

Cài đặt Control Plane và Worker node:

Dưới đây là hướng dẫn đầy đủ và được cập nhật 2024, có thể áp dụng cho Ubuntu 20.04, 22.04 (Jammy) và mới hơn:

1. Cài đặt công cụ hỗ trợ HTTPS cho APT

```
sudo apt-get update
```

```
sudo apt-get install -y apt-transport-https ca-certificates curl gpg git vim
```

2. Thêm khóa GPG và tạo source list đúng cách

```
sudo mkdir -p /etc/apt/keyrings
```

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | gpg --dearmor | sudo tee  
/etc/apt/keyrings/kubernetes-apt-keyring.gpg > /dev/null
```

```
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/  
/" | sudo tee /etc/apt/sources.list.d/kubernetes.list > /dev/null
```

3. Cập nhật và cài đặt Kubernetes:

```
sudo apt-get update
```

```
sudo apt-get install -y kubelet kubeadm kubectl
```

```
sudo apt-mark hold kubelet kubeadm kubectl
```

Kiểm tra sau khi cài đặt

```
kubectl version --client
```

```
kubeadm version
```

```
kubelet --version
```

Tiếp theo là triển khai Kubernetes cluster multi-node (1 control plane + 1 worker)

Bước 1: Cấu hình hệ thống (trên cả 2 VM Ubuntu)

Tắt swap:

Bật các kernel module cần thiết:

Cấu hình sysctl:

```
sudo swapoff -a
```

```
sudo sed -i 's/^/#/' /etc/fstab
```

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
```

```
br_netfilter
```

```
EOF
```

```
sudo modprobe br_netfilter
```

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
```

```
net.bridge.bridge-nf-call-iptables = 1
```

```
net.ipv4.ip_forward = 1
```

```
net.bridge.bridge-nf-call-ip6tables = 1
```

```
EOF
```

```
sudo sysctl --system
```

Bước 2: Cài container runtime (nếu chưa có)

Kubernetes cần một container runtime. Dùng containerd là ổn định và phổ biến nhất:

```
sudo apt-get install -y containerd
```

```
sudo mkdir -p /etc/containerd
```

```
containerd config default | sudo tee /etc/containerd/config.toml
```

Chỉ cần cấu hình để SystemdCgroup = true:

```
sudo vim /etc/containerd/config.toml
```

Tìm dòng:

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
```

Và chỉnh:

```
SystemdCgroup = true
```

Sau đó:

```
sudo systemctl restart containerd  
sudo systemctl enable containerd
```

Bước 3: Tạo cluster trên control plane node

Trên VM control plane:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

Dùng lệnh: `kubeadm join ...` dành cho worker. hoặc có thể lấy lại token này bằng lệnh:

```
sudo kubeadm token create --print-join-command
```

Sau đó cấu hình kubectl:

```
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Bước 4: Cài Flannel CNI (hoặc Calico, Cilium, etc.)

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

Bước 5: Thêm worker node vào cluster

Trên node worker, dùng lệnh `kubeadm join` mà bạn đã nhận từ bước 3. Ví dụ:

```
sudo kubeadm join 192.168.1.100:6443 --token abcdef.0123456789abcdef \  
--discovery-token-ca-cert-hash sha256:xxxxxxxxxxxxxxxxxxxxxx
```

Kiểm tra cluster (trên control plane)

```
kubectl get nodes
```

Mặc định, kubernetes không cho phép chạy pod trên node control plane, để cho phép pod chạy trên node control-plane (dành cho môi trường học tập, lab, máy ảo test)

Chạy lệnh sau để xóa taint:

```
kubectl taint nodes <node-name> node-role.kubernetes.io/control-plane-
```

Ví dụ, nếu node bạn tên là control-plane, thì:

```
kubectl taint nodes control-plane node-role.kubernetes.io/control-plane-
```

Dấu - ở cuối lệnh là để xóa taint.

Cài đặt Ingress Control (NodePort) cho Kubernetes

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.10.0/deploy/static/provider/baremetal/deploy.yaml>

Kiểm tra Ingress Control

kubectl get svc -n ingress-nginx

Hoặc

kubectl get service --all-namespaces

Xác định ingressClass để cấu hình cho ingress service

kubectl get ingressclass

Kiểm tra cấu hình của một Ingress

kubectl get ingress ingress-app1 -o yaml

kubectl get ingress rancher -n cattle-system -o yaml

Kiểm tra logs Ingress

kubectl logs -n ingress-nginx -l app.kubernetes.io/name=ingress-nginx

Dùng forward để kiểm tra một ứng dụng trong pod với Cluster-IP:

kubectl port-forward svc/cluster-ip-app1 18080:8080

Sau khi forward có thể kiểm tra ứng dụng thông qua localhost:18080

Cài đặt, config và kiểm tra cert-manager trong k8s

Bước 1: Cài cert-manager

`kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.18.1/cert-manager.yaml`

Xem pod certificate

`kubectl get pods -n cert-manager`

Giảm ndots cho pod cert-manager trước khi thực hiện bước tiếp theo (nhằm tránh việc phân giải dns sai do ndots cao):

Trong Kubernetes, khi pod chạy, nó sẽ inherit ndots từ /etc/resolv.conf của node hoặc từ config CoreDNS → nhưng ta có thể **override cho từng pod** qua: `dnsConfig.options` trong Deployment

Cụ thể — ví dụ chỉnh Deployment của cert-manager:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cert-manager
  namespace: cert-manager
spec:
  template:
    spec:
      dnsPolicy: ClusterFirst
      dnsConfig:
        options:
          - name: ndots
            value: "1"
```

Với cert-manager — bạn chỉ cần làm:

`kubectl edit deployment -n cert-manager cert-manager`

→ Thêm đoạn:

```
spec:
  template:
    spec:
      dnsPolicy: ClusterFirst
      dnsConfig:
        options:
          - name: ndots
            value: "1"
```

→ Save lại → Kubernetes sẽ tự rollout lại pod cert-manager mới.

Hoặc:

Patch Deployment/cert-manager để set `dnsConfig.options.ndots: 1`

Lệnh cụ thể:

`kubectl patch deployment cert-manager -n cert-manager --type=json -p='[{"op": "add", "path": "/spec/template/spec/dnsConfig", "value": {"options": [{"name": "ndots", "value": "1"}]}]'`

Sau đó restart lại cert-manager:

`kubectl rollout restart deployment cert-manager -n cert-manager`

Bước 2: Tạo ClusterIssuer với [cluster-issuer.yaml](#):

`kubectl apply -f cluster-issuer.yaml`

Kiểm tra Issuer / ClusterIssuer

`kubectl get clusterissuer`

Phải thấy:

NAME	READY	AGE
letsencrypt-http01	True	10m

Bước 5: Kết quả mong muốn

Khi bạn truy cập <https://nexus.txuapp.com>:

Request sẽ qua modem NAT → Ubuntu → ingress controller → cert-manager sẽ xin cert từ Let's Encrypt và lưu trong Secret nexus-tls. Ingress controller tự dùng cert đó → HTTPS chuẩn production

Cài Rancher trong k8s

Bước 1: Thêm repo Helm Rancher

Rancher được cài với Helm chart.

