ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

~~~~~~~~~~*~~~~~~~~~~

ĐỒ ÁN MÔN HỌC

Đề tài: Build an intelligent monitoring system for a Kubernetes cluster using Prometheus and Grafana

Môn học: Đánh giá hiệu năng hệ thống mạng máy tính

Sinh viên thực hiện:	**Tô Công Quân - 22521190**
	Nguyễn Thành Thạo - 22521371
	Nguyễn Đức Toàn - 22521490
	Trần Văn Thuận - 22521448
Giảng viên hướng dẫn:	**ThS. Đặng Lê Bảo Chương**

TP.Hồ Chính Minh - 2025
~~~~~~~~~~

LỜI CẢM ƠN

Lời đầu tiên, cho phép tập thể Nhóm 1 đến từ lớp NT531.P21 xin gửi lời cảm ơn và tri ân sâu sắc đến thầy Đặng Lê Bảo Chương - giảng viên môn Đánh giá hiệu năng hệ thống mạng máy tính, vì sự tận tâm giúp đỡ, chỉ bảo và hướng dẫn trong quá trình thực hiện đề tài. Những hướng dẫn của thầy đóng vai trò quan trọng trong việc hoàn thiện đồ án này. Chúng em chân thành cảm ơn thầy vì kiến thức và kinh nghiệm mà thầy đã chia sẻ với chúng em.

Chúng em cũng muốn gửi lời cảm ơn đến tất cả các thành viên trong nhóm đồ án. Sự đóng góp và nỗ lực của mỗi người trong việc tìm kiếm tài liệu, đưa ra ý tưởng và hoàn thiện đề tài .

Cuối cùng, vì thời gian và năng lực có hạn nên không thể tránh khỏi sai sót trong khi thực hiện đồ án học tập của chúng em. Rất mong sự góp ý và bổ sung của thầy và các bạn để đề tài chúng em trở nên hoàn thiện hơn. Một lần nữa, chân thành cảm ơn thầy và tất cả mọi người đã theo dõi đề tài này.

MỤC LỤC

DANH MỤC HÌNH ẢNH VÀ BẢNG

TÓM TẮT

Đề tài tập trung vào xây dựng hệ thống giám sát thông minh cho cụm Kubernetes sử dụng Prometheus và Grafana. Nhóm triển khai một cụm gồm ba node (một master, hai worker) và cài đặt các công cụ như Prometheus, Grafana, Rancher để theo dõi hiệu năng hệ thống. Prometheus thu thập số liệu hiệu năng như CPU, RAM, trong khi Grafana hiển thị chúng dưới dạng dashboard trực quan. Hệ thống cảnh báo được thiết lập để gửi email khi có sự cố về tài nguyên. Nhóm cũng triển khai các kịch bản thực nghiệm như stress test CPU, giám sát pods và tự động scale khi có nhiều request dẫn đến tăng CPU, Memory đột ngột. Ngoài ra, đề tài còn thử nghiệm tình huống khi một node bị tắt đột ngột để kiểm tra khả năng phản ứng của hệ thống.

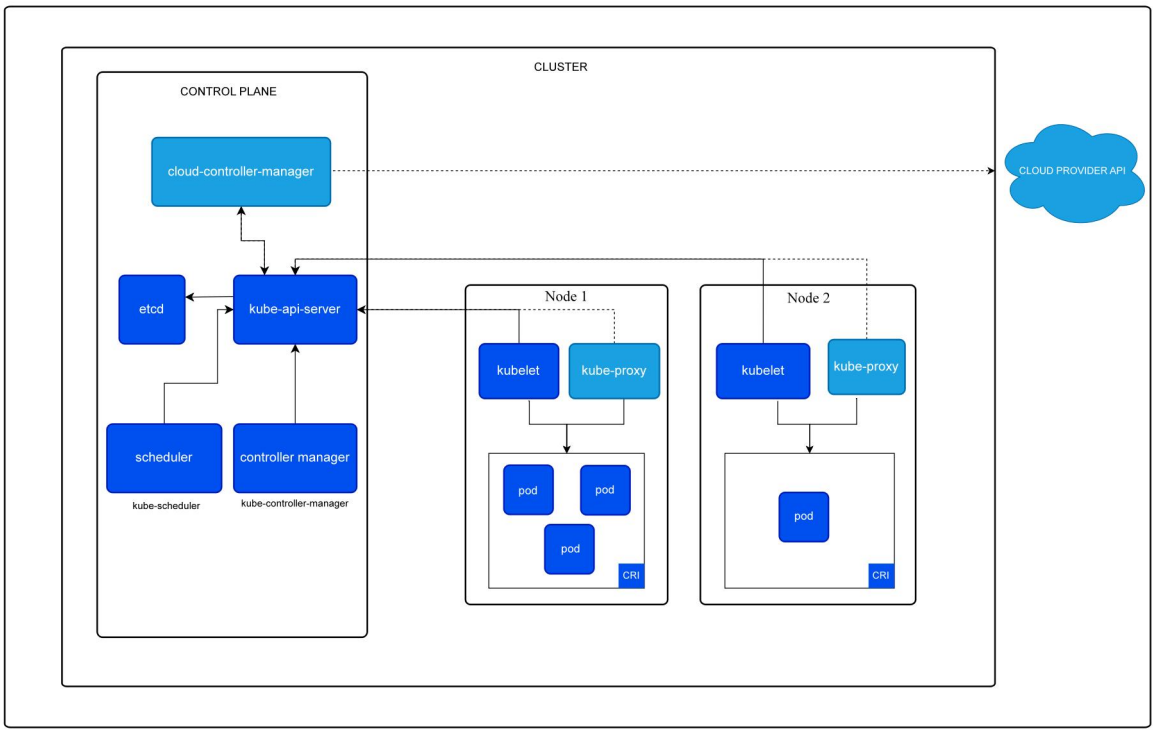
I. TỔNG QUAN

1. Kubernetes

1.1. Khái niệm

Được biết đến với một cái tên khác là K8s, là một hệ thống mã nguồn mở cho việc triển khai, mở rộng và quản lý một cách tự động những ứng dụng dưới dạng container.

1.2. Kiến trúc



Hình 1: Kiến trúc Kubernetes

- Master node (control plane): được hiểu như là server điều khiển các node trong cụm kubernet hoạt động. Gồm:
  - API server: Thành phần giúp cho các thành phần khác bên trong cụm K8s có thể giao tiếp được với nhau
  - Scheduler: Có chức năng lập lịch làm việc, triển khai cho các ứng dụng bên trong

- Controller manager: Quản lý hoạt động của tất cả woker trong K8s, bất kể còn hay đã mất
- Etcd: Là hệ cơ sở dữ liệu của K8s

- Node: hay còn được gọi là worker có chức năng thực hiện những tác vụ được yêu cầu, được giao.
  - Kuberlet: giúp cho node giao tiếp được với Master thông qua API server để hoạt động trơn tru hơn
  - Kube-proxy: Giúp cho các pod trong node có thể giao tiếp với bên ngoài
  - Pod: là đơn vị nhỏ nhất của một node, mỗi node chưa một hoặc nhiều container. Trong thực tế, mỗi pod thường đại diện cho một container, người ta hay gọi pod là server

1.3. Các chứng năng chính

- Cân bằng tải: K8s quản lý nhiều Docker host bằng cách tạo các container cluster (cụm container). Ngoài ra, khi chạy một container trên Kubernetes, việc triển khai replicas (tạo các bản sao giống nhau) có thể đảm bảo cân bằng tải (load balancing) tự động và tăng khả năng chịu lỗi. Đồng thời nhờ có load balancing mà cũng có thể thực hiện autoscaling - tự động tăng giảm số lượng replicas.
- Tự phát hành và tự thu hồi: Docker host còn gọi là Node. Khi sắp xếp các container vào các Node này, có các dạng workload khác nhau như Disk, CPU,…
- Quản lý cấu hình: K8s có chức năng gọi là Service cung cấp load balancing cho một nhóm container cụ thể. Ngoài việc tự động thêm và xóa tại thời điểm scale, nó tự động ngắt kết nối trong trường hợp container bị lỗi.

2. Công cụ quản lý, giám sát

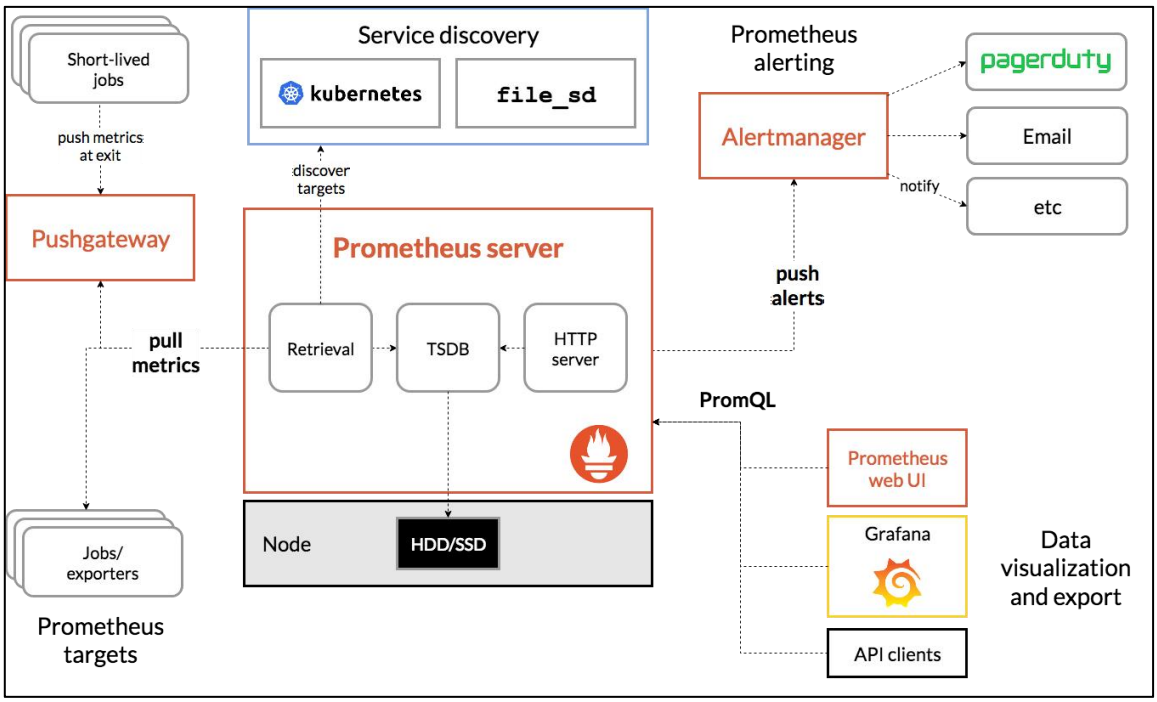
2.1. Prometheus

Prometheus là bộ công cụ giám sát và cảnh báo hệ thống nguồn mở, thu thập và lưu trữ số liệu của mình dưới dạng dữ liệu chuỗi thời gian, tức là thông tin số liệu được lưu trữ với dấu thời gian mà nó được ghi lại, cùng với các cặp khóa-giá trị tùy chọn được gọi là nhãn.

