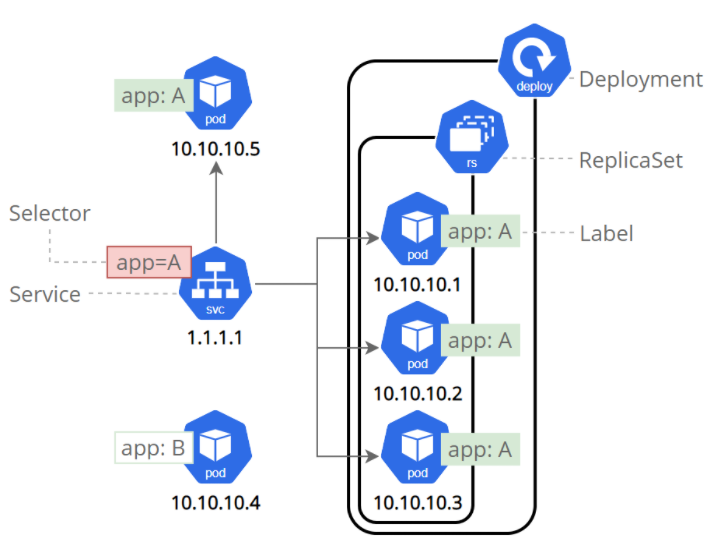


****

**Cài đặt docker và kubenetes**

1)**Cài đặt docker**

* Docker
* Docker-io
* Docker-daemon
* Docker-compose

Cài đặt docker-engine

https://docs.docker.com/engine/install/ubuntu/

# sudo apt-get update && sudo apt-get install apt-transport-https ca-certificates curl software-properties-common -y

2)**Thêm Docker GPG key và apt repository.**

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

3) **Cài đặt Docker community edition**

sudo apt-get update && apt-get install docker-ce

4) **Thêm cấu hình docker daemon**.

cat > /etc/docker/daemon.json <<EOF

{

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2"

}

EOF

5) **Tạo thư mục service**

mkdir -p /etc/systemd/system/docker.service.d

6) **Khởi động lại Docker service**

systemctl daemon-reload

systemctl restart docker

7) **Cài đặt Kubeadm & Kubelet & Kubectl**

**Cài đặt các phụ thuộc cần thiết**

sudo apt-get update && sudo apt-get install -y apt-transport-https curl

8) **Thêm GPG key.**

Thêm:

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add –

# chmod 777 /etc/apt/sources.list.d/

Bình thường có thể khi **apt search** các gói của ubuntu sẽ ko có 1 số ứng dụng như kubeadm, kubelet,… là ta chỉ có thể cài n qua đường dẫn web hoặc update n vào danh sách của apt ubuntu.

Tất cả các gói cài đặt của ubuntu là apt install sẽ có trong đường dẫn

**/ etc/apt/sources.list.d/** và ta sẽ cho thêm các đường dẫn gói kubeadm, kubelet vào danh sách này của ubuntu

Cấp quyền và tạo file kubernetes.list trong đường dẫn này. Đường dẫn ở ảnh dưới



Sau đó sử dụng apt update và search lại

Đã có tools kube



9) **Thêm kubernetes apt repository.**

cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list

deb https://apt.kubernetes.io/ kubernetes-xenial main

EOF



10) **Cập nhật apt và cài đặt kubelet, kubeadm và kubectl.**

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

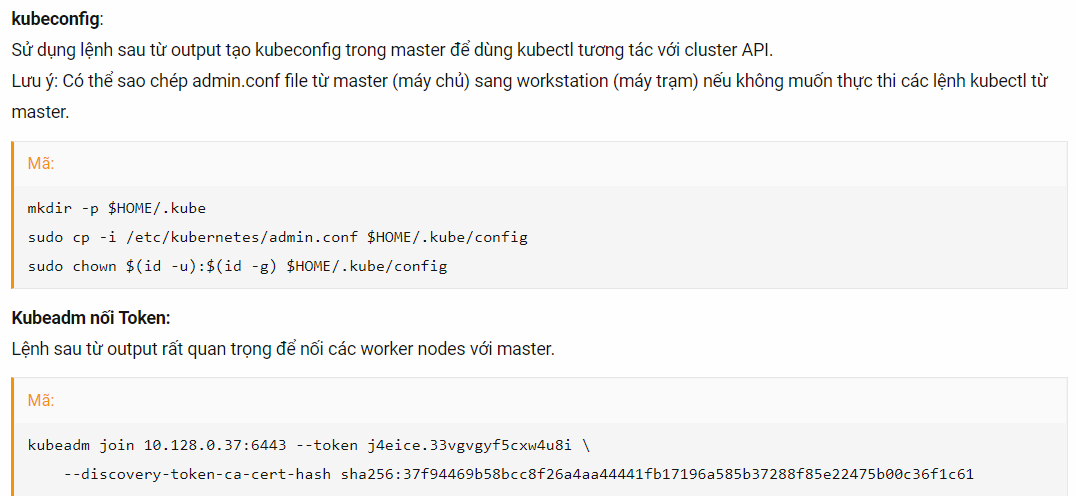
* Bây giờ đã có tất cả phương tiện và công cụ cần thiết để định cấu hình các thành phần Kubernetes bằng kubeadm.

**11) Khởi tạo Kubeadm trên Master Node**

Khởi tạo kubeadm trên master node với lệnh sau. Nó sẽ thiết lập tất cả Kubernetes master components

sudo kubeadm init --pod-network-cidr=192.168.0.0/16

sau đây sẽ có mã và nối tocken



12) **Cài đặt Calico Network Plugin**

kubectl apply -f https://docs.projectcalico.org/v3.8/manifests/calico.yaml

# Cài kubectl, kubeadm

Kiểm tra đã có các kube chưa

# Which kubeadm



Cd vào /user/bin

# cd kube (++ tab tab)



Khi cài process CPU của VM là 1 thì nhân CPU máy sẽ là 2

Và phải =2 thì mới tạo được máy ảo và chạy lệnh kubeadm init

# egrep -c '(vmx|svm)' /proc/cpuinfo



Lỗi host or port ?



