

**BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN**



BÁO CÁO CUỐI KỲ

MÔN HỌC: NHẬP MÔN DỮ LIỆU LỚN

**Đề tài: Xây dựng và giám sát hệ thống microservices trên nền tảng
Docker Swarm**

Giảng viên hướng dẫn: ThS. Huỳnh Nam

Lớp học phần: DHHTTT17B – 420300232902

Sinh viên thực hiện: Dương Thái Bảo – 21037621

Vũ Hải Nam – 21113621

Nguyễn Văn Nguyên – 22000835

Nguyễn Thị Tuyết Lan – 22651091

Nguyễn Thiên Phú – 20073871

Nguyễn Đức Thịnh – 21074131

Đỗ Minh Thư – 22658841

Nguyễn Tuấn Trung – 19477111

TP. Hồ Chí Minh, ngày 15 tháng 04 năm 2025

MỤC LỤC

MỤC LỤC	2
LỜI MỞ ĐẦU	7
Phần 1: Xây dựng hệ thống mạng ảo giả lập trên Virtual Box	8
1. Các loại cấu hình mạng:	8
1.1. NAT	8
1.2. Internal Network Mode:	8
1.3. Bridged Network Mode:	9
1.4. Host-only Networking	10
2. Cấu hình mạng cho hệ thống:	10
Phần 2: Xây dựng cụm máy tính dựa trên Docker Swarm theo yêu cầu	16
1. Docker Swarm là gì?	16
2. Vai trò của Docker Swarm trong hệ thống	17
3. Môi trường triển khai	17
4. Cấu hình máy ảo	17
5. Chuẩn bị môi trường	18
5.1. Cập nhật hệ thống	18
5.2. Cài đặt SSH	18
5.3. Kiểm tra kết nối từ máy local	18
6. Cài đặt và khởi tạo cụm Docker Swarm	19
6.1. Cài đặt Docker	19
a. Cài đặt các gói phụ thuộc:	19
b. Thêm GPG key của Docker:	19

c.	Thêm repository của Docker:	19
d.	Cập nhật danh sách gói và cài đặt Docker:.....	20
e.	Kiểm tra phiên bản Docker:.....	20
f.	Thêm user osboxes vào nhóm docker (để chạy lệnh docker mà không cần sudo): 20	
6.2.	Khởi tạo cụm Docker Swarm.....	20
a.	Khởi tạo cụm.....	20
b.	Kiểm tra trạng thái cụm	20
c.	Thêm worker node	21
7.	Kết luận	22
Phần 3:	Triển khai các gói ứng, tiến hành HA, scaling và load balancing cho hệ thống 23	
1.	Thay đổi Hostname của các máy ảo:.....	23
2.	Promote máy worker thành máy manager:.....	23
3.	Triển khai các gói ứng dụng.....	23
3.1.	Tiến hành build image cho các dịch vụ: hasher, worker, rng và webui:	23
3.2.	Publith các image lên Docker Hub:	24
a.	Đăng nhập Docker Hub:	24
b.	Push images của các dịch vụ lên Docker Hub:.....	24
3.3.	Tạo overlay network trong hệ thống swarm:	25
3.4.	Tiến hành Deploy hệ thống lên swarm theo thứ tự redis, rng, hasher, worker, webui tương ứng với port và gắn vào coinswarmnet:	25
3.5.	Tiến hành HA cho hệ thống swarm:	26
a.	High Availability là gì?	26

b.	HA có tác dụng gì?	26
c.	Triển khai HA cho hệ thống với hệ số n=2:	26
3.6.	Tiến hành Auto scaling và Auto load balancing các dịch vụ:	26
a.	Auto scaling là gì?	26
b.	Auto scaling có tác dụng gì?.....	26
c.	Auto load balancing là gì?	27
d.	Auto load balancing có tác dụng gì?.....	27
e.	Triển khai Auto scaling trong hệ thống:	27
Phần 4:	Giám sát và quản trị hệ thống Docker Swarm	29
1.	Giám sát hệ thống Docker Swarm là gì?	29
2.	Quản trị hệ thống Docker Swarm là gì?	29
3.	Tại sao cần Giám sát và quản trị hệ thống Docker Swarm	30
4.	Triển khai Giám sát và quản trị hệ thống Docker Swarm	30
4.1.	Kiểm tra danh sách các node trong Swarm và trạng thái của chúng:	30
4.2.	Xem danh sách service hiện có.....	30
4.3.	Kiểm tra chi tiết các container đang chạy để phục vụ cho dịch vụ	31
4.4.	Truy xuất và hiển thị các log được tạo ra bởi các container (tasks) đang chạy của dịch vụ	31
4.5.	Hiển thị thông kê sử dụng tài nguyên trực tiếp của các container đang chạy	31
4.6.	Kiểm tra và đo lường CPU / memory và I/O Usage của một node	32
4.7.	Kiểm tra độ trễ phản hồi của các dịch vụ	34
Phần 5:	ELK Stack	35
1.	Giới thiệu	35
2.	Cài đặt.....	35

Phần 6: Prometheus-Grafana-InfluxDB.....	40
1. Giới thiệu về prometheus, grafana và influxdb	40
1.1. Prometheus.....	40
1.2. InfluxDB	40
1.3. Grafana.....	40
1.4. Telegraf.....	40
2. Cài đặt.....	41
2.1. Đảm bảo Docker Swarm hoạt động.....	41
2.2. Cấu hình daemon.json trên mỗi node	41
2.3. Tạo network	41
2.4. Triển khai node-exporter và cadvisor	42
2.5. Cấu hình docker-compose.yml cho prometheus, influxdb, telegraf và grafana	44
2.6. Cấu hình prometheus.yml.....	48
2.7. Cấu hình telegraf.conf	49
a. Triển khai stack monitoring.....	50
b. Kiểm tra service	51
c. Kiểm tra log của các service để đảm bảo các service chạy ổn định	51
3. Giới thiệu các metric được thu thập trên toàn bộ các node	52
4. Hiển thị các thông tin metric trên cụm như mục trên bằng lệnh CLI	53
5. Trình diễn cơ chế tạo pipeline thu thập và lưu trữ các metric với Prometheus với CLI	53
6. Đồng bộ hóa dữ liệu từ Prometheus với InfluxDB	54
7. Truy vấn dữ liệu metrics được thu thập được lưu trữ trong InfluxDB chạy trên swarm.....	54

8. Tạo Dashboard sử dụng Prometheus/Grafana để phân tích , giám sát hiệu suất swarm
55

Phần 7: Reverse proxy sử dụng nginx.....	67
1. Giới thiệu về Reverse Proxy.....	67
2. Giới thiệu về Nginx	67
3. Ý tưởng thiết lập Reverse Proxy	68
4. Triển khai Reverse Proxy với Nginx.....	68
KẾT LUẬN	73

LỜI MỞ ĐẦU

Nhóm 01 xin gửi lời cảm ơn chân thành đến thầy Huỳnh Nam - người đã tận tình hướng dẫn và hỗ trợ nhóm trong suốt quá trình thực hiện đề án này.

Với sự chỉ dạy quý báu của thầy, chúng em đã có cơ hội tiếp cận sâu hơn với mô hình triển khai và giám sát hệ thống microservices trên nền tảng Docker Swarm – một kiến thức vô cùng thực tiễn và có giá trị lớn trong ngành công nghệ hiện nay.

Trong quá trình thực hiện, nhóm đã cố gắng hết mình để hoàn thành đề án một cách nghiêm túc.

Tuy nhiên, do thời gian có hạn và kinh nghiệm còn non kém, chắc chắn sẽ không tránh khỏi những thiếu sót. Nhóm rất mong nhận được sự thông cảm và góp ý từ thầy để có thể cải thiện và học hỏi thêm trong những lần sau.

Một lần nữa, nhóm xin chân thành cảm ơn thầy và kính chúc thầy luôn mạnh khỏe, công tác tốt và tiếp tục truyền cảm hứng cho nhiều thế hệ sinh viên.

Trân trọng,

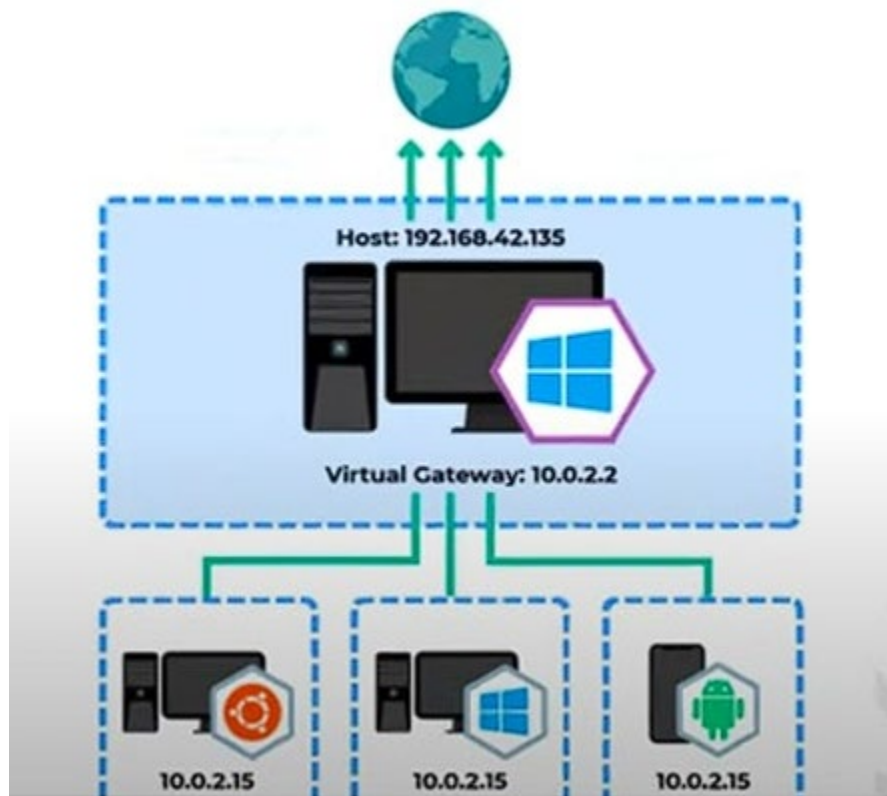
Nhóm 01

PHẦN 1: XÂY DỰNG HỆ THỐNG MẠNG ẢO GIẢ LẬP TRÊN VIRTUAL BOX

1. Các loại cấu hình mạng:

1.1. NAT

NAT là chế độ mạng mặc định. Chế độ này cho phép các máy ảo (VM) truy cập internet (nếu máy chủ của chúng được kết nối), nhưng không thể giao tiếp với nhau.

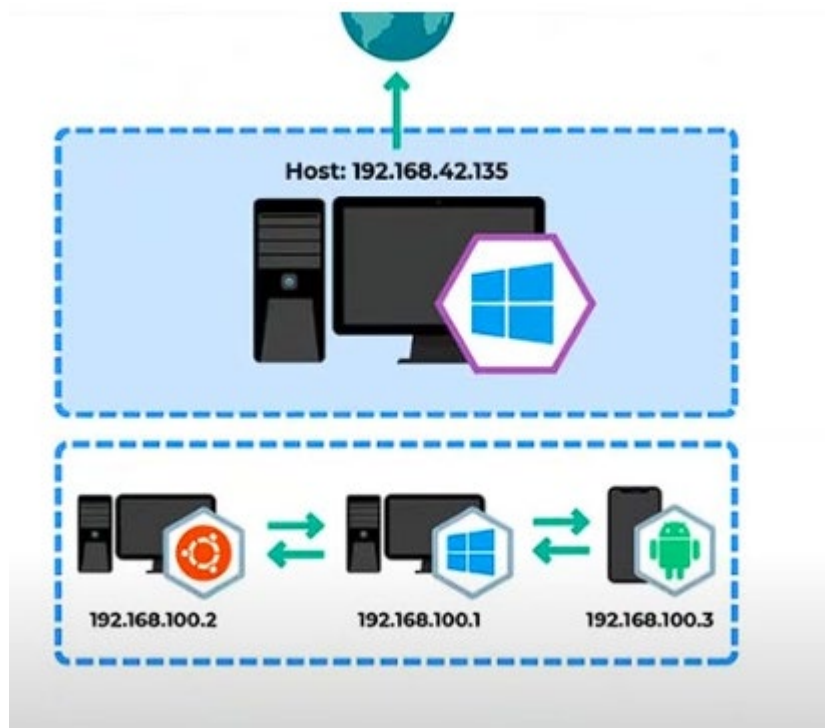


- 3 máy ảo sẽ có cùng IP, có thể kết nối ra ngoài internet
- Thông qua gateway cố định 10.0.2.2
- Từ máy windows không truy cập ngược được do IP giống nhau.
- Các máy đều giống IP nên không thể giao tiếp với nhau.

1.2. Internal Network Mode:

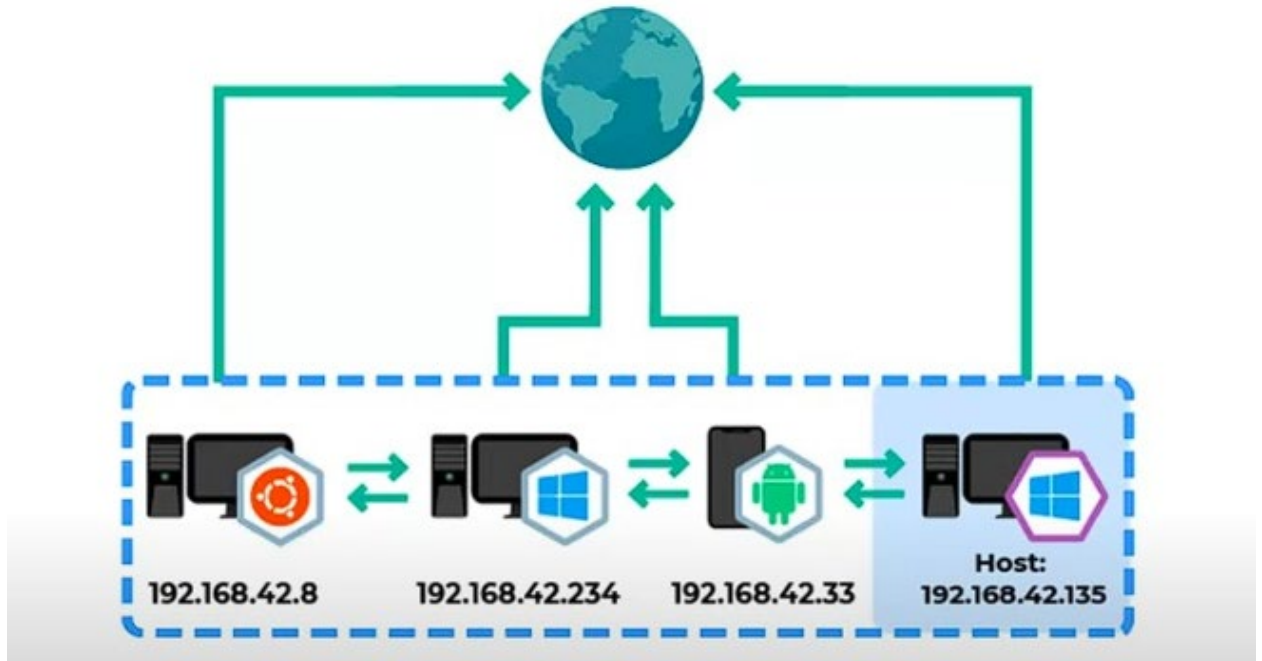
- Cách đơn giản nhất để thiết lập giao tiếp giữa các máy ảo (VM) là đặt tất cả chúng vào một mạng nội bộ bằng cách sử dụng chế độ mạng nội bộ.

- Chế độ Nội bộ không có máy chủ DHCP theo mặc định. Vì vậy, một adapter của máy ảo được gắn vào mạng Nội bộ sẽ không nhận được địa chỉ IP, trừ khi thiết lập adapter sang IP tĩnh trong hệ điều hành của máy ảo hoặc cung cấp một máy chủ DHCP trên mạng Nội bộ, thông qua một máy ảo khác hoặc bằng cách kích hoạt thủ công một máy chủ DHCP thông qua VBoxmanage. Các máy ảo chỉ có thể nói chuyện với nhau, có thể có một IP tĩnh, giao tiếp qua ngoài Internet, hay từ ngoài vào máy ảo không được



1.3. Bridged Network Mode:

- Mỗi một máy ảo sẽ được cấp 1 IP tương đương với host, trên hệ thống mạng chung của máy host đang dùng. DHCP sẽ cấp cho các máy ảo, giao tiếp được giữa các máy, ra Internet.
- Tuy nhiên do dựa vào DHCP => IP sẽ được cấp động, không cố định



1.4. Host-only Networking

- Máy host sẽ join vào hệ thống máy ảo => mạng riêng, gần giống Internal Network, nhưng máy Windows gia nhập vào luôn

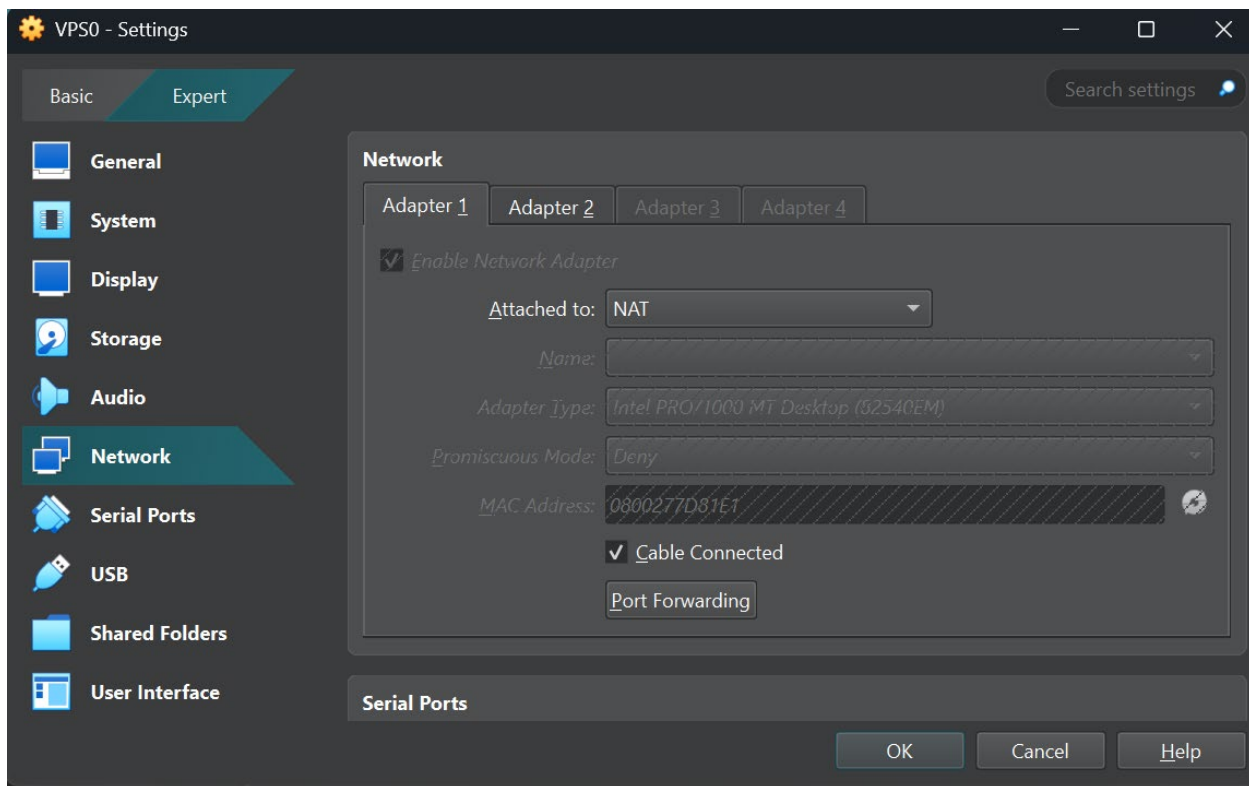
2. Cấu hình mạng cho hệ thống:

Chuẩn bị:

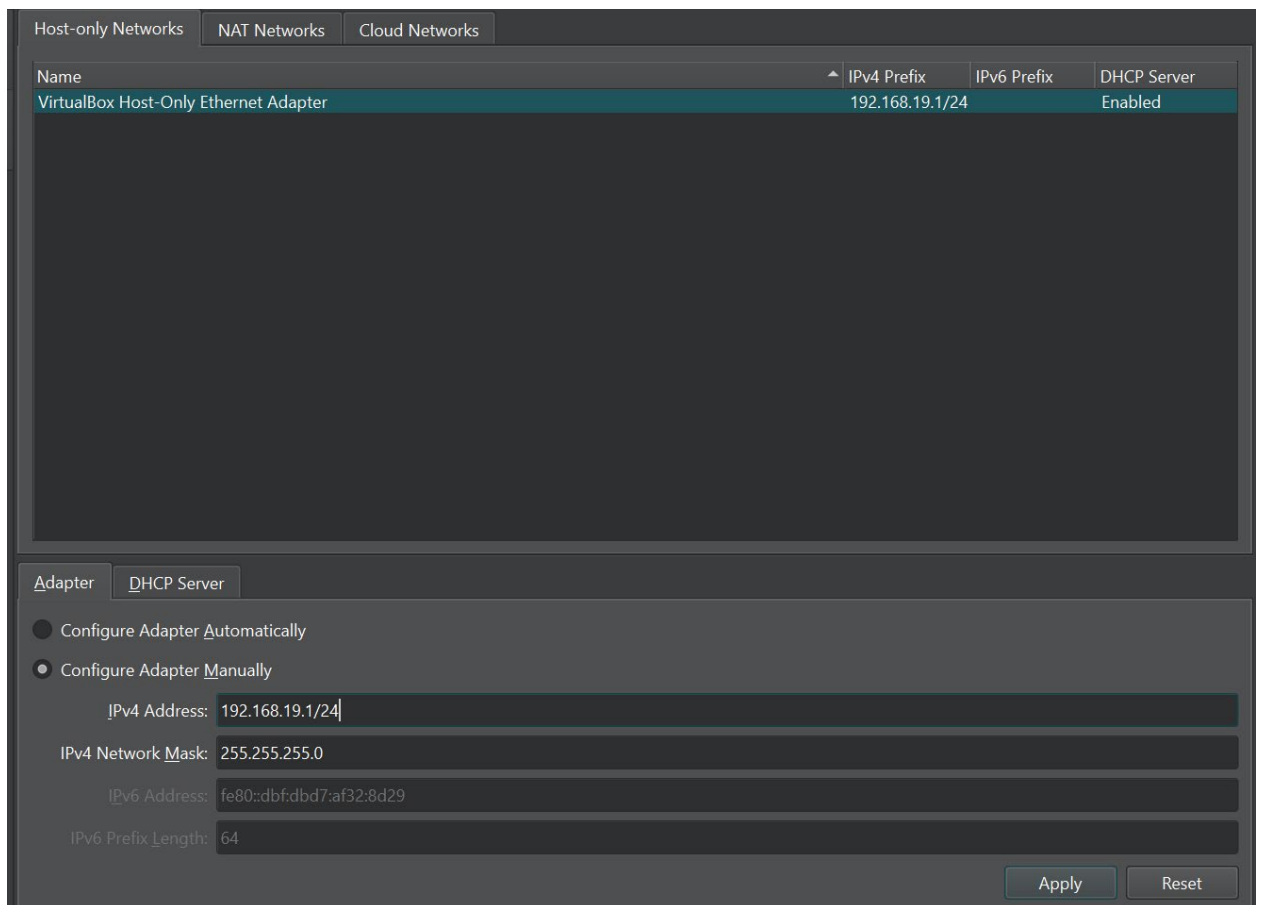
- Cài ssh (server lẫn client), docker engine
- Cài thêm gói network manager tools:
 - `sudo apt update`
 - `sudo apt install ifupdown net-tools openssh-client openssh-server`
- Setup 2 adapter: 1 - nat, 2 - host only network

Thực hiện

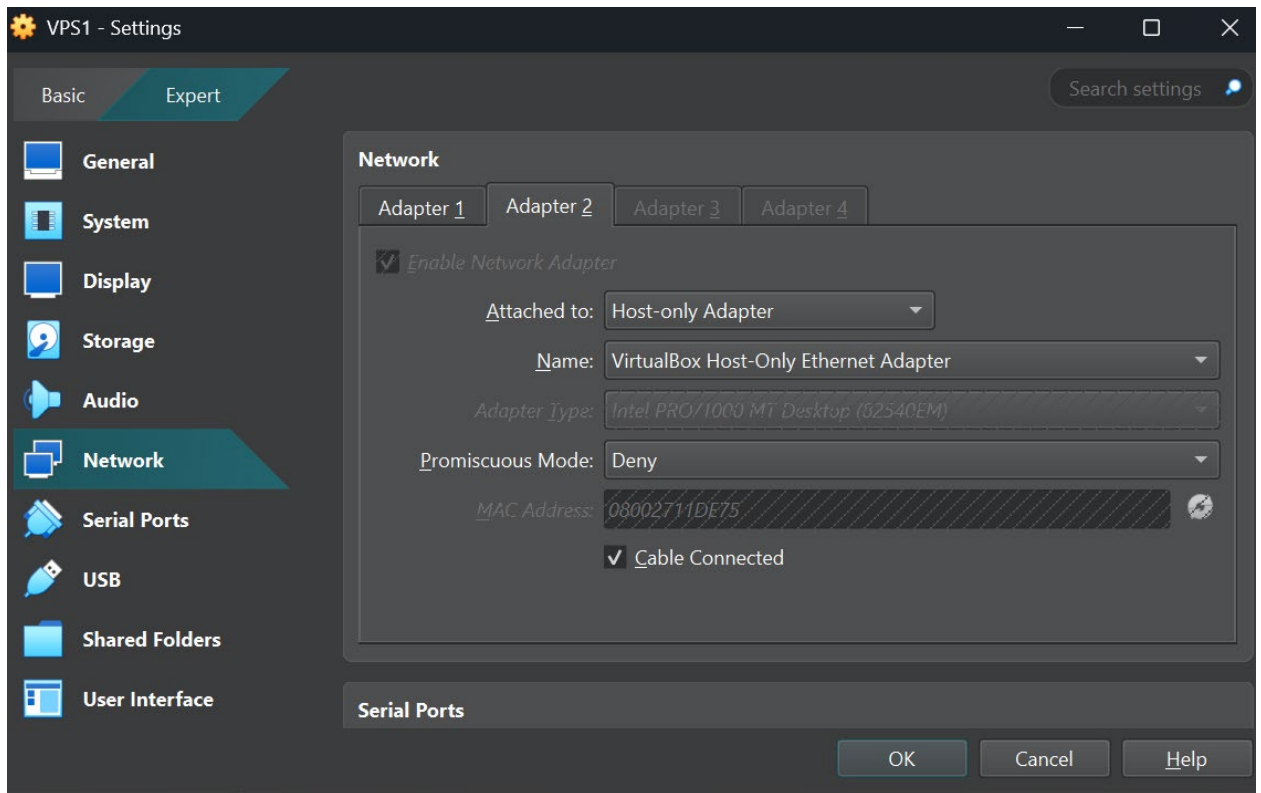
- Cấu hình mạng NAT cho các máy tính. Mỗi máy cấu hình network cho adapter 1 như sau



- Tạo host-only network tự động: Phần IP, thiết lập là 192.168.19.1



- Với từng máy, chọn cấu hình host-only network cho từng máy phù hợp trong tab adapter 2: -> Virtual host-only Ethernet Adapter



Cấu hình mạng Nat và Host-Only Network cho máy master01

- Khởi động máy master01
- Trở về thư mục gốc

pwd

cd /

ls

```
osboxes@osboxes: /  
(8 additional names omitted)  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added '192.168.1.242' (ED25519) to the list of known hosts.  
osboxes@192.168.1.242's password:  
Welcome to Ubuntu 24.10 (GNU/Linux 6.11.0-9-generic x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:       https://ubuntu.com/pro  
  
System information as of Mon Oct  7 17:20:01 UTC 2024  
  
System load:  1.08      Processes:            30  
Usage of /home: unknown  Users logged in:      0  
Memory usage: 7%       IPv4 address for eth0: 10.10.10.2  
Swap usage:   0%  
  
164 updates can be applied immediately.  
88 of these updates are standard security updates.  
To see these additional updates run: apt list --upgradable  
  
osboxes@osboxes:~$ pwd  
/home/osboxes  
osboxes@osboxes:~$ cd /  
osboxes@osboxes:/$ ls  
bin  cdrom  etc  lib  lost+found  mnt  proc  run  snap  swap.img  tmp  var  
boot  dev  home  lib64  media  opt  root  sbin  srv  sys  usr  
osboxes@osboxes:/$
```

- Thiết lập IP tĩnh cho mạng

sudo nano /etc/network/interfaces

```
GNU nano 8.1 /etc/network/interfaces *  
# interfaces(5) file used by ifup(8) and ifdown(8)  
# Include files from /etc/network/interfaces.d:  
source /etc/network/interfaces.d/*  
  
auto lo  
iface lo inet loopback  
  
auto enps03  
iface enps03 inet dhcp  
  
auto enps08  
iface enps08 inet static  
address 192.168.19.101  
  
[ Soft wrapping of overlong lines enabled ]  
Help Write Out Where Is Cut Execute Location Go To Line Set Mark To Bracket Previous  
Exit Read File Replace Paste Justify Redo Done Where Has Next
```

- Sau đó lưu lại, khởi động lại máy ảo

sudo reboot

- Kiểm tra ip mạng bằng lệnh

Ifconfig -a

```

Password:
Welcome to Ubuntu 24.10 (GNU/Linux 6.11.0-9-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Mon Oct  7 17:20:01 UTC 2024

System load:   1.08      Processes:      30
Usage of /home: unknown  Users logged in:  0
Memory usage:  7%       IPv4 address for eth0: 10.10.10.2
Swap usage:    0%

164 updates can be applied immediately.
88 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

osboxes@osboxes:~$ ifconfig -a
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
    inet6 fe80::a00:27ff:fecf:f78a  prefixlen 64  scopeid 0x20<link>
    inet6 fd00::a00:27ff:fecf:f78a  prefixlen 64  scopeid 0x0<global>
    ether 08:00:27:cf:f7:8a  txqueuelen 1000  (Ethernet)
    RX packets 10  bytes 2168 (2.1 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 18  bytes 2216 (2.2 KB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.19.101  netmask 255.255.255.0  broadcast 192.168.19.255
    inet6 fe80::a00:27ff:fee2:7e48  prefixlen 64  scopeid 0x20<link>
    ether 08:00:27:e2:7e:48  txqueuelen 1000  (Ethernet)
    RX packets 96  bytes 8832 (8.8 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 9  bytes 726 (726.0 B)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop txqueuelen 1000  (Local Loopback)
    RX packets 84  bytes 6352 (6.3 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 84  bytes 6352 (6.3 KB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

osboxes@osboxes:~$

```

- Từ máy window, tiến hành kết nối ssh tới máy master01

```

osboxes@osboxes: ~
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\LENOVO>ssh osboxes@192.168.19.101
osboxes@192.168.19.101's password:
Welcome to Ubuntu 24.10 (GNU/Linux 6.11.0-9-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Tue Apr 15 12:59:39 PM UTC 2025

System load:   0.89
Usage of /:    6.9% of 97.87GB
Memory usage:  7%
Swap usage:    0%
Processes:    107
Users logged in:  0
IPv4 address for enp0s3: 10.0.2.15
IPv6 address for enp0s3: fd00::a00:27ff:fecf:f78a

164 updates can be applied immediately.
88 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Tue Apr 15 12:36:21 2025 from 192.168.1.108
osboxes@osboxes:~$

```

- Làm tương tự cho các máy worker còn lại

PHẦN 2: XÂY DỰNG CỤM MÁY TÍNH DỰA TRÊN DOCKER SWARM THEO YÊU CẦU

1. Docker Swarm là gì?

Docker Swarm là một công cụ **orchestration** (điều phối) được tích hợp sẵn trong Docker, cho phép quản lý và triển khai các container trên nhiều máy chủ (nodes) một cách hiệu quả. Docker Swarm biến một nhóm các máy chủ chạy Docker thành một cụm (cluster) duy nhất, hoạt động như một hệ thống thống nhất, giúp đơn giản hóa việc triển khai, mở rộng (scaling), và quản lý các ứng dụng container hóa.

- **Các khái niệm cơ bản trong Docker Swarm:**

- **Node:** Một máy chủ chạy Docker Engine, tham gia vào cụm Swarm. Có hai loại node:
 - **Manager Node:** Node quản lý, chịu trách nhiệm điều phối cụm, phân phối công việc, và duy trì trạng thái của cụm.
 - **Worker Node:** Node làm việc, chỉ thực thi các tác vụ (tasks) được giao từ manager node.
- **Service:** Một định nghĩa của ứng dụng hoặc container cần chạy trong cụm, bao gồm số lượng replicas (bản sao), image, cổng, v.v.
- **Task:** Một instance của container được chạy trên một node, do manager node phân phối.
- **Overlay Network:** Mạng ảo cho phép các container trên các node khác nhau giao tiếp với nhau.

- **Ưu điểm của Docker Swarm:**

- **Dễ sử dụng:** Được tích hợp sẵn trong Docker, không cần cài đặt thêm công cụ.
- **Tự động scaling:** Hỗ trợ mở rộng số lượng container dựa trên nhu cầu.
- **Load balancing:** Tự động phân phối tải giữa các container.
- **High Availability (HA):** Đảm bảo ứng dụng luôn sẵn sàng bằng cách tự động khởi động lại container nếu có sự cố.
- **Quản lý đơn giản:** Sử dụng các lệnh Docker quen thuộc (docker service, docker node, v.v.).

- **So sánh với Kubernetes:**

- Docker Swarm đơn giản hơn Kubernetes, phù hợp với các hệ thống nhỏ hoặc vừa.
- Kubernetes mạnh mẽ hơn, hỗ trợ nhiều tính năng phức tạp, nhưng yêu cầu cấu hình và quản lý phức tạp hơn.

Trong bài toán này, nhóm sử dụng Docker Swarm để xây dựng một cụm máy tính trên môi trường ảo hóa VirtualBox, triển khai các ứng dụng như ELK Stack và các service (webui, worker, v.v.), đảm bảo tính sẵn sàng cao và khả năng mở rộng.

2. Vai trò của Docker Swarm trong hệ thống

Docker Swarm đóng vai trò là nền tảng điều phối cho hệ thống, với các nhiệm vụ chính:

- **Quản lý cụm:** Điều phối các node trong cụm, đảm bảo các service được triển khai và chạy đúng.
- **Triển khai ứng dụng:** Triển khai các stack như elk (Elasticsearch, Logstash, Kibana) và loggingnet (webui, worker, v.v.).
- **Mở rộng và cân bằng tải:** Hỗ trợ scaling (ví dụ: tăng số replicas của worker lên 10) và phân phối tải giữa các container.
- **Giám sát và phục hồi:** Tự động khởi động lại container nếu có sự cố, đảm bảo hệ thống luôn hoạt động.

3. Môi trường triển khai

Để xây dựng cụm Docker Swarm, sử dụng VirtualBox để tạo môi trường ảo hóa. VirtualBox là một phần mềm mã nguồn mở, cho phép tạo và quản lý các máy ảo trên máy tính cá nhân, phù hợp để mô phỏng một hệ thống phân tán.

4. Cấu hình máy ảo

- **Hệ điều hành:** Ubuntu 22.04 LTS (Long Term Support), một phiên bản ổn định và phổ biến của Ubuntu.
- **Tên máy ảo:** vps0.
- **User:** osboxes.
- **Địa chỉ IP:** 192.168.19.10.

- Cấu hình mạng dạng **Host-Only Adapter** để máy local (Windows) có thể truy cập máy ảo qua SSH và trình duyệt (ví dụ: truy cập Kibana tại <http://192.168.19.10:5601>).
- **Cấu hình phần cứng:**
 - RAM: 4GB (đáp ứng yêu cầu chạy nhiều container như ELK Stack, webui, worker, v.v.).
 - CPU: 2 cores.
 - Dung lượng ổ cứng: 20GB (đủ để lưu trữ dữ liệu của Elasticsearch và các container).