2.1.1. Các đặc điểm:

- Mô hình dữ liệu đa chiều với dữ liệu chuỗi thời gian được xác định bằng tên số liệu và cặp khóa/giá trị
- PromQL, ngôn ngữ truy vấn linh hoạt để tận dụng tính đa chiều này
- Không phụ thuộc vào lưu trữ phân tán; các nút máy chủ đơn lẻ là tự chủ
- Thu thập chuỗi thời gian diễn ra thông qua mô hình kéo qua HTTP
- Đẩy chuỗi thời gian được hỗ trợ thông qua cổng trung gian
- Mục tiêu được phát hiện thông qua khám phá dịch vụ hoặc cấu hình tĩnh
- Nhiều chế độ hỗ trợ biểu đồ và bảng điều khiển

2.1.2. Kiến trúc



Hình 2: Kiến trúc prometheus

Prometheus thu thập số liệu từ các công việc được đo lường, trực tiếp hoặc thông qua cổng đẩy trung gian cho các công việc ngắn hạn. Nó lưu trữ tất cả các mẫu đã thu thập cục bộ và chạy các quy tắc trên dữ liệu này để tổng hợp và ghi lại chuỗi thời gian mới từ dữ liệu hiện có hoặc tạo cảnh báo. Grafana hoặc các trình tiêu thụ API khác có thể được sử dụng để trực quan hóa dữ liệu đã thu thập.

2.2. Grafana

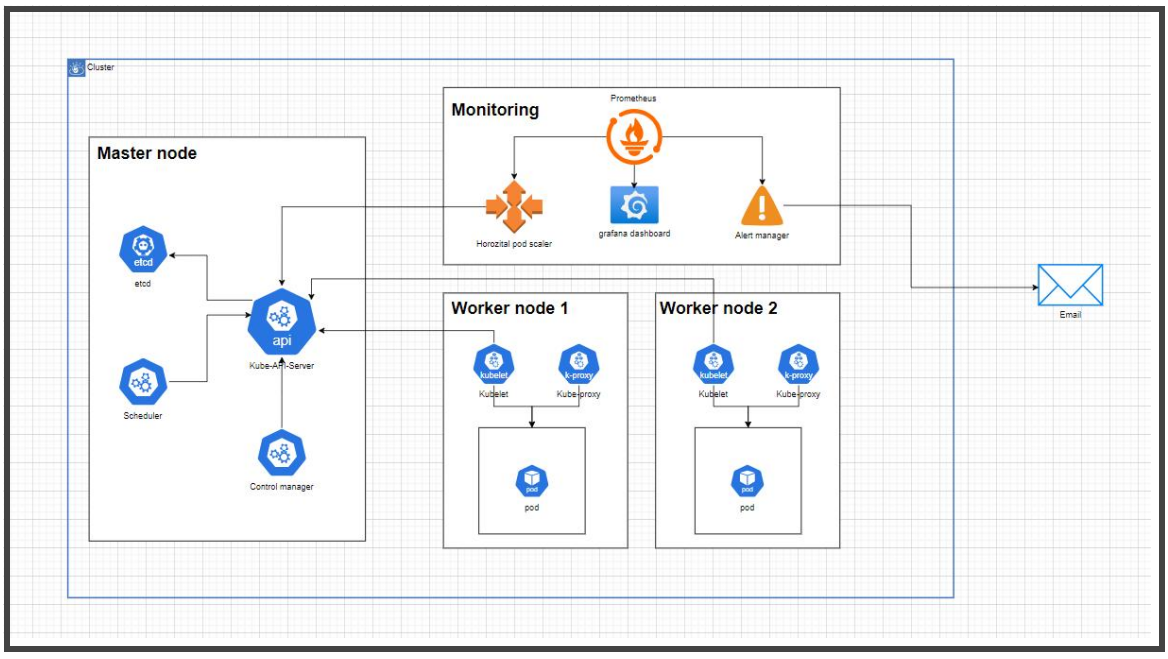
Grafana là một nền tảng mã nguồn mở giúp truy vấn, hiển thị, và cảnh báo dữ liệu từ nhiều nguồn khác nhau. Có thể sử dụng Grafana để theo dõi thông tin theo thời gian thực từ nhiều loại dữ liệu khác nhau, như cơ sở dữ liệu thời gian thực (Time Series Database), SQL, NoSQL, hoặc các dịch vụ cloud thông qua các bảng điều khiển (dashboard) có thể tùy chỉnh. Nó lấy nguồn dữ liệu từ nguồn như Prometheus, Graphite hay ElasticSearch…

Hiểu đơn giản thì Prometheus làm nhiệm vụ cặm cụi lấy thông tin hệ thống, Grafana thì kết nối với Prometheus để lấy dữ liệu và hiện thị lên một cách gọn gàng đẹp đẽ và long lanh nhất.

Các tính năng chính:

- Dashboard Đây là tính năng "đỉnh" nhất của Grafana với rất nhiều tùy biến cũng như hệ thống template cực kỳ đa dạng giúp việc hiển thị dữ liệu trở nên sinh động, trực quan.
- Alerts: Việc đặt ngưỡng cảnh có thể được thực hiện ở Grafana (tương tự cấu hình rule ở Alert Manager vậy).
- Native Support: Được hỗ trợ naitive từ rất nhiều database phổ biến như MySQL, Postgres..
- Built-in Support: Hỗ trợ sẵn các datasource đối với Prometheus, Influx DB, CloudWatch, Graphite, ElasticSearch.

II. MÔ HÌNH



Hình 3: Mô hình triển khai

Ý tưởng thực hiện:

- Triển khai một cụm K8s gồm ba node, một node master và hai node worker.
- Trên node master sẽ cài đặt prometheus và grafana để quản lý, giám sát hiệu năng của cụm, ngoài ra cài đặt thêm Rancher để quản lý cụm K8s bằng ứng dụng web.

- Triển khai một ứng dụng thông qua pod, nó sẽ được cấu hình triển khai thông qua file yaml trên master, khi đó pod sẽ được phân bổ trên tất cả các node worker.
- Cấu hình Alert management để gửi cảnh báo thông qua email khi hiệu năng cụm vượt ngưỡng như CPU, RAM,… Ngoài ra, khi ứng dụng được triển khai, nếu các traffic quá nhiều có thể dẫn tới nghẽn hay sập ứng dụng thì sẽ áp dụng scaling để cân bằng tải, khi đó cũng sẽ gửi email về để thông báo.

III. TRIỂN KHAI

1. Cài đặt cụm Kubernetes

Vì đề tài tập trung vào việc quản lý, giám sát hiệu năng của cụm K8s nên phần cấu hình sẽ tham khảo từ link dưới, phần số 2:

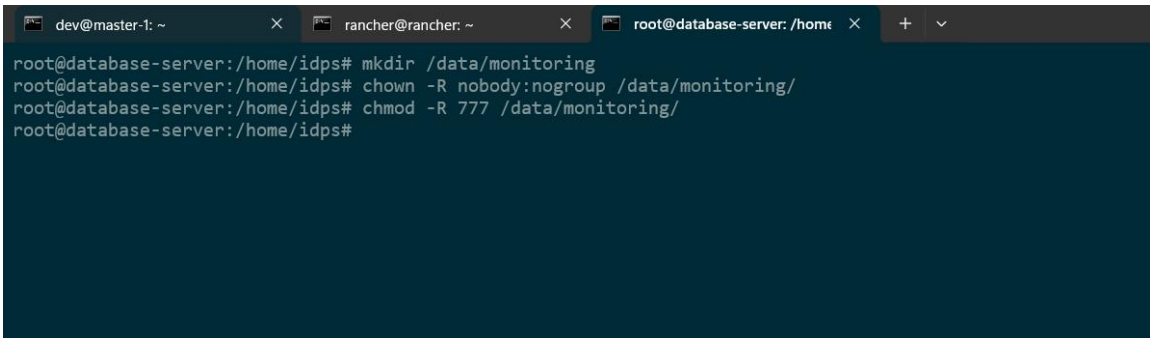
Tại đây

Server	Ip	Host
master-1	192.168.169.134	
master-2	192.168.169.135	
master-3	192.168.169.132	
rancher	192.168.169.136	rancher.quantc.uit
loadbalancer-server	192.168.169.138	<app_name>.quantc.uit
database-server	192.168.169.129	