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
helm repo add rancher-latest https://releases.rancher.com/server-charts/latest
helm repo update
```

Bước 2: Tạo namespace cho Rancher

```
kubectl create namespace cattle
```

Note: cài nfs-client (xem ở phần cấu hình và cài đặt grafana,...)

Bước 3: Cài đặt Rancher

```
helm install rancher rancher-latest/rancher \
--namespace cattle \
--create-namespace \
--set hostname=rancher.txuapp.com \
--set replicas=1 \
--set bootstrapPassword="Phan@123" \
--set ingress.tls.source=secret \
--set ingress.ingressClassName=nginx \
--set extraEnv[0].name=CATTLE_BOOTSTRAP_PASSWORD \
--set extraEnv[0].value="Phan@123" \
--set ingress.extraAnnotations."cert-manager.io/cluster-issuer"="letsencrypt-http01" \
--set persistence.enabled=true \
--set persistence.storageClass=nfs-client \
--set persistence.accessMode=ReadWriteOnce \
--set persistence.size=20Gi \
--wait \
--timeout 10m
```

Note:

- letsencrypt-http01: tên clusterIssuer của cert-manager
- nginx: tên class của ingress controller
- Với tham số cài đặt trên, một ingress với hostname “rancher.txuapp.com” sẽ được tạo ra và trỏ đến pod ứng dụng rancher. Ingress sẽ xin và dùng cert từ cert-manager được chỉ định.
- Lấy password mặc định (trong trường này là Phan@123 vì password được set khi cài đặt):

```
kubectl get secret -n rancher bootstrap-secret -o go-template='{ {.data.bootstrapPassword | base64decode } }'
```

Cài MinIO trong ubuntu (trên docker)

Bước 1: Cài đặt Docker trên Ubuntu

1.1. Cập nhật hệ thống:

```
sudo apt update  
sudo apt upgrade -y
```

1.2. Cài Docker từ Docker repository chính thức:

```
sudo apt install -y \  
ca-certificates \  
curl \  
gnupg \  
lsb-release
```

1.3. Thêm Docker GPG key:

```
sudo mkdir -p /etc/apt/keyrings  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \  
| sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

1.4. Thêm Docker repository:

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

1.5. Cài Docker Engine:

```
sudo apt update  
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

1.6. Thêm user vào nhóm docker (để không cần sudo):

```
sudo usermod -aG docker $USER  
newgrp docker # chạy lại phiên làm việc hoặc logout/login
```

Kiểm tra Docker hoạt động:

```
docker version  
docker run hello-world
```

Bước 2: Tạo thư mục dữ liệu MinIO

```
sudo mkdir -p /mnt/data/minio  
sudo useradd -r $USER -s /sbin/nologin  
sudo chown -R $USER:$USER /mnt/data/minio  
sudo chmod -R 750 /mnt/data/minio
```

Bước 3: Tạo service chạy minio

```
sudo vim /etc/systemd/system/minio.service
```

Nội dung:

[Unit]

Description=MinIO Subnet (Full features)

After=network.target docker.service

Requires=docker.service

[Service]

User=nhuy

Group=nhuy

ExecStartPre=/usr/bin/docker pull quay.io/minio/minio:RELEASE.2024-06-11T03-13-30Z

ExecStart=/usr/bin/docker run --rm \
--name minio \
-p 9000:9000 \
-p 9001:9001 \
minio/minio:RELEASE.2024-06-11T03-13-30Z

```
-e MINIO_ROOT_USER=admin \  
-e MINIO_ROOT_PASSWORD=Phan@123 \  
-e MINIO_IDENTITY_MANAGE=on \  
-e MINIO_PROMETHEUS_AUTH_TYPE=public \  
-e MINIO_BROWSER_REDIRECT_URL=https://console-minio.txuapp.com \  
-v /mnt/data/minio:/data \  
quay.io/minio/minio:RELEASE.2024-06-11T03-13-30Z \  
server /data --console-address ":9001"
```

Restart=always
RestartSec=5
LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

Chạy service:

```
sudo systemctl daemon-reexec  
sudo systemctl daemon-reload  
sudo systemctl enable --now minio
```

Khởi động lại service (khi cần)

```
sudo systemctl start docker  
sudo systemctl restart minio
```

Kiểm tra trạng thái của service:

```
sudo systemctl status minio
```

Tạo butket “jenkins-data” cho jenkins:

Tạo access key cho phép đăng nhập và sử dụng butket “jenkins-data”:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "s3:GetBucketLocation",  
        "s3:ListBucket"  
      ],  
      "Effect": "Allow",  
      "Resource": [  
        "arn:aws:s3:::jenkins-data"  
      ]  
    },  
    {  
      "Action": [  
        "s3:PutObject",  
        "s3:GetObject",  
        "s3:DeleteObject"  
      ],  
      "Effect": "Allow",  
      "Resource": [  
        "arn:aws:s3:::jenkins-data/*"  
      ]  
    }  
  ]  
}
```


Hoặc đơn giản:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:*"],
      "Resource": [
        "arn:aws:s3:::jenkins-data",
        "arn:aws:s3:::jenkins-data/*"
      ]
    }
  ]
}
```

Tạo secret chứa access key đăng nhập minio để gán cho pod sử dụng:

```
kubectl create secret generic minio-creds --from-literal=MINIO_ACCESS_KEY_ID=xxxx --from-literal=MINIO_SECRET_ACCESS_KEY=yyyy -n jenkins
```

Bước 1: Cài đặt NFS Server trên máy Ubuntu (máy chia sẻ volume)

Trên máy Ubuntu có IP ví dụ là 192.168.98.150:

```
sudo apt update
```

```
sudo apt install -y nfs-kernel-server
```

Tạo thư mục dùng để chia sẻ

```
sudo mkdir -p /mnt/data/nfs/jenkins
```

```
sudo chown -R nobody:nogroup /mnt/data/nfs/jenkins
```

```
sudo chmod 777 /mnt/data/nfs/jenkins
```

Cấu hình chia sẻ trong /etc/exports

```
sudo vim /etc/exports
```

Thêm dòng sau:

```
/mnt/data/nfs/jenkins 192.168.98.0/24(rw,sync,no_subtree_check,no_root_squash)
```

Thay 192.168.98.0/24 bằng subnet mạng Kubernetes hoặc IP của node nếu cần chính xác hơn.

Áp dụng cấu hình:

```
sudo exportfs -ra
```

```
sudo systemctl restart nfs-kernel-server
```

Bước 2: Cài đặt NFS Client trên tất cả node của Kubernetes cluster

Trên mỗi node (cả control-plane và worker):

```
sudo apt update
```

```
sudo apt install -y nfs-common
```

Bước 3: Tạo PersistentVolume (PV) và PersistentVolumeClaim (PVC) trong Kubernetes

nfs-pv.yaml:

```
apiVersion: v1
```

```
kind: PersistentVolume
```

```
metadata:
```

```
  name: jenkins-nfs-pv
```

```
spec:
```

```
  capacity:
```

```
    storage: 10Gi
```

```
  accessModes:
```

```
    - ReadWriteMany
```

```
  nfs:
```

```
    server: 192.168.98.150
```

```
    path: /mnt/data/nfs/jenkins
```

```
  persistentVolumeReclaimPolicy: Retain
```

nfs-pvc.yaml:

```
apiVersion: v1
```

```
kind: PersistentVolumeClaim
```

```
metadata:
```

```
  name: jenkins-nfs-pvc
```

```
  namespace: jenkins
```

```
spec:
```

```
  accessModes:
```

```
    - ReadWriteMany
```

```
  resources:
```

```
    requests:
```

```
      storage: 10Gi
```

```
kubectl apply -f nfs-pv.yaml
```

```
kubectl apply -f nfs-pvc.yaml
```

Bước 4: Gắn PVC vào Pod Jenkins

Trong phần volumeMounts và volumes của jenkins và các sidecar như backup-to-minio, thay hostPath bằng:

volumeMounts:

```
- name: jenkins-home
```

```
  mountPath: /var/jenkins_home
```

volumes:

- name: jenkins-home

persistentVolumeClaim:

- claimName: jenkins-nfs-pvc

Kiểm tra

Sau khi khởi động lại pod Jenkins:

`kubectrl get pods -n jenkins -o wide`

Và kiểm tra thử ghi dữ liệu, ví dụ tạo job rồi xem /mnt/data/nfs-jenkins có dữ liệu hay chưa trên máy NFS server.

Cài đặt Nexus trong k8s:

Lấy mật khẩu mặc định của nexus:

```
kubectl exec -it nexus-7fd84bfd8b-z9t47 -n nexus -- cat /nexus-data/admin.password
```

Sau khi nhập mật khẩu mặc định, nexus yêu cầu đặt lại mật khẩu mới cho tài khoản admin. Tài khoản này được dùng để push thư viện lên nexus, cách cấu hình:

Thêm code sau vào file settings.xml trong thư mục .m2:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>nexus</id>
      <username>admin</username>
      <password>bungteDT@123</password>
    </server>
  </servers>
</settings>
```

Cài đặt và kiểm tra docker:

Kiểm tra người dùng trong nhóm "docker"

```
getent group docker
```

Nếu chưa có nhóm "docker" có thể thêm nhóm:

```
sudo groupadd docker
```

Thêm người dùng vào nhóm "docker"

```
sudo usermod -aG docker vo
```

Hoặc thêm người dùng hiện tại vào nhóm "docker"

```
sudo usermod -aG docker $USER
```

Sau khi thêm user vào nhóm "docker" có thể chạy lệnh sau và kiểm tra lại nhóm "docker"

```
newgrp docker
```

1. Kiểm tra CoreDNS có hoạt động đúng không

```
kubectl get pods -n kube-system -l k8s-app=kube-dns -o wide
```

Bạn nên thấy 2 pod coredns-* đang Running.