5. Chuẩn bị môi trường

5.1. Cập nhật hệ thống

```
sudo apt update
```

```
sudo apt upgrade -y
```

5.2. Cài đặt SSH

```
sudo apt install -y openssh-server
```

```
sudo systemctl enable
```

```
ssh sudo systemctl start ssh
```

5.3. Kiểm tra kết nối từ máy local

Trên máy local (Windows), dùng Command Prompt:

```
ssh osboxes@192.168.56.101
```

Kết quả: Đăng nhập thành công vào máy ảo.

```

C:\Users\LENOVO>ssh osboxes@192.168.19.101
osboxes@192.168.19.101's password:
Welcome to Ubuntu 24.10 (GNU/Linux 6.11.0-9-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Tue Apr 15 01:53:59 PM UTC 2025

System load:            0.32
Usage of /:             6.9% of 97.87GB
Memory usage:          7%
Swap usage:            0%
Processes:             113
Users logged in:       0
IPv4 address for enp0s3: 10.0.2.15
IPv6 address for enp0s3: fd00::a5f2:bbc7:b9e9:366e
IPv6 address for enp0s3: fd00::a00:27ff:fe7d:81e1

150 updates can be applied immediately.
74 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Tue Apr 15 13:27:52 2025 from 192.168.19.1
osboxes@osboxes:~$ |

```

6. Cài đặt và khởi tạo cụm Docker Swarm

6.1. Cài đặt Docker

a. Cài đặt các gói phụ thuộc:

```
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
```

b. Thêm GPG key của Docker:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

c. Thêm repository của Docker:

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

d. Cập nhật danh sách gói và cài đặt Docker:

sudo apt update sudo apt install -y docker-ce docker-ce-cli containerd.io

e. Kiểm tra phiên bản Docker:

sudo docker --version

```
osboxes@osboxes:~$ sudo docker --version
Docker version 28.0.4, build b8034c0
osboxes@osboxes:~$ |
```

Kết quả: Docker version 28.0.4, build b8034c0

f. Thêm user osboxes vào nhóm docker (để chạy lệnh docker mà không cần sudo):

sudo usermod -aG docker osboxes

Đăng xuất và đăng nhập lại để áp dụng thay đổi:

exit ssh osboxes@192.168.56.101

6.2. Khởi tạo cụm Docker Swarm

a. Khởi tạo cụm

docker swarm init --advertise-addr 192.168.19.10:2377

--advertise-addr 192.168.19.10: Chỉ định địa chỉ IP của manager node để các node khác (nếu có) có thể tham gia cụm.

• Kết quả:

```
osboxes@osboxes:~$ docker swarm init --advertise-addr 192.168.19.101:2377
Swarm initialized: current node (8ieq95u3lqf7hcgq3smlofs6o) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-48tksfwaij3lu0s3k9wp3c7gjqtio9vr8cjjc7xcxgdiz572qy-48q39v8s8pt630fhdm71gde3s 192.168.19.101:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
osboxes@osboxes:~$ |
```

Cụm Docker Swarm được khởi tạo thành công, và vps0 trở thành manager node.

b. Kiểm tra trạng thái cụm

sudo docker node ls

```
osboxes@osboxes:~$ sudo docker node ls
[sudo] password for osboxes:
ID                HOSTNAME    STATUS    AVAILABILITY    MANAGER STATUS    ENGINE VERSION
8ieq95u3lqf7hcgq3sm1ofs6o *  osboxes    Ready     Active           Leader             28.0.4
osboxes@osboxes:~$
```

vps0 là node duy nhất trong cụm, có trạng thái Ready, vai trò Leader, và phiên bản Docker Engine là 28.0.4.

c. Thêm worker node

Trong trường hợp bài toán yêu cầu nhiều node, nhóm tạo thêm máy ảo trên VirtualBox để làm worker node. Các bước như sau:

i. Tạo các máy ảo vps1,...vps10

Thực hiện clone lại máy ảo vps0, đặt tên là vps1, ... vps10, cấu hình mạng NAT là Host only adapter, cấu hình RAM 2GB

ii. Tham gia cụm Swarm

Trên vps0, lấy token để worker node tham gia:

docker swarm join-token worker

```
osboxes@osboxes:~$ docker swarm join-token worker
To add a worker to this swarm, run the following command:

docker swarm join --token SWMTKN-1-48tksfwaij31u0s3k9wp3c7gjqtio9vr8cjjc7xcxgdiz572qy-48q39v8s8pt630fhdm71gde3s 192.168.19.101:2377
osboxes@osboxes:~$
```

Trên vps1, chạy lệnh tham gia:

```
osboxes@osboxes:~$ docker swarm join --token SWMTKN-1-48tksfwaij31u0s3k9wp3c7gjqtio9vr8cjjc7xcxgdiz572qy-48q39v8s8pt630fhdm71gde3s 192.168.19.101:2377
This node joined a swarm as a worker.
osboxes@osboxes:~$
```

Kiểm tra lại trên vps0:

Kết quả:

```
osboxes@osboxes:~$ sudo docker node ls
ID                HOSTNAME    STATUS    AVAILABILITY    MANAGER STATUS    ENGINE VERSION
8ieq95u3lqf7hcgq3sm1ofs6o *  osboxes    Ready     Active           Leader             28.0.4
osboxes@osboxes:~$ sudo docker node ls
ID                HOSTNAME    STATUS    AVAILABILITY    MANAGER STATUS    ENGINE VERSION
8ieq95u3lqf7hcgq3sm1ofs6o *  osboxes    Ready     Active           Leader             28.0.4
lp9b4q5f2113ulzb46dsr8i22  osboxes    Ready     Active           Leader             28.0.4
osboxes@osboxes:~$
```

Lập lại cho các node còn lại

```
osboxes@master01:~/nginx$ sudo docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
yqr7nc9g3wndo46amkxxuljyy *	master01	Ready	Drain	Leader	28.0.1
vmx4dvq9zj9zjlpdavbcb45v7	master02	Ready	Active	Reachable	28.0.1
2gx129vkm9ju366hmqlkue70g5	master03	Ready	Active	Reachable	28.0.1
ifjz7jhvbu5j5bjrox5mram3l	vps01	Ready	Active		28.0.1
td25pl4vzs5tn2co9u9od4ht4	vps02	Ready	Active		28.0.1
pu922hoj0i6fvx1sc4zr50g9h	vps03	Ready	Active		28.0.1
k7olt0f7i62txgdumqbcht7g1	vps04	Ready	Active		28.0.1
fwm77ef27yu9d52za3ji3xy6c	vps05	Ready	Active		28.0.1
69w8u56tcfls8ymu1snqdklej	vps06	Ready	Active		28.0.1
sj6lumi7wi307n4wk6u8phwdg	vps07	Ready	Active		28.0.1

7. Kết luận

Cụm Docker Swarm đã được xây dựng thành công trên môi trường VirtualBox với một node duy nhất (vps0) đóng vai trò manager node. Phiên bản Docker Engine được sử dụng là 24.0.5, đáp ứng yêu cầu về phiên bản. Mạng overlay (coinswarmnet) đã được cấu hình để đảm bảo các service trong các stack như elk (ELK Stack) và loggingnet (các service như webui, worker, v.v.) có thể giao tiếp hiệu quả. Hệ thống đã sẵn sàng cho các bước triển khai tiếp theo trong các phần sau của báo cáo.

PHẦN 3: TRIỂN KHAI CÁC GÓI ỨNG, TIẾN HÀNH HA, SCALING VÀ LOAD BALANCING CHO HỆ THỐNG

1. Thay đổi Hostname của các máy ảo:

`sudo hostname`

`sudo hostnamectl set-hostname <tên muốn set>`

Ví dụ: `sudo hostnamectl set-hostname vps0`

Sau khi thay đổi xong, reboot lại:

`sudo reboot`

Thay đổi Hostname không thay đổi tên tài khoản đăng nhập vào máy ảo.

```
osboxes@master01:~/nginx$ sudo docker node ls
ID                                HOSTNAME    STATUS    AVAILABILITY    MANAGER STATUS    ENGINE VERSION
yqr7nc9g3wndo46amkxxuljyy *      master01    Ready     Drain            Leader             28.0.1
vmx4dvq9zj9zjlpdavbcb45v7        master02    Ready     Active           Reachable          28.0.1
2gx129vkm9ju366hmqkue70g5        master03    Ready     Active           Reachable          28.0.1
ifjz7jhvbu5j5bjrox5mram3l        vps01      Ready     Active           Reachable          28.0.1
td25pl4vzs5tn2co9u9od4ht4        vps02      Ready     Active           Reachable          28.0.1
pu922hoj0i6fvx1sc4zr50g9h        vps03      Ready     Active           Reachable          28.0.1
k7olt0f7i62txgdumqbcht7g1        vps04      Ready     Active           Reachable          28.0.1
fwm77ef27yu9d52za3ji3xy6c        vps05      Ready     Active           Reachable          28.0.1
69w8u56tcfls8ymu1snqdklej        vps06      Ready     Active           Reachable          28.0.1
sj6lumi7wi307n4wk6u8phwdg        vps07      Ready     Active           Reachable          28.0.1
```

2. Promote máy worker thành máy manager:

`sudo docker node promote vps1`

```
osboxes@vps0:~$ sudo docker node promote vps1
Node vps1 promoted to a manager in the swarm.
osboxes@vps0:~$ sudo docker node ls
ID                                HOSTNAME    STATUS    AVAILABILITY    MANAGER STATUS    ENGINE VERSION
m8we3e4b19ua0uia4u75nurpe *      vps0       Ready     Active           Leader             27.5.1
f72xul4iuhha4x9x3edpv3og1        vps1       Ready     Active           Reachable          27.5.1
```

3. Triển khai các gói ứng dụng.

3.1. Tiến hành build image cho các dịch vụ: hasher, worker, rng và webui:

- Chuyển thư mục hiện hành vào rng: `cd rng/`
- Tiến hành build image rng: `sudo docker build -t rng .`
- Kiểm tra build thành công: `sudo docker image ls`
- Output:

```

osboxes@vps0:~$ pwd
/home/osboxes
osboxes@vps0:~$ cd orchestration-workshop-mrnam/
osboxes@vps0:~/orchestration-workshop-mrnam$ cd dockercoins/
osboxes@vps0:~/orchestration-workshop-mrnam/dockercoins$ ls
docker-compose.yml hasher rng webui worker
osboxes@vps0:~/orchestration-workshop-mrnam/dockercoins$ cd rng/
osboxes@vps0:~/orchestration-workshop-mrnam/dockercoins/rng$ ls
Dockerfile rng.py
osboxes@vps0:~/orchestration-workshop-mrnam/dockercoins/rng$ sudo docker build -t rng .
[+] Building 6.4s (8/8) FINISHED
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 127B
-> [internal] load metadata for docker.io/library/python:alpine
-> [internal] load .dockerignore
-> => transferring context: 2B
-> [1/3] FROM docker.io/library/python:alpine@sha256:323a717dc4a010fee21e3f1aac738ee10bb485de4e7593ce242b36ee48d6b352
-> [internal] load build context
-> => transferring context: 28B
-> CACHED [2/3] RUN pip install Flask
-> CACHED [3/3] COPY rng.py /
-> exporting to image
-> => exporting layers
-> => writing image sha256:4f836d6d6ac54c57533261cfc8506224cc76ed2cf6d2ad63dd294f4ee8c70bb
-> => naming to docker.io/library/rng
osboxes@vps0:~/orchestration-workshop-mrnam/dockercoins/rng$ sudo docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
rng          latest   4f836d6d6ac5   4 days ago    58.7MB
osboxes@vps0:~/orchestration-workshop-mrnam/dockercoins/rng$

```

- Kiểm tra sau khi đã build xong image của các dịch vụ bằng lệnh
sudo docker image ls:

```

osboxes@vps0:~/orchestration-workshop-mrnam/dockercoins$ sudo docker image
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
worker       latest   e5e6c9fdc6a7   4 days ago    59.7MB
webui        latest   8144faa4429a   4 days ago    221MB
hasher       latest   c49e0b0a316d   4 days ago    58.7MB
rng          latest   4f836d6d6ac5   4 days ago    58.7MB

```

3.2. Publish các image lên Docker Hub:

a. Đăng nhập Docker Hub:

- Dùng lệnh: sudo docker login, hệ thống sẽ trả về 1 đoạn mã dùng 1 lần, copy đoạn mã đó.
- Truy cập <https://login.docker.com/activate> rồi tiến hành dán vào hộp thoại.
- Đăng nhập thành công:

```

WARNING! Your password will be stored unencrypted in /root/.docker/config.json
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores
Login Succeeded

```

b. Push images của các dịch vụ lên Docker Hub:

- Tag và push image lên Docker Hub:
sudo docker tag rng kennex666/rng:coinswarm
sudo docker push kennex666/rng:coinswarm
- Làm tương tự cho các dịch vụ còn lại.

- Check trên Docker Hub xem image đã được push lên chưa

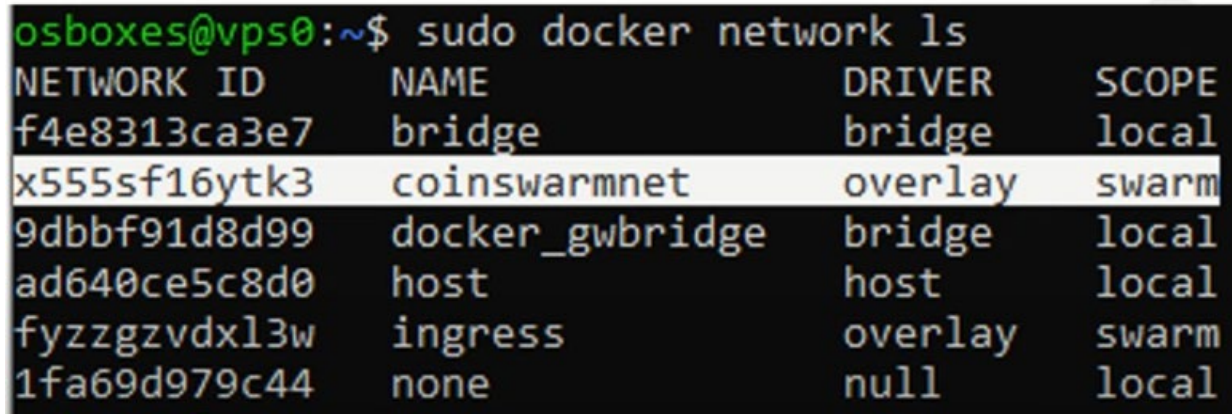
3.3. Tạo overlay network trong hệ thống swarm:

- Tạo overlay network có mã hóa:

```
sudo docker network create --scope=swarm --attachable -d overlay --opt
encrypted coinswarmnet
```

- Liệt kê danh sách các network:

```
sudo docker network ls
```



NETWORK ID	NAME	DRIVER	SCOPE
f4e8313ca3e7	bridge	bridge	local
x555sf16ytk3	coinswarmnet	overlay	swarm
9dbbf91d8d99	docker_gwbridge	bridge	local
ad640ce5c8d0	host	host	local
fyzzgzvdxl3w	ingress	overlay	swarm
1fa69d979c44	none	null	local

3.4. Tiến hành Deploy hệ thống lên swarm theo thứ tự redis, rng, hasher, worker, webui tương ứng với port và gắn vào coinswarmnet:

```
sudo docker service create --name redis --network coinswarmnet redis
```

```
sudo docker service create --name rng -p 8001:80 --network coinswarmnet
kennex666/rng:coinswarm
```

```
sudo docker service create --name hasher -p 8002:80 --network coinswarmnet
kennex666/hasher:coinswarm
```

```
sudo docker service create --name worker --network coinswarmnet
kennex666/worker:coinswarm
```

```
sudo docker service create --name webui -p 8000:80 --network coinswarmnet
kennex666/webui:coinswarm
```

3.5. Tiến hành HA cho hệ thống swarm:

a. High Availability là gì?

High Availability (HA) là khả năng của một hệ thống duy trì hoạt động ổn định và liên tục trong thời gian dài, ngay cả khi xảy ra sự cố về phần cứng, phần mềm hoặc mạng. Hệ thống có HA sẽ có các cơ chế như dự phòng (redundancy), phát hiện lỗi (health checking) và chuyển đổi dự phòng (failover) để đảm bảo thời gian ngừng hoạt động (downtime) là tối thiểu. HA hoạt động theo công thức $(2n-1)$ máy manager với n là số Single point failure.

b. HA có tác dụng gì?

HA có tác dụng đảm bảo hệ thống vẫn có thể hoạt động bình thường khi có 1 lượng máy master có vấn đề (sập, lỗi hoặc mất kết nối với hệ thống swarm).

c. Triển khai HA cho hệ thống với hệ số $n=2$:

```
sudo docker node update --availability drain vps0
```

```
sudo docker node update --availability drain vps1
```

3.6. Tiến hành Auto scaling và Auto load balancing các dịch vụ:

a. Auto scaling là gì?

Auto Scaling là cơ chế tự động điều chỉnh số lượng tài nguyên hoặc phiên bản dịch vụ (ví dụ: container, VM, pod, v.v.) dựa trên mức độ tải thực tế của hệ thống. Khi nhu cầu tăng cao, hệ thống sẽ tự động mở rộng tài nguyên (scale out); khi tải giảm, hệ thống sẽ thu hẹp tài nguyên lại (scale in) để tối ưu chi phí và hiệu năng.

b. Auto scaling có tác dụng gì?

- Đáp ứng linh hoạt khối lượng công việc thay đổi theo thời gian.
- Tối ưu hóa hiệu suất hệ thống và trải nghiệm người dùng.
- Tiết kiệm tài nguyên và chi phí vận hành, bằng cách chỉ sử dụng những tài nguyên cần thiết tại từng thời điểm.

c. Auto load balancing là gì?

Auto Load Balancing là cơ chế tự động phân phối lưu lượng truy cập hoặc yêu cầu từ người dùng đến nhiều nút (node), máy chủ hoặc container khác nhau trong hệ thống, giúp cân bằng tải và tránh tình trạng quá tải tại một điểm đơn lẻ.

d. Auto load balancing có tác dụng gì?

- Tối ưu hiệu năng hệ thống và tốc độ phản hồi.
- Đảm bảo hệ thống hoạt động ổn định ngay cả khi có lưu lượng lớn.
- Tăng khả năng mở rộng và tính sẵn sàng của hệ thống.
- Hỗ trợ triển khai kiến trúc phân tán (distributed system) một cách hiệu quả.

e. Triển khai Auto scaling trong hệ thống:

`sudo docker service scale worker=15`

```
osboxes@master01:~/nginx$ sudo docker service scale worker=15
worker scaled to 15
overall progress: 15 out of 15 tasks
1/15: running [=====>]
2/15: running [=====>]
3/15: running [=====>]
4/15: running [=====>]
5/15: running [=====>]
6/15: running [=====>]
7/15: running [=====>]
8/15: running [=====>]
9/15: running [=====>]
10/15: running [=====>]
11/15: running [=====>]
12/15: running [=====>]
13/15: running [=====>]
14/15: running [=====>]
15/15: running [=====>]
verify: Service worker converged
```

`sudo docker service scale rng=10`

```

osboxes@master01:~/nginx$ sudo docker service scale rng=10
rng scaled to 10
overall progress: 10 out of 10 tasks
1/10: running [=====>]
2/10: running [=====>]
3/10: running [=====>]
4/10: running [=====>]
5/10: running [=====>]
6/10: running [=====>]
7/10: running [=====>]
8/10: running [=====>]
9/10: running [=====>]
10/10: running [=====>]
verify: Service rng converged

```

sudo docker service scale hasher=10