Bảng thống số các server

2. Cài đặt công cụ quản lý, giám sát

Trên server database-server, trước hết cần tạo thư mục riêng cho prometheus

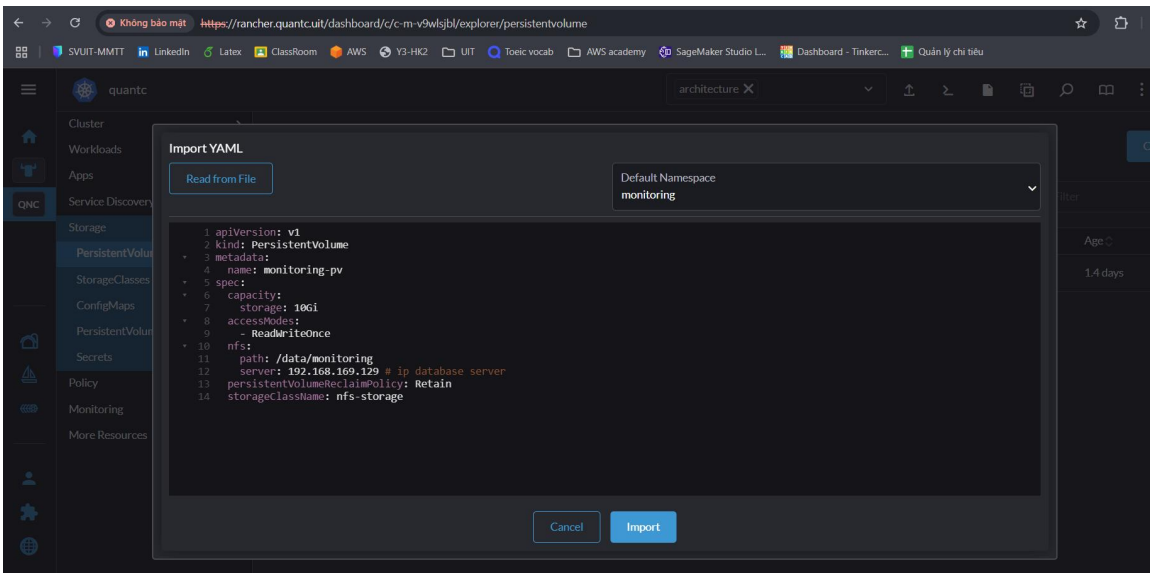


Hình 4: Tạo thư mục riêng lưu trữ cho prometheus

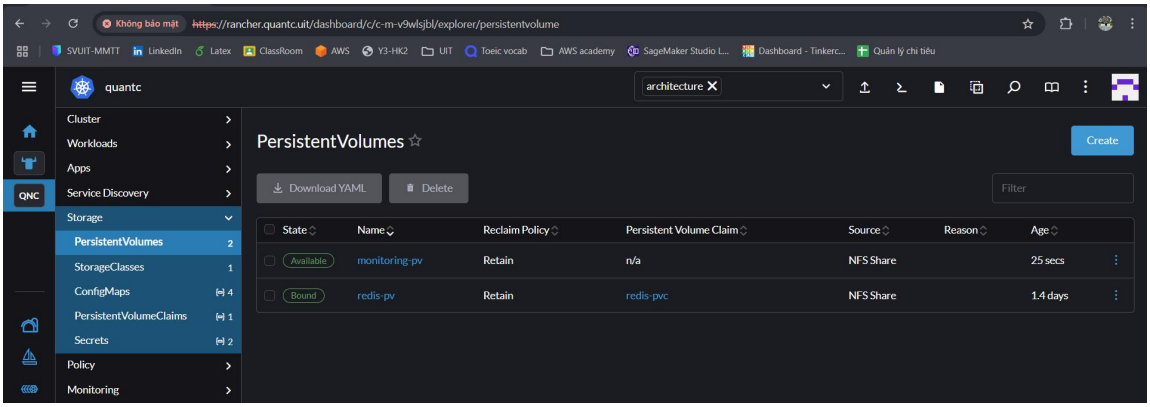
Cấu hình PV cho prometheus

```
dev@master-1:~$ kubectl create ns monitoring
namespace/monitoring created
dev@master-1:~$
```

Hình 5: Tạo namespace riêng cho việc monitoring



Hình 6: Import file yaml để cấu hình PV



Hình 7: Kiểm tra PV đã được tạo trên Rancher

Thêm repo prometheus-community vào helm repo

```
dev@master-1:~$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" already exists with the same configuration, skipping
dev@master-1:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "metric-server" chart repository
...Successfully got an update from the "ingress-nginx" chart repository
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⎈Happy Helming!⎈
dev@master-1:~$
```

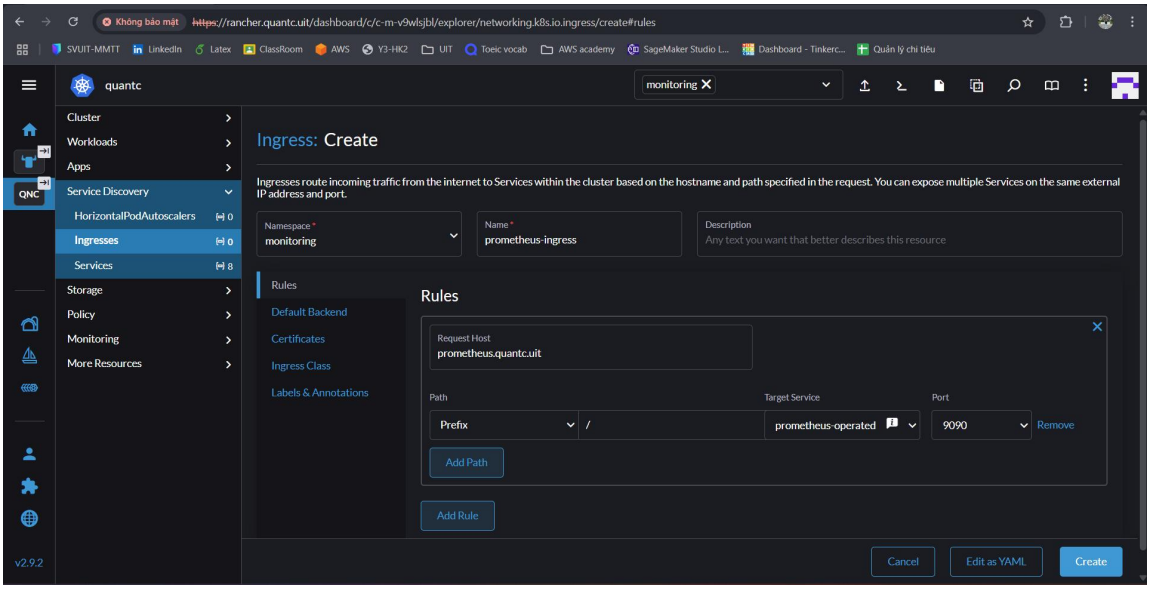
Hình 8: Thêm repo prometheus-community

Tiến hành cài đặt

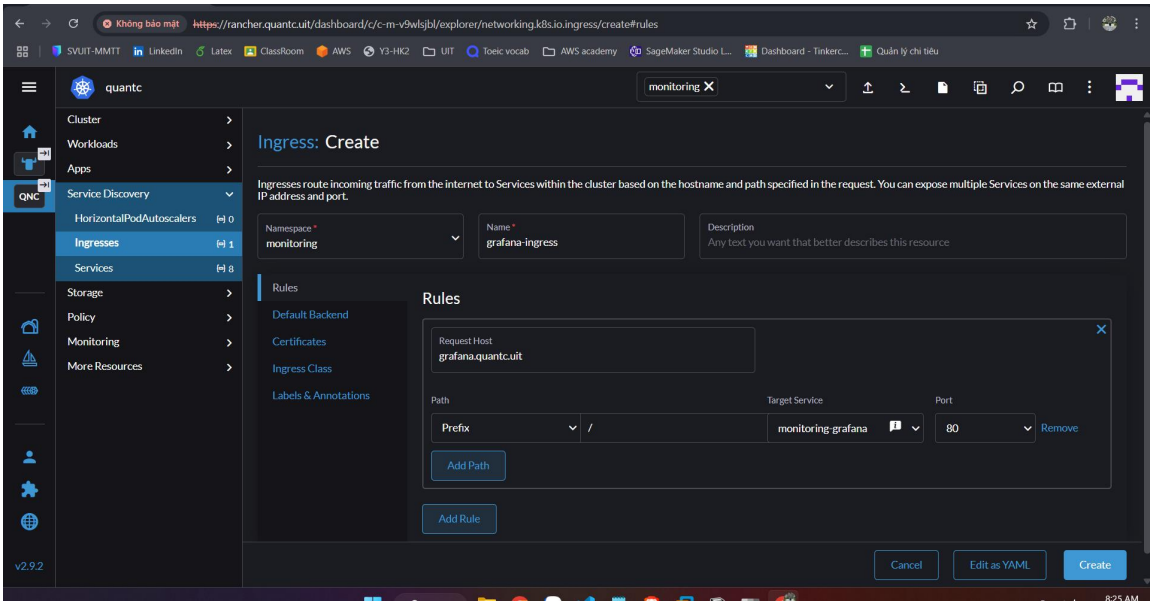
```
dev@master-1:~$  helm upgrade --install monitoring prometheus-community/kube-prometheus-stack \
> --namespace monitoring \
> --set prometheus.prometheusSpec.storageSpec.volumeClaimTemplate.spec.accessModes[0]=ReadWriteOnce \
> --set prometheus.prometheusSpec.storageSpec.volumeClaimTemplate.spec.resources.requests.storage=10Gi \
> --set prometheus.prometheusSpec.storageSpec.volumeClaimTemplate.spec.storageClassName=nfs-storage
Release "monitoring" has been upgraded. Happy Helming!
NAME: monitoring
LAST DEPLOYED: Thu Apr  3 01:06:36 2025
NAMESPACE: monitoring
STATUS: deployed
REVISION: 2
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=monitoring"

Get Grafana 'admin' user password by running:

  kubectl --namespace monitoring get secrets monitoring-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo

Access Grafana local instance:

  export POD_NAME=$(kubectl --namespace monitoring get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=monitoring" -oname)
  kubectl --namespace monitoring port-forward $POD_NAME 3000

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
dev@master-1:~$
```

Hình 9: Cài đặt prometheus-stack

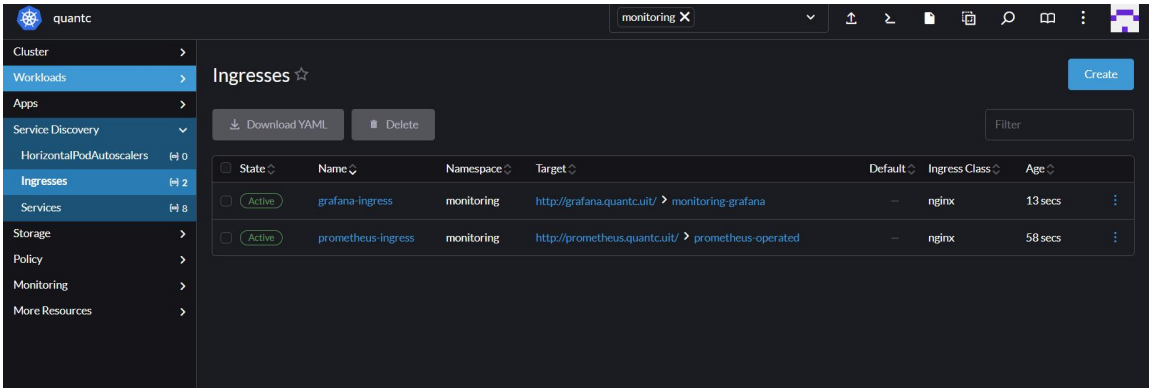
Tạo ingress cho service, lưu ý chọn ingress class là nginx, cho prometheus và grafana để có thể truy cập được trên trình duyệt