Fix

sudo mkdir /etc/docker

cat <<EOF | sudo tee /etc/docker/daemon.json

{​​​​​​​​

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {​​​​​​​​

"max-size": "100m"

}​​​​​​​​,

"storage-driver": "overlay2"

}​​​​​​​​

EOF

​

sudo systemctl enable docker

sudo systemctl daemon-reload

sudo systemctl restart docker

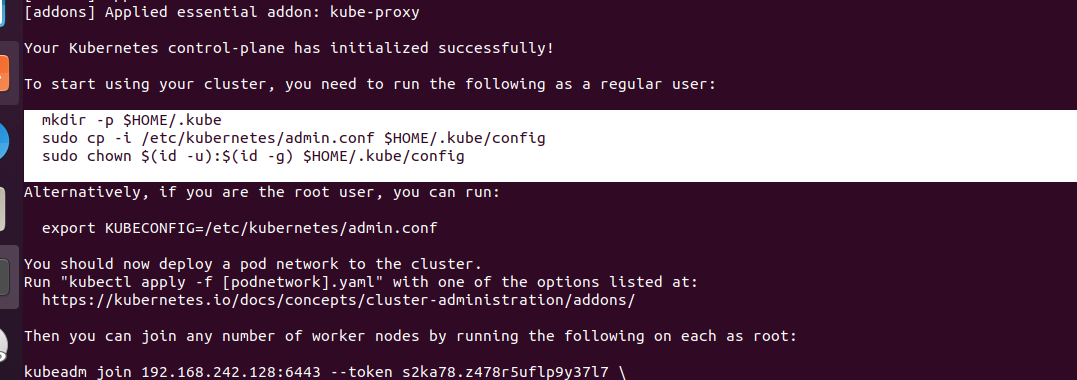
# Xóa swap : vì cái này ko chạy trên swap

swapoff -a

kubeadm reset

rm -rf /root/.kube

kubeadm init



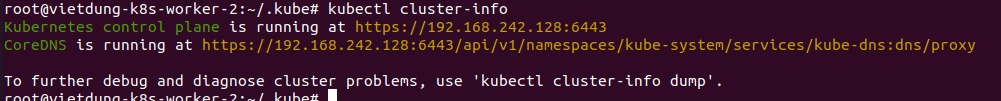
Xog cop tiếp 3 dòng kia ốp vào

 oke đã chạy

Con server là kubernetes cluster

* Xem thông tin về kubernetes cluster

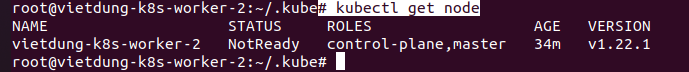
# kubectl cluster-info



Kia là địa chỉ để các API n liên lạc

* Kiểm tra trong cluster này có những node nào

# kubectl get node (mới có 1 node là master, chính là con local đang chạy)



Khi báo NotReady thì sử dụng lệnh này

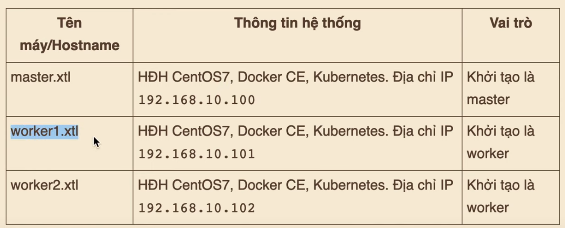
# kubectl apply -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

* Kiểm tra thông tin chi tiết của node đó

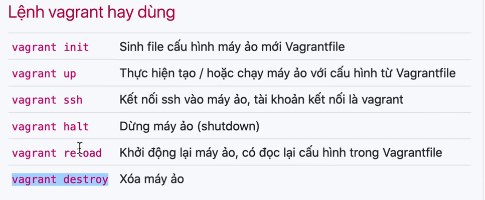
# kubectl describe node/vietdung-k8s-worker-2

Triển khai hệ thống hoàn chỉnh có nhiều node

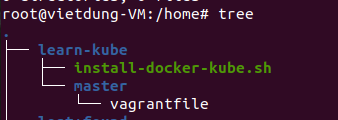
Chúng ta tạo kubernetes cluster có 3 máy



Công cụ tương tác với virtualbox là vagrant : tự động hóa tạo các máy ảo

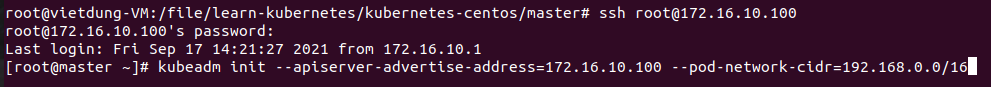


vagrantfile: file chứa cấu hình máy ảo chúng ta cần tạo



Ssh vào node master chạy lệnh

kubeadm init --apiserver-advertise-address=172.16.10.100 --pod-network-cidr=192.168.0.0/16



kubectl apply -f https://docs.projectcalico.org/v3.10/manifests/calico.yaml

* Lệnh lấy đường dẫn kết nối node worker

# kubeadm token create --print-join-command

* xem các node có trong máy (xem chi tiết)

# kubectl get no -o wide

* xem thông tin node

# kubectl describe no/[NodeName]

* label tương tự như các tab để add vào trong các node (miêu tả 1 số t.tin)
* thêm nhãn

# kubectl label node vietdung-vm nodeabc=dechayphp

vietdung-vm: là tên của node

nodeabc=dechayphp: tên nhãn để gắn vào node

* lấy node có nhãn nào đó

# kubectl get node -l “nodeabc=dechayphp”

* xóa nhãn

# kubectl label node vietdung-vm nodeabc-

**I.chuyển đổi xem các node ở console của máy local**

* Xem nội dung cấu hình hiện tại của kubectl

kubectl config view

* Lấy file config cluster

Tại máy master ở trên, có file cấu hình cho tại /root/.kube/config, ta copy file cấu hình này ra lưu thành file config-mycluster (không ghi đè vào config hiện tại của máy HOST)