2. Kiểm tra logs của CoreDNS

```
kubectl logs -n kube-system -l k8s-app=kube-dns
```

Tìm xem có dòng lỗi nào liên quan jenkins.jenkins.svc.cluster.local hoặc các truy vấn bị từ chối.

3. Kiểm tra lại resolv.conf của Pod curl-test

```
kubectl exec -n jenkins curl-test -- cat /etc/resolv.conf
```

Output chuẩn nên là:

```
search jenkins.svc.cluster.local svc.cluster.local cluster.local
nameserver 10.96.0.10
options ndots:5
```

Sau đó kiểm tra lại DNS:

```
nslookup jenkins.jenkins.svc.cluster.local
curl -v http://jenkins.jenkins.svc.cluster.local:8080
```

Tạo secret gắn cho pod kaniko sử dụng cho đăng nhập aws và push code lên ecr:

```
kubectl create secret generic aws-creds-kaniko --from-literal=AWS_ACCESS_KEY_ID=aws-access-key-id --
from-literal=AWS_SECRET_ACCESS_KEY=aws-secret-key --from-literal=AWS_REGION=ap-southeast-1 -n
jenkins
```

```
kubectl create secret generic aws-creds-backend --from-literal=AWS_ACCESS_KEY_ID=xxxxxx --from-
literal=AWS_SECRET_ACCESS_KEY=yyyyy --from-literal=AWS_DEFAULT_REGION=ap-southeast-1 -n
backend
```

```
kubectl create secret generic minio-creds --from-literal=MINIO_ACCESS_KEY_ID=xxxx --from-
literal=MINIO_SECRET_ACCESS_KEY=yyyyy -n jenkins
```

Tạo secret cho phép jenkins agent (kubectl) chạy deployment kéo image từ ecr về tạo container trong kubernetes:

```
kubectl create secret docker-registry ecr-secret --docker-server=211125364313.dkr.ecr.ap-southeast-
1.amazonaws.com --docker-username=AWS --docker-password=$(aws ecr get-login-password --region ap-
southeast-1) --docker-email=vovantungdt123@gmail.com -n backend
```

```
kubectl create secret generic aws-credentials --from-file=credentials=$HOME/.aws/credentials -n jenkins
```

(secret đọc file, cung cấp cho tạo aws/credentials trong pod jenkins trên kubernetes. Trong lệnh dùng cần dùng "\$HOME" thay vì "~" bởi vì "~" là shell nên không được hiểu trong câu lệnh kubectl, còn "\$HOME" là biến môi trường nên sẽ luôn đúng cho các trường hợp)

(Lưu ý: trước khi tạo, chạy aws config với secret được lấy từ aws với quyền phù hợp hoặc secret từ root account)

Cài đặt Grafana, prometheus, alertmanager

Bước 1: Cài đặt Helm

Nếu bạn chưa cài Helm trên máy control plane:

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
helm version
```

Cài đặt StorageClass cho nfs server với tên nfs-client (dùng StorageClass này để mount dữ liệu của pod đến nfs server):

```
helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/
helm repo update
```

Xác định thư mục mount trên NFS server

Trên NFS server (192.168.98.150), bạn cần đảm bảo:

Tạo thư mục dùng để chia sẻ

```
sudo mkdir -p /mnt/data/nfs/nfsclinet
sudo chown -R nobody:nogroup /mnt/data/nfs/nfsclinet
sudo chmod 777 /mnt/data/nfs/ nfsclinet
```

Cấu hình chia sẻ trong /etc/exports

```
sudo vim /etc/exports
```

Thêm dòng sau:

```
/mnt/data/nfs/nfsclinet 192.168.98.0/24(rw,sync,no_subtree_check,no_root_squash)
```

Áp dụng cấu hình:

```
sudo exportfs -ra
sudo systemctl restart nfs-kernel-server
```

path: phải trùng **chính xác** với thư mục bạn đã cấp quyền.

Cài đặt StorageClass “nfs-client” với [nfs-client-values.yaml](#)

```
helm repo update
helm upgrade --install nfs-client \
  nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \
  --namespace nfs-storage --create-namespace \
  -f nfs-values.yaml
```

Kiểm tra StorageClass

```
kubectl get sc
```

Có thể sử dụng Local Path Provisioner (không lưu tập trung và chủ động như nfs-client)

```
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml
```

storageClassName là local-path:

Tạo namespace cho Prometheus

```
kubectl create namespace monitoring
```

Xóa pvc cũ để upgrade prometheus-stack (optional)

```
kubectl delete pvc -n monitoring alertmanager-prometheus-stack-kube-prom-alertmanager-db-alertmanager-
prometheus-stack-kube-prom-alertmanager-0
kubectl delete pvc -n monitoring prometheus-prometheus-stack-kube-prom-prometheus-db-prometheus-
prometheus-stack-kube-prom-prometheus-0
```

Cấu hình alert send mail to gmail account:

Bước 2: Thêm repo Prometheus cộng đồng

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
```

Bước 4: Cài Prometheus + các thành phần đi kèm

```
helm install prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring -f monitoring-
values.yaml
helm upgrade --install prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring -f
monitoring-values.yaml
```

Bước 5: Kiểm tra

```
kubectl get pods -n monitoring
```

```
kubectl describe pod alertmanager-prometheus-stack-kube-prom-alertmanager-0 -n monitoring
kubectl get secret prometheus-stack-grafana -n monitoring -o jsonpath="{.data.admin-password}" | base64 -d
Hoặc: username và password có thể được đặt khi cài trong file (monitoring-values.yaml)
```

Khi cài đặt prometheus, grafana, alertmanager với StorageClass nfs-client, k8s sẽ tự động tạo pvc mount thư được gắn StorageClass nfs-client, và được mount vào thư mục của nfs-client đã được chỉ định khi cài đặt trên nfs-client

Cài đặt loki:

helm repo add grafana <https://grafana.github.io/helm-charts>

helm repo update

Do mặc định loki và promtail sử dụng các tên miền đầy đủ để giao tiếp với nhau (đã gắn sẵn phần searches chẳng hạn cluster.local). Do đó để việc giao tiếp các thành phần bên trong loki và promtail có thể gọi nhau và hoạt động chính xác, cần giảm ndots của các pods của loki và promtail xuống giá trị 1. Tuy nhiên, hiện tại với cấu ở file loki-values.yaml và promtail-values.yaml chưa thể cấu hình được giá trị này nên cần sử dụng **Kyverno** để tự động đặt lại giá trị ndots của các pods trong một namespace theo một giá trị xác định. Để thực hiện việc này, ta sẽ tổ chức loki và promtail trong namespace “logging” và dùng **Kyverno** để đặt lại ndots cho loki và promtail trong namespace này.