Hình 10: Tạo ingress cho prometheus



Hình 11: Tạo ingress cho grafana



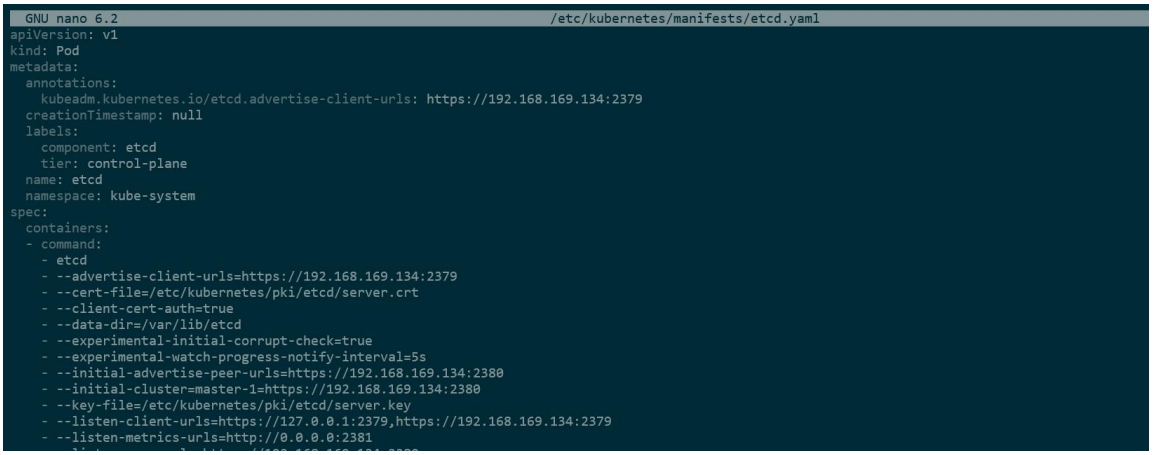
Hình 12: Kiểm tra hai ingress đã được tạo

Kiểm tra prometheus nếu có một số target bị down thì có thể thực hiện một số bước

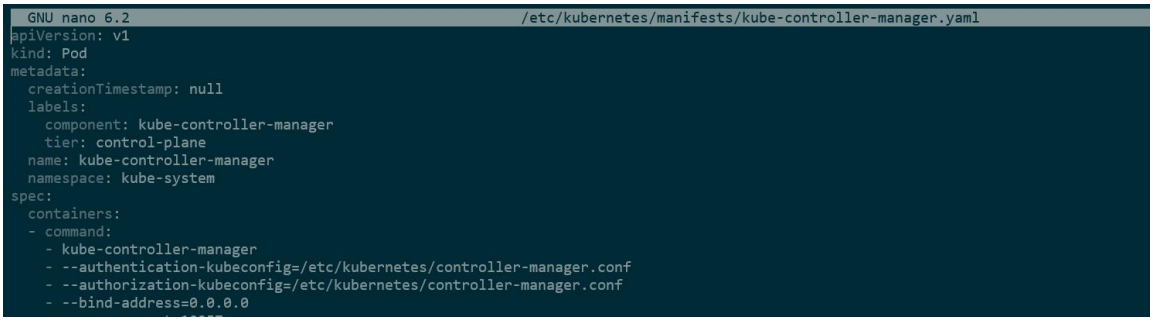
Tạo một file custom-values.yaml sau đó upgrade realease đó lại và trỏ vào file đó. Nội dung code sẽ ở trong file này (Phần 5 - Triển khai kube prometheus stack) . Sau đó chạy lệnh để upgrade

`helm upgrade monitoring prometheus-community/kube-prometheus-stack -n monitoring -f /<>path/custom-values.yaml`

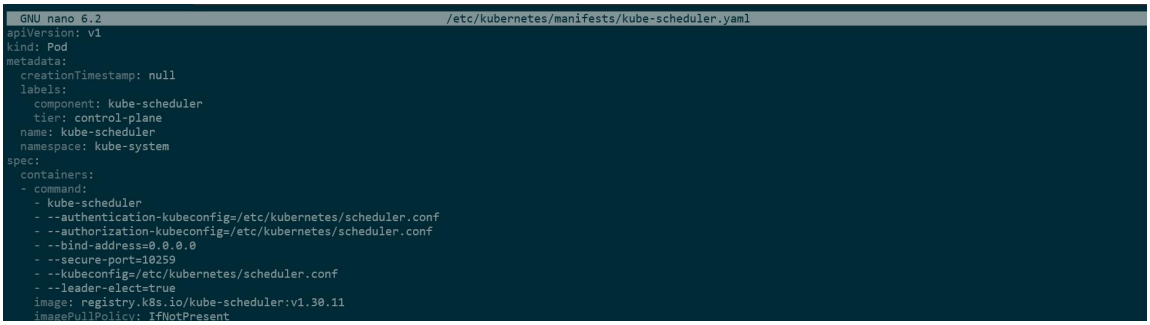
Chỉnh sửa các listen-metric từ 127.0.0.1 → 0.0.0.0, bind-addess cũng tương tự



Hình 13: Chỉnh sửa listen-metrics-url

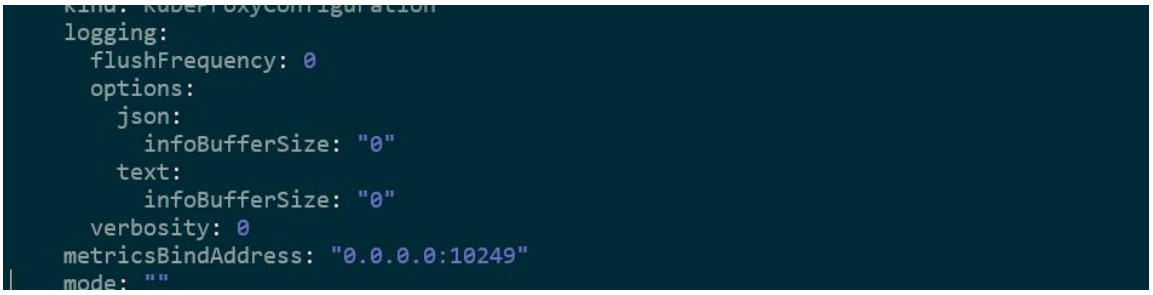


Hình 14: Chỉnh sửa bind-address



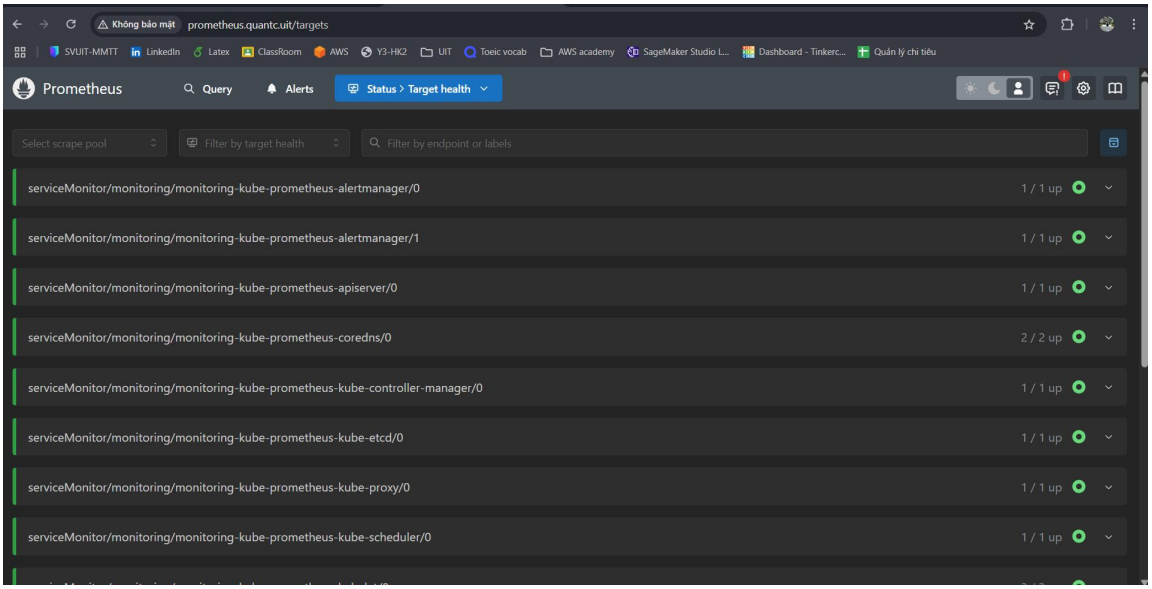
Hình 15: Cỉnh sửa bind-address (2)

Sau khi sửa thì sẽ còn kube-proxy bị down, vào edit config map, thêm 0.0.0.0:10249. Chạy lệnh: kubectl edit configmap kube-proxy -n kube-system -o yaml

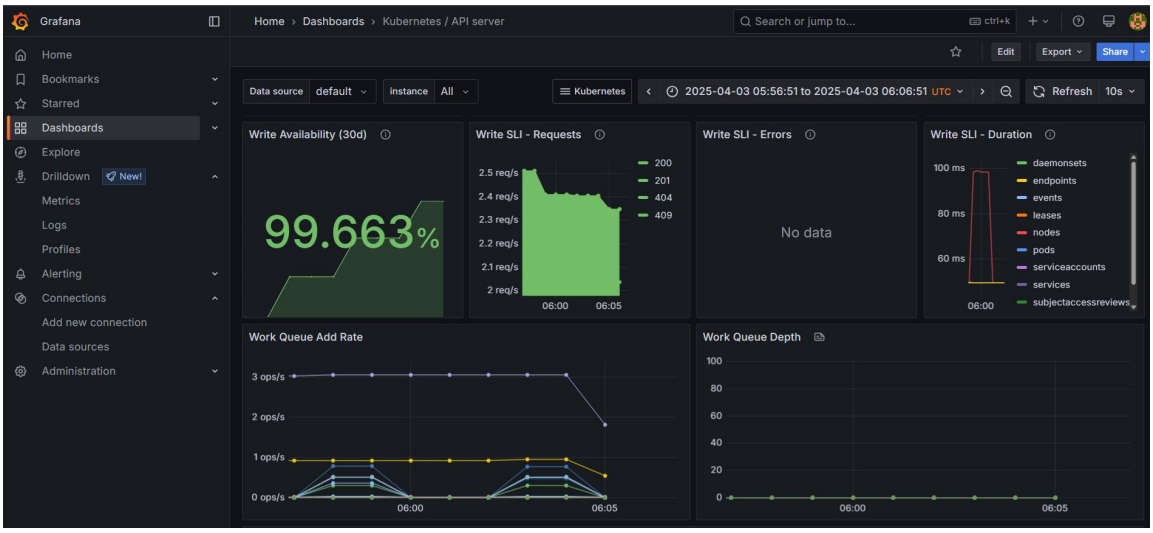


Hình 16: Chỉnh sửa configmap kube-proxy

Vào ingress, truy cập prometheus và grafana tư link của ingress. Lưu ý cần dùng phương pháp add host để có thể truy cập.