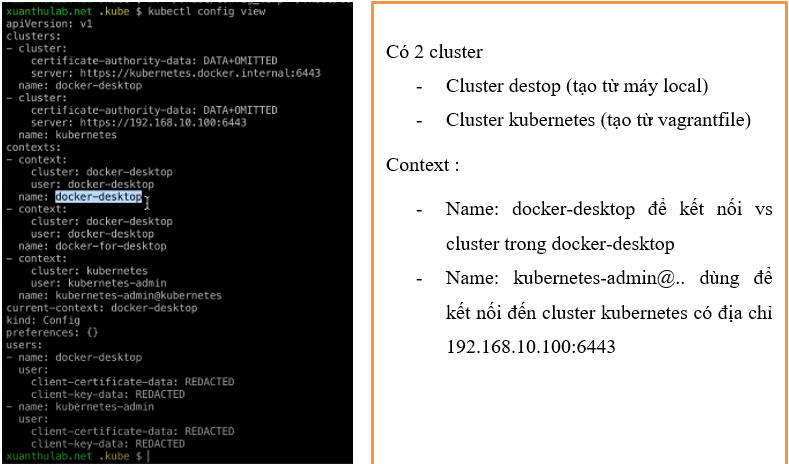
scp root@172.16.10.100:/etc/kubernetes/admin.conf ~/.kube/config-mycluster

* Trộn 2 file config cluster của destop và cluster của master.xtl

export KUBECONFIG=~/.kube/config:~/.kube/config-mycluster

kubectl config view --flatten > ~/.kube/config\_temp

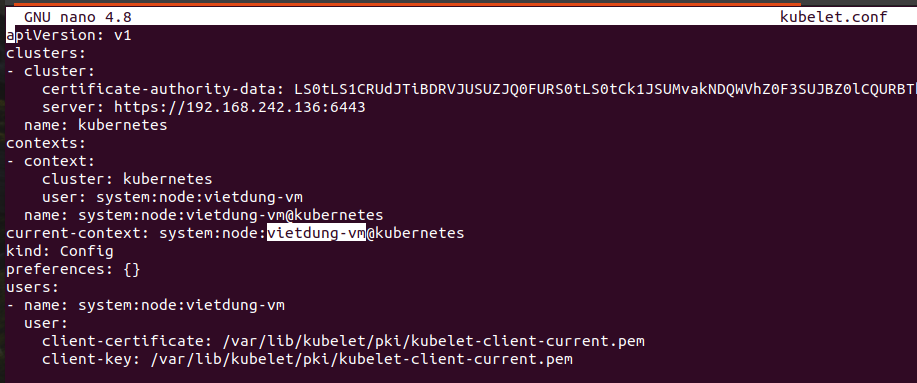
mv ~/.kube/config\_temp ~/.kube/config

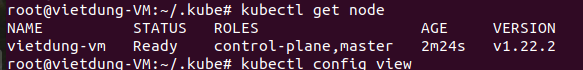


Ngữ cảnh làm việc hiện tại bgio là **current-context** là docker-destop. Nghĩa là kubectl đang làm việc với docker-destop. Tức là đang ở console của máy local thì kiểm tra node ta chỉ xem được node cluster của máy local (docker-destop)

Lưu ý: file cấu hình của host name kube này nằm ở file

**/etc/kubernetes/ kubelet.conf** và n đọc ở file cấu hình này

****



* Để xem kubectl có những ngữ cảnh nào

# kubectl config get-contexts



Dấu \* ở đầu là đang làm việc với docker-desktop

* Đổi ngữ cảnh làm việc (kết nối đến cluster nào)

# kubectl config use-context kubernetes-admin@kubernetes

# Tạo dashboard

Tải file này về lưu thành tên dashboard-v2-beta6.yaml

curl https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta6/aio/deploy/recommended.yaml > dashboard-v2-beta6.yaml

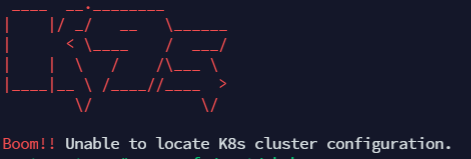
open-vm-tools-desktop

hoặc

open-vm-tools

# K9S

Lỗi khi khởi động k9s



* Kiểm tra lại xem cluster có bị down ko

kubectl get nodes

nếu down thì tạo lại kube cluster trên master

sau đó:

export KUBECONFIG=$HOME/.kube/config

sudo vi /etc/profile

export KUBECONFIG=$HOME/.kube/config

mkdir /root/.k9s

chmod 777 /root/.k9s

oke sau đó chạy lại k9s

# Kubectl

Lệnh **kubectl** tương tác với Cluster, cú pháp chính

kubectl [command] [TYPE] [NAME] [flags]

Trong đó:

* [command] là lệnh, hành động như apply, get, delete, describe ...
* [TYPE] kiểu tài nguyên như ns, no, po, svc ...
* [NAME] tên đối tượng lệnh tác động
* [flags] các thiết lập, tùy thuộc loại lệnh