Bước 1: Cài Kyverno bằng Helm

helm repo add kyverno <https://kyverno.github.io/kyverno/>

helm repo update

helm install kyverno kyverno/kyverno -n kyverno --create-namespace

Yaml để change ndots=1 cho namespace “logging”

[set-ndots-policy.yaml](#)

Hoặc có thể patch từng pod

```
kubectl patch statefulset loki-backend -n logging --type=json -p '[{"op": "add", "path":  
"/spec/template/spec/dnsConfig", "value": {"options": [{"name": "ndots", "value": "1"}] } ]'
```

Do loki cần sử dụng s3 minio để lưu trữ và truy xuất các index và chunks cho phép lưu trữ và truy xuất log. Do đó cần tạo secret chứa thông tin truy cập minio.

[loki-s3-secret.yaml](#)

Loki sử dụng nfs kiểu StorageClass để lưu trữ dữ liệu cho các pods. Do đó, cần tạo một StorageClass với tên **nfs-client** (đã được tạo và sử dụng cho Grafana và Prometheus)

Cài đặt loki với loki-values.yaml:

helm upgrade --install loki grafana/loki --namespace logging --create-namespace -f loki-values.yaml

[loki-values.yaml](#)

helm uninstall loki -n logging

Lấy một ứng dụng DaemonSet trong namespace:

kubectl get ds -n logging

Kiểm tra container loki trong pod loki-backend để xem log:

kubectl logs -n logging loki-backend-0 -c loki

Note: Sau khi chạy loki với cấu hình và cài đặt như trên, các thành phần trong loki sẽ tự động thực hiện việc ghi và đọc dữ liệu trên **minio**. Cụ thể, khi có yêu cầu truy vấn log từ grafana (đã kết nối với loki qua loki gateway), loki-gateway sẽ gửi yêu cầu đến **loki-read** để thực hiện lấy dữ liệu được lưu trữ trên minio trước đó bởi **loki-write**, kết quả truy vấn sẽ được gửi về grafana qua loki-gateway. Khi promtail thu thập dữ liệu và gửi đến loki-gateway, loki-gateway sẽ chuyển yêu cầu cho loki-write thực hiện ghi dữ liệu lên minio dưới dạng các index và chunks. Do đó việc truy xuất dữ liệu giữa các thành phần Grafana, loki, promtail và minio được thực hiện một cách tự động mà không cần phải thực hiện thêm các hành động thủ công nào cho việc lưu trữ và truy xuất logs. Khi cài đặt lại loki hoặc promtail với cấu hình cũ sẽ có thể truy xuất được đầy đủ logs đã được lưu trữ trên minio. Việc cần làm là xóa các dữ liệu logs quá cũ trên minio để tránh dữ liệu lưu trữ nhiều dữ liệu cũ không cần thiết (có thể thực hiện với cronjob)

Cài đặt promtail:

helm repo add grafana <https://grafana.github.io/helm-charts>

helm repo update

Vì promtail cần thu thập logs trong thư mục tập trung (nfs server), do đó sẽ tạo một pvc mount thư mục trên nfs server (192.168.98.150:/mnt/data/nfs/backend_logs) vào thư mục trong promtail (/mnt/backend_logs)

Tạo pv, pvc với [promtail-logs-pv-pvc.yaml](#)

Note: Do promtail thu thập log trong thư mục được mount cùng với thư mục các ứng dụng backend cùng mount trên nfs server, nên cần mount chính xác thư mục để promtail có thể truy cập chính xác các file log trên nfs server, và điều quan trọng là cần tổ chức lưu trữ file logs trong các ứng dụng khi mount lên nfs server sao cho có cấu trúc `/<namespace>/<app>/*.log`, ví dụ ứng dụng “hrm” có namespace “backend” sẽ lưu logs với cấu trúc thư mục `/logs/backend/hrm/*.log`. Việc này giúp promtail dựa vào cấu hình để đặt các labels cho namespace và app trước khi gửi đến loki (để làm thông tin truy vấn logs).

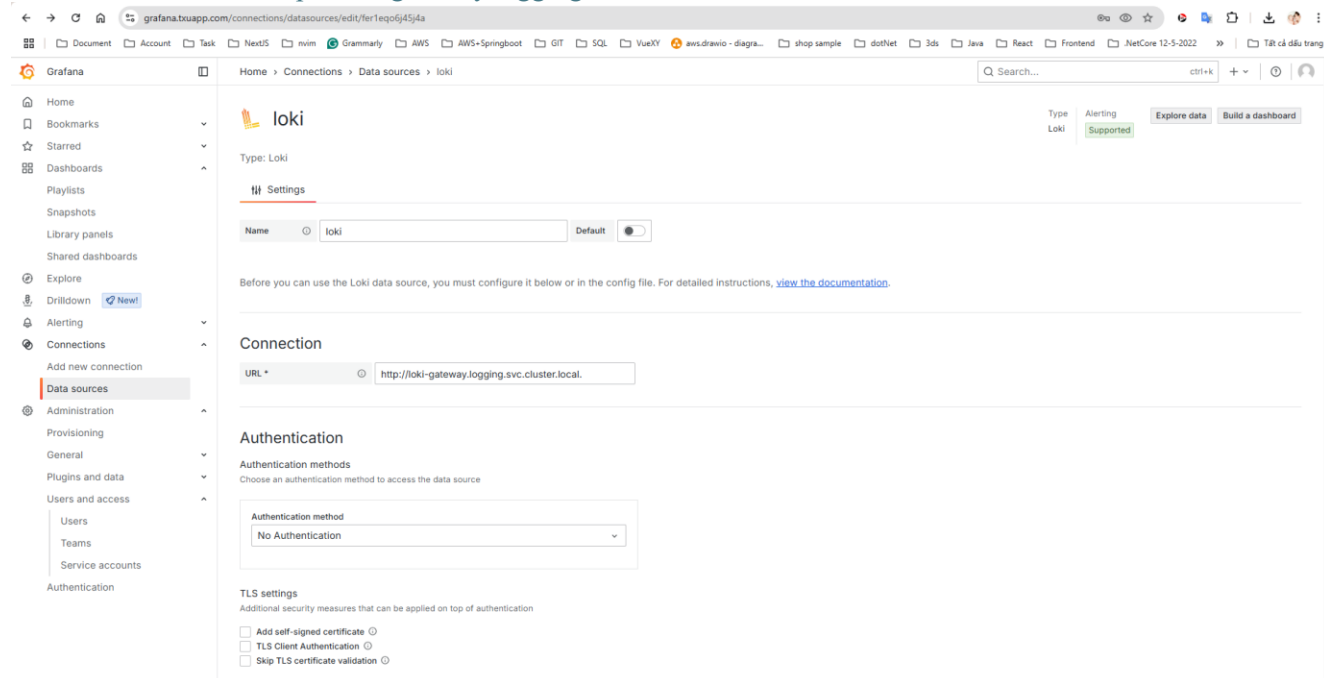
Cài đặt promtail với [promtail-values.yaml](#)

helm upgrade --install promtail grafana/promtail --namespace logging -f promtail-values.yaml
helm uninstall promtail -n logging

Sử dụng Grafana để theo dõi logs:

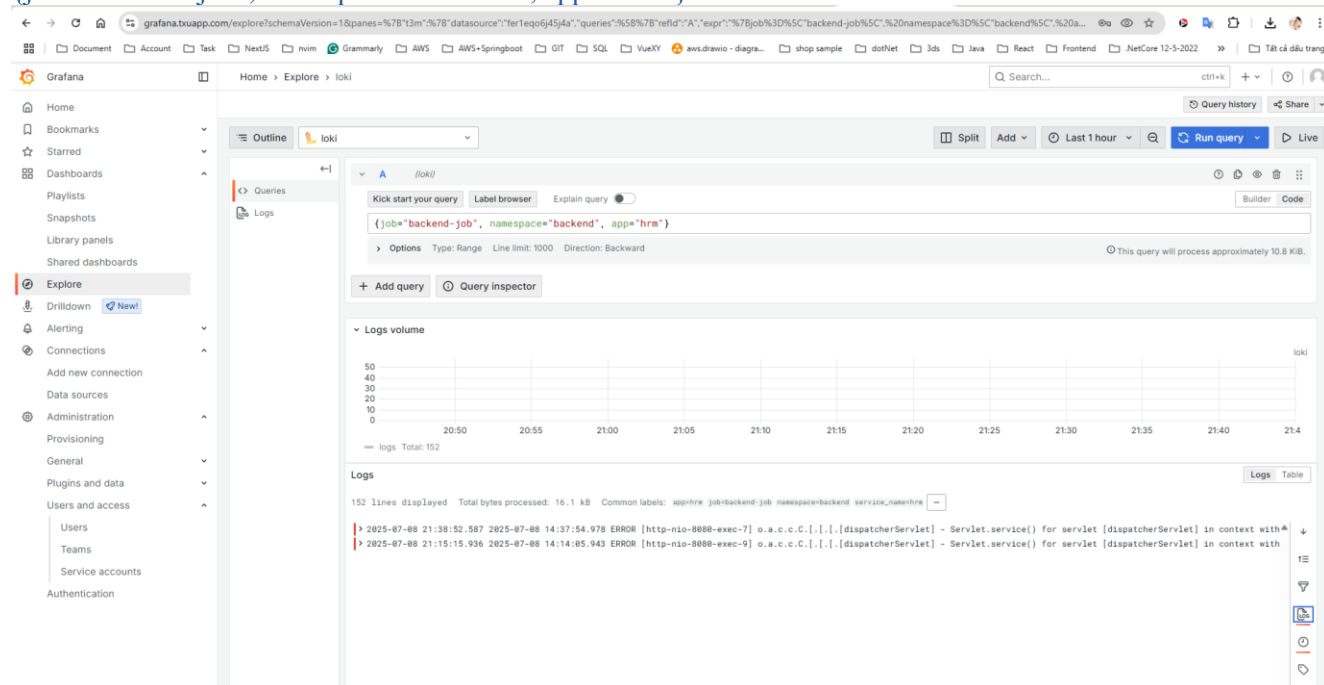
Home -> Data Source -> Add Data Source -> Loki

Mục Connection-url: <http://loki-gateway.logging.svc.cluster.local>.



Home -> Explore -> Loki

`{job="backend-job", namespace="backend", app="hrm"}`



Mục tiêu:

1. Cấu hình ứng dụng **tự động scale (HPA)** khi CPU vượt ngưỡng 70%.
2. Gửi request liên tục đến ứng dụng để **làm tăng CPU load** và kiểm chứng autoscaling.

Kích hoạt metrics server (nếu chưa có)

Kubernetes cần metrics-server để theo dõi CPU/memory:

`kubectly apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml`

Sửa metrics-server deployment

Chạy lệnh sau để **thêm cờ --kubelet-insecure-tls** cho container metrics-server:

`kubectly edit deployment metrics-server -n kube-system`

Trong phần containers.args, sửa hoặc thêm:

containers:

- name: metrics-server

image: k8s.gcr.io/metrics-server/metrics-server:v0.6.x

args:

- --cert-dir=/tmp

- --secure-port=4443

- --kubelet-preferred-address-types=InternalIP,Hostname,InternalDNS,ExternalDNS,ExternalIP

- --kubelet-insecure-tls # Thêm dòng này vào cấu hình

Sau khi lưu, Kubernetes sẽ tự cập nhật pod.

Sau 1-2 phút, test:

`kubectly top pod -n backend`

hoặc

`kubectly get pod -n kube-system | grep metrics`

Nếu bạn thấy CPU/Memory usage → đã OK.

Xem hpa:

`kubectly get hpa -n backend`

Scale instance:

`kubectly -n backend scale deploy hrm --replicas=3`

Cấu hình scale cho replica:

apiVersion: apps/v1

kind: Deployment

metadata:

spec:

template:

spec:

containers:

- name: hrm

resources:

requests:

cpu: "1000m"

limits:

cpu: "1000m"

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: hrm

namespace: backend

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: hrm

minReplicas: 1

maxReplicas: 4

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 70

Cấu hình để alertmanager gửi mail cảnh báo:

Sử dụng tính năng gửi mail của tài khoản gmail để cấu hình gửi mail cảnh báo. Cần lấy mật khẩu gửi mail của tài khoản gmail, truy cập: <https://myaccount.google.com/apppasswords> để tạo một mật khẩu dùng cho việc cấu hình gửi mail.

Sau khi có mật khẩu dùng cho việc gửi mail từ tài khoản gmail, cần tạo một file [alertmanager.yaml](#) để sử dụng tạo một secret dùng config alertmanager:

```
kubectl create secret generic smtp-mail-config --from-file=alertmanager.yaml=alertmanager.yaml -n monitoring
```

Tránh ghi đè cấu hình lên secret:

```
kubectl patch secret smtp-mail-config -n monitoring -p '{"immutable":true}'
```

Sử dụng secret đã tạo trong cấu hình của alertmanager:

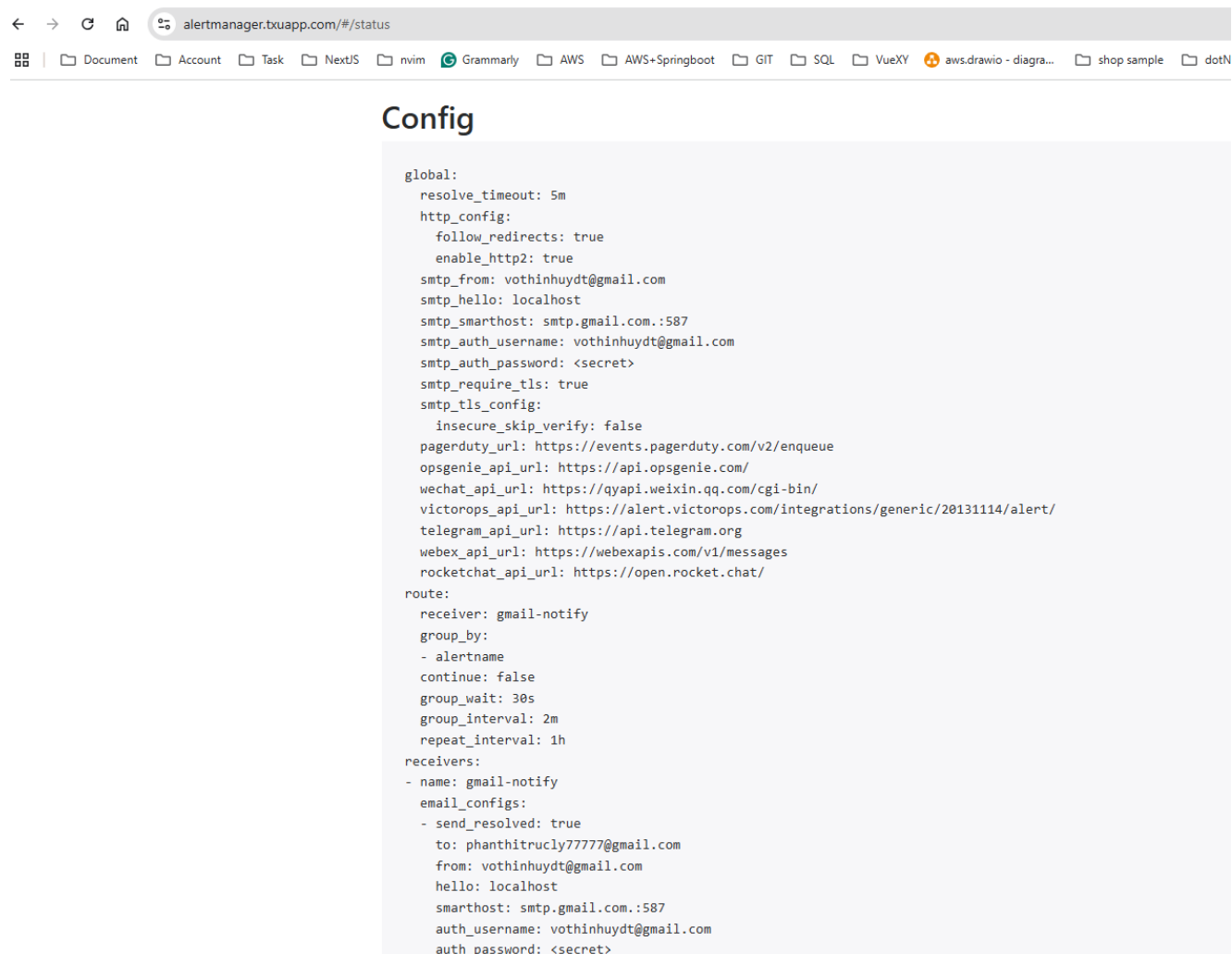
alertmanager:

alertmanagerSpec:

configSecret: smtp-mail-config

Sau khi cài đặt alertmanager với cấu hình từ secret như trên, kiểm tra cấu hình qua:

<https://alertmanager.txuapp.com/#/status>



The screenshot shows a web browser window with the address bar displaying `alertmanager.txuapp.com/#/status`. The browser's tab bar shows several open tabs including Document, Account, Task, NextJS, nvim, Grammarly, AWS, AWS+Springboot, GIT, SQL, VueXY, aws.drawio - diagrama..., shop sample, and dotN. The main content area of the page is titled "Config" and displays a YAML configuration for the alertmanager. The configuration includes global settings like timeouts, http config, and various API endpoints for PagerDuty, OpsGenie, WeChat, VictorOps, Telegram, Webex, and RocketChat. It also defines a route for 'gmail-notify' and a list of receivers, including a detailed email configuration for 'gmail-notify' with fields for name, email configs, send resolved status, to/from addresses, hello message, smarthost, auth username, and auth password.

```
global:
  resolve_timeout: 5m
  http_config:
    follow_redirects: true
    enable_http2: true
  smtp_from: vothinhuydt@gmail.com
  smtp_hello: localhost
  smtp_smarthost: smtp.gmail.com.:587
  smtp_auth_username: vothinhuydt@gmail.com
  smtp_auth_password: <secret>
  smtp_require_tls: true
  smtp_tls_config:
    insecure_skip_verify: false
  pagerduty_url: https://events.pagerduty.com/v2/enqueue
  opsgenie_api_url: https://api.opsgenie.com/
  wechat_api_url: https://qyapi.weixin.qq.com/cgi-bin/
  victorops_api_url: https://alert.victorops.com/integrations/generic/20131114/alert/
  telegram_api_url: https://api.telegram.org
  webex_api_url: https://webexapis.com/v1/messages
  rocketchat_api_url: https://open.rocket.chat/
route:
  receiver: gmail-notify
  group_by:
  - alertname
  continue: false
  group_wait: 30s
  group_interval: 2m
  repeat_interval: 1h
receivers:
- name: gmail-notify
  email_configs:
  - send_resolved: true
    to: phanthitruclly77777@gmail.com
    from: vothinhuydt@gmail.com
    hello: localhost
    smarthost: smtp.gmail.com.:587
    auth_username: vothinhuydt@gmail.com
    auth_password: <secret>
```

Mặc định, khi cấu hình gửi mail cho alertmanager, thì tất cả các rules thỏa điều kiện và được prometheus load sẽ được alertmanager xử lý và gửi cảnh báo qua mail đã cấu hình. Và khi cài đặt mặc định, có rất nhiều rules của hệ thống tạo sẽ được prometheus load. Do đó, cần thêm cấu hình cho prometheus để lọc các rules (giữ rule

custom, tránh load những rules mặc định của hệ thống). Để làm việc này ta cần thực hiện thêm cấu hình lọc cho prometheus như sau:

Cấu hình yaml cài đặt prometheus	Yaml cài đặt rules prometheus
<pre>prometheus: prometheusSpec: ruleNamespaceSelector: matchNames: - monitoring - backend ruleSelectorNilUsesHelmValues: false ruleSelector: matchLabels: role: hrm-rules # Đây dùng để lọc rules</pre>	<pre>apiVersion: monitoring.coreos.com/v1 kind: PrometheusRule metadata: name: hrm-replica-alert namespace: backend labels: role: hrm-rules # Sử dụng role này để map với cấu hình trong prometheus release: prometheus-stack # Phải khớp với release helm spec: groups: - name: hrm.rules rules: - alert: HrmHighReplicaCount expr: kube_deployment_status_replicas{deployment="hrm", namespace="backend"} >= 4 for: 2m labels: severity: warning annotations: summary: "HRM deployment has scaled up (>= 04 replicas)" description: "Triển khai HRM trong không gian 'backend' đã vượt hơn 03 bản sao"</pre>

Thêm rule cho prometheus theo dõi khi scale instance hrm lên 4:

`kubectl apply -f hrm-replica-alert.yaml`

Xem rules:

`kubectl get prometheusrules -n backend`

`kubectl get prometheusrules -A`

`kubectl delete prometheusrule -n backend --all`

`kubectl rollout restart deploy -n monitoring prometheus-prometheus-stack-kube-prometheus`

Tất cả các rules của prometheus trong hệ thống đều có thể xem bởi lệnh: `kubectl get prometheusrules -A`. Tuy nhiên chỉ có những rules map với cấu hình lọc của prometheus mới được load và xem được trên UI của prometheus. Những rules này sẽ được alertmanager xử lý (gửi mail nếu alertmanager đã được cấu hình)

```
tung@control-plane:~/data/kubernetes-doc/deployment/grafana-prometheus$ kubectl get prometheusrules -A
NAMESPACE   NAME                                                                                                     AGE
backend     hrm-replica-alert                                                                                       3h46m
monitoring   prometheus-stack-kube-prom-alertmanager.rules                                                         21m
monitoring   prometheus-stack-kube-prom-config-reloaders                                                            21m
monitoring   prometheus-stack-kube-prom-etcd                                                                        21m
monitoring   prometheus-stack-kube-prom-general.rules                                                                21m
monitoring   prometheus-stack-kube-prom-k8s.rules.container-cpu-usage-second                                         21m
monitoring   prometheus-stack-kube-prom-k8s.rules.container-memory-cache                                             21m
monitoring   prometheus-stack-kube-prom-k8s.rules.container-memory-rss                                               21m
monitoring   prometheus-stack-kube-prom-k8s.rules.container-memory-swap                                              21m
monitoring   prometheus-stack-kube-prom-k8s.rules.container-memory-working-s                                          21m
monitoring   prometheus-stack-kube-prom-k8s.rules.container-resource                                                 21m
monitoring   prometheus-stack-kube-prom-k8s.rules.pod-owner                                                          21m
monitoring   prometheus-stack-kube-prom-kube-apiserver-availability.rules                                             21m
monitoring   prometheus-stack-kube-prom-kube-apiserver-burnrate.rules                                                21m
monitoring   prometheus-stack-kube-prom-kube-apiserver-histogram.rules                                              21m
monitoring   prometheus-stack-kube-prom-kube-apiserver-slos                                                          21m
monitoring   prometheus-stack-kube-prom-kube-prometheus-general.rules                                                21m
monitoring   prometheus-stack-kube-prom-kube-prometheus-node-recording.rules                                         21m
monitoring   prometheus-stack-kube-prom-kube-scheduler.rules                                                         21m
monitoring   prometheus-stack-kube-prom-kube-state-metrics                                                           21m
monitoring   prometheus-stack-kube-prom-kubelet.rules                                                                21m
monitoring   prometheus-stack-kube-prom-kubernetes-apps                                                             21m
monitoring   prometheus-stack-kube-prom-kubernetes-resources                                                         21m
monitoring   prometheus-stack-kube-prom-kubernetes-storage                                                           21m
monitoring   prometheus-stack-kube-prom-kubernetes-system                                                            21m
monitoring   prometheus-stack-kube-prom-kubernetes-system-apiserver                                                  21m
monitoring   prometheus-stack-kube-prom-kubernetes-system-controller-manager                                         21m
monitoring   prometheus-stack-kube-prom-kubernetes-system-kube-proxy                                                 21m
monitoring   prometheus-stack-kube-prom-kubernetes-system-kubelet                                                    21m
monitoring   prometheus-stack-kube-prom-kubernetes-system-scheduler                                                  21m
monitoring   prometheus-stack-kube-prom-node-exporter                                                                21m
monitoring   prometheus-stack-kube-prom-node-exporter.rules                                                         21m
monitoring   prometheus-stack-kube-prom-node-network                                                                21m
monitoring   prometheus-stack-kube-prom-node.rules                                                                    21m
monitoring   prometheus-stack-kube-prom-prometheus                                                                    21m
monitoring   prometheus-stack-kube-prom-prometheus-operator                                                         21m
```

prometheus.txuapp.com/alerts

Document Account Task NextJS nvim Grammarly AWS AWS+Springboot GIT Tắt cả dấu trang