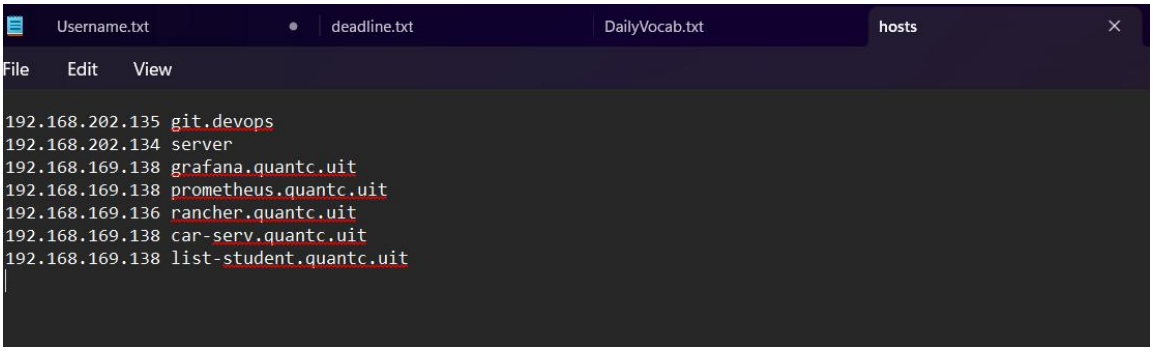


Hình 17: Kiểm tra target health



Hình 18: Xem dashboard API server

Thông tin add host



Hình 19: Thông tin add host

3. Cấu hình gửi cảnh báo đơn giản dựa trên hiệu năng của cụm (giám sát nodes)

Kịch bản: Gửi cảnh báo về email khi có một trong 3 node có CPU hay Memory vượt quá 70%.

Demo: link

Viết một file service kind là Prometheus rule để thiết lập rule (Phần Danh gia hieu nang - Toggle thứ 2), sau đó áp dụng cấu hình

```
dev@master-1:~/custom-monitoring$ kubectl apply -f /home/dev/custom-monitoring/monitor-node.yaml
prometheusrule.monitoring.coreos.com/node-resource-usage-monitoring created
```

Hình 20: Áp dụng cấu hình prometheus rule

Cấu hình alertManager để nhận cảnh báo qua email

```
global:
  smtp_smarthost: 'smtp.gmail.com:587'
  smtp_from: 'tocongquan315@gmail.com'
  smtp_auth_username: 'tocongquan315@gmail.com'
  smtp_auth_password: '<app password>'
route:
  group_by: ['alertname', 'instance']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 5m
  receiver: 'email'
receivers:
- name: 'email'
  email_configs:
  - to: 'tocongquan315@gmail.com'
    send_resolved: true
```

Hình 21: Cấu hình nhận email

Lấy nội dung của file config trên cuyển thành bas64 bằng lệnh: `base64 -w 0 alertmanager.yaml`

Tạo một file cấu hình secret, với <mã base> là mã được base64 ở trên. Sau đó áp dụng cấu hình

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: alertmanager-monitoring-kube-prometheus-alertmanager
  namespace: monitoring
type: Opaque
data:
  alertmanager.yaml: <ma base>
```

Hình 22: Cấu hình secret

Cập nhật cấu hình alert manager trong file cusom-values.yaml, thêm đoạn code vào cuối file

```yaml
alertmanager:
  enabled: true
  config:
    global:
      resolve_timeout: 5m
  alertmanagerSpec:
    configSecret: alertmanager-monitoring-kube-prometheus-alertmanager
```

Hình 23: Thêm đoạn code vào sau trong file custom-values.yaml

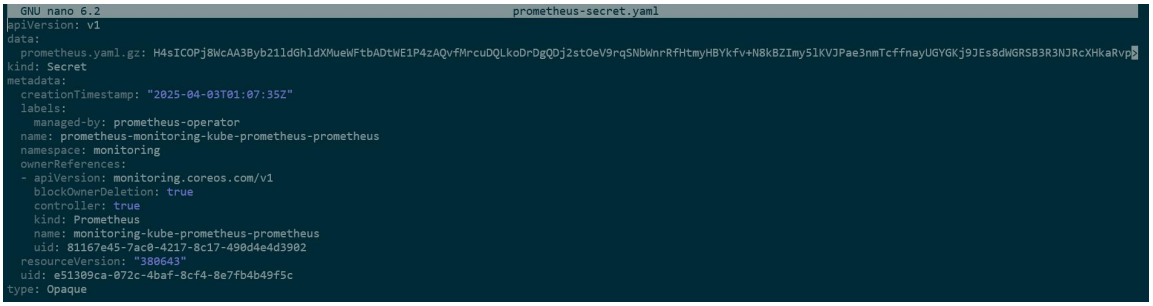
Sau đó cập nhật lại helm

```
dev@master-1:~/custom-monitoring$ helm upgrade monitoring prometheus-community/kube-prometheus-stack -n monitoring -f /home/dev/custom-monitoring/custom-values.yaml
Release "monitoring" has been upgraded. Happy Helming!
NAME: monitoring
LAST DEPLOYED: Sat Apr  5 11:28:42 2025
NAMESPACE: monitoring
STATUS: deployed
REVISION: 13
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=monitoring"

Get Grafana 'admin' user password by running:

  kubectl --namespace monitoring get secrets monitoring-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo

Access Grafana local instance:

  export POD_NAME=$(kubectl --namespace monitoring get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=monitoring" -oname)
  kubectl --namespace monitoring port-forward $POD_NAME 3000

