Prometheus và ứng dụng trong giám sát môi trường cloud

# **Chương 1: Đặt vấn đề**

Prometheus là bộ công cụ giám sát hệ thống mã nguồn mở. Bắt đầu thành lập từ 2012 bởi Sound Cloud, hiện đã được chuyển giao cho Cloud Native Computing Foundation (CNCF) - trở thành một dự án mã nguồn mở độc lập, với tư cách là dự án được ưu tiên phát triển lớn thứ 2, chỉ sau Kubernetes . Hiện tại được rất nhiều công ty, tổ chức áp dụng và có một cộng đồng hỗ trợ mạnh.

Hiện tại, môi trường cloud đang được đề xuất giám sát bởi TICK và Telemetry. TICK cho hệ thống monitor vật lý và các dịch vụ liên quan openstack chạy trên đó, Telemetry giám sát cho tài nguyên ảo hóa. Cách giám sát gặp phải một số hạn chế như: TICK có phần cảnh báo chưa tốt, không triển khai được HA, telemetry được cộng đồng đánh giám không cao do hiệu năng kém và cảnh báo khó dùng. Nghiên cứu này sẽ tìm hiểu sử dụng prometheus nhằm giám sát cho môi trường Cloud cho cả về tài nguyên vật lý và ảo hóa

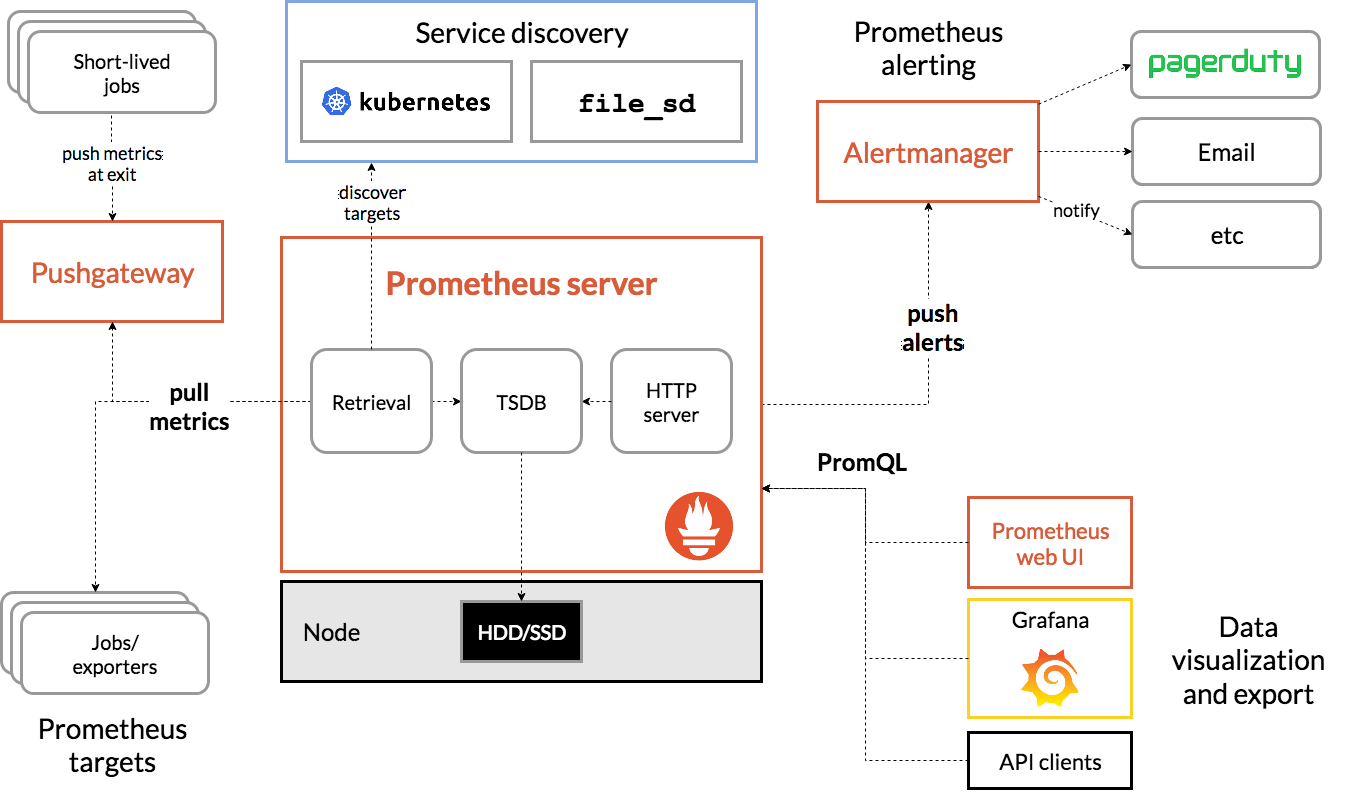
Báo cáo gồm các chương:

* Chương 2 Kiến trúc : các khái niệm và các thành phần của prometheus.
* Chương 3 Khởi động: giới thiệu cài đặt thử nghiệm.
* Chương 4 Thu thập dữ liệu: lấy dữ liệu giám sát cho prometheus, đặc biệt dùng cho nhiệm vụ giám sát cloud
* Chương 5 Lưu trữ dữ liệu: thiết lập mô hình lưu trữ các dữ liệu giám sát trên.
* Chương 6 Truy vấn và cảnh báo: giới thiệu cách thức truy vấn dữ liệu, cấu hình của thành phần cảnh báo
* Chương 7 Kịch bản sử dụng: các ca sử dụng với hệ thống , ví dụ như xem đồ thị giám sát, đặt cảnh báo .
* Chương 8 Kiến trúc triển khai: mô hình bố trí các thành phần, đảm bảo HA
* Chương 9 Hướng dẫn cài đặt: cài đặt hệ thống bằng ansible
* Chương 10 Kết luận: ưu, nhược điểm và các so sánh với hệ thống khác.

Hiện trạng công việc: làm việc đến chương 4, các chương sau đang dừng ở mức hiểu tổng quan chưa có thử nghiệm.

# **Chương 2: Kiến trúc**

Kiến trúc của prometheus được mô tả như hình dưới



Kiến trúc prometheus có thể chia thành 4 phần:

* Khối thu thập: lấy dữ liệu giám sát, các đối tượng của khối này chia làm 2 loại: pushgateway (push method) và job-exporter (pull method)
* Khối xử lý - prometheus server: quản lý việc xử lý lưu trữ dữ liệu trên và thực hiện các truy vấn
* Khối cảnh báo - alert manager: xử lý và gửi cảnh báo đi.
* Khối truy xuất: biểu diễn dữ liêu(grafana) và cung cấp api để phần mềm khác truy xuất.

Luồng hoạt động cơ bản:

B1. Dữ liệu giám sát được:

* Expoter lấy từ chính host hoặc vật lý hoặc gọi api. Nó mở một cổng (thường là 9\*\*\*) để cho các thành phần khác gọi http GET vào. Các exporter được chia sẻ opensource rất nhiều, thu thập đủ các thể loại dịch vụ.
* Các job nhỏ tại các host thu thập, đẩy cho pushgateway. Pushgateway mở cổng 9091 cho các thành phần khác GET lấy dữ liệu này.
  + Các job này thường là các chương trình rất nhỏ, tự viết từ python, java, go. Dùng các thư viện prometheus cung cấp để POST các dữ liệu vào cổng của pushgateway.
  + Các job này rất hiếm có thư viện sẵn trên mạng, thường là các batch job mà lấy dữ liệu sẽ khó khăn khi cào khi dùng exporter.
  + Nó cũng được sử dụng đến nếu tại mô hình mạng: khối xử lý bị tường lửa chặn khi truy cập **đến** khối thu thập.

B2. Tiếp theo, prometheus server sẽ cào các đầu vào này.