Gán nhãn cho node worker1

# kubectl label no worker1 nodeabc=dechayungdungphp

Liệt kê node nào có nhãn kia

# kubectl get no -l "nodeabc=dechayungdungphp"

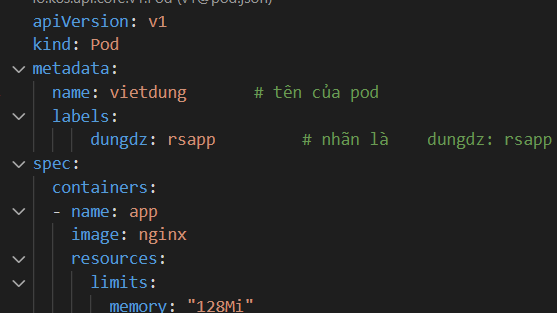
Xóa nhãn

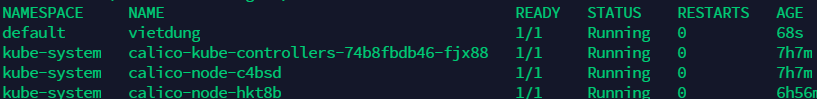
# kubectl label no worker1 nodeabc-

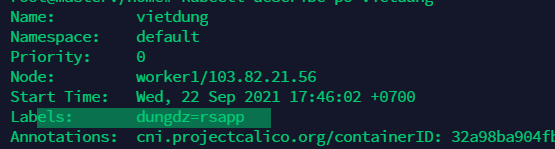
Xóa pod

# kubectl detele po [podname]

File yaml







# Pode / Node

Tạo trực tiếp

# kubectl create deploy nginx-vietdung --image=nginx --replicas=2

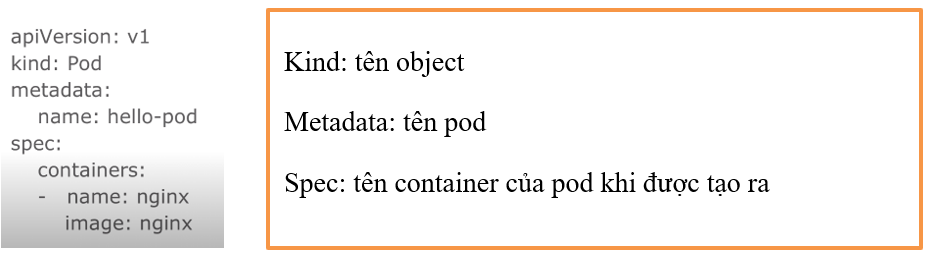
kubectl get po -o wide

Định dạng file pod



Cấu trúc file yaml

Gồm 4 thành phần



* Xem file yaml của pod

kubectl get po/[podname] -o yaml

* Chỉnh sửa pod trên giao diện console

kubectl edit po [PodName] (dạng vi)

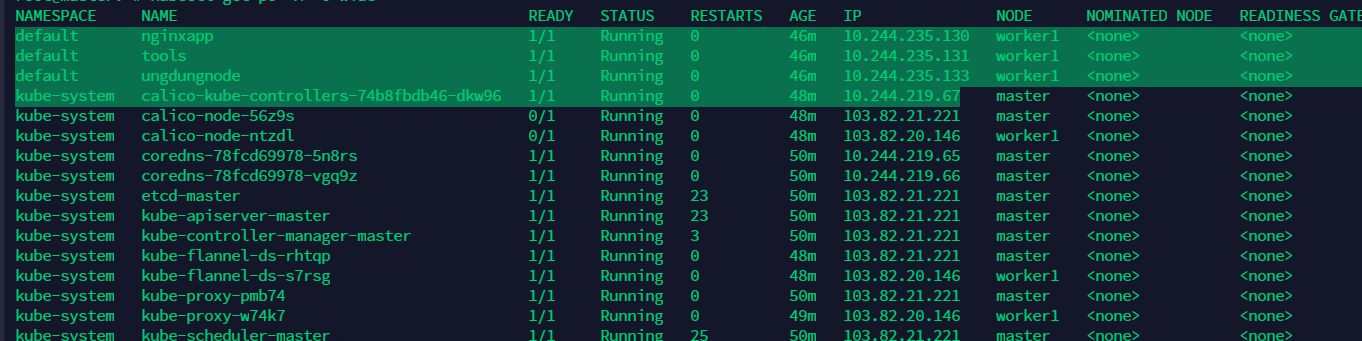
kubectl edit po [PodName] -o yaml (dạng nano)

* Vào container của pod

kubectl exec -it [PodName] -c [containerName] bash

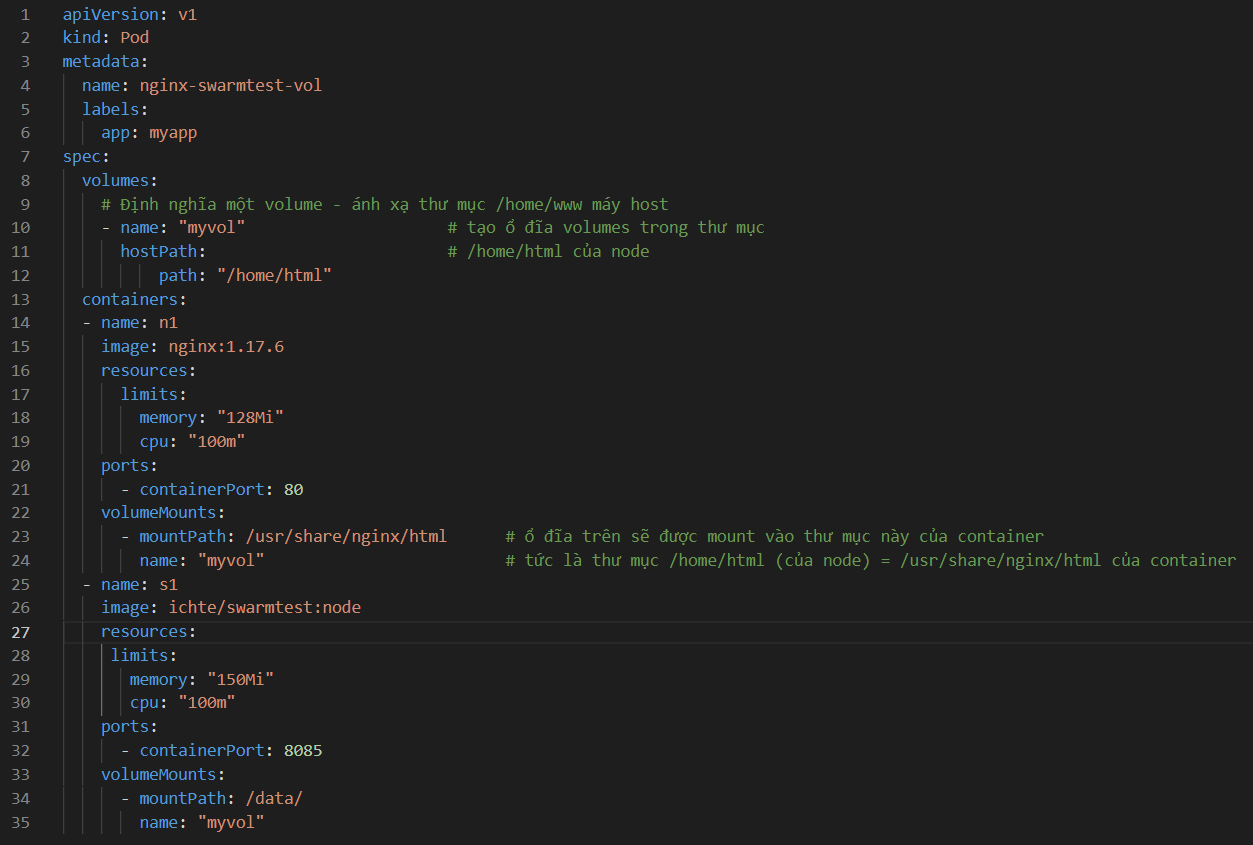
nếu pod đó có chạy nhiều container thì phải chỉ định danh là vào container nào của pod, nếu ko có mặc định nó sẽ vào container đầu tiên