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
dev@master-1:~/custom-monitoring$
```

Hình 24: Cập nhật helm

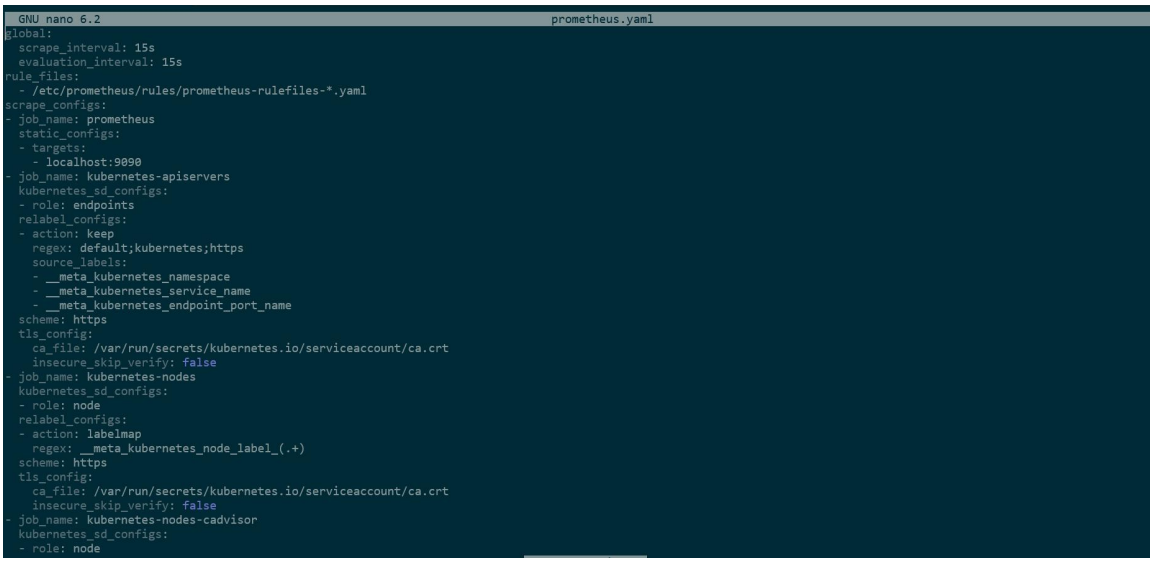
Xuất secret và lưu vào 1 file bằng lệnh

kubectl get secret -n monitoring prometheus-monitoring-kube-prometheus-prometheus -o yaml > prometheus-secret.yaml



Hình 25: File secret sau khi được xuất ra

Lấy dữ liệu đã được mã hoá thành base64 trong phần data chuyển lại thành file yaml. Khi mã hoá xong, tạo một file yaml tên prometheus.yaml sau đó chỉnh sửa thêm phần rule_files: thêm đường dẫn



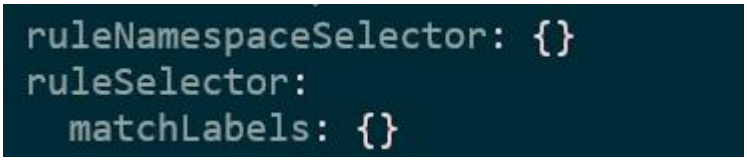
Hình 26: Chỉnh sửa file prometheus.yaml

Tiếp đó sẽ đóng gói file này lại và chuyển sang base64.

Hình 26: Hai file gz và base64

Lấy dữ liệu base64 đó và sửa lại phần data của file prmetheus-secret.yaml và áp dụng cấu hình. Reload lại bằng cách xoá pod prometheus-monitoring, nó sẽ tự động tạo lại pod mới với lệnh: kubectl delete pod -n monitoring prometheus-monitoring-kube-prometheus-prometheus-0

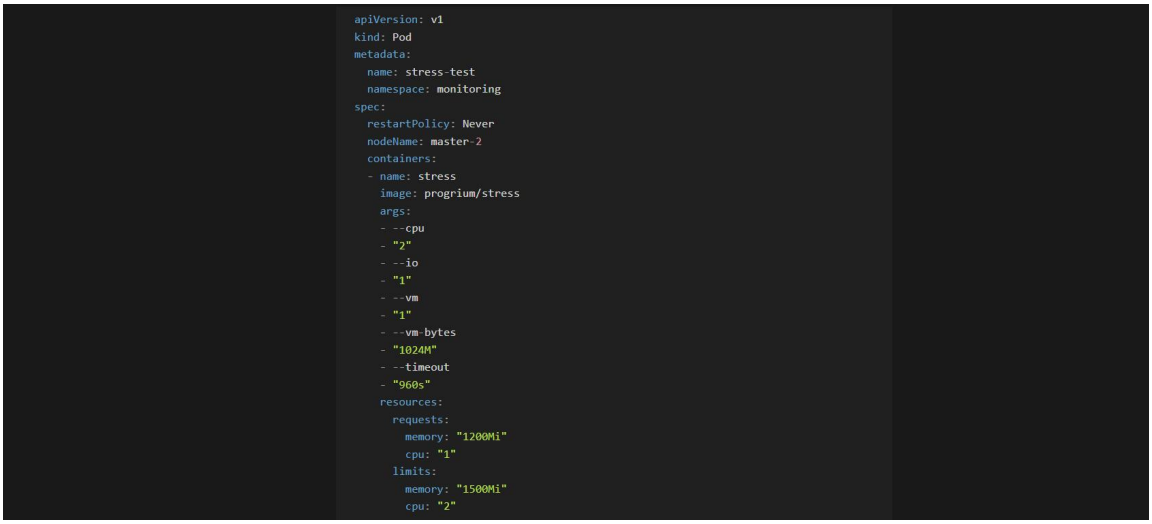
Cần chỉnh lại tham số này là {} để Prometheus Operator sẽ theo dõi tất cả PrometheusRule trong namespace monitoring, chạy lệnh: kubectl edit prometheus -n monitoring monitoring-kube-prometheus-prometheus



Hình 27: Chỉnh sửa tham số {}

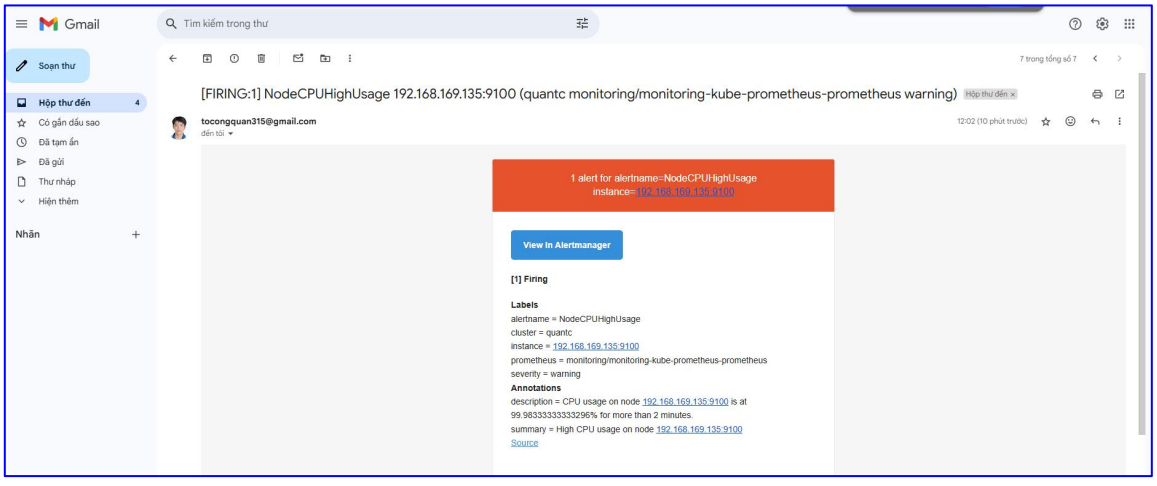
Thử nghiệm kịch bản:

Tạo một pod với image stress node bất kì, ví dụ worker 2. Sau đợi đợi phản hồi email



Hình 28: Thử nghiệm kịch bản 1

Apply cấu hình và đợi nhận email



Hình 29: Email cảnh báo kịch bản 1

4. Triển khai một ứng dụng và giám sát hiệu năng của cụm (giám sát pods)

Kịch bản: Cấu hình cảnh báo khi một deployment nhận quá nhiều request, dẫn đến CPU hoặc Memory tăng cao dẫn đến phải auto scaling autoscaling

Demo: link

Vì kịch bản một nhóm đã cấu hình các bước cần thiết, nên ở kịch bản này chỉ cần cấu hình một file PrometheusRule để bắt sự kiện.

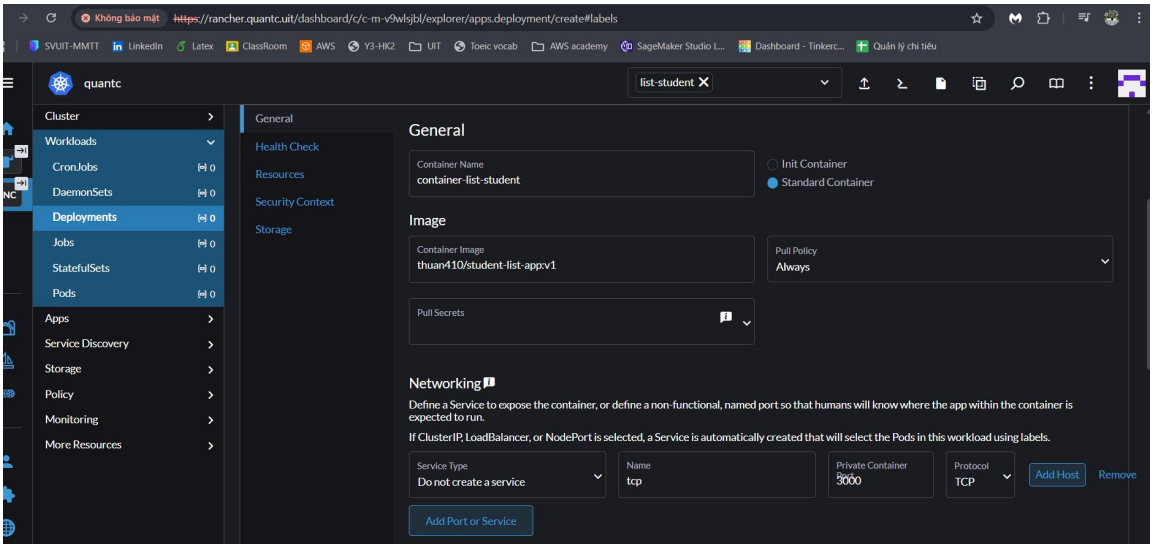
Đầu tiên, cần triển khai một ứng dụng trên cụm k8s

Tạo một namespace riêng cho dự án list-student