Mỗi đối tượng cần cào là 1 *target*, các target chung một mục đích (vd giám sát các dịch vụ database) gọi là 1 *job.* Mỗi *target* trong một nhóm được định danh bằng nhãn *instance* (thường là hostIP:port vd localhost:9100).

Mô hình dữ liệu: cấu trúc một bản ghi được lưu có:

<metric name>{<label name>=<label value>, ...} : value

Các chỉ mục được đánh theo label và timestamp.

Ví dụ:

*api\_http\_requests\_total{job=“web”, instance=“10.240.117.27:9009”, method="POST", handler="/messages"} : 20*

Đồng thời, khối này cũng quản lý luôn việc xử lý cho các alert rule : truy vấn, trả ra cảnh báo.

Backend lưu trữ cho khối trên có thể là local storage- lưu vào file local hoặc remote storage -sử dụng database của bên thứ 3 để lưu trữ. chi tiết sẽ đề cập ở chương 5.

B3. Các cảnh báo được khối xử lý chuyển cho khối cảnh báo -alert manager. Khối này làm nhiệm vụ chủ yếu là tổ chức và handle cho cảnh báo(send mail, telegram…). Nhiệm vụ tổ chức được thể hiện ví dụ như:

- Khi mạng bị phân mảnh, khiến cho hàng chục ứng dụng không tiếp cận được với cơ sở dữ liệu, người dùng không muốn nhận cả chục cảnh báo đó, mà chỉ cần một cảnh báo có không truy cập được csdl, và các máy ảo bị hảnh hưởng là a,b,c….

- Khi có một cảnh báo về có 1 cụm máy không truy cập được, ta có thể cấu hình liên quan đến tắt tất cả các cảnh báo khác liên quan tới cụm này

- Tắt cảnh báo liên quan đến một số host trong khoảng thời gian đang sửa chữa

B4. Grafana truy vấn các dữ liệu giám sát trên và vẽ đồ thị cho người sử dụng.

# **Chương 3. Khởi động**

**Cài đặt exporter**

Exporter ở đây là node exporter: thu thập thông tin tài nguyên của máy chủ vật lý.

Chạy lệnh:

|  |
| --- |
| docker run -d --name node-exporter --restart unless-stopped \  --net="host" \  --pid="host" \  -v "/:/host:ro,rslave" \  quay.io/prometheus/node-exporter \  --path.rootfs /host |

**Cài đặt Prometheus server**

Chuẩn bị file cấu hình

|  |
| --- |
| mkdir /prometheus-data  mkdir /prometheus-conf  cat <<EOF > /prometheus-conf/prometheus.yml  global:  scrape\_interval: 15s  # Attach these labels to any time series or alerts when communicating with  # external systems (federation, remote storage, Alertmanager).  external\_labels:  monitor: 'lab-monitor'  scrape\_configs:  # The job name is added as a label 'job=<job\_name> ' to any timeseries scraped from this config.  - job\_name: 'node-exporter'  # Override the global default and scrape targets from this job every 5 seconds.  scrape\_interval: 5s  static\_configs:  - targets: [**'controller:9100'**]  EOF |

(Thay controller:9100 bằng <hostname>:9100)

Chạy lệnh:

|  |
| --- |
| mkdir /prometheus-data  chown nobody /prometheus-data  docker run -p 9090:9090 -d --name prometheus --restart unless-stopped \  -v /prometheus-conf:/prometheus/conf \  -v /prometheus-data:/prometheus/data \  prom/prometheus \  --config.file=/prometheus/conf/prometheus.yml \  --storage.tsdb.path /prometheus/data \  --web.enable-admin-api |