IP:

 địa chỉ IP 10.244.x.x/16 là ip trong mạng nội bộ của cluster

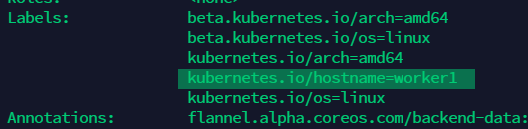
Và các pod này thông với nhau

Tạo pod 2 container



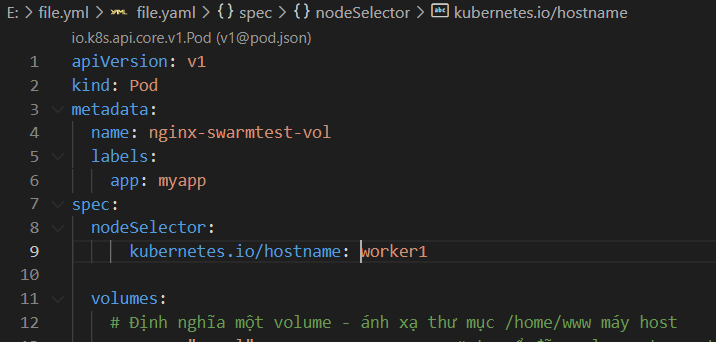
* ấn định 1 pod khi tạo ra chỉ chạy trên 1 node nào đó thôi

B1: kiểm tra cái label (nhãn nào đặc trưng cho node worker1)



Labels này mang giá trị của node worker1 và chỉ worker1 mới có nhãn này

Thêm vào file yaml của pod đó



# Replicaset

Phần **selector** là các nhãn của **pod** để **selector** chuyển hướng yêu cầu đến



replicaSet là kiểu scale pod theo số lượng (cố định 2, 3 pod)

Deployment là kiểu scale pod theo thông số ram, cpu, có thể tăng, giảm số lượng pod, có thể update, rollback service

**Deployment** quản lý một nhóm các Pod - các Pod được nhân bản, nó tự động thay thế các Pod bị lỗi, không phản hồi bằng pod mới nó tạo ra. Như vậy, deployment đảm bảo ứng dụng của bạn có một (hay nhiều) Pod để phục vụ các yêu cầu.

**Deployment** sử dụng mẫu Pod (Pod template - chứa định nghĩa / thiết lập về Pod) để tạo các Pod (các nhân bản replica), khi template này thay đổi, các Pod mới sẽ được tạo để thay thế Pod cũ ngay lập tức.

* kiểm tra danh sách replicaSet

# `kubectl get rs -o wide

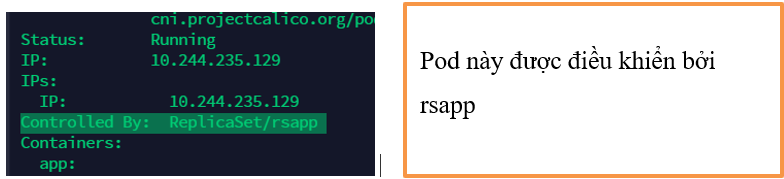
* kiểm tra tất cả các pod, rs deployment, service

# kubectl get all -o wide

Selector là miêu tả các nhãn để chọn ra các pod mà ReplicaSet quản lý



Kiểm tra pod: 3 pod này được giám sát và điều khiển bởi ReplicaSet



* liệt kê các pod có nhãn là app=rsapp

# kubectl get pod -l "app=rsapp"

* Xóa nhãn trong pod

# kubectl label pod rsapp-4n8qc app-

app- : key của nhãn app=rsapp (key-value : label)

* khi xóa các ReplicaSet thì các pod quản lý bởi replicaSet đều bị xóa

# kubectl delete rs rsapp

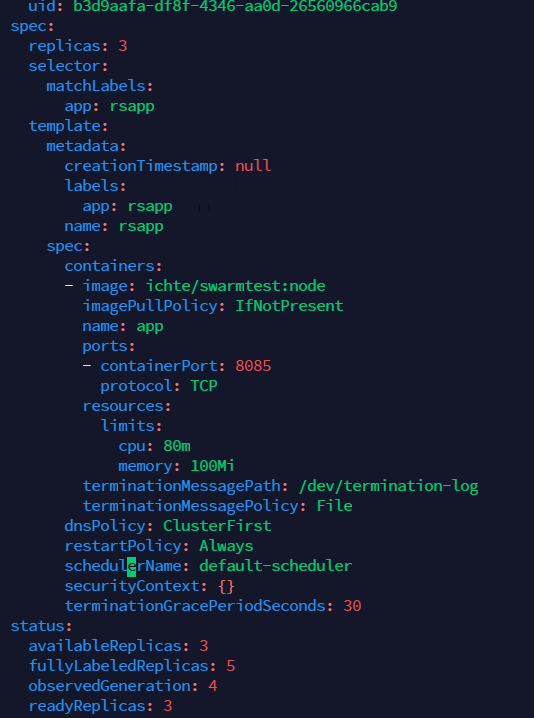
**Lưu ý**: khi tạo 1 pod mà pod đó có nhãn nằm trong sự quản lý replicaSet thì khi tạo Pod đó thì pod đó ko tạo được thì số lượng Pod của replica đã đủ