```
dev@master-1:~$ kubectl get ns -n list-student
NAME                          STATUS   AGE
cattle-fleet-system           Active   5d12h
cattle-impersonation-system   Active   5d12h
cattle-system                 Active   5d12h
cattle-ui-plugin-system       Active   5d12h
default                       Active   6d12h
ingress-nginx                 Active   36h
kube-node-lease               Active   6d12h
kube-public                   Active   6d12h
kube-system                   Active   6d12h
list-student                  Active   25s
local                         Active   5d12h
my-test-ns                    Active   40h
prometheus                    Active   6d12h
```

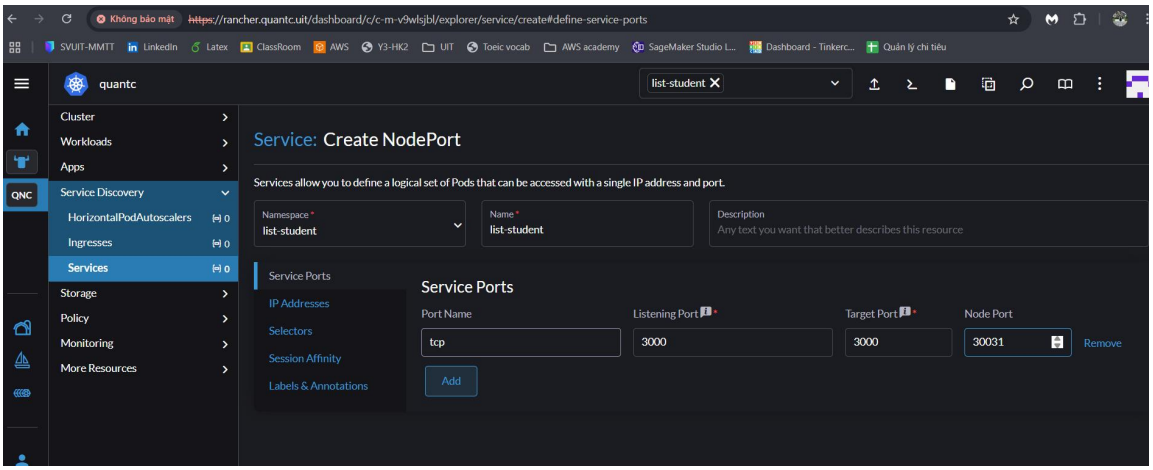
Hình 30: Tạo namespace cho dự án

Vào Rancher server tạo một deployment. Vì hiện tại images trong hình bị lỗi, nên có thể đổi sang image sau: *zquan315/list-student:v1*



Hình 31: Tạo một deployment

Tiếp theo, tạo service cho dự án, theo thực tế thì nên tạo ClusterIP để đảm bảo tính bảo mật, vì với service này, chỉ có ip nội bộ trong cụm mới truy cập được. Tuy nhiên, để thực hiện demo dễ dàng hơn, nhóm sẽ tạo service NodePort

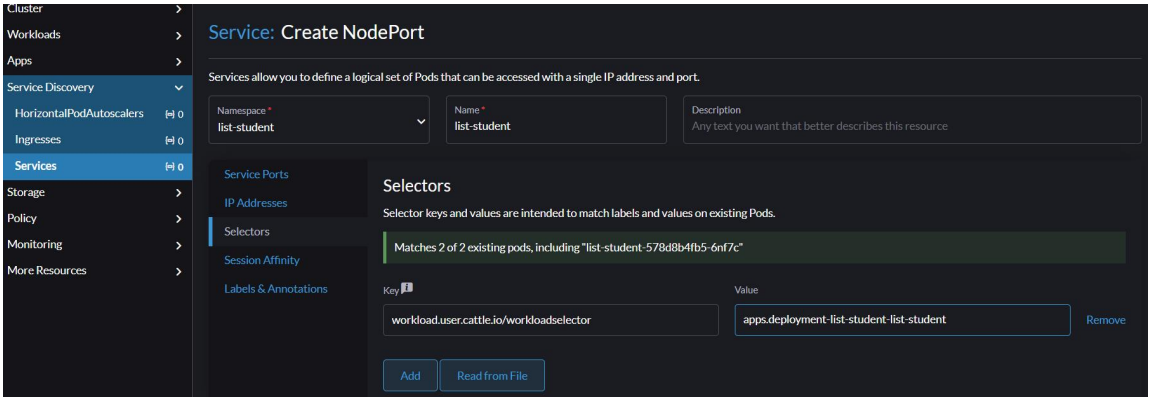


Hình 32: Tạo service NodePort

Ở phần Selectors này, ta sẽ vào file yaml của Deployment của list-tudent đã tạo ở trên để lấy key và value của nó

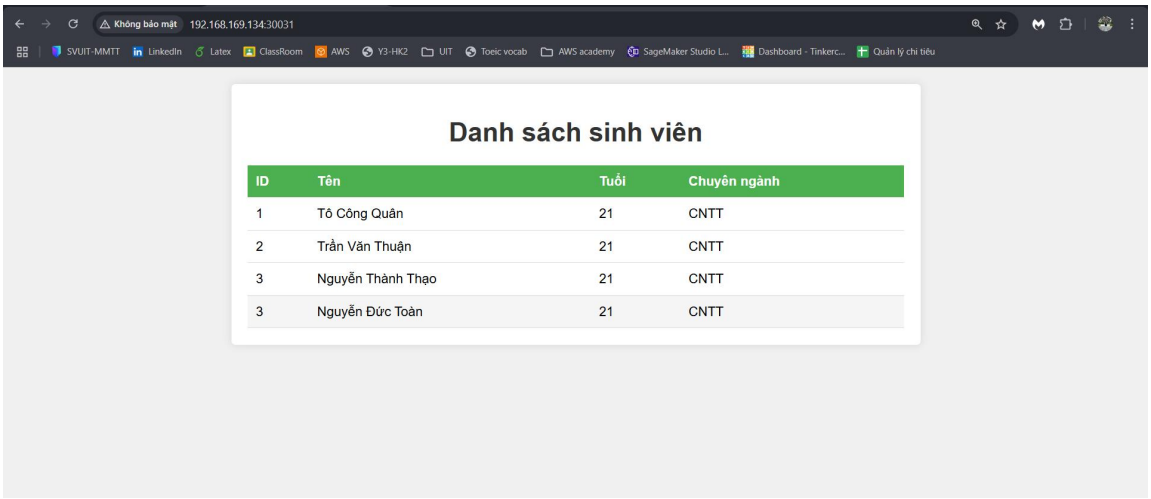
```
110   selector:
111     matchLabels:
112       workload.user.cattle.io/workloadselector: apps.deployment-list-student-list-student
```

Hình 33: Key và values



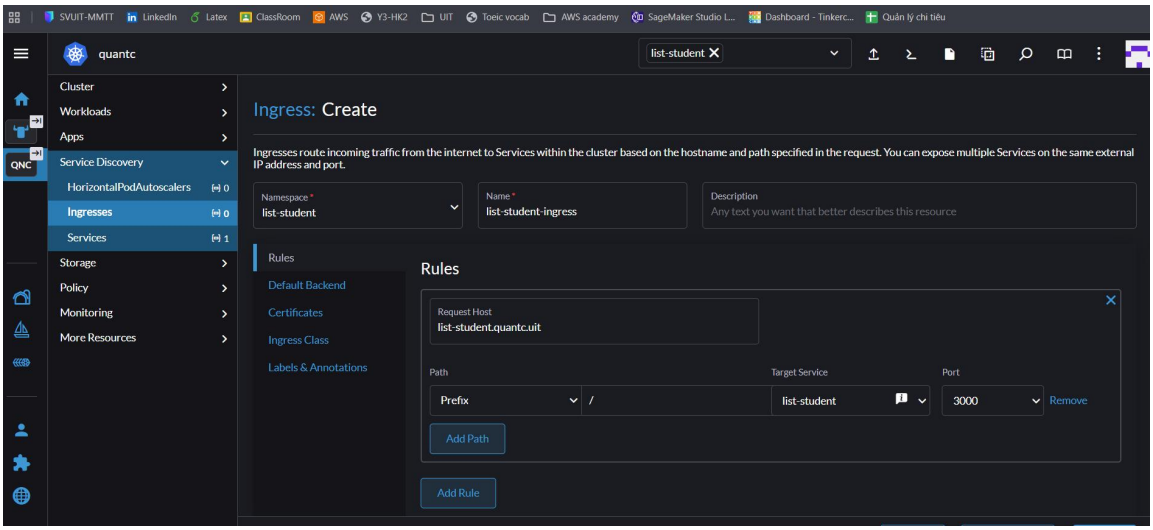
Hình 34: Lấy key và values dán vào

Bây giờ với ip của 3 node trong cụm và port 30031 thì sẽ truy cập được trang web

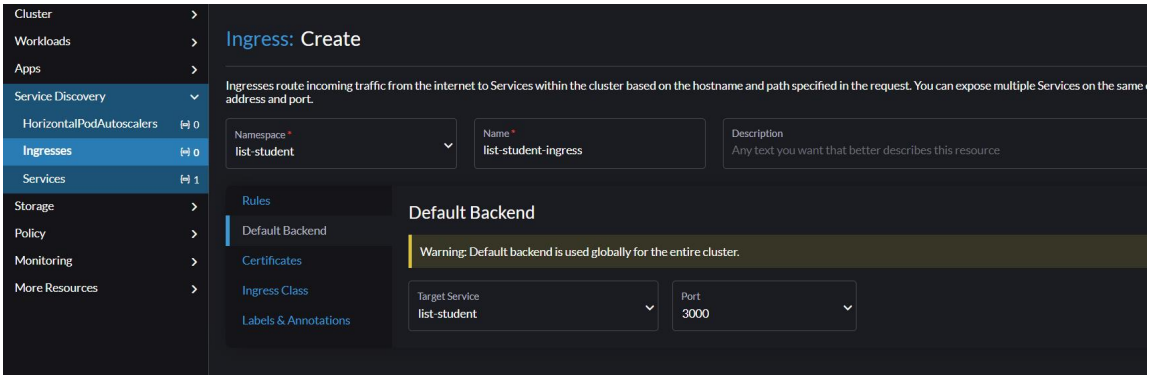


Hình 35: Truy cập ứng dụng

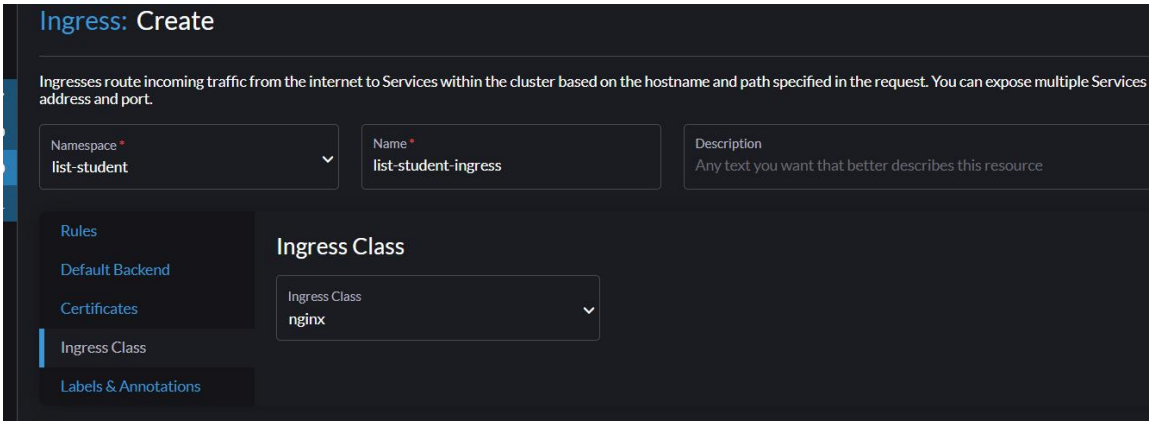
Tạo ingress cho dự án



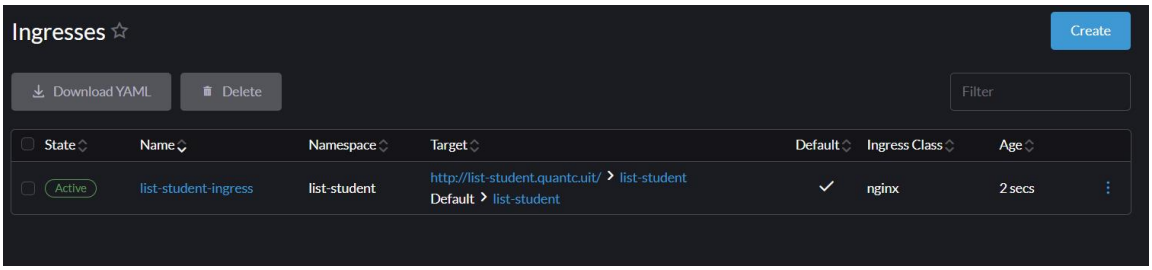
Hình 36: Tạo ingress cho dự án (1)



Hình 37: Tạo ingress cho dự án (2)

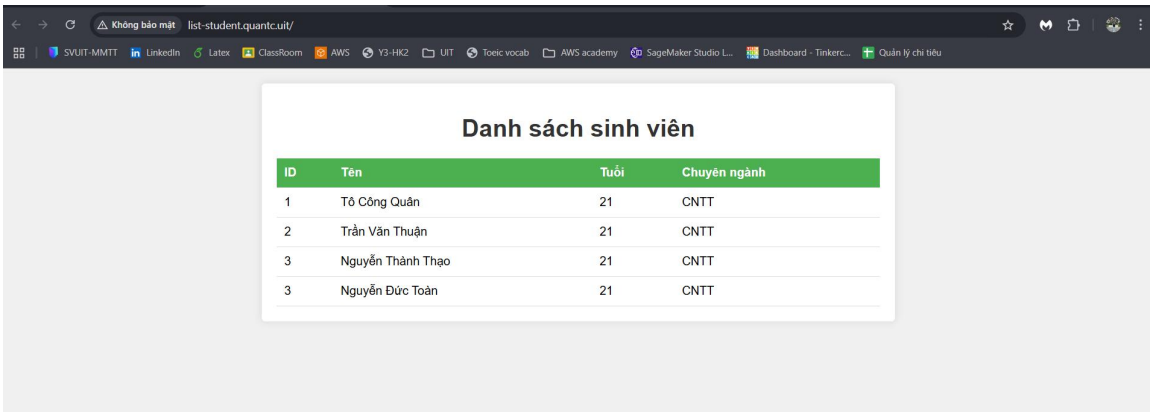


Hình 38: Tạo ingress cho dự án (3)



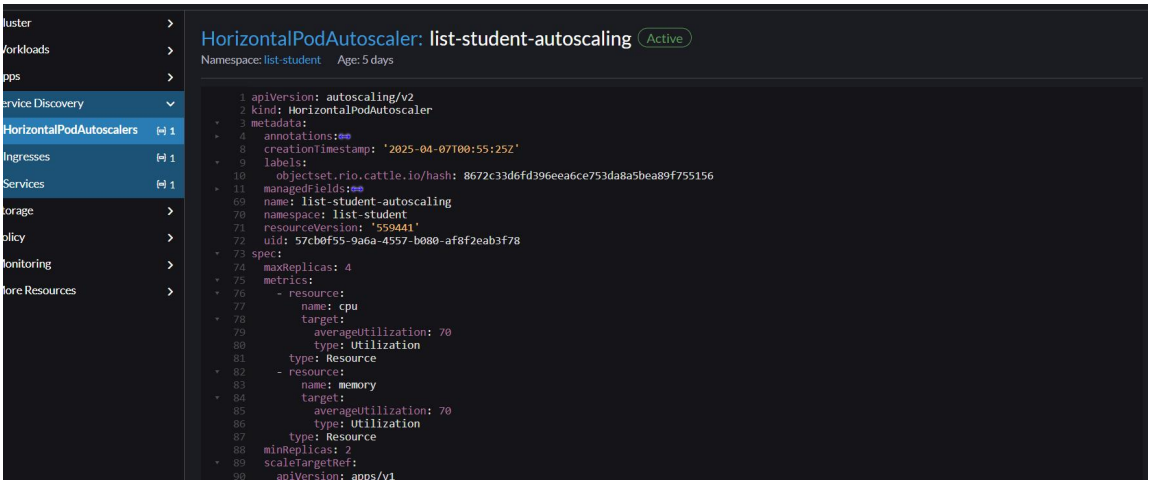