**Cài đặt grafana**

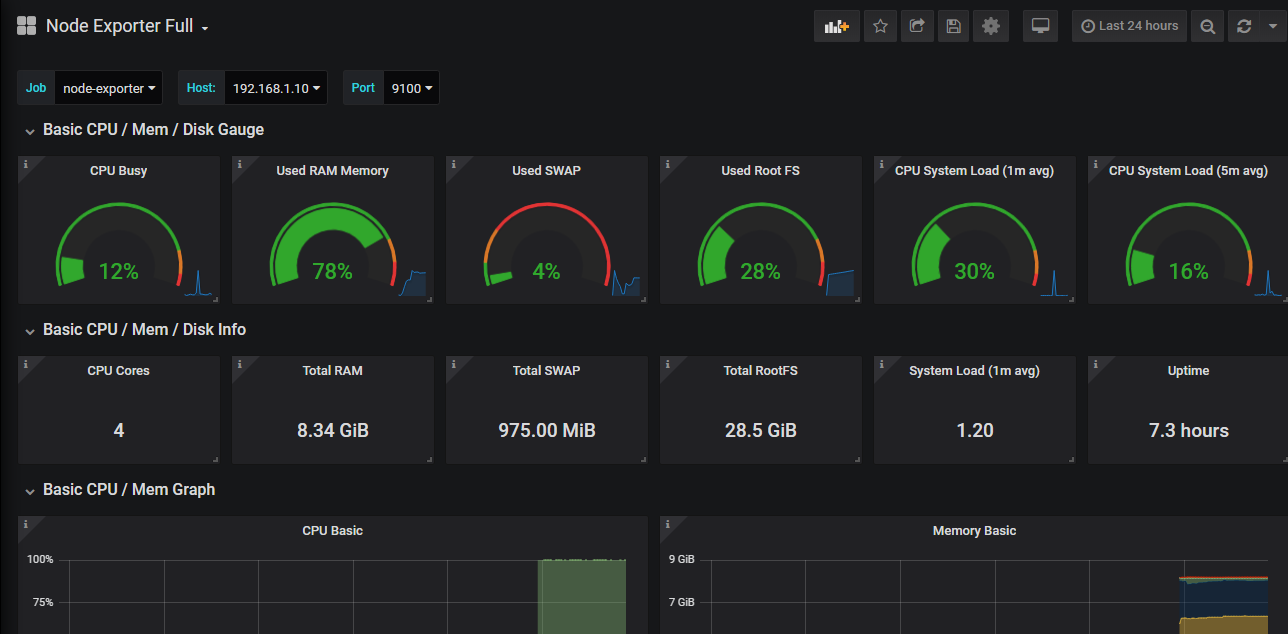
Chạy lệnh:

|  |
| --- |
| mkdir -p /grafana-data/grafana-storage  touch /grafana-data/grafana.ini  chown -R 472 /grafana-data  docker run -d -p 3000:3000 \  --name grafana \  -v /grafana-data/grafana.ini:/etc/grafana/grafana.ini \  -v /grafana-data/grafana-storage:/var/lib/grafana \  grafana/grafana |

Import dashboard template tại:

<https://github.com/cloud-guru/prometheus/blob/master/node-exporter.json>

Kết quả: giao diện quản trị tại grafana



# Chương 4: Thu thập dữ liệu.

## **4.1. Các yêu cầu giám sát**

Hệ thống cần giám sát 3 thành phần:

* Tài nguyên vật lý.
* Các dịch vụ của openstack cùng các dịch vụ liên quan (control plane).
* Tài nguyên ảo hóa: các instance, volume, network ảo (data plane).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thành phần** | **Đối tượng** | **Thông số** | **Cảnh báo** | **Hiện trạng thu thập**  **dữ liệu** |
| Tài nguyên vật lý | host | Thống kê số lượng node và trạng thái (status/healthy) của các node vật lý  Monitor tài nguyên: CPU, Memory, Disk, Network bandwith/throughput | cảnh báo node mất kết nối, cảnh báo tài nguyên vượt ngưỡng (CPU, RAM, Disk, Network throughput) | Đủ |
| Dịch vụ liên quan đến openstack | mysql | Thống kê số lượng Questions, số lượng Read queries (Select), số lượng Write queries | Cảnh báo khi có sự thay đổi bất thường làm giảm query throughput (Số lượng Questions) | Đủ |
| Thống kê số lượng query runtime mỗi schema, số lượng query errors, số lượng query chậm | Cảnh báo khi số lượng query runtimes, số lượng query errors tăng lên bất thường. Cảnh báo khi có quá nhiều số lượng query chậm | Đủ. |
| Thống kê số lượng connection đang mở, số lượng connection đang hoạt động, số lượng connection bị từ chối so server bị lỗi, số lượng connection bị từ chối do vượt quá tham số max\_connections, số lượng connection bị hủy bỏ (chưa được chấp nhận kết nối) | Cảnh báo khi số lượng connection vượt quá max\_connections, số lượng connection bị hủy bỏ | Đủ |
| Rabbitmq | Monitor số lượng messages được published in/out một exchange, số lượng message unroutable | Cảnh báo số lượng message unroutable sau một khoảng thời gian nhất định | Đủ |
| Dung lượng ổ cứng chiếm bởi RabbitMQ trên mỗi OpenStack Node  Dung lượng RAM sử dụng bởi RabbitMQ trên mỗi OpenStack Node | Cảnh báo khi dung lượng ổ cứng trống thấp hơn threshold cấu hình cho RabbitMQ  Cảnh báo khi memory usage của RabbitMQ quá cao | Đủ |
| memcache | Số bytes đã được dùng để cache  Số bytes tối đa được cho phép trong cache  Current open connections  Số lượng GET/SET requests nhận bởi server mỗi giây  Hits (số lượng GET requests thành công mỗi giây)  Misses (số lượng GET requests bị từ chối mỗi giây)  Read (Số lượng bytes/s gửi từ network và đọc bởi server)  Written (số lượng bytes/s gửi từ server ra bên ngoài) |  | có exporter - chưa thử nghiệm |
| Docker | Số lượng docker container trên mỗi OpenStack Node  Số lượng docker container UP/DOWN trên mỗi OpenStack Node  Với mỗi docker container, monitor các metrics: CPU usage (%), Memory usage/limit, Network I/O throughput, Disk I/O throughput | Cảnh báo khi container thay đổi trạng thái (started, stopped, restarted) | có exporter - chưa thử nghiệm. |
| HAproxy | Frontend Metrics (Các metrics về client connections và requests)  Backend Metrics (Các metrics thể hiện khả năng sẵn sàng và tình trạng của backend server) |  | có exporter - chưa thử nghiệm. |
| Openstack service  (tổng quan) | health status của mỗi dịch vụ (*Keystone, Glance, Nova, Neutron, Cinder, Heat, Masakari*) trên toàn cụm Openstack (OK, WARNING, CRITICAL) | Cảnh báo nếu health status của bất kì OpenStack service nào không OK | Đủ |
| Tổng quan compute resources   * Tổng số vCPUs * Tổng dung lượng RAM * Tổng dung lượng ổ cứng |  | Chưa có |
|  | Keystone | Tổng số roles  Số tenants theo trạng thái (enabled/disabled)  Số users theo trạng thái (enabled/disabled) |  | chưa có |
| Glance | Số lượng images và visibility  Tổng dung lượng chiếm bởi các images phân loại và visibility  Số lượng snapshot images phân loại s và visibility  Tổng dung lượng tính theo bytes của các snapshots phân loại và visibility |  | có, chưa đủ |
| Nova | Tổng số các instances  Nova service state  Tổng số nova services |  | đủ |
| Neutron | Neutron agents state  Tổng số neutron agents |  | đủ |
| Tổng số địa chỉ floating IP  Số lượng neutron network  Số lượng available IPs của mỗi network  Số lượng virtual ports theo owner và state |  | đủ |
| Cinder | Cinder service state.  Tổng số Cinder services . |  | exporter thư viện trên mạng lỗi phần cinder |
|  | Số lượng snapshots  Tổng kích thước của các snapshots  Số lượng volumes  Tổng kích thứơc các volume |  | exporter thư viện trên mạng lỗi phần cinder |
| Tài nguyên ảo hóa | Libvirt | Tổng dung lượng ổ đĩa sẵn sàng tính theo GB  Tổng dung lượng memory sẵn sàng  Tổng số vCPUs sẵn sàng  Tổng số instances đang chạy  Tổng số tasks đang thực thi  Tổng dung lượng ổ cứng sử dụng bởi các instances  Tổng dung lượng memory sử dụng bởi các instances  Tổng số vCPUs được sử dụng bởi các instances |  | có, chưa đủ |
| Instance | CPU allocate  memory usage  memory allocate  disk I/O  network I/O |  | Thiếu thu thập % cpu, % memory |

