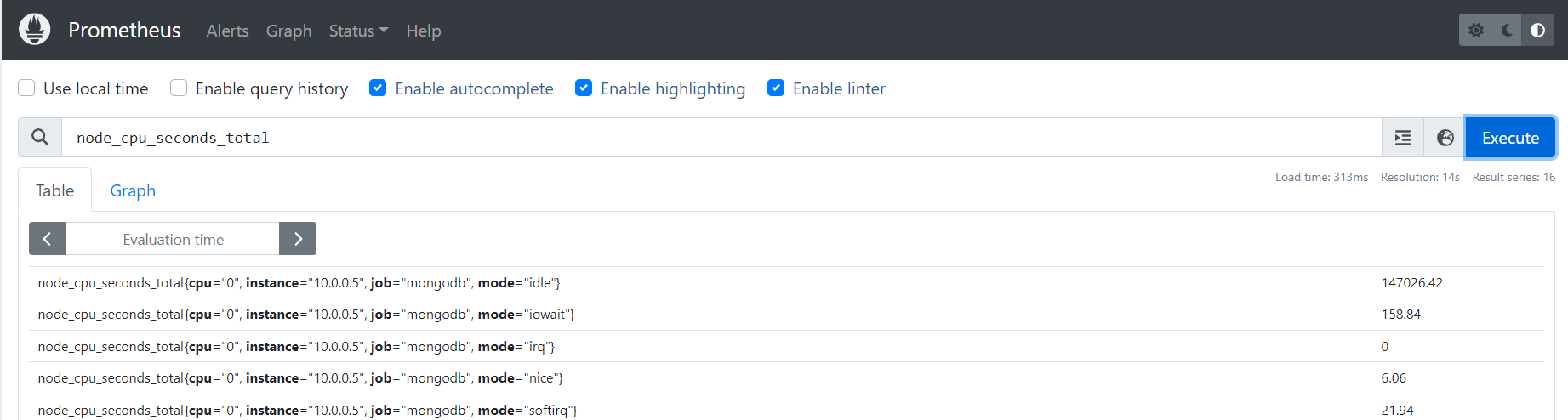
Vì vậy grafana chỉ cần add prometheus vào là sẽ lấy được cả log của node\_exporter

Logs đẩy về prometheus, prometheus đẩy về grafana

## Tính CPU

## CPU Utilization

Metric mà Node Exporter thu thập về cho CPU Utilization là node\_cpu\_seconds\_total, để kiểm tra ta truy cập địa chỉ http://localhost:9090/graph và gõ vào node\_cpu\_seconds\_total{}.

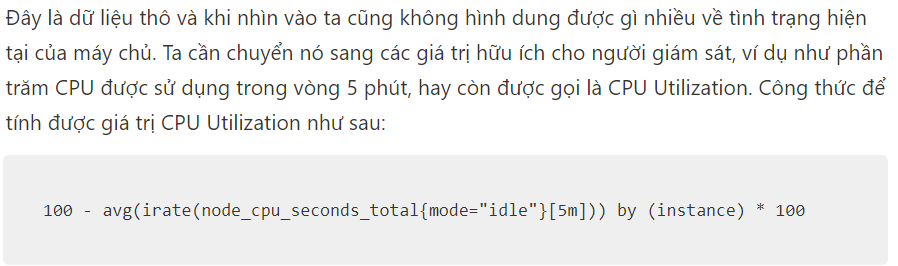


node\_cpu\_seconds\_total{**cpu**="0", **instance**="10.0.0.5", **job**="mongodb", **mode**="idle"}