HPA

Chỉnh sửa ReplicaSet trực tiếp

# kubectl edit rs rsapp

rsapp: là tên của ReplicaSet, có hoặc ko cũng được



Chỉ cần chỉnh mỗi thông số replicas: 3 thành 3 còn lại phần status thì ko cần chỉnh, n tự cập nhật

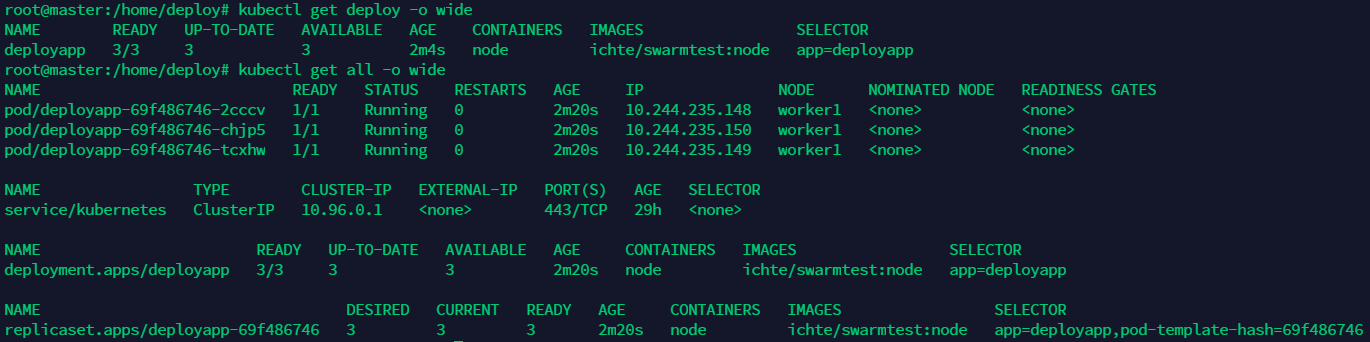
# Deployment

**Deployment** quản lý một nhóm các Pod - các Pod được nhân bản, nó tự động thay thế các Pod bị lỗi, không phản hồi bằng pod mới nó tạo ra. Như vậy, deployment đảm bảo ứng dụng của bạn có một (hay nhiều) Pod để phục vụ các yêu cầu.

**Deployment** sử dụng mẫu Pod (Pod template - chứa định nghĩa / thiết lập về Pod) để tạo các Pod (các nhân bản replica), khi template này thay đổi, các Pod mới sẽ được tạo để thay thế Pod cũ ngay lập tức.

* kiểm tra các deploy

# kubectl get deploy -o wide



Khi deploy tạo ra nó tạo ra replicaSet

Deploy cũng có nhãn để quản lý

Khi ReplicaSet tạo ra, nó tạo ra các Pod

Điểm khác biệt của **deploy**

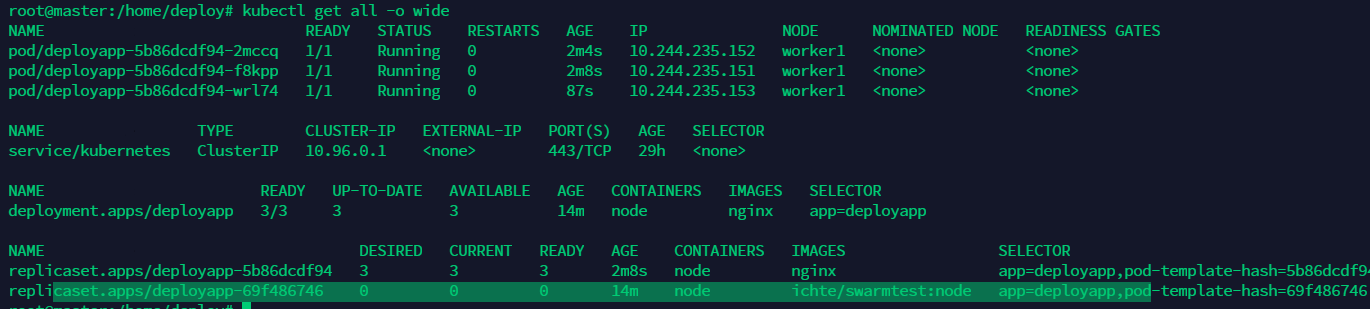
Khi chỉnh sủa lại file deploy hoặc edit file yaml deploy

* chỉnh sửa lại file deploy.yaml bằng nano thì apply lại nó
* hoặc kubectl edit deploy [deployName]
* thay thử image cũ của n bằng image mới là nginx

khi thay đổi như thế nó xóa hết các pod, replicaSet cũ, đồng thời nó sẽ tạo ra các pod- replicaSet mới

nhưng điều đặc biệt là deployment này ko xóa replicaSet đó đi mà giữ lại

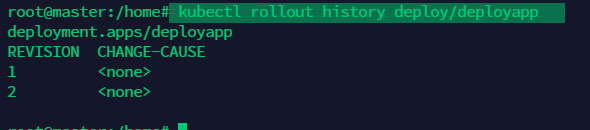
mục đích: để rollback



Replica cũ đã bị xóa hết nhưng vẫn tồn tại

* để xem các số lần cập nhật

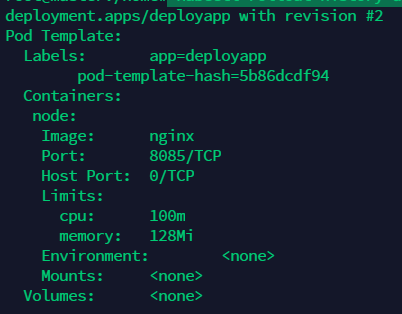
# kubectl rollout history deploy/deployapp