```

osboxes@master01:~/nginx$ sudo docker service scale hasher=10
hasher scaled to 10
overall progress: 10 out of 10 tasks
1/10: running [=====>]
2/10: running [=====>]
3/10: running [=====>]
4/10: running [=====>]
5/10: running [=====>]
6/10: running [=====>]
7/10: running [=====>]
8/10: running [=====>]
9/10: running [=====>]
10/10: running [=====>]
verify: Service hasher converged

```

sudo docker service scale webui=5

```

osboxes@master01:~/nginx$ sudo docker service scale webui=5
webui scaled to 5
overall progress: 5 out of 5 tasks
1/5: running [=====>]
2/5: running [=====>]
3/5: running [=====>]
4/5: running [=====>]
5/5: running [=====>]
verify: Service webui converged

```

sudo docker service scale redis=2

```

osboxes@master01:~/nginx$ sudo docker service scale redis=2
redis scaled to 2
overall progress: 2 out of 2 tasks
1/2: running [=====>]
2/2: running [=====>]
verify: Service redis converged

```

PHẦN 4: GIÁM SÁT VÀ QUẢN TRỊ HỆ THỐNG DOCKER SWARM

1. Giám sát hệ thống Docker Swarm là gì?

Giám sát Docker Swarm là quá trình theo dõi và đánh giá hiệu suất, tình trạng và các thông số quan trọng khác của cụm Swarm (node, container, service, log).

Mục tiêu:

- Theo dõi và đánh giá hiệu suất, tình trạng và các thông số quan trọng của cụm Docker Swarm.
- Phát hiện sớm các vấn đề tiềm ẩn hoặc sự cố xảy ra.
- Đảm bảo tính ổn định và khả dụng của các ứng dụng chạy trên Swarm.

Các hoạt động giám sát bao gồm:

- Giám sát tài nguyên: Theo dõi việc sử dụng CPU, bộ nhớ, mạng và ổ đĩa của các nút trong cụm.
- Giám sát dịch vụ: Kiểm tra tình trạng của các dịch vụ đang chạy, số lượng bản sao, thời gian hoạt động và các lỗi phát sinh.
- Giám sát mạng: Theo dõi lưu lượng, độ trễ

2. Quản trị hệ thống Docker Swarm là gì?

Quản trị hệ thống Docker Swarm là tập hợp các hoạt động và quy trình liên quan đến việc thiết lập, duy trì, và tối ưu hóa một cụm Docker Swarm. Điều này bao gồm nhiều khía cạnh, từ việc triển khai ban đầu đến việc giám sát liên tục và xử lý sự cố.

Mục tiêu:

- Thiết lập, duy trì và tối ưu hóa cụm Docker Swarm.
- Triển khai và quản lý các ứng dụng trên Swarm.
- Đảm bảo an ninh và hiệu suất của hệ thống.

Các hoạt động quản trị bao gồm:

- Thiết lập và cấu hình cụm
- Triển khai và quản lý dịch vụ
- Giám sát và bảo trì hệ thống
- Quản lý tài nguyên.

3. Tại sao cần Giám sát và quản trị hệ thống Docker Swarm

Giám sát cung cấp dữ liệu và thông tin cần thiết cho quản trị.

Quản trị sử dụng thông tin từ giám sát để đưa ra các quyết định và hành động phù hợp.

Giám sát giúp phát hiện các vấn đề cần được giải quyết bởi quản trị.

Quản trị thiết lập các cơ chế giám sát để đảm bảo hệ thống hoạt động tốt.

4. Triển khai Giám sát và quản trị hệ thống Docker Swarm

4.1. Kiểm tra danh sách các node trong Swarm và trạng thái của chúng:

sudo docker node ls

```
osboxes@master01:~/nginx$ sudo docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
yqr7nc9g3wndo46amkxxuljyy *	master01	Ready	Drain	Leader	28.0.1
vmx4dvq9zj9zjlpdavbcb45v7	master02	Ready	Active	Reachable	28.0.1
2gx129vkm9ju366hmqlkue70g5	master03	Ready	Active	Reachable	28.0.1
ifjz7jhvbu5j5bjrox5mram3l	vps01	Ready	Active		28.0.1
td25pl4vzs5tn2co9u9od4ht4	vps02	Ready	Active		28.0.1
pu922hoj0i6fvxlsc4zr50g9h	vps03	Ready	Active		28.0.1
k7olt0f7i62txgdumqbcht7g1	vps04	Ready	Active		28.0.1
fwm77ef27yu9d52za3ji3xy6c	vps05	Ready	Active		28.0.1
69w8u56tcfls8ymu1snqdklej	vps06	Ready	Active		28.0.1
sj6lumi7wi307n4wk6u8phwdg	vps07	Ready	Active		28.0.1

Tại đây sẽ hiện danh sách các node trong cụm. Giúp chúng ta biết được đang thực hiện trên node nào, trạng thái hoạt động của từng node, mức độ sẵn sàng nhận container và phiên bản Docker Engine. Điều này giúp chúng ta quản trị hệ thống tốt hơn theo dõi, đánh giá tình trạng và vai trò của từng node trong Docker Swarm, từ đó ra quyết định triển khai, bảo trì hoặc mở rộng hệ thống một cách hiệu quả.

4.2. Xem danh sách service hiện có

sudo docker service ls

Các dịch vụ đang chạy trong Docker swarm sẽ được hiển thị một cách tổng quát : ID, tên service và tên image Docker để chạy container. Ngoài ra còn cung cấp ta các thông tin như:

- “MODE” thể hiện chế độ triển khai service. Cụ thể replicated: service được triển khai theo số lượng bản sao (replica). global: mỗi node trong Swarm sẽ chạy đúng 1 bản sao.

- “REPLICAS” Hiển thị số bản sao đang chạy / tổng số bản sao thiết lập.
- PORTS :Mapping cổng giữa host và container, giúp truy cập dịch vụ từ bên ngoài.

4.3. Kiểm tra chi tiết các container đang chạy để phục vụ cho dịch vụ `sudo docker service ps <tên service>`

Để xem chi tiết phân phối của các service đang chạy. Cung cấp thông tin về các node đang chạy dịch vụ, trạng thái mong muốn (DESIRED STATE) và trạng thái thực tế (CURRENT STATE) đồng thời cả lỗi nếu task đó không hoạt động.

4.4. Truy xuất và hiển thị các log được tạo ra bởi các container (tasks) đang chạy của dịch vụ

`sudo docker service logs <tên service>`

Ta còn có thể xem log của từng task

`sudo docker service ps <tên service>`

`sudo docker logs <ID>`

4.5. Hiển thị thống kê sử dụng tài nguyên trực tiếp của các container đang chạy

`sudo docker stats`

Kết quả:

CONTAINER ID	NAME	CPU %	MEM	
USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
659e81fbf177	prometheus.1.m9ry2m4ngrlfzs4tbixqun0q9			1.00%
47MiB / 3.324GiB	1.38%	18.4kB / 8.97kB	63.4MB / 65.5kB	8
f2c2914af6f8				
cadvisor.miz3esii35v97gc9vhakm3id1.sd3qtn5ryau39lc5i26015jbm	2.11%			
39.26MiB / 3.324GiB	1.15%	880B / 0B	21MB / 0B	11

Các metric:

CONTAINER ID: ID của container đang chạy

NAME: Tên container, có thể thấy là service trong Docker Swarm

CPU %: Tỷ lệ CPU đang sử dụng so với tổng CPU cấp cho container

MEM USAGE / LIMIT: RAM đang dùng / RAM được cấp phép (host RAM nếu không giới hạn)

MEM %: % RAM đang dùng so với RAM limit

NET I/O: Tổng lưu lượng mạng vào / ra container

BLOCK I/O: Dữ liệu đọc / ghi từ ổ đĩa (volume, bind mount...)

PIDS: Số tiến trình (process) đang chạy trong container

4.6. Kiểm tra và đo lường CPU / memory và I/O Usage của một node

CPU & Memory: top hoặc htop

I/O: Dùng iostat (từ sysstat)

sudo apt install sysstat

iostat -dx 1

Kết quả:

Device	r/s	rkB/s	rrqm/s	%rrqm	r_await	rareq-sz	w/s	wkB/s	wrqm/s	%wrqm	w_await	wareq-sz	d/s	dkB/s	drqm/s	%drqm	d_await	dareq-sz	f/s	f_await	aq-sz	%util
dm-0	25.52	473.92	0.00	0.00	6.23	18.57	68.48	332.97	0.00	0.00	7.63	4.86	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.68
	11.26																					
loop0	0.00	0.00	0.00	0.00	0.00	1.27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
sda	22.50	476.45	3.16	12.31	5.01	21.18	25.43	333.02	43.10	62.89	5.84	13.10	0.00	0.00	0.00	0.00	0.00	0.00	7.11	5.06	0.30	10.30

Network I/O: ifstat, vnstat hoặc nload


```
sudo apt install ifstat
ifstat -i enp0s3 1
```

```
enp0s3
KB/s in KB/s out
0.00  27.59
0.29  31.63
0.29  31.14
0.29  29.67
0.00  18.13
0.00  31.95
```

- Kiểm tra tài nguyên được khai báo:

```
sudo docker node inspect worker1 --pretty
```

- Dùng cAdvisor để giám sát tài nguyên các container trên từng node

- Cài đặt:

```
VERSION=v0.49.1 # use the latest release version from
https://github.com/google/cadvisor/releases
```

```
sudo docker run \
```

```
--volume=/:/rootfs:ro \
```

```
--volume=/var/run:/var/run:ro \
```

```
--volume=/sys:/sys:ro \
```

```
--volume=/var/lib/docker:/var/lib/docker:ro \
```

```
--volume=/dev/disk:/dev/disk:ro \
```

```
--publish=8080:8080 \
```

```
--detach=true \
```

```
--name=cadvisor \
```

```
--privileged \
```

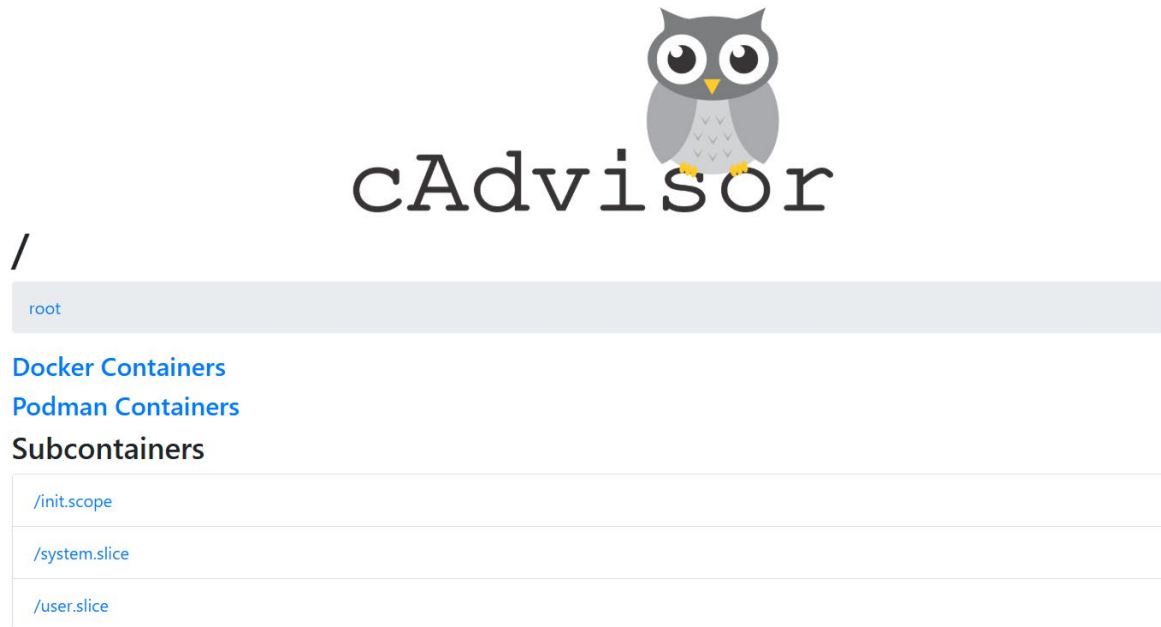
```
--device=/dev/kmsg \
```

```
gcr.io/cadvisor/cadvisor:\$VERSION
```

- Kiểm tra xem đã cài Cadvisor rồi

`sudo docker ps | grep cadvisor`

Truy cập vào đường dẫn để xem: `http://<ip_may>:808`



4.7. Kiểm tra độ trễ phản hồi của các dịch vụ

- Cài đặt : `sudo apt install httping`
- Kiểm tra: `sudo httping -c 10 <ip_may>:<port_service> <ten_service>`

```
osboxes@vps0:~$ sudo httping -c 10 192.168.19.10:8001 rng
[sudo] password for osboxes:
PING 192.168.19.10:8001 (/):
connected to 192.168.19.10:8001 (177 bytes), seq=0 time= 74.99 ms
connected to 192.168.19.10:8001 (177 bytes), seq=1 time= 19.29 ms
connected to 192.168.19.10:8001 (177 bytes), seq=2 time=  9.38 ms
connected to 192.168.19.10:8001 (177 bytes), seq=3 time=185.40 ms
connected to 192.168.19.10:8001 (177 bytes), seq=4 time= 11.07 ms
^CGot signal 2
--- http://192.168.19.10:8001/ ping statistics ---
5 connects, 5 ok, 0.00% failed, time 5583ms
```

PHẦN 5: ELK STACK

1. Giới thiệu

ELK Stack là bộ công cụ mã nguồn mở do công ty Elastic phát triển, được sử dụng rộng rãi trong việc xử lý, phân tích và trực quan hóa dữ liệu lớn theo thời gian thực. Tên "ELK" là viết tắt của ba thành phần chính: Elasticsearch, Logstash và Kibana, mỗi thành phần đảm nhận một vai trò riêng biệt nhưng kết hợp chặt chẽ để tạo thành một hệ sinh thái mạnh mẽ.

Elasticsearch là một công cụ tìm kiếm và phân tích phân tán, được xây dựng dựa trên Apache Lucene. Nó lưu trữ dữ liệu dưới dạng tài liệu JSON, cho phép xử lý nhanh các truy vấn tìm kiếm phức tạp và phân tích dữ liệu theo thời gian thực. Có khả năng mở rộng, hỗ trợ tìm kiếm full-text và phân tích dữ liệu gần như tức thời.

Logstash là công cụ thu thập, xử lý và chuyển đổi dữ liệu, đóng vai trò như một "đường ống" (pipeline) để đưa dữ liệu từ nhiều nguồn vào Elasticsearch. Logstash hỗ trợ nhập dữ liệu từ các nguồn như file log, cơ sở dữ liệu, hoặc luồng dữ liệu thời gian thực, sau đó lọc và định dạng dữ liệu trước khi lưu trữ.

Kibana là công cụ trực quan hóa dữ liệu, cung cấp giao diện web thân thiện để người dùng khám phá và trình bày dữ liệu từ Elasticsearch. Với Kibana, người dùng có thể tạo biểu đồ, bảng điều khiển (dashboard), và các hình ảnh hóa dữ liệu trực quan để theo dõi và phân tích thông tin. Kibana tích hợp nhiều hình ảnh hóa vào một bảng điều khiển duy nhất để giám sát toàn diện, giúp biến dữ liệu thô thành thông tin dễ hiểu, hỗ trợ ra quyết định nhanh chóng trong các lĩnh vực như giám sát hệ thống hoặc phân tích kinh doanh.

2. Cài đặt

- **Bước 1:** Bạn cần một Docker Swarm Cluster với nhiều node (ít nhất 1 manager và 1 worker).

Kiểm tra swarm bằng lệnh: *docker node ls*

```
osboxes@master01:~$ sudo docker node ls
[sudo] password for osboxes:
ID                                HOSTNAME    STATUS    AVAILABILITY    MANAGER STATUS    ENGINE VERSION
yqr7nc9g3wndo46amkxxuljyy *     master01    Ready    Drain           Reachable         28.0.1
vmx4dvq9zj9zjlpdavbcb45v7       master02    Ready    Active          Reachable         28.0.1
2gx129vkm9ju366hmqkue70g5       master03    Ready    Active          Leader            28.0.1
ifjz7jhvbu5j5bjrox5mram3l       vps01      Ready    Active          Reachable         28.0.1
td25pl4vzs5tn2co9u9od4ht4       vps02      Ready    Active          Reachable         28.0.1
pu922hoj0i6fvx1sc4zr50g9h       vps03      Ready    Active          Reachable         28.0.1
k7olt0f7i62txgdumqbcht7g1       vps04      Ready    Active          Reachable         28.0.1
fwm77ef27yu9d52za3ji3xy6c       vps05      Ready    Active          Reachable         28.0.1
69w8u56tcfls8ymu1snqdklej       vps06      Ready    Active          Reachable         28.0.1
sj6lumi7wi307n4wk6u8phwdg       vps07      Ready    Active          Reachable         28.0.1
osboxes@master01:~$
```

- Bước 2: Tạo Docker Network

Tạo một overlay network để các service ELK có thể giao tiếp với nhau.

```
osboxes@master01:~$ sudo docker network create --driver overlay elk_network
uynsp7jn8r0b6bhoypuza678o
osboxes@master01:~$
```

- Bước 3: Tải file elk-app của nhóm, Giải nén file ra sau đó đẩy lên master01

Mở cmd tại root (chứa elk-app, không mở trong elk-app):

Gõ: `scp -r elk-app osboxes@192.168.19.10:/home/osboxes`

- Bước 4: Triển khai ELK trên Docker Swarm

a. Đăng nhập master01

b. Chuyển đến folder vừa đẩy lên (/home/osboxes/elk-app), gõ

`sudo docker config create logstash_config logstash.conf`

c. Sau đó deploy bằng lệnh `sudo docker stack deploy -c docker-compose.yml`

`elk`

```
osboxes@master01:~/elk-app$ sudo docker stack deploy -c docker-compose.yml elk
yaml: line 4: found character that cannot start any token
osboxes@master01:~/elk-app$ sudo docker stack deploy -c docker-compose.yml elk
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network elk_elk_network
Creating service elk_kibana
Creating service elk_elasticsearch
Creating service elk_logstash
osboxes@master01:~/elk-app$
```