* Hai labels instance và job giúp xác định metric này được thu thập từ đâu

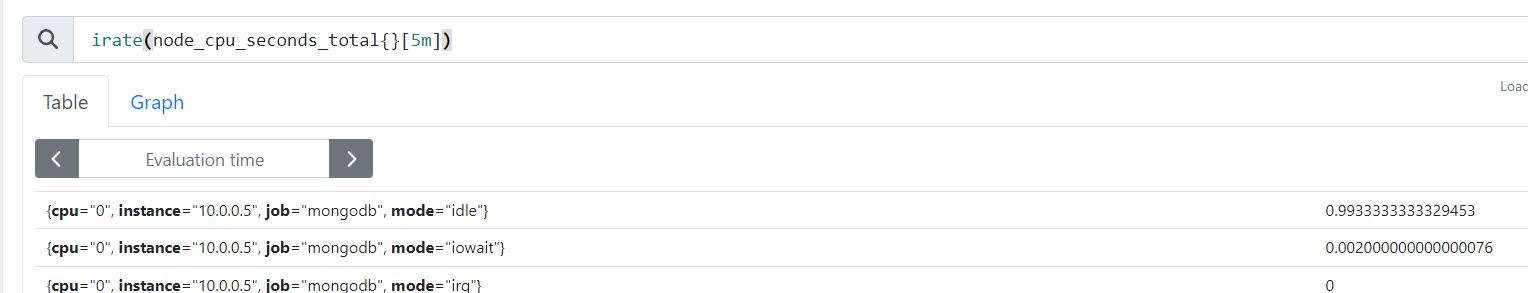
Label cpu chỉ định thứ tự *core* của CPU trong máy chủ, ví dụ ta hay nghe CPU có 2 core hay 4 core

* Label mode là chế độ của CPU như: user, system, idle, …



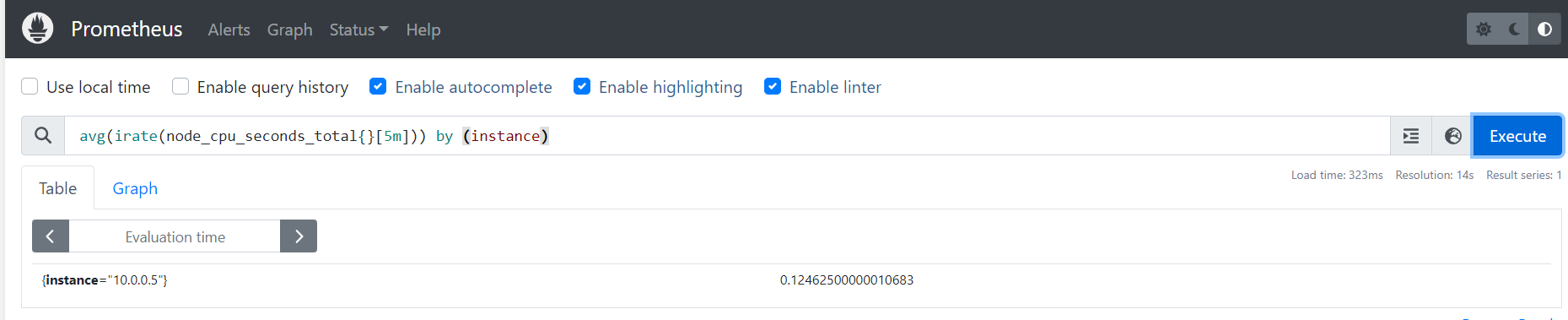
- tính được CPU Utilization đầu tiên ta cần tính *per-second rate* của CPU. tỷ lệ trong vòng 5 phút:

irate(node\_cpu\_seconds\_total{}[5m])



Sau đó ta tổng hợp các giá trị này theo Instance:

avg(irate(node\_cpu\_seconds\_total{}[5m])) by (instance)



**avg(...) by (instance)**: Tính trung bình giá trị được tính toán trước đó theo từng instance, tức là theo từng máy chủ.

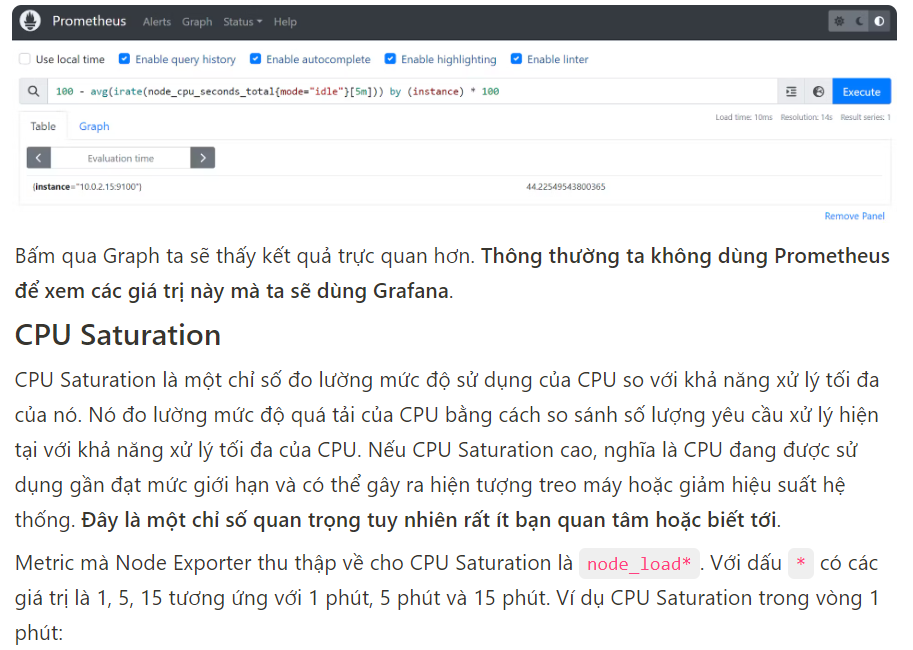
Nên để tính được tài nguyên CPU đang được sử dụng, ta sẽ lấy tổng tài nguyên của CPU (100) trừ đi cho tài nguyên đang ở trạng thái idle. Phần trăm tài nguyên đang ở trạng thái idle:

avg(irate(node\_cpu\_seconds\_total{mode="idle"}[5m])) by (instance) \* 100

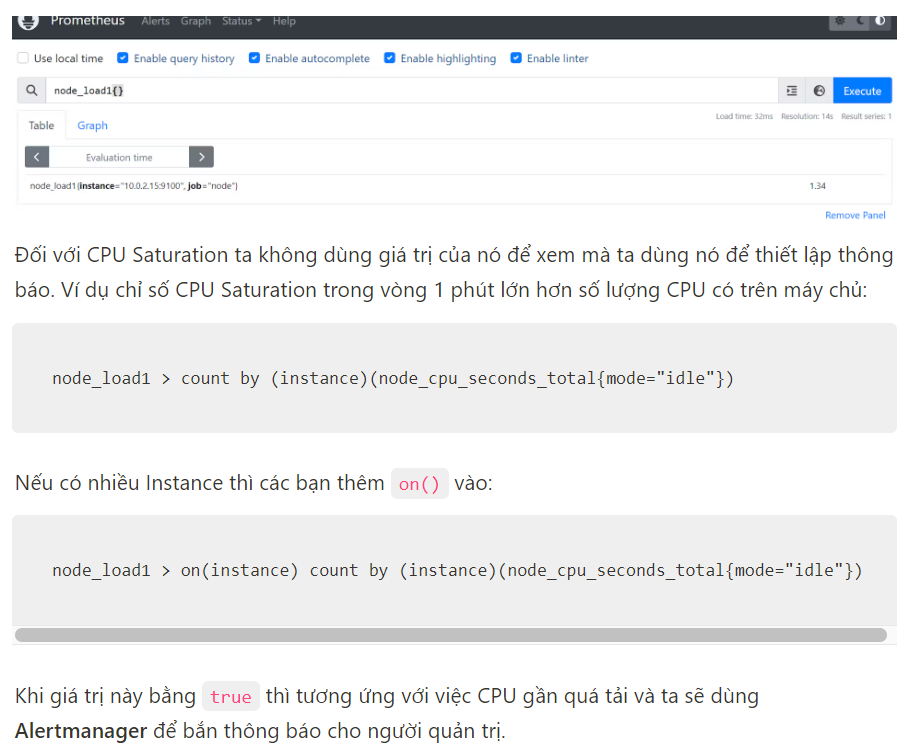
Phần trăm CPU đang được sử dụng (CPU Utilization):

100 - avg(irate(node\_cpu\_seconds\_total{mode="idle"}[5m])) by (instance) \* 100

* idle: Chế độ này đại diện cho thời gian CPU không được sử dụng hoặc đang rảnh rỗi.
* system: Chế độ này đại diện cho thời gian CPU được sử dụng cho các hoạt động của hệ thống, chẳng hạn như phục vụ các truy vấn I/O.
* user: Chế độ này đại diện cho thời gian CPU được sử dụng cho các tiến trình người dùng.
* Idle là CPU trống, free thì 1 - idle = CPU đang sử dụng



node\_load1{}



# Triển khai PMM

Install with docker

Add data source promethues với loki



Cái này chỉ là import những dashboarch có sẵn

Promtail (đẩy log) -> loki -> grafana

Prometheus pull metric từ (node exporter, mongo\_exporter) -> grafana

Prometheus sẽ scrape mấy thằng này, trừ thằng loki

## Monitor mongodb

1. Bắn log mongodb

Cài promtail

Promtail (đẩy log) -> loki -> grafana

PROMTAIL\_VERSION=$(curl -s "https://api.github.com/repos/grafana/loki/releases/latest" | grep -Po '"tag\_name": "v\K[0-9.]+')