Số lần cập nhật tính từ trên xuống dưới. vd 2 là lần cuối cùng

* xem số lần cập nhật thứ 2

# kubectl rollout history deploy/deployapp --revision=2

 thông số

* quay trở về bản cập nhật số 1 (roolback)
* Cập nhật image mới trong POD - ví dụ thay image của container node bằng image mới httpd
* kubectl set image deploy/deployapp node=httpd --record
* Để xem quá trình cập nhật của deployment
* kubectl rollout status deploy/deployapp
* Khi cập nhật, ReplicaSet cũ sẽ hủy và ReplicaSet mới của Deployment được tạo, tuy nhiên ReplicaSet cũ chưa bị xóa để có thể khôi phục lại về trạng thái trước (rollback).
* Bạn cũng có thể cập nhật tài nguyên POD theo cách tương tự, ví dụ giới hạn CPU, Memory cho container với tên app-node
* kubectl set resources deploy/deployapp -c=node --limits=cpu=200m,memory=200Mi

deployapp là tên deploy

Khi cần quay lại phiên bản cũ nào đó, ví dụ bản revision 1

kubectl rollout undo deploy/deployapp --to-revision=1

Nếu muốn quay lại bản cập nhật trước gần nhất

kubectl rollout undo deploy/deploy

scale pod (thay đổi chỉ số replica (số lượng POD) của Deployment)

nó cũng tương tự như scale của replica nhưng vì deployment ưu điểm của nó có thể rollback

* scale thủ công

# kubectl scale deploy/deployapp --replicas=10

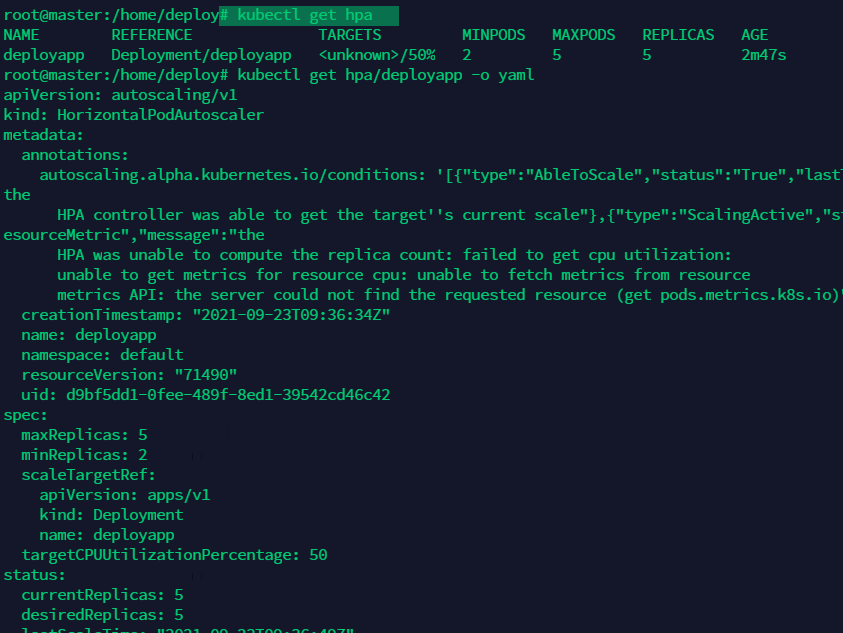
* scale tự động (số lượng pod tối thiểu, tối đa)

kubectl autoscale deploy/deployapp --min=2 --max=5 --cpu-percent=50



Khi thực hiện lệnh này nó tự tạo ra 1 HPA autoscale

Vì nó cũng đã tạo ra hpa rồi, ta có thể lưu lại cái HPA yaml này vào 1 file



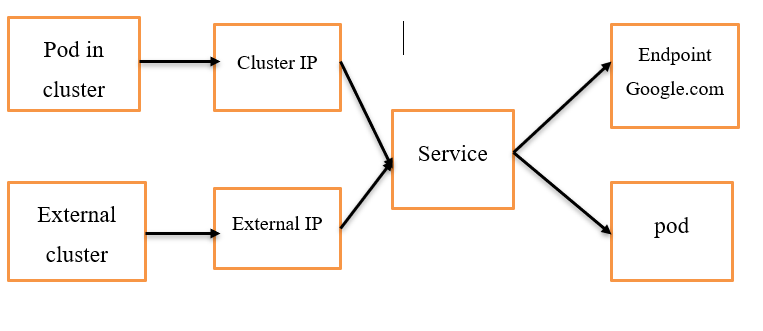
Lưu file hpa này lại

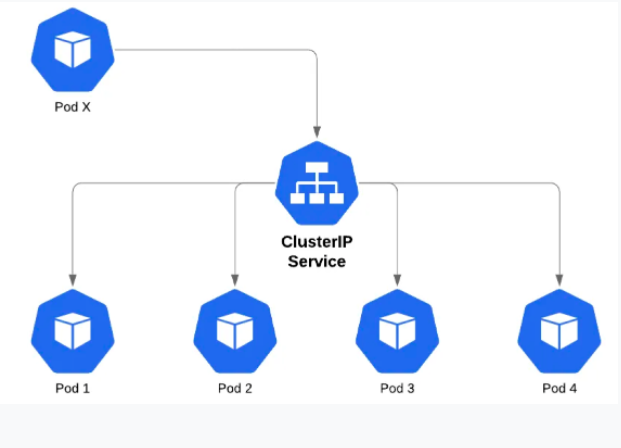
# kubectl get hpa/deployapp -o yaml > 2.hpa.yaml