- Bước 5: Kiểm tra trạng thái

Xem danh sách các service: `sudo docker service ls`

```
osboxes@master01:~/elk-app$ sudo docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
vsce45vmf0d6	cadvisor	global	9/9	gcr.io/cadvisor/cadvisor:latest	
askih2pygnd2	elk_elasticsearch	replicated	0/1	docker.elastic.co/elasticsearch/elasticsearch:8.5.1	*:9200-
>9200/tcp					
ybnlyt41nvp3	elk_kibana	replicated	0/1	docker.elastic.co/kibana/kibana:8.5.1	*:5601-
>5601/tcp					
lbyy20kw8mn4	elk_logstash	replicated	0/1	docker.elastic.co/logstash/logstash:8.5.1	*:5044-
>5044/tcp					
q1hf7t69rh36	hasher	replicated	10/10	kennex666/hasher:coinswarm	*:8002-
>80/tcp					
mxkj03wtn17q	redis	replicated	2/2	redis:latest	
kya1w992745o	rng	replicated	10/10	kennex666/rng:coinswarm	*:8001-
>80/tcp					
5gxh4qft4oql	webui	replicated	5/5	kennex666/webui:coinswarm	*:8000-
>80/tcp					
9ccmz61b895a	worker	replicated	15/15	kennex666/worker:coinswarm	

Xem logs của từng service:

```
docker service logs elk_elasticsearch
```

```
docker service logs elk_logstash
```

```
docker service logs elk_kibana
```

Gán driver gref vào service:

```
sudo docker service update \
```

```
--log-driver=gelf \
```

```
--log-opt gelf-address=udp://192.168.19.10:12201 \
```

```
--log-opt tag="{{.Name}}" \
```

```
rng
```

```
sudo docker service update \
```

```
--log-driver=gelf \
```

```
--log-opt gelf-address=udp://192.168.19.10:12201 \
```

```
--log-opt tag="{{.Name}}" \
```

```
worker
```

```
sudo docker service update \
```

```
--log-driver=gelf \
```

```
--log-opt gelf-address=udp://192.168.19.10:12201 \
```

```
--log-opt tag="{{.Name}}" \
```

webui

```
sudo docker service update \
```

```
--log-driver=gelf \
```

```
--log-opt gelf-address=udp://192.168.19.10:12201 \
```

```
--log-opt tag="{{.Name}}" \
```

hasher

- **Bước 7: Truy cập Kibana**

- Mở trình duyệt, vào: <http://192.168.19.10:5601> (ip của manager)
- Mở mục Elastic -> Discover -> Create new Data
- Nhập tên log
- Pattern nhập: log*
- Nhấn Create

Thông tin bổ sung:

- hasher, rng sẽ gồm info cơ bản gồm ip, ngày, method http, trạng thái http, uri

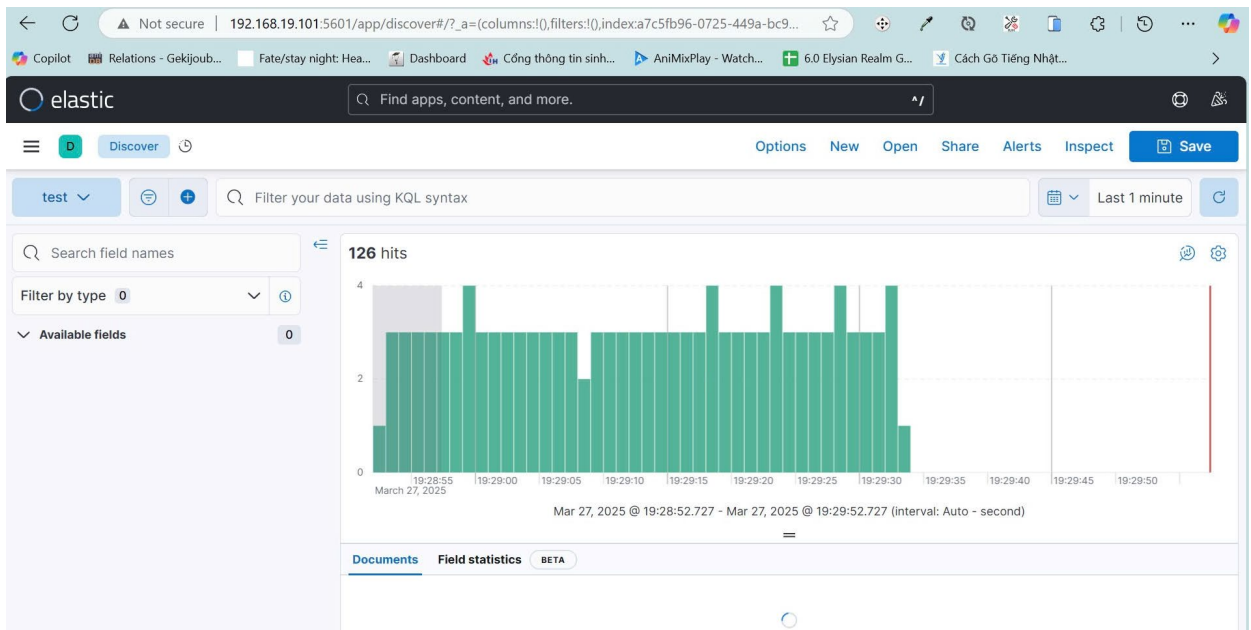
Mẫu: 10.0.1.34 - - [14/Apr/2025 21:55:42] "GET /32 HTTP/1.1" 200 -

- Webui có log liên quan đến redis khi bị lỗi, log khi khởi động. (message).

Để thấy được log của webui sẽ rất khó vì không có log liên tục, dùng lệnh *sudo docker service scale webui=0* sau đó về 5 sẽ thấy log.

- Worker sẽ có thông tin ``INFO: __main__:5 units of work done, updating hash counter``
- Tất cả đều parse về json, trừ webui do nó chỉ có message.

Kết quả sau khi hoàn thành:



PHẦN 6: PROMETHUS-GRAFANA-INFLUXDB

1. Giới thiệu về prometheus, grafana và influxdb

1.1. Prometheus

Prometheus là một hệ thống giám sát và cảnh báo mã nguồn mở, được phát triển bởi SoundCloud và sau đó trở thành dự án của Cloud Native Computing Foundation (CNCF). Prometheus được thiết kế để thu thập và lưu trữ dữ liệu thời gian thực (time-series data) dựa trên mô hình pull, nghĩa là nó chủ động truy vấn (scrape) các endpoint HTTP của các mục tiêu (targets) để lấy metric.

Vai trò: Prometheus được triển khai trên swarm và thu thập về tài nguyên hệ thống như CPU, RAM, disk,... trên các node trong hệ thống swarm.

1.2. InfluxDB

InfluxDB là một cơ sở dữ liệu time-series mã nguồn mở, được thiết kế đặc biệt để xử lý và lưu trữ dữ liệu theo thời gian (time-series data) với hiệu suất cao. InfluxDB thường được sử dụng trong các hệ thống giám sát, IoT, và phân tích dữ liệu thời gian thực. Dữ liệu trong InfluxDB được tổ chức dưới dạng bucket (tương tự database) và được quản lý theo tổ chức (organization), với hệ thống token để kiểm soát quyền truy cập, đảm bảo tính bảo mật và linh hoạt trong quản lý.

Vai trò: InfluxDB đóng vai trò là nơi lưu trữ lâu dài các metric thu thập từ Prometheus. Telegraf đẩy dữ liệu từ Prometheus vào bucket trên InfluxDB, sau đó dữ liệu có thể được truy vấn qua InfluxDB UI hoặc API.

1.3. Grafana

Grafana là một nền tảng mã nguồn mở để trực quan hóa và phân tích dữ liệu, được sử dụng rộng rãi trong các hệ thống giám sát. Grafana cho phép người dùng tạo các dashboard tương tác, hiển thị dữ liệu dưới dạng biểu đồ, đồ thị, và bảng.

1.4. Telegraf

Telegraf là một agent thu thập dữ liệu mã nguồn mở, được phát triển bởi InfluxData. Telegraf đóng vai trò trung gian, thu thập dữ liệu từ nhiều nguồn (inputs) và ghi vào nhiều đích đến (outputs).

Vai trò: Telegraf thu thập metric từ Prometheus qua endpoint /metrics và ghi vào InfluxDB và đảm bảo dữ liệu được lưu trữ lâu dài để phân tích sau này.

2. Cài đặt

2.1. Đảm bảo Docker Swarm hoạt động

Trên node manager(ví dụ vps0), kiểm tra trạng thái Swarm:

```
sudo docker node ls
```

```
[sudo] password for osboxes:
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
oz1kdrdljk6daghqphmdeg8nq *	vps0	Ready	Active	Leader	28.0.4
m5c39xnqwgjx3iy4wds7771q	vps1	Ready	Active	Reachable	28.0.4
dwck7xqkhw5y7a9i0895w8sy0	vps2	Down	Active	Unreachable	28.0.4
mbddkt7w5t2s8dj2d6onflz36	vps3	Down	Active		28.0.4
epudzh075trq0aqxh4snwx3co	vps4	Down	Active		28.0.4

Nếu có node nào bị down thì khởi động lại node đó.

2.2. Cấu hình daemon.json trên mỗi node

Truy cập vào từng node bằng ssh, sau đó chỉnh sửa /etc/docker/daemon.json

```
sudo nano /etc/docker/daemon.json
```

Thêm nội dung này vào, sau đó lưu lại

```
{  
  "metrics-addr": "0.0.0.0:9323",  
  "experimental": true,  
  "dns": ["8.8.8.8", "8.8.4.4", "127.0.0.11"]  
}
```

Thực hiện restart lại docker daemon

```
sudo systemctl restart docker
```

Và kiểm tra lại trạng thái sau khi restart, nếu thấy running thì docker đã được restart lại và ổn định.

```
sudo systemctl status docker
```

Thực hiện cho tất cả các node còn lại

2.3. Tạo network

Thực hiện lệnh để tạo 1 network riêng cho prometheus, grafana, influxdb và telegraf

```
sudo docker network create --driver overlay monitoring_network
```

Kiểm tra xem có network chưa

```
sudo docker network ls
```

```
osboxes@vps0:~$ sudo docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
b81dea391969	bridge	bridge	local
pe7ht66jrqrq	coinswarmnet	overlay	swarm
a12912e77565	docker_gwbridge	bridge	local
wzg8oyadgktw	elk_elk_network	overlay	swarm
uqub61lyq0np	elk_network	overlay	swarm
2119259d4010	host	host	local
vaxunih9zypy	ingress	overlay	swarm
42g6jpt7g6gn	monitoring_network	overlay	swarm
06708249c4d0	mrnamnet	bridge	local
5821ed03abf4	none	null	local
adixq69sh2p6	sys-metrics_default	overlay	swarm

2.4. Triển khai node-exporter và cadvisor

Hai service là sub_monitoring_cadvisor và sub_monitoring_node-exporter giúp thu thập thông tin hệ thống và container, vì Prometheus không thể tự thu thập thông tin nếu ko có các exporter.

Ở giao diện vps0, ta về lại /home/osboxes, sau đó tạo thư mục sub_monitoring
Tiến hành tạo file docker-compose.yml

```
sudo nano docker-compose.yml
```

Sau đó dán nội dung dưới và lưu lại

```
version: '3.8'
```

```
services:
```

```
node-exporter:
```

```
image: prom/node-exporter:latest
```

```
deploy:
```

```
mode: global
```

```
volumes:
```

- /proc:/host/proc:ro

- /sys:/host/sys:ro

- /:/rootfs:ro

command:

- '--path.procfs=/host/proc'

- '--path.sysfs=/host/sys'

- '--path.rootfs=/rootfs'

ports:

- target: 9100

published: 9100

mode: host

networks:

- monitoring_network

cadvisor:

image: gcr.io/cadvisor/cadvisor:latest

deploy:

mode: global

volumes:

- /:/rootfs:ro

- /var/run:/var/run:ro

- /sys:/sys:ro

- /var/lib/docker:/var/lib/docker:ro

- /dev/disk:/dev/disk:ro

ports:

- target: 8080

published: 8080

mode: host

networks:

- *monitoring_network*

networks:

monitoring_network:

external: true

Sau đó thực hiện deploy stack *sub_monitoring*

sudo docker stack deploy -c docker-compose.yml sub_monitoring

Thực hiện kiểm tra lại các service

sudo docker service ls

Nếu thấy đã có đủ replica chạy trên toàn bộ các node thì đã thành công.

d3tm7xmd4725	sub_monitoring_cadvisor	global	5/2	gcr.io/cadvisor/cadvisor:latest
iapzx903mjkl	sub_monitoring_node-exporter	global	5/2	prom/node-exporter:latest

2.5. Cấu hình docker-compose.yml cho prometheus, influxdb, telegraf và grafana

Về lại dir /home/osboxes, tạo 1 dir là *monitoring*, sau đó tạo file *docker-compose.yml*

sudo nano docker-compose.yml

Sau đó dán nội dung vào, lưu lại.

version: '3.8'

services:

influxdb:

image: influxdb:latest

networks:

- *monitoring_network*

ports:

- *"8086:8086"*

volumes:

- *influxdb_data:/var/lib/influxdb2*

deploy:

placement:

constraints: [node.role == manager]

replicas: 1

environment:

- *DOCKER_INFLUXDB_INIT_MODE=setup*
- *DOCKER_INFLUXDB_INIT_USERNAME=admin*
- *DOCKER_INFLUXDB_INIT_PASSWORD=admin123*
- *DOCKER_INFLUXDB_INIT_ORG=osboxes*
- *DOCKER_INFLUXDB_INIT_BUCKET=osboxes*
- *DOCKER_INFLUXDB_INIT_TOKEN=my-init-token-omggggg*
- *INFLUXD_HTTP_BIND_ADDRESS=0.0.0.0:8086*
- *DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=myadmintoken*

sysctls:

- *net.ipv6.conf.all.disable_ipv6=1*

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost:8086/ping"]

interval: 10s

timeout: 5s

retries: 5

start_period: 30s

prometheus:

image: prom/prometheus:latest

networks:

- *monitoring_network*

ports:

- *"9090:9090"*

deploy:

placement:

```

    constraints: [node.role == manager]
  replicas: 1
  configs:
    - source: prometheus_config
      target: /etc/prometheus/prometheus.yml
  volumes:
    - prometheus_data:/prometheus
    - /var/run/docker.sock:/var/run/docker.sock:ro
  healthcheck:
    test: ["CMD", "wget", "--spider", "http://localhost:9090/-/healthy"]
    interval: 10s
    timeout: 5s
    retries: 5
    start_period: 30s
  dns:
    - 8.8.8.8
    - 8.8.4.4

grafana:
  image: grafana/grafana:latest
  networks:
    - monitoring_network
  ports:
    - "3000:3000"
  volumes:
    - grafana_data:/var/lib/grafana
  deploy:
    placement:
      constraints: [node.role == manager]

```

replicas: 1

environment:

- *GF_SECURITY_ADMIN_USER=admin*
- *GF_SECURITY_ADMIN_PASSWORD=admin*

telegraf:

image: telegraf:latest

user: root # ➤ *THÊM DÒNG NÀY!*

networks:

- *monitoring_network*

configs:

- *source: telegraf_config*
- target: /etc/telegraf/telegraf.conf*

volumes:

- */var/run/docker.sock:/var/run/docker.sock:ro* # ➤ *Cần để đọc docker info*

deploy:

mode: global # ➤ *QUAN TRỌNG: mỗi node chạy 1 container Telegraf*

restart_policy:

condition: on-failure

delay: 5s

max_attempts: 20

window: 60s

healthcheck:

test: ["CMD", "curl", "-f", "http://influxdb:8086/health"]

interval: 10s

timeout: 2s

retries: 10

start_period: 30s

```
dns:  
- 8.8.8.8  
- 8.8.4.4  
- 127.0.0.11
```

```
networks:  
monitoring_network:  
external: true
```

```
volumes:  
influxdb_data:  
grafana_data:  
prometheus_data: # Thêm volume này để lưu dữ liệu Prometheus
```

```
configs:  
prometheus_config: # Thêm config cho Prometheus  
external: true  
telegraf_config:  
external: true
```

2.6. Cấu hình prometheus.yml

Ở dir monitoring, tạo file prometheus.yml

```
sudo nano prometheus.yml
```

Sau đó dán nội dung bên dưới vào và lưu lại

```
global:  
scrape_interval: 15s  
  
scrape_configs:
```



```

- job_name: 'prometheus'
  static_configs:
    - targets: ['localhost:9090']

- job_name: 'node-exporter'
  dns_sd_configs:
    - names: ['tasks.sub_monitoring_node-exporter']
      type: A
      port: 9100
      refresh_interval: 15s

- job_name: 'cadvisor'
  dns_sd_configs:
    - names: ['tasks.sub_monitoring_cadvisor']
      type: A
      port: 8080
      refresh_interval: 15s

```

2.7. Cấu hình telegraf.conf

Ở dir monitoring, tạo file telegraf.conf

```
sudo nano telegraf.conf
```

Sau đó dán nội dung bên dưới vào và lưu lại

```

[agent]
interval = "10s"
round_interval = true
metric_batch_size = 1000
metric_buffer_limit = 10000
collection_jitter = "0s"
flush_interval = "10s"
flush_jitter = "0s"

```

```

precision = ""
hostname = ""
omit_hostname = false
[[outputs.influxdb_v2]]
  urls = ["http://influxdb:8086"]
  token = "my-init-token-omggggg"
  organization = "osboxes"
  bucket = "osboxes"
[[inputs.cpu]]
  percpu = true
  totalcpu = true
  collect_cpu_time = false
  report_active = true
[[inputs.mem]]
[[inputs.swap]]
[[inputs.system]]
[[inputs.disk]]
  ignore_fs = ["tmpfs", "devtmpfs", "overlay"]
[[inputs.net]]
[[inputs.processes]]
[[inputs.netstat]]
[[inputs.diskio]]

```

a. Triển khai stack monitoring

Deploy config và docker file

```

sudo docker config create prometheus_config ./prometheus.yml
sudo docker config create telegraf_config ./telegraf.conf
sudo docker stack deploy -c docker-compose.yml monitoring

```

Sau khi chạy, chờ khoảng 3-4 phút để các service chạy ổn định, có thể kiểm tra bằng lệnh

sudo docker stack ps monitoring