sudo mkdir /opt/promtail

sudo wget -qO /opt/promtail/promtail.gz "https://github.com/grafana/loki/releases/download/v${PROMTAIL\_VERSION}/promtail-linux-amd64.zip"

sudo gunzip /opt/promtail/promtail.gz

sudo chmod a+x /opt/promtail/promtail

sudo ln -s /opt/promtail/promtail /usr/local/bin/promtail

sudo wget -qO /opt/promtail/promtail.yaml "https://raw.githubusercontent.com/grafana/loki/v${PROMTAIL\_VERSION}/clients/cmd/promtail/promtail-local-config.yaml"

promtail -version

sudo tee /etc/systemd/system/promtail.service<<EOF

[Unit]

Description=Promtail client for sending logs to Loki

After=network.target

[Service]

ExecStart=/opt/promtail/promtail -config.file=/opt/promtail/promtail.yaml

Restart=always

TimeoutStopSec=3

[Install]

WantedBy=multi-user.target

EOF

sudo tee /opt/promtail/promtail.yaml<<EOF

server:

  http\_listen\_port: 9080

  grpc\_listen\_port: 0

positions:

  filename: /tmp/positions.yaml

clients:

  - url: http://192.168.48.1:3100/loki/api/v1/push

scrape\_configs:

- job\_name: MongoDB\_logs

  static\_configs:

  - targets:

      - localhost

    labels:

      job:  mongodb-1

**\_\_path\_\_**: /var/log/mongodb/\*.log

EOF

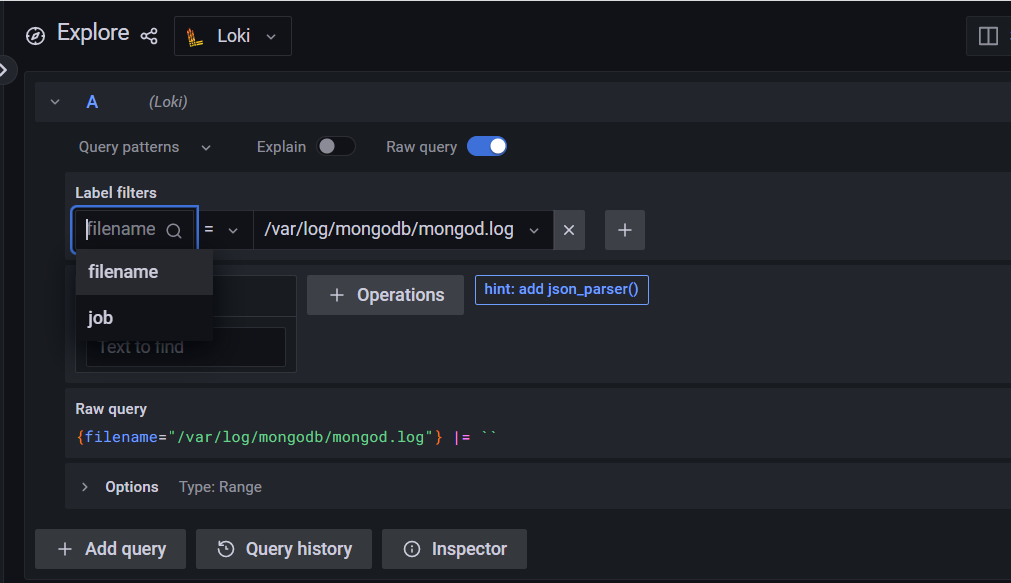
sudo systemctl enable promtail

sudo service promtail restart

sudo systemctl status promtail

đẩy log đến loki

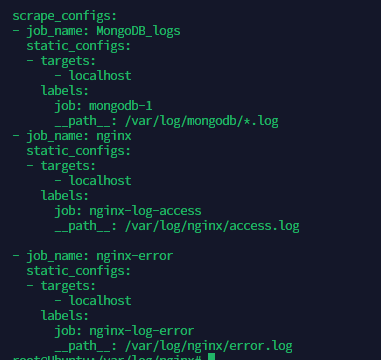
* check log promtail oke chưa
* check loki service oke chưa
* check data source được add chưa