Prometheus Query Alerts Status

Filter by rule group state Filter by rule name or labels

hrm.rules

/etc/prometheus/rules/prometheus-prometheus-stack-kube-prom-prometheus-rulefiles-0/backend-hrm-replica-alert-7d95ae8a-3da5-4447-8f68-a0678b91cd46.yaml

INACTIVE (1)

HrmHighReplicaCount

kube_deployment_status_replicas{deployment="hrm",namespace="backend"} >= 4

for: 1m

severity="warning"

description Triển khai HRM trong không gian 'backend' đã vượt hơn 03 bản sao

summary HRM deployment has scaled up (>= 04 replicas)

The screenshot shows the Prometheus web interface at `prometheus.txuapp.com/rules`. The top navigation bar includes 'Query', 'Alerts', and 'Status > Rule health'. The main content area displays the 'hrm.rules' configuration file located at `/etc/prometheus/rules/prometheus-prometheus-stack-kube-prom-prometheus-rulefiles-0/backend-hrm-replica-alert-7d95ae8a-3da5-4447-8f68-a0678b91cd46.yaml`. The rule is named 'HrmHighReplicaCount' and is triggered by the query `kube_deployment_status_replicas{deployment="hrm", namespace="backend"} >= 4`. The rule is currently 'OK' and has a severity of 'warning'. The description states: 'Triển khai HRM trong không gian 'backend' đã vượt hơn 03 bản sao' (HRM deployment in namespace 'backend' has exceeded more than 3 replicas). The summary states: 'HRM deployment has scaled up (>= 04 replicas)'.

Sử dụng JMeter để tăng load của ứng dụng và replicas của pod tăng lên đến ngưỡng trong rule và kiểm tra cảnh báo (trên UI và mail)

Kiểm tra replicas của ứng dụng hrm trên UI của ptometheus (replicas:4), trong Query:

`kube_deployment_status_replicas{deployment="hrm", namespace="backend"}`

The screenshot shows the Prometheus web interface at `prometheus.txuapp.com/query?g0.expr=kube_deployment_status_replicas%7Bdeployment%3D%22hrm%22%2Cnamespace%3D%22backend%22%7D`. The query is entered in the search bar, and the 'Execute' button is clicked. The results are displayed in a table format, showing the evaluation time and the result series. The result series is a single value of 4, indicating that there are 4 replicas of the 'hrm' deployment in the 'backend' namespace.

Kiểm tra bằng lệnh shell trong k8s, thấy các replica của ứng dụng hrm được tạo ra và tăng đến ngưỡng định nghĩa trong rule:

`kubect top pod -n backend`

```
tung@control-plane:~/data/kubernetes-doc/deployment/grafana-prometheus$ kubectl top pod -n backend
NAME                                CPU(cores)   MEMORY(bytes)
hrm-6866bc5b86-2jn52               1003m        291Mi
hrm-6866bc5b86-qr6nm               515m         311Mi
hrm-6866bc5b86-v6wm2               697m         292Mi
hrm-6866bc5b86-wmjsg              1001m        256Mi
tung@control-plane:~/data/kubernetes-doc/deployment/grafana-prometheus$
```

Kiểm tra trên Prometheus UI, thấy trạng thái rule đã chuyển sang trạng thái **“firing”** (được kích hoạt để gửi cảnh báo đến alertmanager xử lý). Một rule có 3 trạng thái inactive, pending và firing.

The screenshot shows the Prometheus Alerts page in a web browser. The URL is `prometheus.txuapp.com/alerts`. The page has a navigation bar with 'Prometheus', 'Query', 'Alerts' (active), and 'Status'. Below the navigation bar, there are filters for 'Filter by rule group state' and 'Filter by rule name or labels'. The main content area shows a list of alerts. The first alert is 'hrm.rules' with a path to the rule file and a 'FIRING (1)' status. The alert details for 'HrmHighReplicaCount' are shown below, including the alert expression, severity, description, summary, and alert labels. The alert is currently in a 'FIRING' state.

hrm.rules /etc/prometheus/rules/prometheus-prometheus-stack-kube-prom-prometheus-rulefiles-0/backend-hrm-replica-alert-7d95ae8a-3da5-4447-8f68-a0678b91cd46.yaml **FIRING (1)**

HrmHighReplicaCount **FIRING (1)**

`kube_deployment_status_replicas{deployment="hrm",namespace="backend"} >= 4`

for: 1m

severity: "warning"

description Triển khai HRM trong không gian 'backend' đã vượt hơn 03 bản sao

summary HRM deployment has scaled up (>= 04 replicas)

Alert labels	State	Active Since	Value
alertname="HrmHighReplicaCount" container="kube-state-metrics" deployment="hrm" endpoint="http" instance="10.244.2.167:8080" job="kube-state-metrics" namespace="backend" pod="prometheus-stack-kube-state-metrics-6865fd9979-4jq9c" service="prometheus-stack-kube-state-metrics" severity="warning"	FIRING	1m 2.02s	4

description Triển khai HRM trong không gian 'backend' đã vượt hơn 03 bản sao

summary HRM deployment has scaled up (>= 04 replicas)

Kiểm tra trên alertmanager UI, thấy nội dung gửi mail khi cảnh báo được prometheus kích hoạt (dựa vào rule đã định nghĩa)

Filter

Group

Receiver: All

Silenced

Inhibited

Muted

+

Silence

Custom matcher, e.g. `env="production"`

⊕ Expand all groups

— gmail-notify alertname="HrmHighReplicaCount" + 1 alert

2025-07-10T11:07:23.775Z — Info📄 Source🔧 Silence🔗 Link

description: Triển khai HRM trong không gian 'backend' đã vượt hơn 03 bản sao

summary: HRM deployment has scaled up (>= 04 replicas)

container="kube-state-metrics" + deployment="hrm" + endpoint="http" + instance="10.244.2.167:8080" + job="kube-state-metrics" +

namespace="backend" + pod="prometheus-stack-kube-state-metrics-6865fd9979-4jq9c" +

prometheus="monitoring/prometheus-stack-kube-prom-prometheus" + service="prometheus-stack-kube-state-metrics" + severity="warning" +

Cài đặt Kong gateway:

1. Cài đặt CustomResourceDefinitions (CRDS):

`kubectl apply -f https://github.com/Kong/charts/raw/main/charts/kong/crds/custom-resource-definitions.yaml`

Hoặc

`kubectl apply -f custom-resource-definitions.yaml`

Gỡ cài đặt CRDS

`kubectl delete -f custom-resource-definitions.yaml`

Hoặc xóa tất cả crds:

`kubectl get crd | grep kong | awk '{print $1}' | xargs kubectl delete crd`

Xem các cdr của kong đang tồn tại:

`kubectl get crds | grep kong`

2. Cài Kong với PostgreSQL (database mode)

`helm repo add kong https://charts.konghq.com`

`helm repo update`

`helm upgrade --install kong kong/kong --namespace kong --create-namespace -f kong-values.yaml`

Note: Kong với chế độ DB sẽ cho phép lưu các cấu hình ở database, kong bao gồm 03 pod (01 pod kong, và 01 pod PostgreSQL, 01 pod kong ingress controller). KIC sẽ lắng nghe và sync các cấu hình về kong thông qua kong admin. Các cấu hình KIC sync về kong bao gồm: routes khi các ingress có gắn IngressClass của kong, KongConsumer cùng với Secret chứa thông tin credentials cho KongConsumer. Do đó sau khi cài đặt, mặc định các Ingress được gắn IngressClass kong sẽ được kong route và cho đi qua hệ thống của mình, nếu các ingress chưa có plugins của kong nào được áp thì các request sẽ được tự do đi qua cho đến khi được áp plugins thì kong sẽ xử lý request theo các logic xử lý plugins đã áp lên đó.