```
osboxes@vps0:~$ sudo docker stack ps monitoring
```

ID	NAME	PORTS	IMAGE	NODE	DESIRED STATE	CURRENT STATE
r2ig0kvnmj20	monitoring_grafana.1		grafana/grafana:latest	vps1	Running	Running 14 minutes ago
joz0qbq0ff5f	_ monitoring_grafana.1		grafana/grafana:latest	vps2	Shutdown	Running 2 hours ago
acspif5hoapf	monitoring_influxdb.1		influxdb:2.7	vps1	Running	Running 13 minutes ago
mqnovwev23ia	_ monitoring_influxdb.1		influxdb:2.7	vps2	Shutdown	Running 2 hours ago
fg6nlwxskbsu	monitoring_prometheus.1		prom/prometheus:latest	vps0	Running	Running 15 minutes ago
uedwgqqck4tl	_ monitoring_prometheus.1		prom/prometheus:latest	vps0	Shutdown	Failed 16 minutes ago
n9ar7726zsg6	_ monitoring_prometheus.1		prom/prometheus:latest	vps0	Shutdown	Shutdown 16 minutes ago
pu8e2jlswo2	_ monitoring_prometheus.1		prom/prometheus:latest	vps2	Shutdown	Rejected 2 hours ago
4ib2zenv60i5	_ monitoring_prometheus.1		prom/prometheus:latest	vps2	Shutdown	Rejected 2 hours ago
swmj3yzct37a	monitoring_telegraf.1		telegraf:latest	vps0	Running	Running 14 minutes ago
d2wqzqfpa4ai	_ monitoring_telegraf.1		telegraf:latest	vps1	Shutdown	Rejected 16 minutes ago
pxuf695jg8rn	_ monitoring_telegraf.1		telegraf:latest	vps0	Shutdown	Failed 16 minutes ago
9jc3j374ntr9	_ monitoring_telegraf.1		telegraf:latest	vps0	Shutdown	Shutdown 16 minutes ago

b. Kiểm tra service

sudo docker service ls

```
osboxes@master01:~/nginx$ sudo docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
0tekoykvkdl1	elk_elasticsearch	replicated	1/1	docker.elastic.co/elasticsearch/elasticsearch:8.5.1	*:9200->9200/tcp
kkp8z9kl567w	elk_kibana	replicated	1/1	docker.elastic.co/kibana/kibana:8.5.1	*:5601->5601/tcp
z874px7i5teh	elk_logstash	replicated	2/2	docker.elastic.co/logstash/logstash:8.5.1	*:12201->12201/udp
q1hf7t69rh36	hasher	replicated	12/10	kennex666/hasher:coinswarm	*:8002->80/tcp
zosx4fjz6nn	monitoring_grafana	replicated	1/1	grafana/grafana:latest	*:3000->3000/tcp
n4cn19fb3468	monitoring_influxdb	replicated	1/1	influxdb:latest	*:8086->8086/tcp
latovtoiv9rr	monitoring_prometheus	replicated	1/1	prom/prometheus:latest	*:9090->9090/tcp
kgf89iwbtc4u	monitoring_telegraf	global	8/8	telegraf:latest	
e0hlrksu2skm	proxy_reverse-proxy	replicated	1/1	nginx:latest	*:80->80/tcp
mxkj03wtn17q	redis	replicated	3/2	redis:latest	
kyalw992745o	rng	replicated	16/10	kennex666/rng:coinswarm	*:8001->80/tcp
30j4u0rjfro9	sub_monitoring_cadvisor	global	9/8	gcr.io/cadvisor/cadvisor:latest	
m6duvd9x81ia	sub_monitoring_node-exporter	global	9/8	prom/node-exporter:latest	
5gxbh4qft4oql	webui	replicated	8/5	kennex666/webui:coinswarm	*:8000->80/tcp
9ccmz61b895a	worker	replicated	25/15	kennex666/worker:coinswarm	

c. Kiểm tra log của các service để đảm bảo các service chạy ổn định

- Prometheus

sudo docker service logs monitoring_prometheus

- Telegraf

sudo docker service logs monitoring_telegraf

- Influxdb

sudo docker service logs monitoring_influxdb

3. Giới thiệu các metric được thu thập trên toàn bộ các node

CPU, RAM (active hoặc inactive memory), disk usage trên tất cả các node

- Nguồn: Metric từ node-exporter
- Metric:
 - CPU: `node_cpu_seconds_total` (thời gian CPU theo mode: user, system, idle, v.v.)
 - RAM (Active/Inactive): `node_memory_Active_bytes`, `node_memory_Inactive_bytes`
 - Disk Usage: `node_filesystem_size_bytes`, `node_filesystem_free_bytes`

Tất cả các process và state của chúng

- Nguồn: Metric từ node-exporter
- Metric:
 - Số process đang chạy: `node_procs_running`
 - Số process bị block: `node_procs_blocked`

Số lượng file đang mở, sockets và trạng thái của chúng

- Nguồn: Metric từ node-exporter
- Metric:
 - Số file descriptor đang mở: `node_filefd_allocated`
 - Số socket: `node_sockstat_sockets_used`

Các hoạt động của I/O (disk, network), trên node hoặc tác động dung lượng (volume)

- Nguồn: node-exporter và cadvisor
- Metric:
 - Disk I/O: `node_disk_read_bytes_total`, `node_disk_written_bytes_total`
 - Network I/O: `node_network_receive_bytes_total`, `node_network_transmit_bytes_total`
 - Volume (từ cadvisor): `container_fs_usage_bytes`

Phản cứng vật lý (nếu có thể): fan speed, CPU temperature, v.v.

- Nguồn: node-exporter (phụ thuộc vào phần cứng hỗ trợ)
- Metric:
 - Nhiệt độ CPU: `node_hwmon_temp_celsius`
 - Tốc độ quạt: `node_hwmon_fan_rpm`

4. Hiển thị các thông tin metric trên cụm như mục trên bằng lệnh CLI

5. Trình diễn cơ chế tạo pipeline thu thập và lưu trữ các metric với Prometheus với CLI

Bước 1: Thu thập metric từ node:

- node-exporter chạy trên mỗi node (vps0, vps1, vps2,...) để thu thập metric hệ thống như CPU (node_cpu_seconds_total), RAM (node_memory_Active_bytes), disk (node_filesystem_size_bytes), v.v.
- cadvisor chạy trên mỗi node để thu thập metric container như dung lượng volume (container_fs_usage_bytes).
- Cả hai cung cấp metric qua endpoint HTTP (:9100 cho node-exporter, :8080 cho cadvisor).

Bước 2: Prometheus thu thập metric:

- Prometheus sử dụng DNS Service Discovery (dns_sd_configs) để tự động phát hiện các instance của node-exporter và cadvisor trên 3 node.
- Prometheus truy vấn (scrape) các endpoint này mỗi 15 giây (scrape_interval: 15s) và lưu trữ metric dưới dạng time-series.

Bước 3: Telegraf đồng bộ dữ liệu

- Telegraf truy vấn endpoint /metrics của Prometheus (http://monitoring_prometheus:9090/metrics) để lấy toàn bộ metric đã thu thập.
- Telegraf sử dụng plugin outputs.influxdb_v2 để ghi dữ liệu vào InfluxDB (bucket mybucket, tổ chức myorg).

Thực hiện trên CLI:

- Kiểm tra metric từ node-exporter
curl <http://vps0:9100/metrics>

```
osboxes@vps0:~$ curl http://vps0:9100/metrics
# HELP go_gc_duration_seconds A summary of the wall-time pause (stop-the-world) duration in garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 4.6623e-05
go_gc_duration_seconds{quantile="0.25"} 5.9234e-05
go_gc_duration_seconds{quantile="0.5"} 6.3385e-05
go_gc_duration_seconds{quantile="0.75"} 7.1326e-05
go_gc_duration_seconds{quantile="1"} 0.005884956
go_gc_duration_seconds_sum 0.021453407
go_gc_duration_seconds_count 183
# HELP go_gc_gogc_percent Heap size target percentage configured by the user, otherwise 100. This value is set by the GOGC environment variable, and the runtime/debug.SetGCPercent function. Sourced from /gc/gogc:percent
# TYPE go_gc_gogc_percent gauge
go_gc_gogc_percent 100
# HELP go_gc_gomemlimit_bytes Go runtime memory limit configured by the user, otherwise math.MaxInt64. This value is set by the GOMEMLIMIT environment variable, and the runtime/debug.SetMemoryLimit function. Sourced from /gc/gomemlimit:bytes
# TYPE go_gc_gomemlimit_bytes gauge
go_gc_gomemlimit_bytes 9.223372036854776e+18
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 8
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.23.7"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated in heap and currently in use. Equals to /memory/classes/heap/objects:bytes.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.079024e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated in heap until now, even if released already. Equals to /gc/heap/allocs:bytes.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 3.13279128e+08
```

- Kiểm tra Prometheus thu thập dữ liệu, thực hiện kiểm tra với metric trên CPU là `node_cpu_seconds_total`

curl http://vps0:9090/api/v1/query?query=node_cpu_seconds_total

```
osboxes@vps0:~$ curl http://192.168.19.101:9090/api/v1/query?query=node_cpu_seconds_total
{"status":"success","data":{"resultType":"vector","result":[{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.5:9100","job":"node-exporter","mode":"idle"},"value":[1744640646.748,"156.44"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.5:9100","job":"node-exporter","mode":"iowait"},"value":[1744640646.748,"0.93"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.5:9100","job":"node-exporter","mode":"irq"},"value":[1744640646.748,"0"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.5:9100","job":"node-exporter","mode":"nice"},"value":[1744640646.748,"0"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.5:9100","job":"node-exporter","mode":"softirq"},"value":[1744640646.748,"1109.12"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.5:9100","job":"node-exporter","mode":"steal"},"value":[1744640646.748,"0"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.5:9100","job":"node-exporter","mode":"system"},"value":[1744640646.748,"359.17"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.5:9100","job":"node-exporter","mode":"user"},"value":[1744640646.748,"260.69"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.14:9100","job":"node-exporter","mode":"idle"},"value":[1744640646.748,"81.53"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.14:9100","job":"node-exporter","mode":"iowait"},"value":[1744640646.748,"0.64"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.14:9100","job":"node-exporter","mode":"irq"},"value":[1744640646.748,"0"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.14:9100","job":"node-exporter","mode":"nice"},"value":[1744640646.748,"0"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.14:9100","job":"node-exporter","mode":"softirq"},"value":[1744640646.748,"1166.43"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.14:9100","job":"node-exporter","mode":"steal"},"value":[1744640646.748,"0"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.14:9100","job":"node-exporter","mode":"system"},"value":[1744640646.748,"383.64"]}, {"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"10.0.4.14:9100","job":"node-exporter","mode":"user"},"value":[1744640646.748,"240.25"]}]}]} osboxes@vps0:~$
```

6. Đồng bộ hóa dữ liệu từ Prometheus với InfluxDB

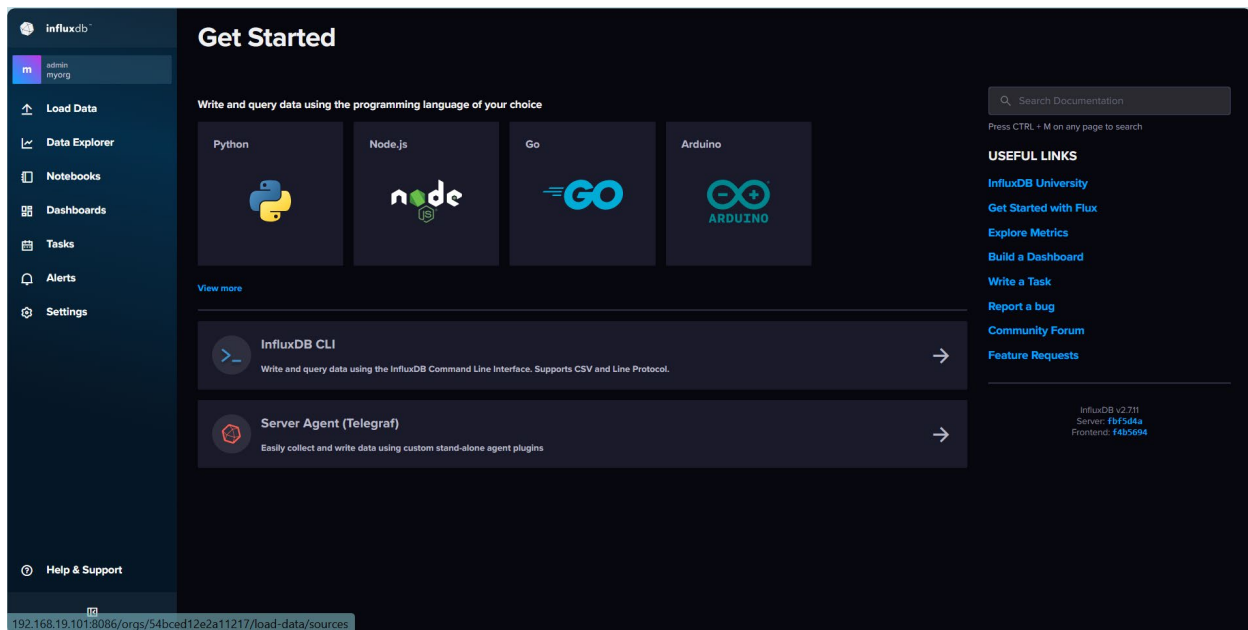
Cơ chế: Dùng telegraf thu thập metric từ Prometheus qua endpoint `/metrics` và ghi vào InfluxDB bằng plugin `outputs.influxdb_v2`. Cấu hình telegraf đã được viết ở trên.

7. Truy vấn dữ liệu metrics được thu thập được lưu trữ trong InfluxDB chạy trên swarm

Ở đây nhóm sử dụng UI để thực hiện truy vấn

Truy cập vào <http://192.168.19.10:8086>

Nhập username và password (đã cấu hình trên `docker-compose.yml`)



Vào Data Explorer, chọn mybucket, chọn 1 metric để truy vấn, ở đây ví dụ là `node_cpu_seconds_total`

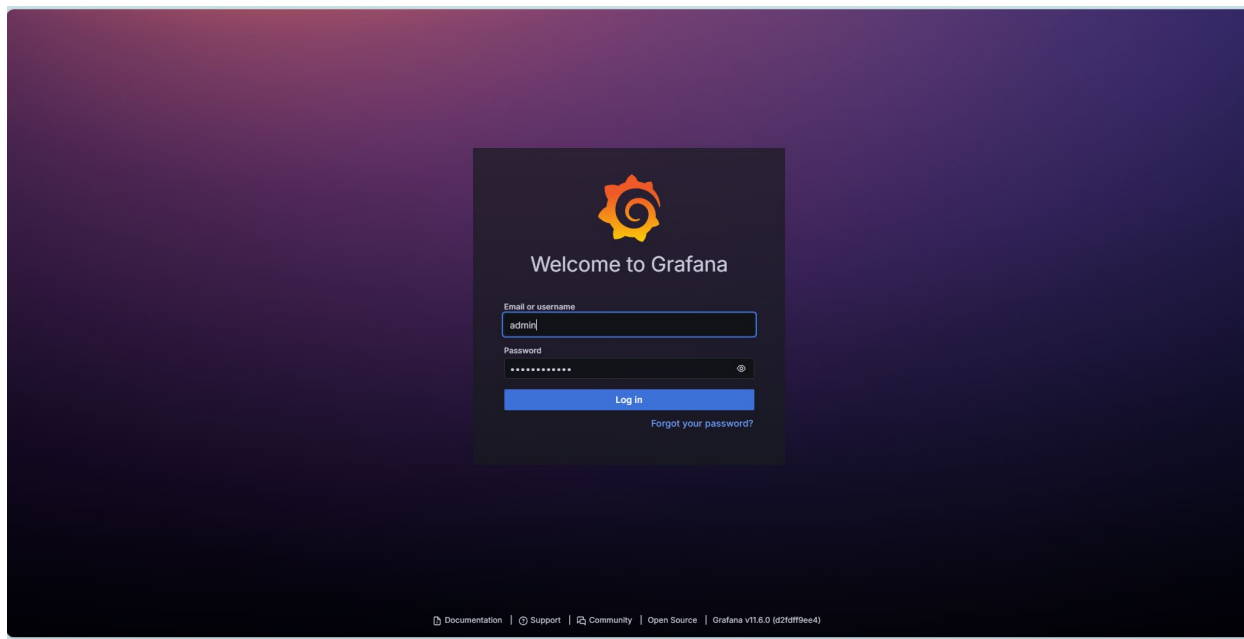
8. Tạo Dashboard sử dụng Prometheus/Grafana để phân tích , giám sát hiệu suất swarm

Kiểm tra service grafana

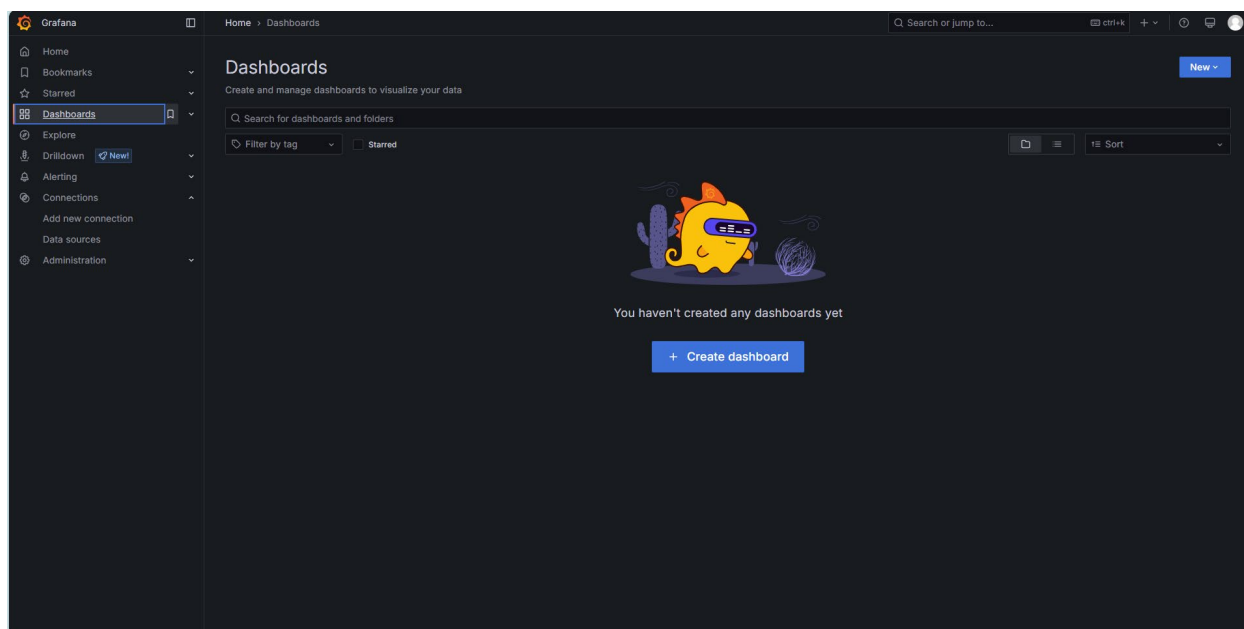
`sudo docker service ls`

```
revxtmmwyuy  monitoring_grafana      replicated  2/1      grafana/grafana:latest  *:3000
```