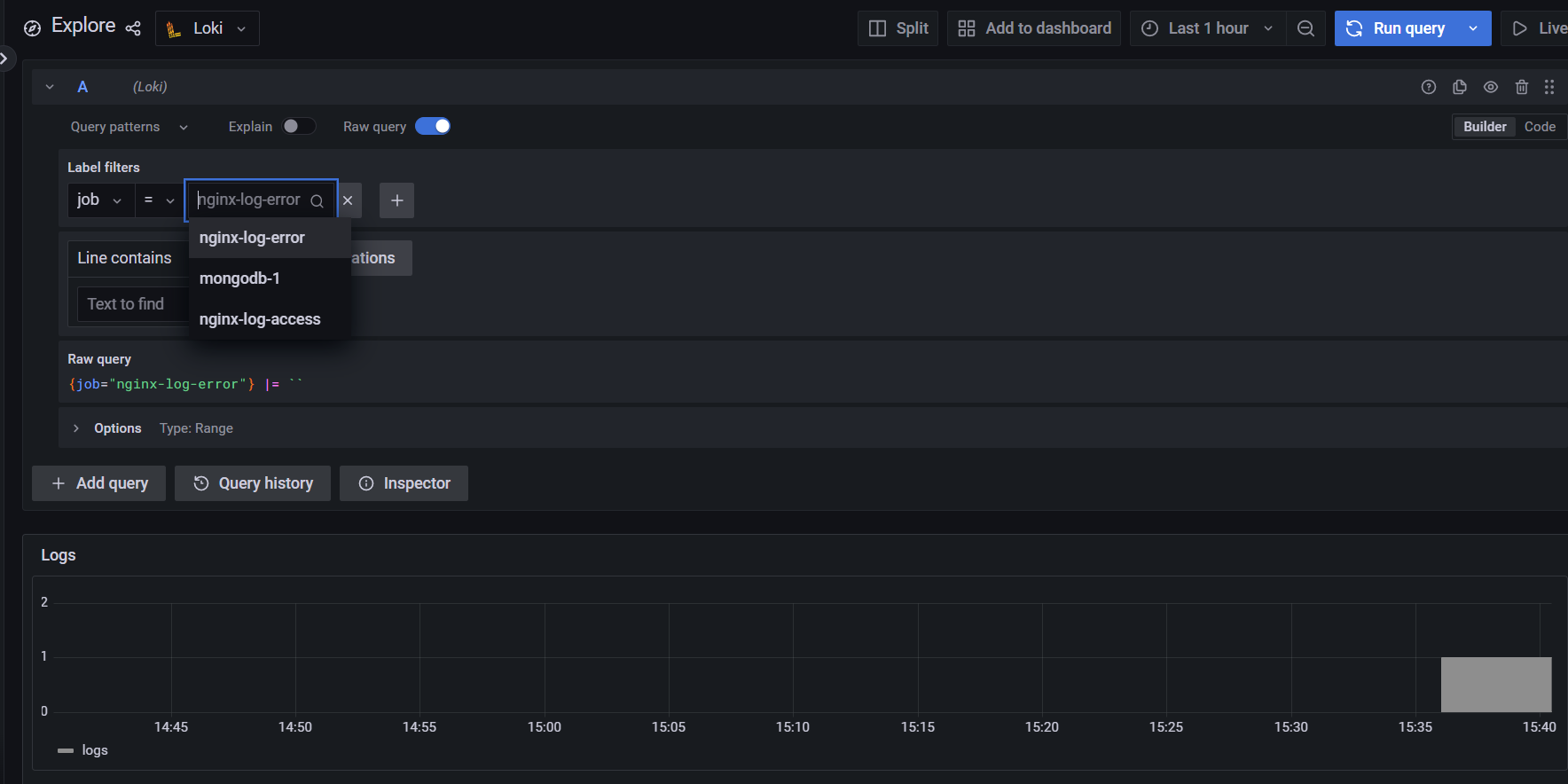


Và oke, 1 là filename

* 2 là cái job (là cái label mình vừa add ở file config promtail)

Ta cũng có thể chia chi tiết cho log error và log access trong promtail





Chia log chi tiết thế này oke hơn.