(Các exporter trong monitor service openstack dự kiến em sẽ viết lại dùng python, exporter tham khảo được trên mạng phần này thiếu nhiều thông tin, cùng với đó là viết bằng go, không tự làm chủ được. Ví dụ sau này muốn monitor thêm vitrage, octavia,… cũng cần tự custom lại exporter )

## **4.2. Cài đặt**

**Monitor mysql:**

Tạo quyền trong mysql:

|  |
| --- |
| GRANT PROCESS, REPLICATION CLIENT, SELECT ON \*.\* TO 'prometheus'@'localhost' IDENTIFIED BY 'Welcome123' WITH MAX\_USER\_CONNECTIONS 3;  GRANT PROCESS, REPLICATION CLIENT, SELECT ON \*.\* TO 'prometheus'@'%' IDENTIFIED BY 'Welcome123' WITH MAX\_USER\_CONNECTIONS 3; |

Chạy exporter container:

|  |
| --- |
| docker run -d --name mysqld-exporter --restart unless-stopped \  -p 9104:9104 \  --network 'host' \  -e DATA\_SOURCE\_NAME="prometheus:Welcome123@(controller:3306)/" \  prom/mysqld-exporter |

Đây là exporter từ nguồn:

https://github.com/prometheus/mysqld\_exporter

Thêm vào prometheus.yml

|  |
| --- |
| scrape\_configs:  - job\_name: 'mysql'  scrape\_interval: 10s  static\_configs:  - targets: ['controller:9104'] |