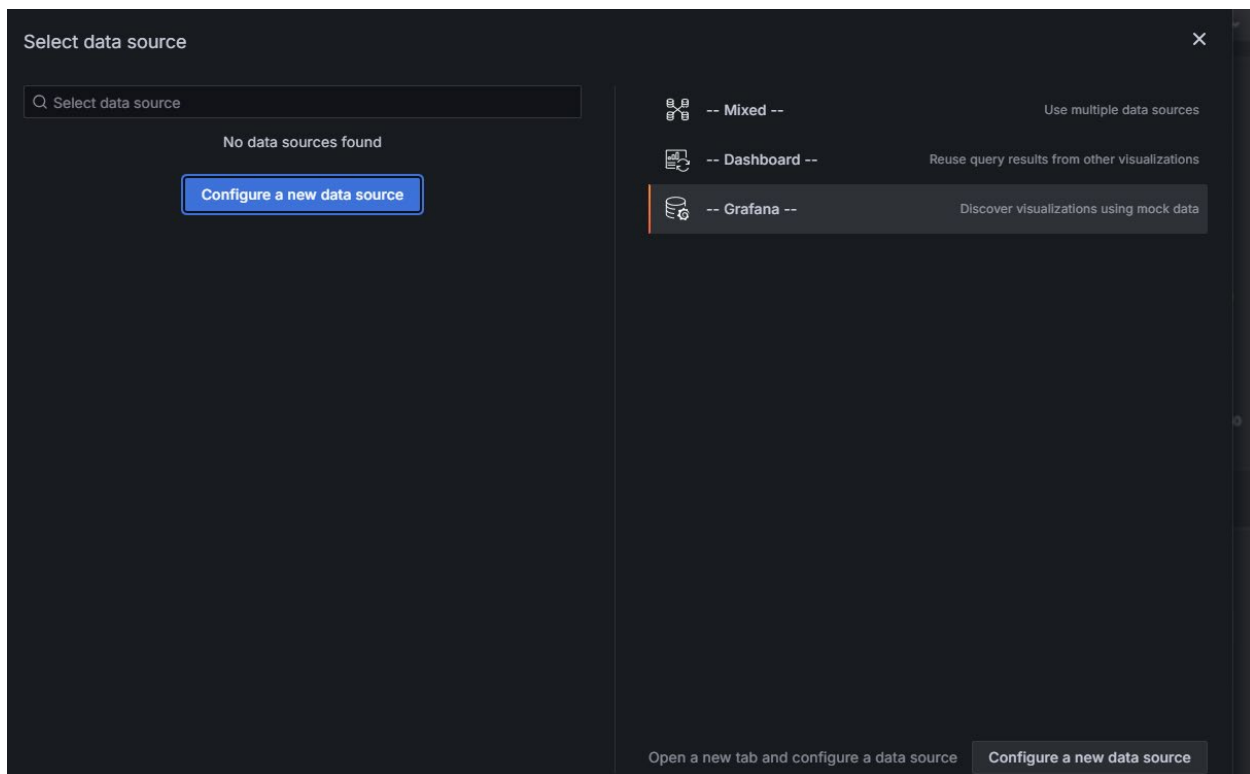
Truy cập vào `192.168.19.10:3000`, đăng nhập bằng username và password đã được cấu hình trong `docker-compose.yml`



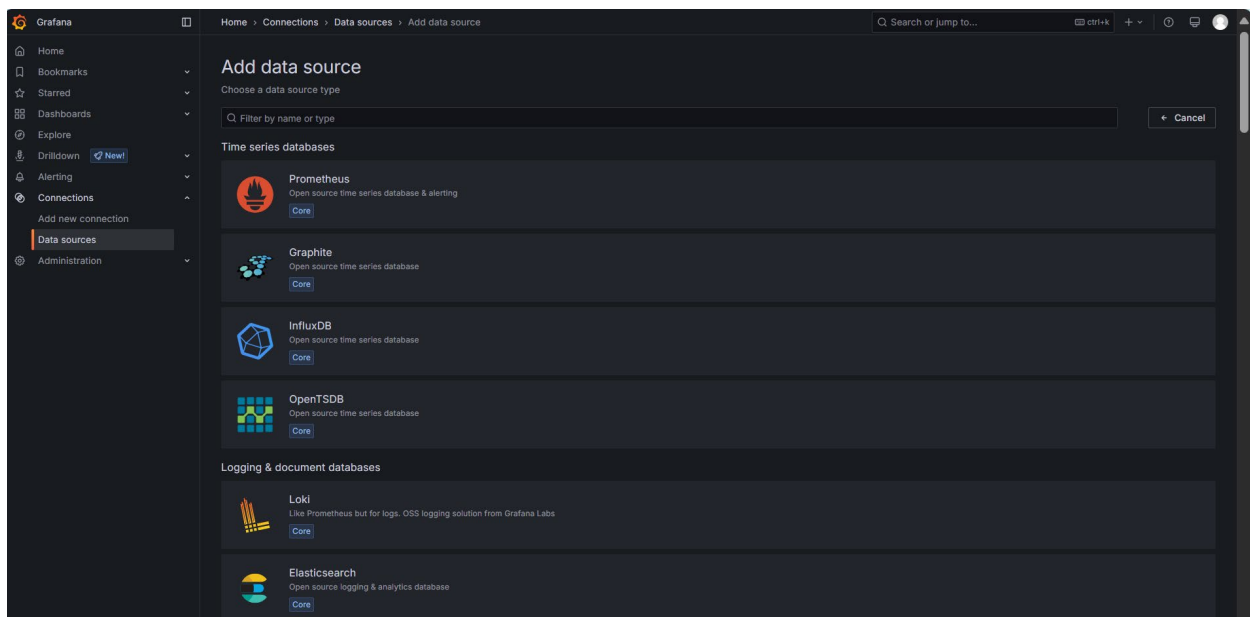
Vào Dashboards, chọn create dashboard



Cấu hình data source cho grafana



Chọn prometheus



Nhập URL của prometheus vào


Connection

Prometheus server URL *

Sau đó save&test, nếu thành công, quay về và chọn source là prometheus

Select data source

Q Select data source

 prometheus **default** Prometheus

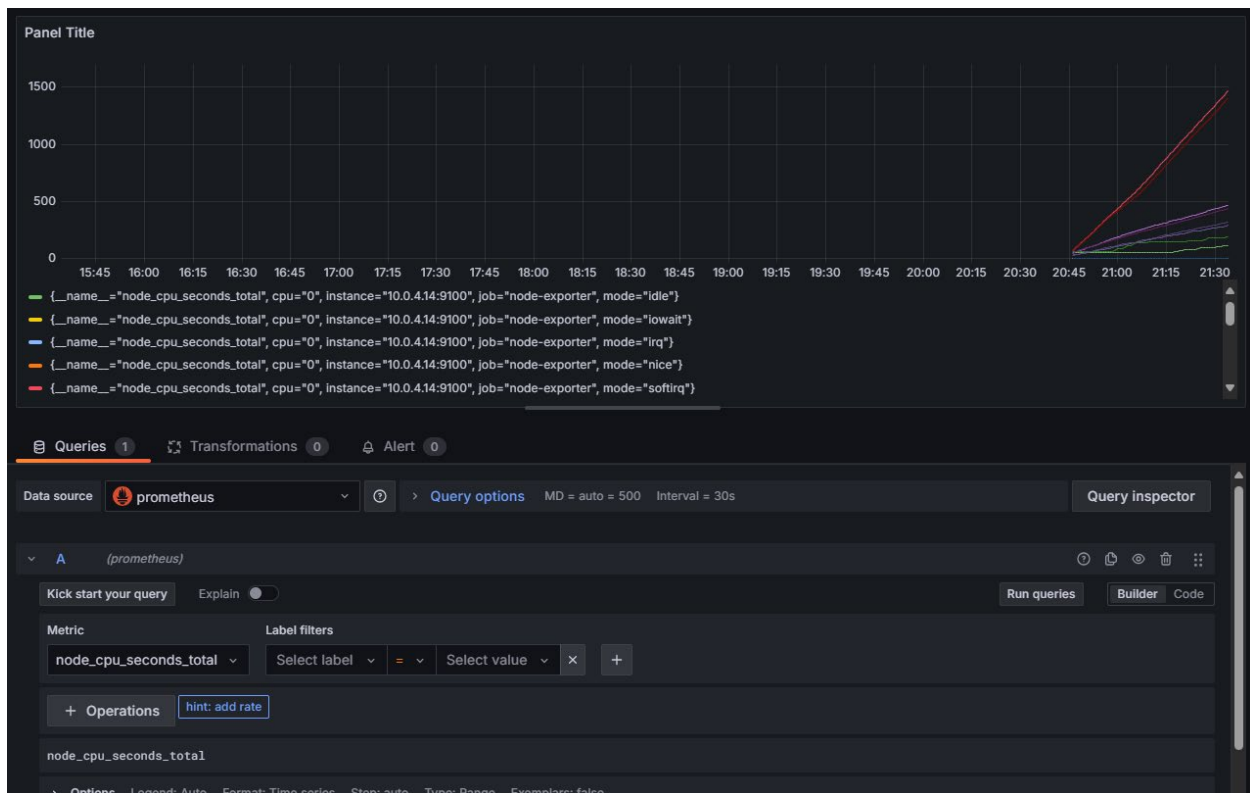
-- Mixed -- Use multiple data sources

-- Dashboard -- Reuse query results from other visualizations

-- Grafana -- Discover visualizations using mock data

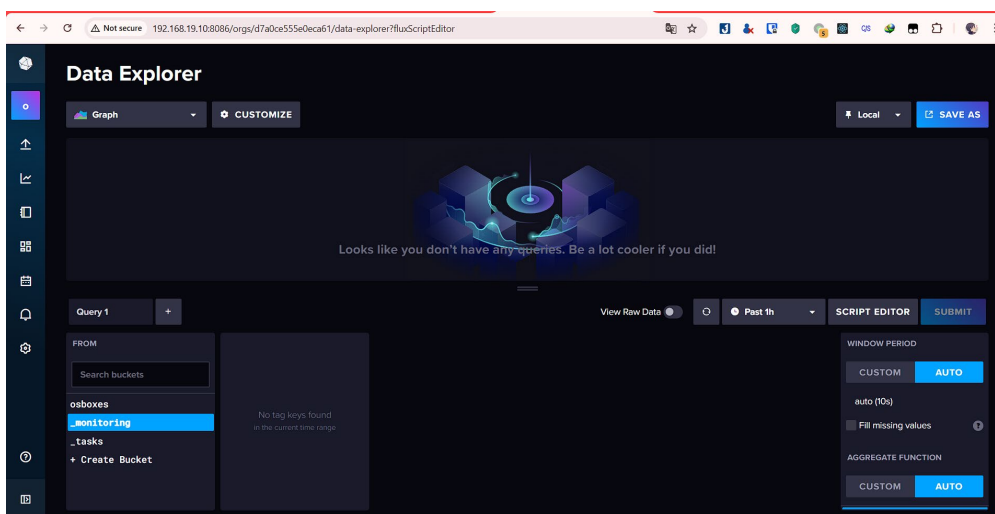
Open a new tab and configure a data source [Configure a new data source](#)

Sau đó thực hiện query, ví dụ query metric `node_cpu_seconds_total`



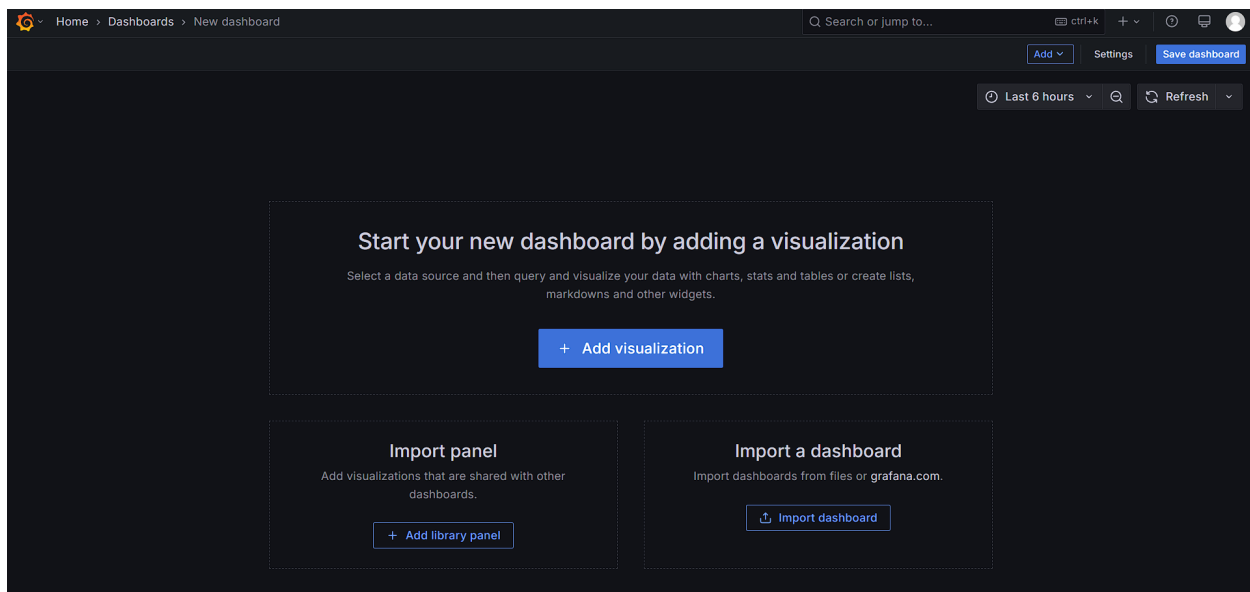
Config để lấy dữ liệu sang InfluxDB:

Truy cập <http://192.168.19.10:8086/> (admin/admin123)

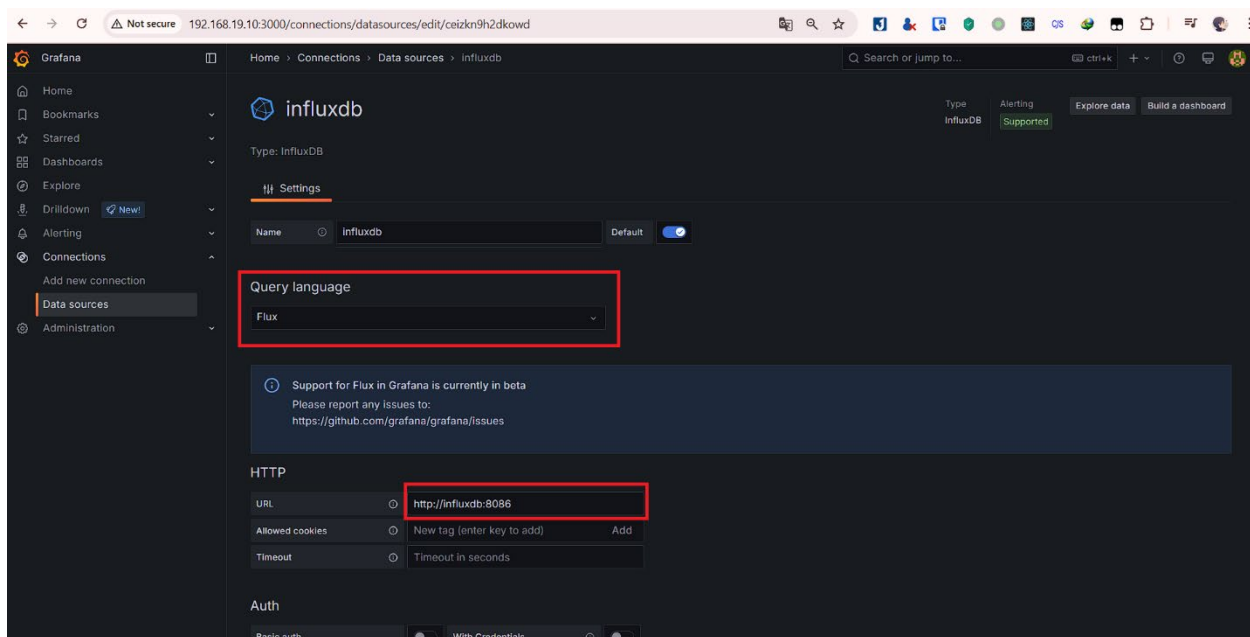


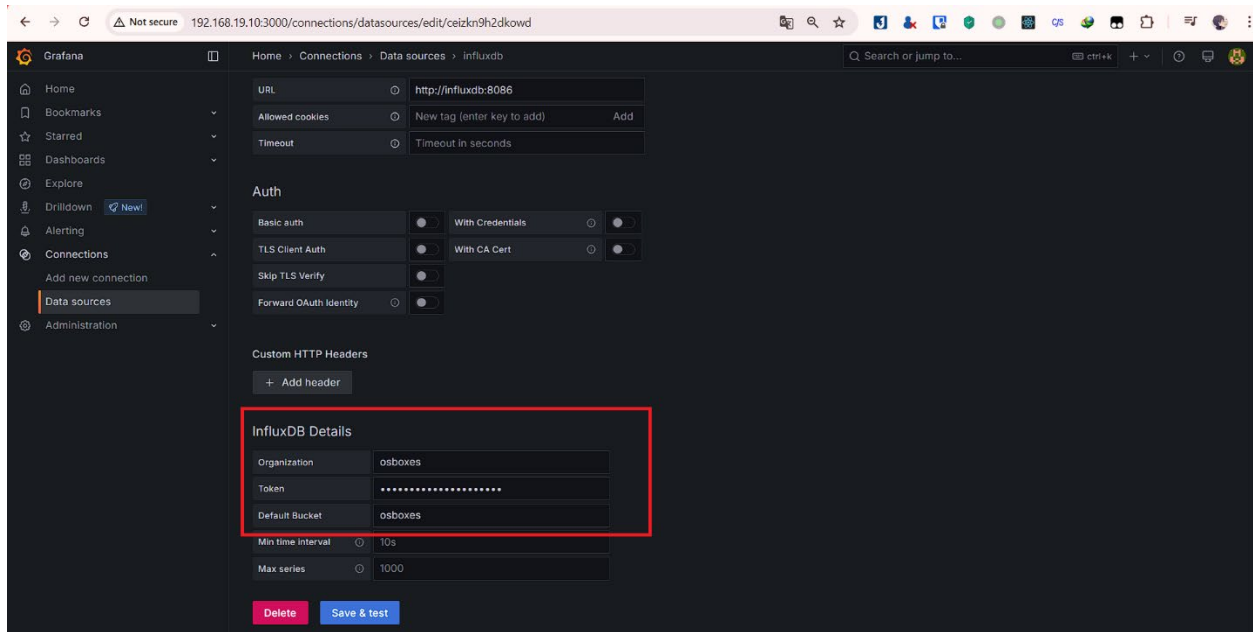
Vào data source tạo mới token.