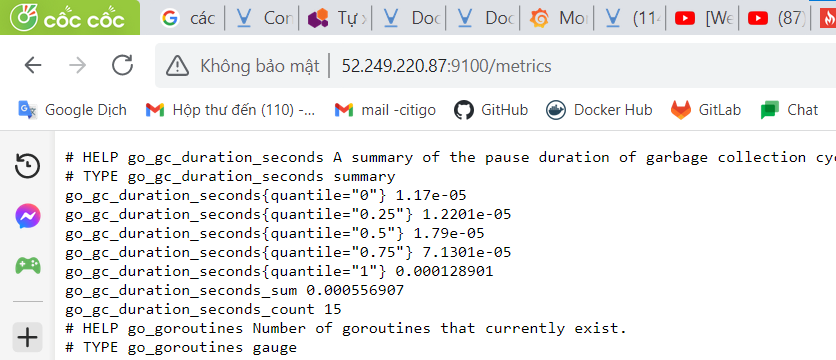
Oke xog phần log

1. Phần dashboarsh metric

* Monitor node

Install node\_exporter

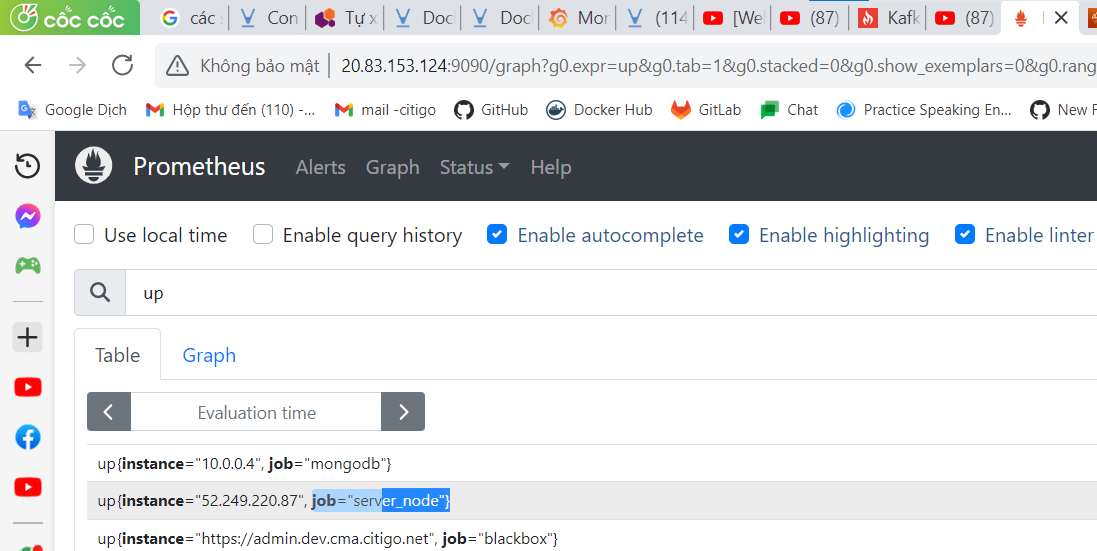
* Check node\_exporter đã hoạt động chưa



Oke node\_exporter đã pull được metric

* Check prometheus đã scrape node exporter được chưa

Vào prometheus check query



Oke n đã scrape được

* Vậy là node\_exporter xog

## Cảnh báo mongodb down

Set 1 cảnh báo cho mongodb

groups:

- name: mongodb

  rules:

  - alert: MongodbDown

    expr: mongodb\_up == 0

    for: 0m

    labels:

      severity: critical

    annotations:

      summary: MongoDB Down (instance {{ $labels.instance }})

      description: "MongoDB instance is down\n  VALUE = {{ $value }}\n  LABELS = {{ $labels }}"

lưu ý phần label để set alert

    labels:

      severity: critical

* Alert manager có 2 phần quan trọng: **route** và **receivers**

route:

  group\_by: ['alertname']

  receiver: 'slack-notifications'

  routes:

  - receiver: 'slack-alert1'

    group\_wait: 10s

    group\_interval: 1m

    repeat\_interval: 3m        # cứ 3 phút là lặp lại 1 lần

    match:

      severity: 'critical'

  - receiver: 'slack-alert2'

    group\_wait: 30s

    group\_interval: 1m

    repeat\_interval: 3m

    match:

      severity: 'warning'

định nghĩa 1 route có tên là **slack-notifications** để định tuyến các cảnh báo đến **receiver** Slack

Nếu các cảnh báo có nhãn **severity: critical**, chúng sẽ được gửi đến **receiver: slack-alert1**

Nhãn được gắn vào từng cảnh báo

routes:

  - receiver: 'receiver name'

  - labels:

* **receiver**: định nghĩa **receiver** sẽ được sử dụng cho các **alert** có thuộc tính **severity** bằng **critical**.
* **group\_by**: xác định các thuộc tính của **alert** để nhóm chúng vào các nhóm cảnh báo khác nhau.
* **group\_wait**: định nghĩa thời gian tối đa để đợi các **alert** cùng nhóm được nhận trước khi gửi chúng đến các **receiver**.
* **group\_interval**: xác định thời gian tối thiểu giữa các lần gửi cùng một nhóm cảnh báo cho các **receiver**.
* **repeat\_interval**: định nghĩa khoảng thời gian giữa các lần gửi lại thông báo cảnh báo nếu **alert** vẫn đang được kích hoạt và chưa được giải quyết.
* **routes**: xác định các tuyến đường để định tuyến các **alert** đến các **receiver**.