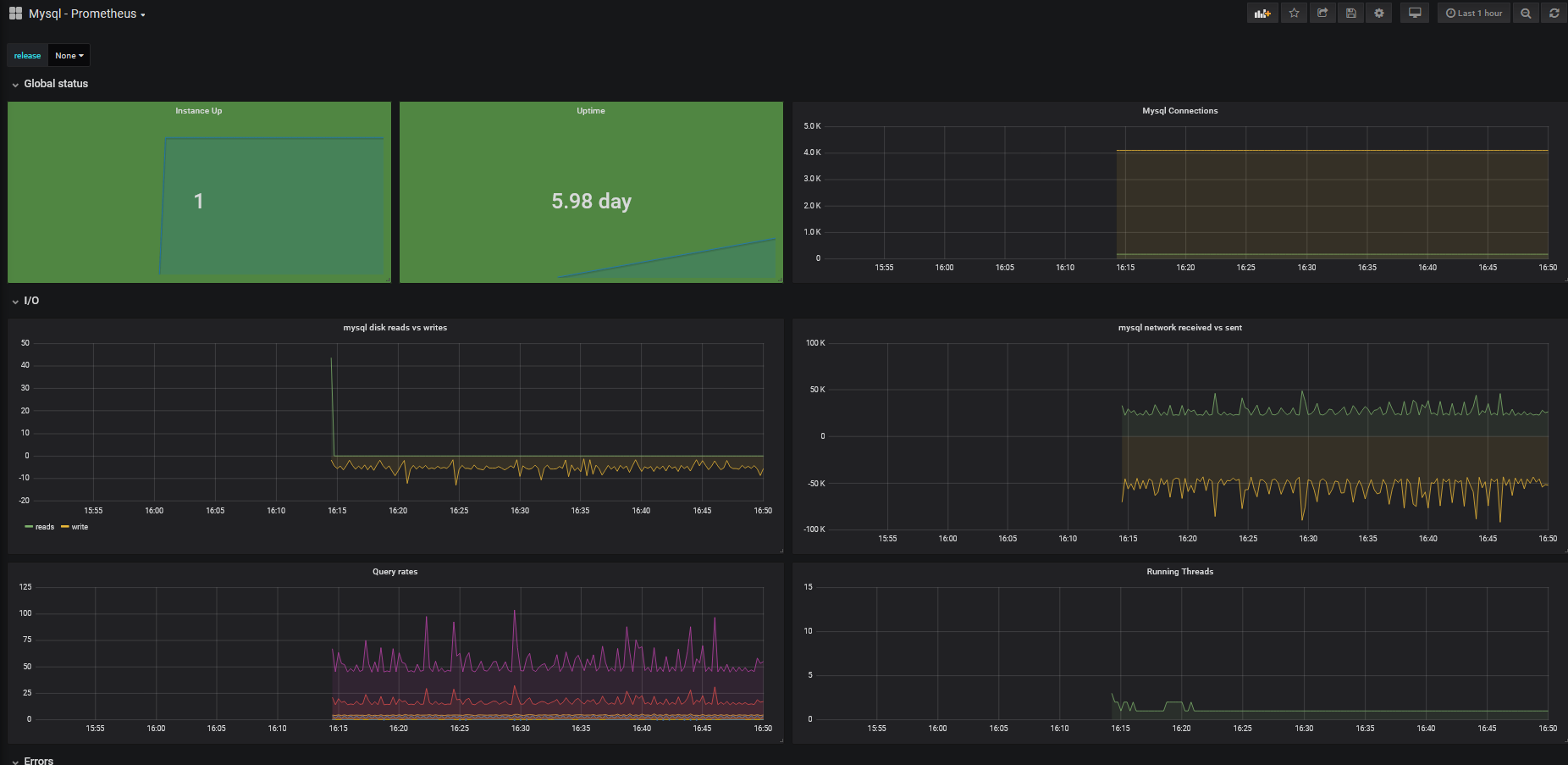
Restart prometheus

|  |
| --- |
| docker restart prometheus |

Tại grafana, import dashboard theo template tại:

<https://github.com/cloud-guru/prometheus/blob/master/grafana/myqld-exporter.json>

Kết quả:



**Rabbitmq exporter:**

Tạo quyền trong rabbitmq:

|  |
| --- |
| rabbitmqctl add\_user prometheus prometheus  rabbitmqctl set\_user\_tags prometheus monitoring  rabbitmqctl set\_permissions prometheus ".\*" "" ".\*"  rabbitmq-plugins enable rabbitmq\_management |

Chạy exporter container:

|  |
| --- |
| docker run -d --name rabbitmq-exporter --restart unless-stopped \  -p 9419:9419 \  --network 'host' --restart unless-stopped \  -e RABBIT\_URL="http://10.240.207.27:15672" \  -e RABBIT\_USER="prometheus" \  -e RABBIT\_PASSWORD="prometheus" \  kbudde/rabbitmq-exporter |

Đây là exporter từ nguồn:

<https://github.com/kbudde/rabbitmq_exporter>

Thêm vào prometheus.yml

|  |
| --- |
| scrape\_configs:  - job\_name: 'rabbitmq'  scrape\_interval: 10s  static\_configs:  - targets: ['controller:9419'] |