Khi cài kong cần chú ý, đối với các custom plugins sẽ được đưa vào kong (cụ thể là ở container kong proxy của kong). Custom plugin bao gồm 02 file handler.lua và schema.lua nằm trong thư mục với tên của custom plugins, và cần copy đến thư mục `/usr/local/share/lua/5.1/kong/plugins` của container kong proxy, và nếu ta cần tạo custom plugins có tên là `txu-plugins` thì hai file trên sẽ được đặt trong thư mục `/usr/local/share/lua/5.1/kong/plugins/txu-plugins`, ngoài ra cần cấu hình biến môi trường để kong biết load custom plugins đã đưa vào kong (`ENV KONG_PLUGINS="bundled,txu-plugins"`) trong dockerfile, image docker mới được build từ image kong nạp thêm cấu custom plugins mới, do đó khi cài kong bằng image docker mới sẽ có thêm custom plugins này.

Khi xây dựng logic xử lý cho các custom plugins, cần chú ý plugins `jwt` mặc định trong kong sẽ xử lý cors, nghĩa là `jwt` sẽ gắn các header (cho phép cors) vào response trả về cho các request, do đó các ingress được gắn `jwt` mặc định sẽ được chấp nhận cors, với custom plugins nếu không áp cùng với `jwt` vào các ingress cần phải tự xử lý cors, và các response trả về nếu không gắn header báo cho trình duyệt biết chấp nhận cors thì trình duyệt sẽ không chấp nhận. Ngoài ra hành vi của trình duyệt trước khi gửi request thật sẽ gửi một request OPTIONS để thăm dò cors của hệ thống, nên với các plugins mặc định và custom plugins đều cần xử lý chấp nhận request OPTIONS từ trình duyệt, cách xử lý chấp nhận là return sớm nếu request là OPTIONS, vì request OPTIONS không chứa thông tin xác thực (token), nên nếu không return sớm request này thì các logic xử lý bên dưới của custom plugins sẽ từ chối nó (OPTIONS).

Trong custom plugin, có thể trả lời một request đến một route không có trong hệ thống (dummy-service), chẳng hạn ta thiết kế một xử lý để trả lời một request gọi đến path `/get-role` trong hệ thống, vì chức năng này custom plugins có thể xử lý được dựa trên thông tin token trong request gửi đến mà không cần đến một service thật. Để làm được điều này ta cần thêm một route với path là `/get-role` cho kong bằng cách tạo một ingress cho kong quản lý với path `/get-role`. Ingress (route) này sẽ trỏ đến một dummy-service (service không tồn tại). Và sau đó chỉ cần xử lý route này trong custom plugin. Do đây cũng như một path thật cho phép request gửi đến, nên cũng cần xử lý request OPTIONS và request thật cho path này. Khi xử lý request OPTION và request thật cần chú ý phải sử dụng `return kong.response.exit(200, {role: ""})` để kong trả về kết quả cho client ngay, tránh dùng `return` vì kong sẽ chuyển tiếp yêu cầu đến dummy-service, sẽ gây lỗi.

3. Cài Kong Ingress Controller (KIC)

Cài kong ingress controller với yaml

`kubectl apply -f kong-ingress-controller-values.yaml`

Kiểm tra kong admin

`kubecttl port-forward svc/kong-admin -n kong 8001:8001`

Kiểm tra các routes đang có trong kong, các routes này do KIC đồng bộ đến kong admin khi có các các Ingress gắn IngressClass mà KIC quản lý. Khi cài KIC, Ingress class này sẽ được tạo và khai báo để KIC quản lý.

`curl http://localhost:8001/routes`

Consumer được tạo dựa trên một custom resource (CRDS) KongConsumer, là đại diện người dùng mà các plugins trong kong cần sử dụng để lấy thông tin xử lý cho xác thực người dùng. Consumer trong sử dụng secret chứa credentials (thông tin xác thực như: chữ ký secret, iss)

Kiểm tra các consumers có trong kong:

`curl http://localhost:8001/consumers`

Jwts (các secret chứa credentials cho các plugins trong kong sử dụng là thông tin xác thực);

`curl http://localhost:8001/jwts`

Các plugins được dùng để áp lên các Ingress, khi đó các xử lý của plugins trong logic xử lý sẽ được áp lên (có hiệu lực với các routes được kong tạo ra dựa trên ingress). Các plugins được tạo ra dựa trên CRD KongPlugins, có thuộc tính plugin là tên của plugins trong kong (có thể là các plugins mặc định như jwt, hoặc có thể là một custom plugins). Khi có nhiều plugins cùng áp lên một ingress thì nó sẽ chịu tác động đồng thời của các plugins đó. Chẳng hạn, khi một ingress gắn plugins jwt và một custom plugins thì nó chịu logic xác thực token của jwt đồng thời chịu xử lý (phân quyền,..) của custom plugins.

`curl http://localhost:8001/plugins`

Kiểm tra các deployment cso trong namespaces kong:

`Kubecttl get deploy -n kong`

Kiểm tra chi tiết một deployment của kong, chẳng hạn:

`kubecttl get deploy/kong -n kong -o -yaml`

`kubecttl get deploy/kong-ingress-controller -n kong -o yaml | grep -A 20 "containers:"`

Kiểm tra mô tả chi tiết một deployment, thường để xem các trạng thái lỗi đầu tiên khi triển khai pod:

`Kubecttl describe pod <pod-name> -n kong`

Kiểm tra log của container proxy trong pod kong:

`kubecttl logs deploy/kong -n kong -c proxy -f | grep <key_serch>`

`kubecttl logs deploy/kong -n kong -c proxy -f | grep txu-plugins`

Kiểm tra container proxy trong pod kong

`kubecttl exec -it deploy/kong -n kong -c proxy -- /bin/sh`

Kiểm tra thư mục chứa custom plugins đã thêm vào thông qua image docker

`cd /usr/local/share/lua/5.1/kong/plugins/txu-plugins`

Kiểm tra IngressClass:

`Kubecttl get ingressclass`

Xem thông tin của crds kongconsumer

`kubecttl get crd -n kong | grep konghq`

`kubecttl get crd kongconsumers.configuration.konghq.com -o yaml`

Xem KongConsumer:

```
kubectrl get kongconsumer -n backend  
kubectrl get kongconsumer -n backend -o yaml  
kubectrl get kongconsumer txu-csm -n backend -o yaml  
kubectrl get kongplugin -n backend
```

Sửa deployment của KIC

```
kubectrl edit deploy kong-ingress-controller -n kong  
kubectrl rollout restart deployment kong-ingress-controller -n kong  
kubectrl rollout restart deploy/kong-ingress-controller -n kong
```

Xem log của KIC

```
kubectrl logs -n kong deploy/kong-ingress-controller --tail=100 -f  
kubectrl logs -n kong deploy/kong-ingress-controller --tail=100 | grep consumer  
kubectrl logs -n kong deploy/kong-ingress-controller -c ingress-controller | grep jwt-auth  
kubectrl logs -n kong deploy/kong-ingress-controller -f | grep txu-iss
```

Linh tinh

```
kubectrl api-resources | grep kong  
kubectrl get secret txu-jwt -n backend -o yaml
```