## cấu hình alertmanaget.yml

global:

  resolve\_timeout: 1m

  slack\_api\_url: 'https://hooks.slack.com/services/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'

route:

  receiver: slack-notifications   # receiver default, mặc định phải set 1 receiver default

  group\_by: ['alertname']

  group\_wait: 30s

  group\_interval: 1m

  repeat\_interval: 3m

  routes:

  - match:

      severity: critical

    receiver: slack-notifications

  - match:

      severity: warning

    receiver: slack-notifications-log

receivers:

- name: 'slack-notifications'

  slack\_configs:

  - channel: '#test'

    send\_resolved: true

    icon\_url: https://avatars3.githubusercontent.com/u/3380462

    title: |-

     [{{ .Status | toUpper }}{{ if eq .Status "firing" }}:{{ .Alerts.Firing | len }}{{ end }}] {{ .CommonLabels.alertname }} for {{ .CommonLabels.job }}

     {{- if gt (len .CommonLabels) (len .GroupLabels) -}}

       {{" "}}(

       {{- with .CommonLabels.Remove .GroupLabels.Names }}

         {{- range $index, $label := .SortedPairs -}}

           {{ if $index }}, {{ end }}

           {{- $label.Name }}="{{ $label.Value -}}"

         {{- end }}

       {{- end -}}

       )

     {{- end }}

    text: >-

     {{ range .Alerts -}}

     \*Alert:\* {{ .Annotations.title }}{{ if .Labels.severity }} - `{{ .Labels.severity }}`{{ end }}

     \*Description:\* {{ .Annotations.description }}

     \*Details:\*

       {{ range .Labels.SortedPairs }} • \*{{ .Name }}:\* `{{ .Value }}`

       {{ end }}

     {{ end }}

- name: 'slack-notifications-log'

  slack\_configs:

    - send\_resolved: true

      channel: '#test'

      title: '{{ if eq .Status "firing" }}:fire:{{ else }}:white\_check\_mark:{{ end }} [{{ .Status | toUpper }}] {{ .CommonAnnotations.title }} '

      text: >-

        {{ range .Alerts }}

          \*Description:\* {{ .Annotations.description }}

          \*Severity:\* `{{ .Labels.severity }}`

          \*Graph:\* {{ if eq .Labels.category "logs" }}<{{ .Annotations.logurl }}|:chart\_with\_upwards\_trend:>{{ else }}<{{ .GeneratorURL }}|:chart\_with\_upwards\_trend:>{{ end }} \*Dashboard:\* <{{ .Annotations.dashboard }}|:bar\_chart:> \*Runbook:\* <{{ .Annotations.runbook }}|:spiral\_note\_pad:>

          \*Details:\*

          {{ range .Labels.SortedPairs }} - \*{{ .Name }}:\* `{{ .Value }}`

          {{ end }}

        {{ end }}

**send\_resolved** là một trường được sử dụng trong cấu hình của Alertmanager để xác định xem có gửi thông báo khi một **alert** được giải quyết hay không. Nếu **send\_resolved** được đặt thành **true**, khi một **alert** được giải quyết, Alertmanager sẽ gửi một thông báo để thông báo rằng **alert** đã được giải quyết.

## lệnh kiểm tra slack có bắn alert ko

curl -X POST -H 'Content-Type: application/json' -d '[{"labels":{"severity":"critical"},"annotations":{"description":"This is an example alert"}}]' http://10.0.0.4:9093/api/v1/alerts