Hình 39: Kiểm tra ingress

Bây giờ chỉ cần nhấn vào link truy cập ở trước dấu > là sẽ mở được trang web



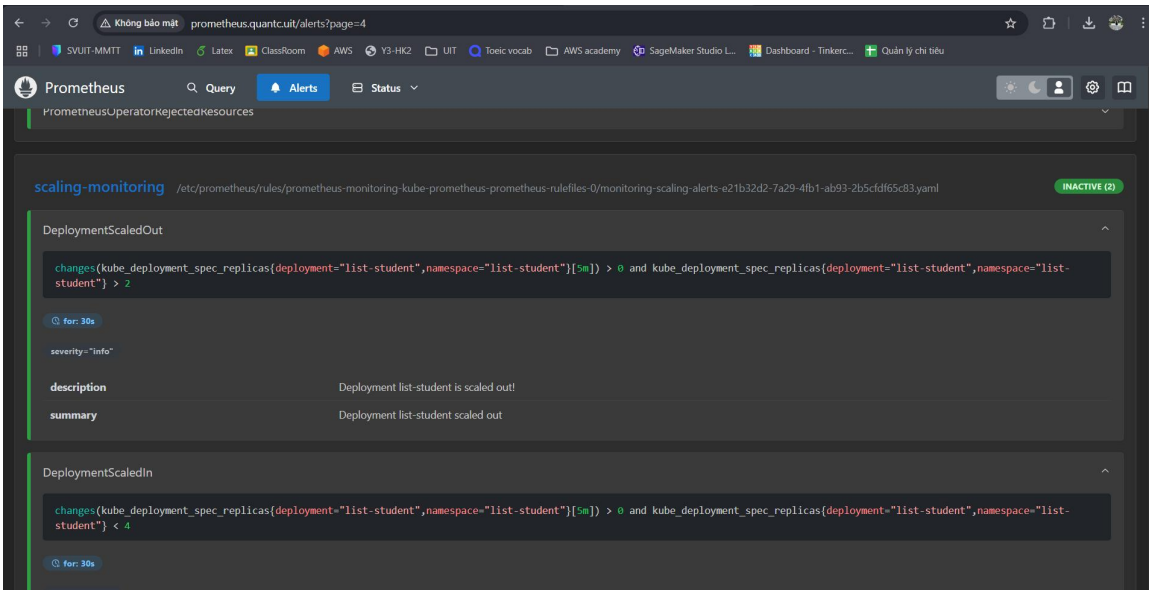
Hình 40: Truy cập web

Tạo một service HPA



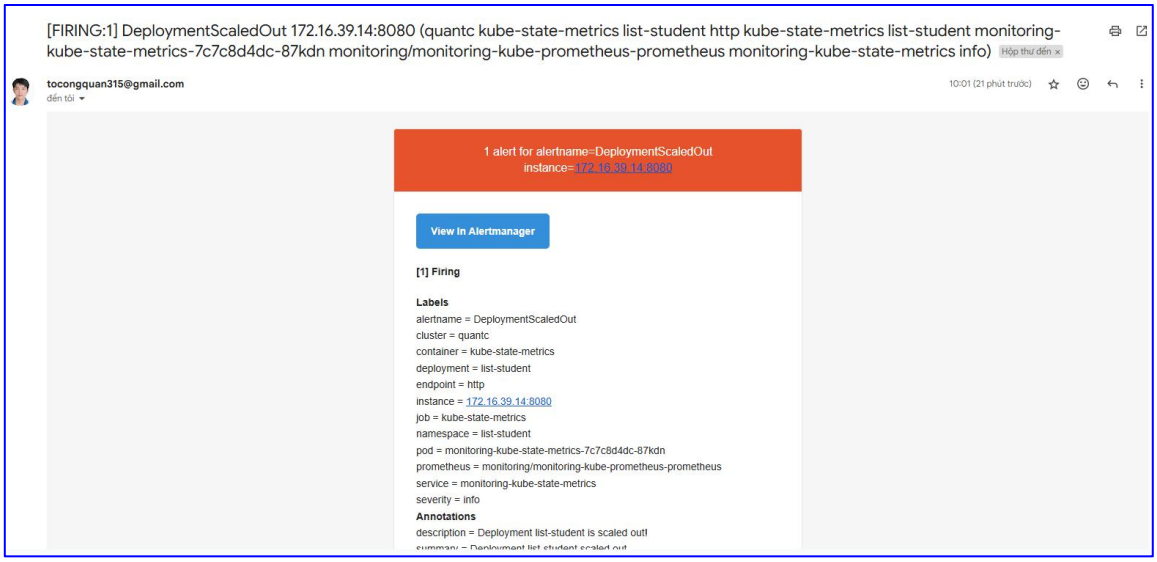
Hình 42: File yaml của HPA

Viết một file prometheusRule để bắt sự kiện (Phần Danh gia hieu nang - toggle thứ 3), áp dụng cấu hình và kiểm tra trên prometheus

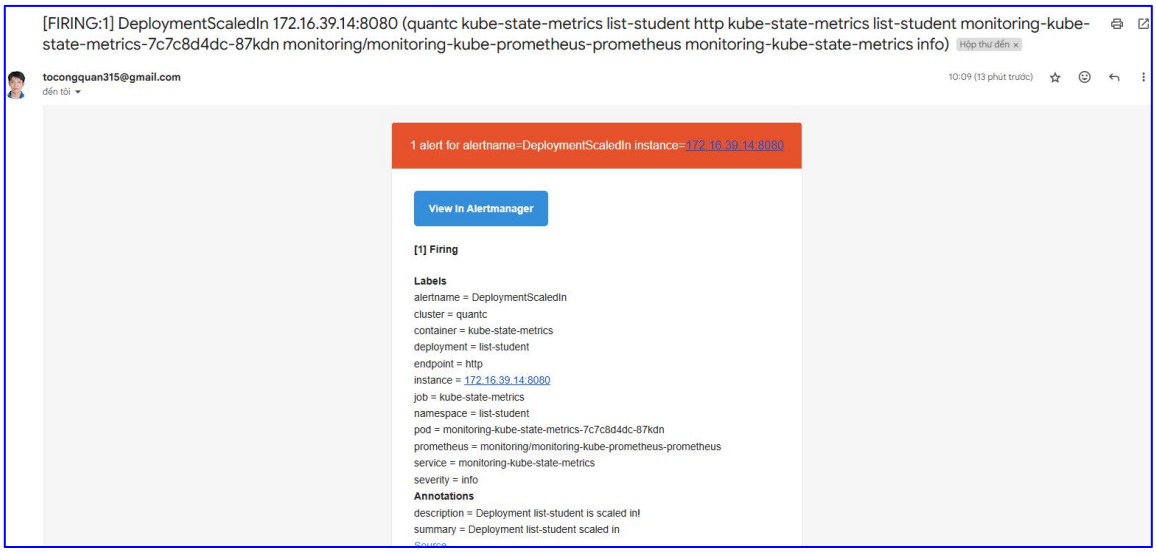


Hình 42: Kiểm tra Rule

Tạo một file để thực hiện lịch bản bằng cách tạo một pod thực hiện curl gửi request liên tục đến web, sau đó áp dụng cấu hình. Đợi một lúc sẽ có email



Hình 43: Email khi scale-out



Hình 44: Email khi scale-in

5. Bổ sung kịch bản - Theo dõi cụm k8s khi shutdown một node

Demo: link

IV. KẾT LUẬN

Đề tài đã xây dựng và triển khai hệ thống giám sát hiệu năng cụm Kubernetes một cách hiệu quả. Việc tích hợp Prometheus và Grafana cho phép nhóm theo dõi tài nguyên theo thời gian thực và phát hiện sớm các bất thường trong hệ thống. Qua các kịch bản thử nghiệm, hệ thống phản hồi tốt, gửi cảnh báo kịp thời và hỗ trợ autoscaling hiệu quả. Đề tài không chỉ giúp nhóm học thêm nhiều kỹ năng kỹ thuật mà còn củng cố kiến thức về kiến trúc hệ thống phân tán và công cụ giám

sát. Tuy nhiên, vẫn còn tiềm năng cải thiện, như bổ sung cảnh báo nâng cao hoặc tích hợp thêm công cụ AI để phân tích dự đoán. Kết quả đạt được là bước đầu quan trọng hướng tới xây dựng các hệ thống giám sát tự động và thông minh trong thực tế.

BẢNG PHÂN CÔNG

STT	Họ và tên	MSSV	Phân công
1	Tô Công Quân	22521190	Viết báo cáo, làm slide, cài đặt máy ảo, cài đặt kubernetes, cấu hình HPA để auto scaling
2	Nguyễn Thành Thạo	22521371	Cài đặt prometheus và grafana, cấu hình prometheus để nhận các metrics từ các node và pod
3	Nguyễn Đức Toàn	22521490	Cài đặt prometheus và grafana, cấu hình prometheus để nhận các metrics từ các node và pod
4	Trần Văn Thuận	22521448	Thiết lập các rules cảnh báo về hiệu năng của các node và gửi về email

TÀI LIỆU THAM KHẢO

- **Devopsedu (Khoá kubernetes từ cơ bản đến thực tế)**
- **Notion (Kubernetes: Basic to practical)**