Truy cập Grafana: <http://192.168.19.10:3000>



Vào connections -> data source, điền thông tin vào các ô khoanh đỏ.





Setup hoàn tất, ta có thể import dashboard bằng file json sau:

```
{
  "title": "Telegraf Node Overview (Flux)",
  "timezone": "browser",
  "panels": [
    {
      "id": 0,
      "gridPos": {
        "h": 8,
        "w": 12,
        "x": 0,
        "y": 0
      },
      "type": "timeseries",
      "title": "CPU Usage (%)",
      "targets": [
        {
          "datasource": {
```

```

    "type": "influxdb",
    "uid": "-"
  },
  "query": "from(bucket: \"osboxes\") |> range(start: -1h) |> filter(fn: (r) =>
r._measurement == \"cpu\" and r._field == \"usage_system\" and r.cpu == \"cpu-
total\")",
  "refId": "A"
}
]
},
{
  "id": 1,
  "gridPos": {
    "h": 8,
    "w": 12,
    "x": 12,
    "y": 0
  },
  "type": "timeseries",
  "title": "Memory Used (%)",
  "targets": [
    {
      "datasource": {
        "type": "influxdb",
        "uid": "-"
      },
      "query": "from(bucket: \"osboxes\") |> range(start: -1h) |> filter(fn: (r) =>
r._measurement == \"mem\" and r._field == \"used_percent\")",
      "refId": "B"
    }
  ]
}
]
}

```

```

    }
  ]
},
{
  "id": 2,
  "gridPos": {
    "h": 8,
    "w": 12,
    "x": 0,
    "y": 8
  },
  "type": "timeseries",
  "title": "Swap Used (%)",
  "targets": [
    {
      "datasource": {
        "type": "influxdb",
        "uid": "-"
      },
      "query": "from(bucket: \"osboxes\") |> range(start: -1h) |> filter(fn: (r) =>
r._measurement == \"swap\" and r._field == \"used_percent\")",
      "refId": "C"
    }
  ]
},
{
  "id": 3,
  "gridPos": {
    "h": 8,

```

```

    "w": 12,
    "x": 12,
    "y": 8
  },
  "type": "timeseries",
  "title": "Disk Used (%)",
  "targets": [
    {
      "datasource": {
        "type": "influxdb",
        "uid": "-"
      },
      "query": "from(bucket: \"osboxes\") |> range(start: -1h) |> filter(fn: (r) =>
r._measurement == \"disk\" and r._field == \"used_percent\" and r.path == \"/\")",
      "refId": "D"
    }
  ]
},
{
  "id": 4,
  "gridPos": {
    "h": 8,
    "w": 12,
    "x": 0,
    "y": 16
  },
  "type": "timeseries",
  "title": "System Uptime (s)",
  "targets": [

```

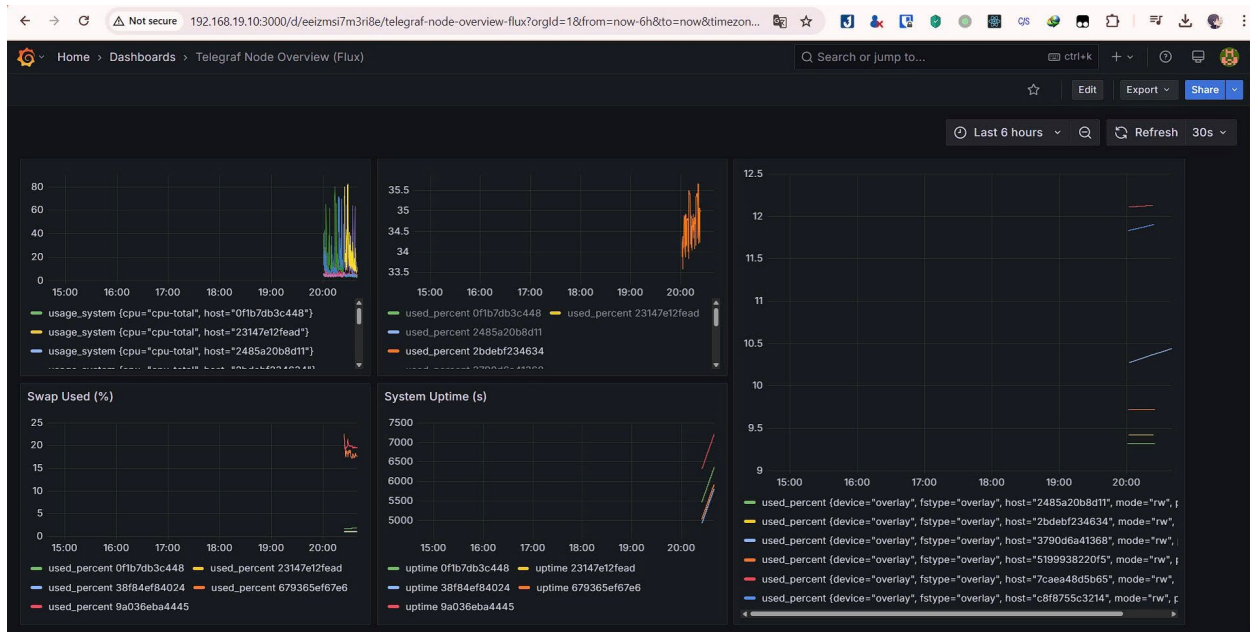


```

{
  "datasource": {
    "type": "influxdb",
    "uid": "-"
  },
  "query": "from(bucket: \"osboxes\") |> range(start: -1h) |> filter(fn: (r) =>
r._measurement == \"system\" and r._field == \"uptime\")",
  "refId": "E"
}
]
},
"templating": {
  "list": []
},
"annotations": {
  "list": []
},
"schemaVersion": 30,
"version": 1,
"refresh": "30s"
}

```

Kết quả sau khi import dashboard:



PHẦN 7: REVERSE PROXY SỬ DỤNG NGINX

1. Giới thiệu về Reverse Proxy

Reverse Proxy là một kỹ thuật quan trọng trong kiến trúc hệ thống phân tán, đóng vai trò trung gian giữa client và các dịch vụ backend. Không giống như Forward Proxy (đại diện cho client để truy cập internet), Reverse Proxy nhận yêu cầu từ client, chuyển tiếp chúng đến các máy chủ backend phù hợp, sau đó trả kết quả về client mà không để lộ cấu trúc nội bộ của hệ thống. Điều này mang lại nhiều lợi ích thiết yếu cho các hệ thống hiện đại.

Đầu tiên, Reverse Proxy giúp cân bằng tải (load balancing) bằng cách phân phối yêu cầu đến nhiều máy chủ backend, đảm bảo không có máy chủ nào bị quá tải, từ đó tăng hiệu suất và độ tin cậy. Thứ hai, nó tăng cường bảo mật bằng cách ẩn đi thông tin của các máy chủ backend, đồng thời có thể tích hợp các cơ chế như xác thực, mã hóa SSL/TLS, hoặc chặn các yêu cầu độc hại. Cuối cùng, Reverse Proxy hỗ trợ quản lý tập trung, cho phép định tuyến (routing) yêu cầu dựa trên URL, header, hoặc các tiêu chí khác, giúp đơn giản hóa việc quản lý nhiều dịch vụ trong một hệ thống phân tán.

2. Giới thiệu về Nginx

Nginx là một máy chủ web và Reverse Proxy mã nguồn mở, được phát triển bởi Igor Sysoev vào năm 2004, và hiện là một trong những công cụ phổ biến nhất trong lĩnh vực này. Ban đầu được thiết kế để giải quyết vấn đề hiệu suất của các máy chủ web truyền thống, Nginx nổi bật với khả năng xử lý hàng nghìn kết nối đồng thời với hiệu suất cao, nhờ kiến trúc bất đồng bộ (asynchronous) và hướng sự kiện (event-driven). Ngoài vai trò là máy chủ web, Nginx còn được sử dụng rộng rãi như một Reverse Proxy, cân bằng tải, và bộ đệm (caching) nhờ tính linh hoạt và hiệu quả của nó.

Nginx hỗ trợ cấu hình dễ dàng thông qua file cấu hình `nginx.conf`, cho phép định nghĩa các quy tắc định tuyến, proxy, và xử lý yêu cầu một cách chi tiết. Công cụ này cũng tích hợp tốt với các hệ thống container và môi trường phân tán như Docker Swarm, nhờ khả năng khám phá dịch vụ và xử lý các yêu cầu động. Với cộng đồng người dùng lớn và

tài liệu phong phú, Nginx là lựa chọn lý tưởng để triển khai Reverse Proxy trong các hệ thống hiện đại, đặc biệt khi cần quản lý nhiều dịch vụ với các yêu cầu định tuyến phức tạp.

3. Ý tưởng thiết lập Reverse Proxy

Tất cả các dịch vụ phía sau reverse proxy sẽ được truy cập như sau:

Đường dẫn	Chuyển tiếp đến
/grafana	grafana:3000
/influxdb	influxdb:8086
/prometheus	prometheus:9090
/webui	webui:8000
/rng	rng:8001
/hasher	hasher:8002
/kibana	kibana: 5601

4. Triển khai Reverse Proxy với Nginx

Bước 1: Ta cần tạo một nginx.conf có nội dung như sau:

```
events {}

http {
    server {
        listen 80;

        location /grafana/ {
            proxy_pass http://grafana:3000/;
            rewrite ^/grafana/?(.*)$ /$1 break;

            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```

        proxy_set_header X-Forwarded-Proto $scheme;
    }
    location /influxdb/ {
        rewrite ^/influxdb(/.*)$ $1 break;
        return 301 http://192.168.19.10:8086$1;
    }
    location /prometheus/ {
        proxy_pass http://prometheus:9090/;
        rewrite ^/prometheus/(.*)$ /$1 break;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /webui/ {
        proxy_pass http://webui:80/;
        rewrite ^/webui/(.*)$ /$1 break;
    }

    location /rng/ {
        proxy_pass http://rng:80/;
        rewrite ^/rng/(.*)$ /$1 break;
    }

    location /hasher/ {
        proxy_pass http://hasher:80/;
        rewrite ^/hasher/(.*)$ /$1 break;
    }

```

```

    }
    location /kibana/ {
        proxy_pass http://kibana:5601/;
        rewrite ^/kibana/?(.*)$ /$1 break;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }
}
}

```

Bước 2: Tạo một file docker compose để deploy lên swarm:

version: '3.8'

services:

reverse-proxy:

image: nginx:latest

ports:

- "80:80"

configs:

- source: nginx_conf

target: /etc/nginx/nginx.conf

networks:

- proxy

- *monitoring_network*
- *elk_elk_network*
- *coinswarmnet*

configs:

nginx_conf:

file: ./nginx.conf

networks:

proxy:

driver: overlay

attachable: true

monitoring_network:

external: true

elk_elk_network:

external: true

coinswarmnet:

external: true

Bước 3: Thực thi lệnh bên dưới để deploy:

sudo docker config create nginx_conf ./nginx.conf

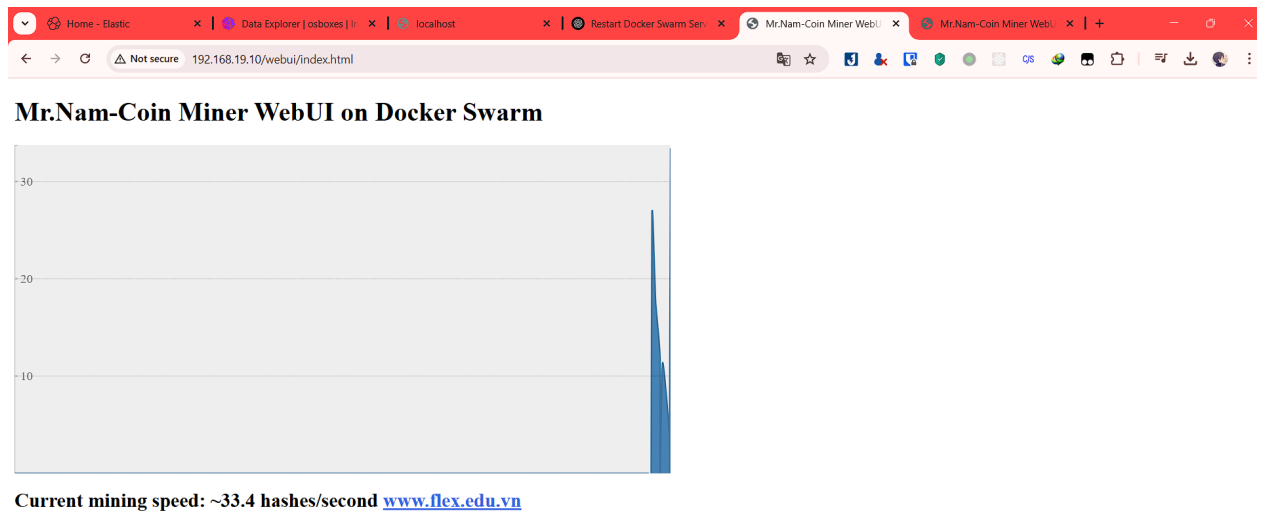
sudo docker stack deploy -c docker-compose.yml proxy

Sau khi deploy thành công, ta có thể truy cập vào một số trang như:

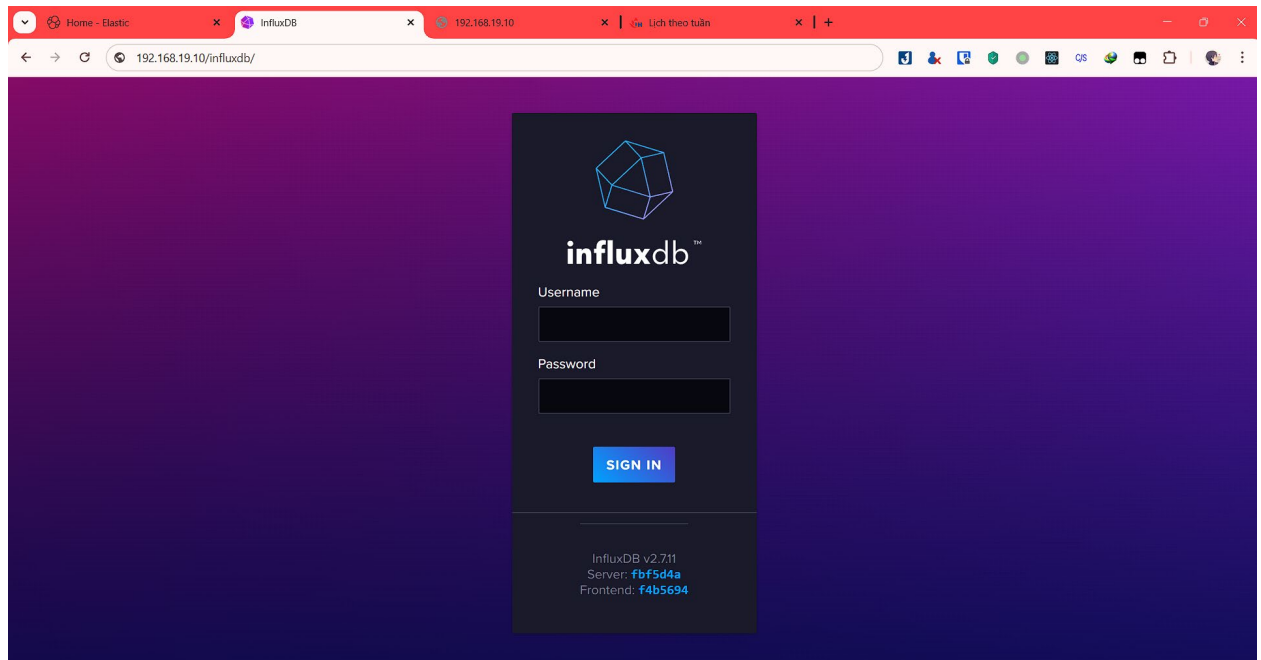
Hasher:



Webui:



InfluxDB:



Một số dịch vụ có thể cần phải config lại base path để hoạt động, đối với dịch vụ không hỗ trợ base path sẽ gây lỗi không truy cập được, ta có hướng giải quyết cơ bản:

- Có thể xem xét việc mở cổng mới qua reverse proxy
- Có thể điều hướng đến trang gốc

KẾT LUẬN

Qua quá trình thực hiện đề tài “**Xây dựng và giám sát hệ thống Microservices trên nền tảng Docker Swarm**”, nhóm đã có cơ hội tiếp cận và trải nghiệm thực tế với nhiều công nghệ hiện đại đang được ứng dụng rộng rãi trong lĩnh vực dữ liệu lớn và điện toán đám mây.

Từ việc xây dựng hạ tầng mô phỏng bằng VirtualBox đến triển khai các cụm máy Docker Swarm và hệ thống giám sát đi kèm, đề tài đã giúp nhóm rèn luyện và tích lũy được nhiều kiến thức và kỹ năng quan trọng, cụ thể như sau:

- Hiểu rõ cơ chế hoạt động của các chế độ mạng trong VirtualBox, từ đó biết cách cấu hình phù hợp để tạo môi trường mô phỏng cụm máy thực tế.
- Làm chủ quá trình cài đặt và triển khai Docker Swarm, quản lý node master/worker, hiểu về overlay network, HA (High Availability), Load Balancing và Auto Scaling trong môi trường phân tán.
- Thực hành triển khai và đánh giá hiện tượng bottleneck trong hệ thống Microservices, đồng thời biết cách scale dịch vụ để tối ưu hiệu năng.
- Thiết lập hệ thống giám sát hiện đại bằng Prometheus, InfluxDB và Grafana, giúp theo dõi tài nguyên hệ thống theo thời gian thực và hỗ trợ việc phân tích – trực quan hóa hiệu suất hệ thống.
- Cài đặt và cấu hình hệ thống thu thập và phân tích log bằng ELK Stack (Elasticsearch – Logstash – Kibana), từ đó xây dựng một quy trình hoàn chỉnh để kiểm soát và xử lý log trong hệ thống phân tán.
- Biết sử dụng dòng lệnh (CLI) để thao tác và kiểm tra pipeline thu thập – lưu trữ – truy vấn metric/log, giúp tăng tính linh hoạt khi vận hành hệ thống trong thực tế.
- Sử dụng nginx để tạo ra reverse proxy.

Đề tài không chỉ mang tính chất kỹ thuật mà còn rèn luyện cho nhóm khả năng làm việc nhóm, phân chia công việc, giải quyết sự cố và báo cáo kết quả. Đây là một trải nghiệm quý giá giúp nhóm làm quen với cách làm việc trong môi trường doanh nghiệp thực tế, chuẩn bị hành trang vững vàng cho các dự án lớn sau này.