# Alert manager

Slow query mongodb

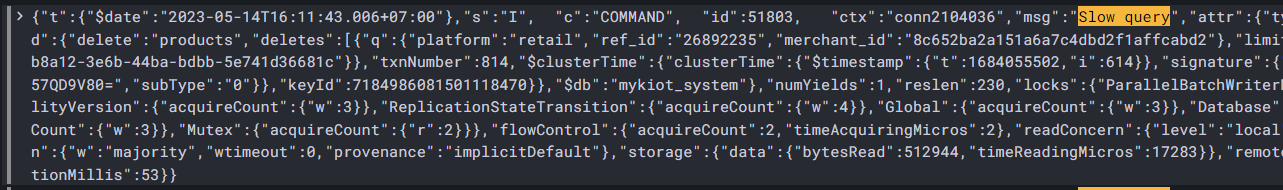
check slow query mongodb

- câu lệnh check

db.getProfilingStatus()



hiện tại đang cấu hình với request là 50ms thì sẽ bắn vào log mongodb coi như là 1 query chậm



VD ở đây durationMillis = 53 > 50

Nếu chưa set thì set bằng cách chui vào mongodb

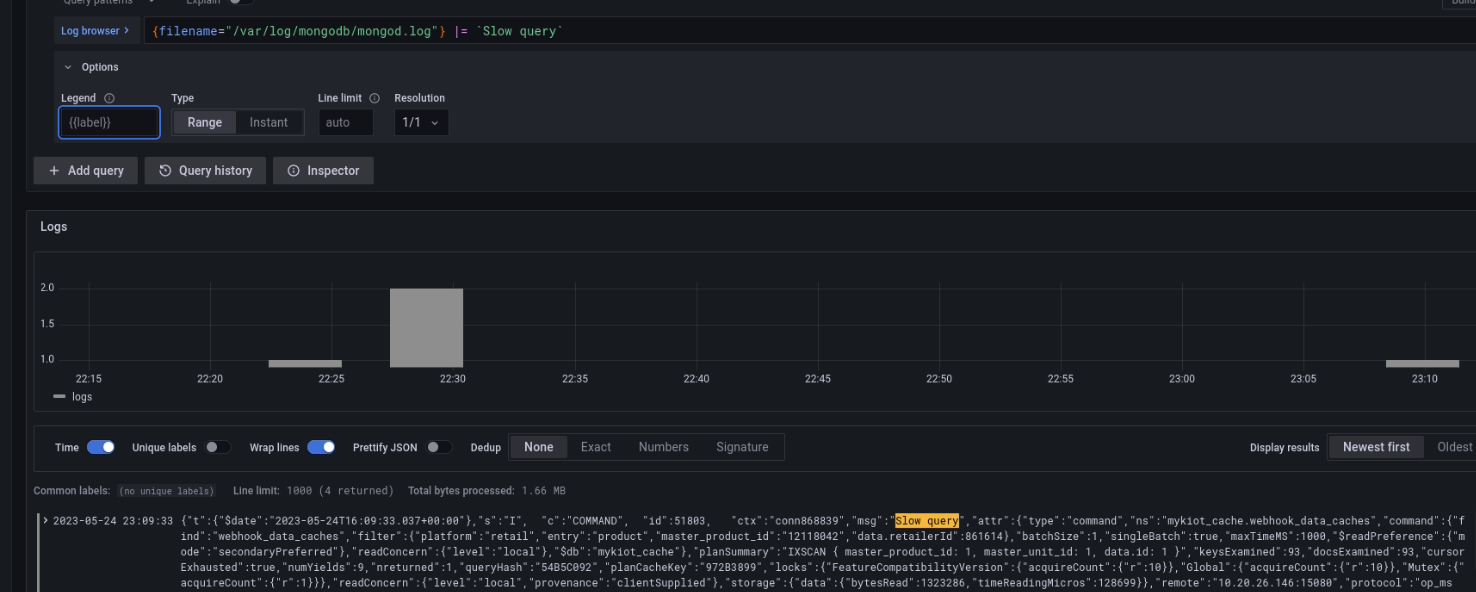
use database

db.system.profile.find({millis: {$gt: 1000}})

để lọc các truy vấn chậm hơn 1 giây.

bthuong để là $gt: 50, 100

**{filename="/var/log/mongodb/mongod.log"} |= `Slow query`**

****

count\_over\_time({filename="/var/log/mongodb/mongod.log"} |= "Slow query"[1m])

tổng số query trong 1 phút

Alert slow query

groups:

- name: example

  rules:

  - alert: SlowQueryDetected

    expr: count\_over\_time({filename="/var/log/mongodb/mongod.log"} |= "Slow query"[5m]) > 100

    for: 1m

    labels:

      severity: warning

    annotations:

      summary: "Slow query detected"

      description: "Slow query was detected in MongoDB log"

## đẩy log error lên slack