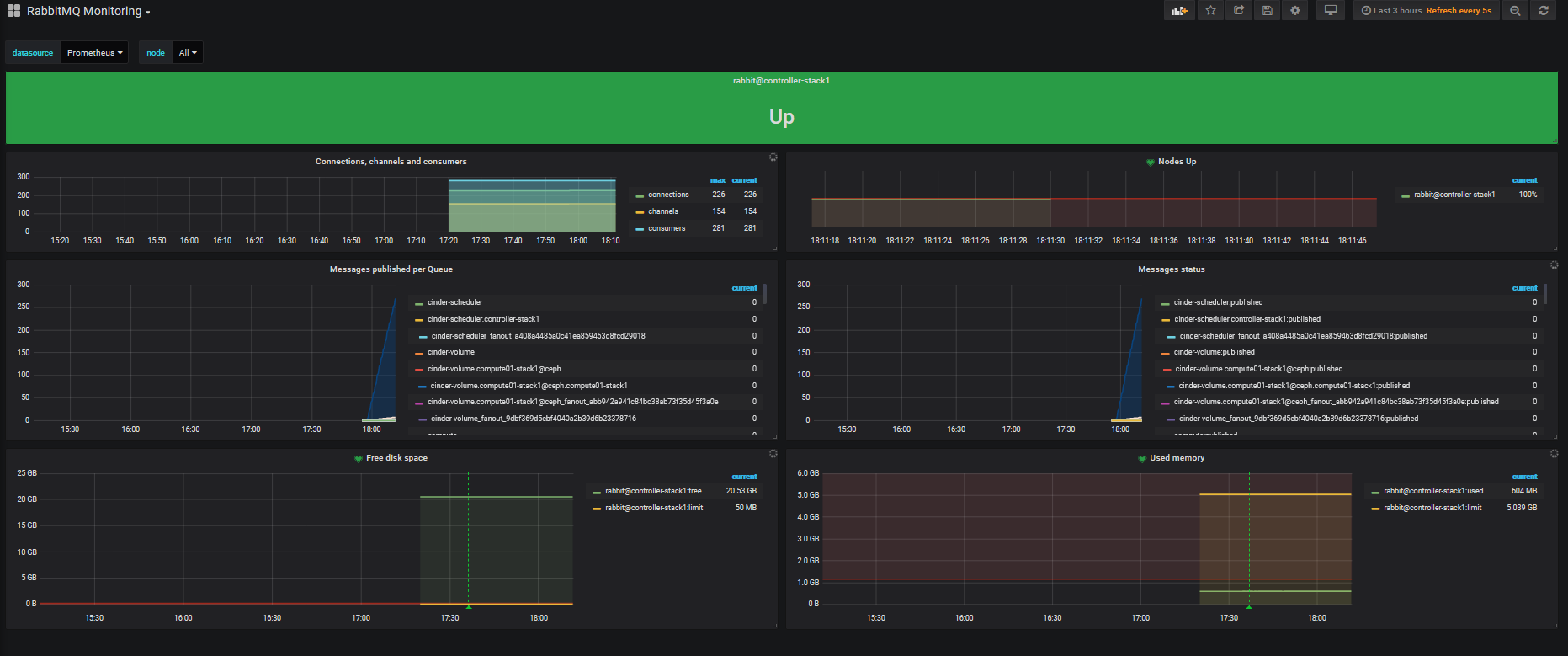
Restart prometheus

|  |
| --- |
| docker restart prometheus |

Tại grafana, import dashboard theo template tại:

<https://github.com/cloud-guru/prometheus/blob/master/grafana/rabbitmq-exporter.json>

Kết quả



# **Chương 5: Lưu trữ dữ liệu.**

## **5.1. Local storage**

## **5.2. Remote adapter**

# **Chương 6: Truy vấn và cảnh báo**

# **Chương 7: Các kịch bản sử dụng.**

# **Chương 8: Kiến trúc triển khai.**

Cluster prometheus

<https://prometheus.io/docs/prometheus/latest/storage/>

|  |
| --- |
| AppOptics: write  Chronix: write  Cortex: read and write  CrateDB: read and write  Elasticsearch: write  Gnocchi: write  Graphite: write  InfluxDB: read and write  IRONdb: read and write  Kafka: write  M3DB: read and write  OpenTSDB: write  PostgreSQL/TimescaleDB: read and write  SignalFx: write  Splunk: write  TiKV: read and write  VictoriaMetrics: write  Wavefront: write |

Phổ biến khuyên dùng: Cortex, InfluxDB , PostgreSQL/TimescaleDB do chúng có mô hình dữ liệu và cách query tương đồng.

# **Chương 9: Hướng dẫn cài đặt**

# **Chương 10: Kết luận**