**Metrics Server trên Kubernetes**

Metrics Server: giám sát tài nguyên như cpu, bộ nhớ của pod, node

# Service

****



Service trong kubernetes hđ như 1 hệ thống cân bằng tải (như proxy)

1 servive được tạo ra trong kubernetes có tên, 1 địa chỉ ip, qua đó chúng ta gửi các truy vấn, các yêu cầu đến các service, server trên các pod thông qua service thay vì truy vấn đến từng pod 1, từng ip của pod đó (mà chưa kể mỗi khi scale các pod này lại có 1 ip mới, các ip là ko cố định )

* ta phải dùng tới service: service tương tự như frontend của hệ thống cân bằng tải. service sẽ chuyển hướng những yêu cầu đến các pod mà n quản lý

khi Service này chuyển hướng đến các server, service bên trong đều gọi là Endpoint (các endpoint này tương tự là các backend)

khi service tạo ra thì type mặc định là clusterIP

có 4 kiểu type

* ClusterIP
* ExternalName
* LoadBalancer
* NodePort

File yaml của service

apiVersion: v1

kind: Service

metadata:

  name: svc1

spec:

  type: ClusterIP

  ports:

    - name: port1

      port: 80

      targetPort: 80

mà hiện tại file service này ko thiết lập **selector** tức là label (label sinh ra để service, deploy, replica quản lý các pod của mình)

nếu ko khai báo label thì thằng service này mặc định n tìm trên cluster, hệ thống có endpoint nào có tên là svc1 không. Nếu endpoint đó có tên cùng tên với service thì n lấy endpoint đó là endpoint của service

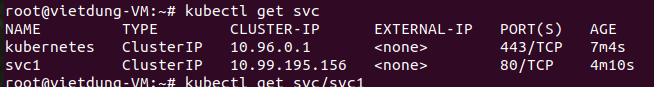
file yaml của endpoint

apiVersion: v1

v

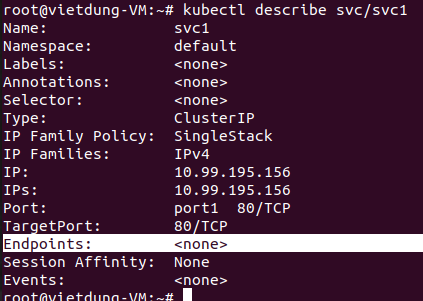
* Kiểm tra các service có trên cluster

# kubectl get svc

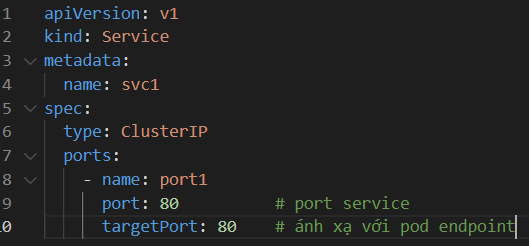
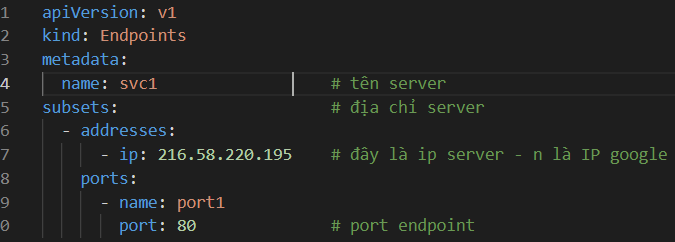


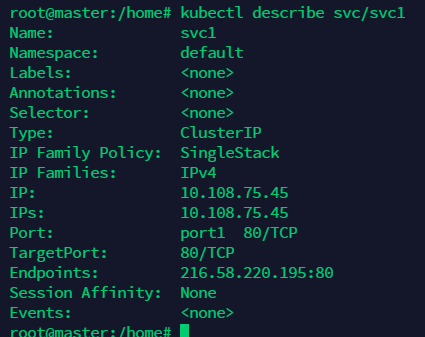
* Kiểm tra thông tin của service

# kubectl describe svc/svc1

 chưa được gắn với endpoint nào

Khai báo endpoint cho service này



Bgio gửi resquest tới service port 80 thì đồng nghĩa với việc yêu cầu đó được gửi tới endpoint port 80

# Phần 2: tạo ra 2 Pod làm endpoint

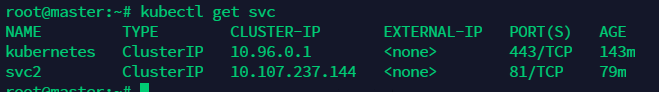
Tạo 2 pod, 1 pod chạy nginx, 1 pod chạy httpd và làm endpoint cho thằng service

Khi request đến ip của service thì n sẽ đến các pod, n là mô hình frontend-backend

* Service là font-end
* Các pod chạy container là backend



Vào worker1 curl đến ip của service



# curl 10.107.237.144:81

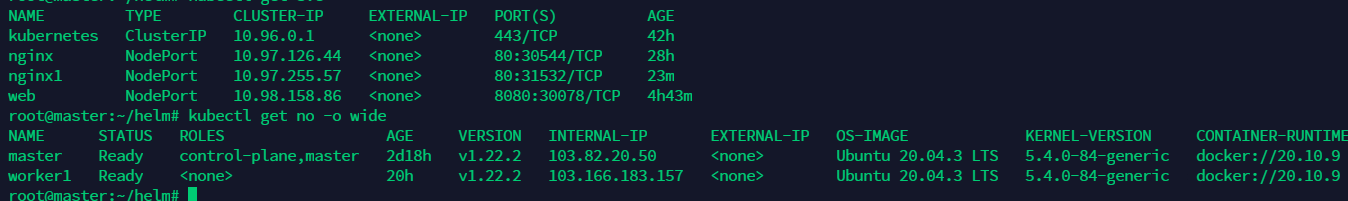
Và ta truy cập service này bằng địa chỉ cluster IP

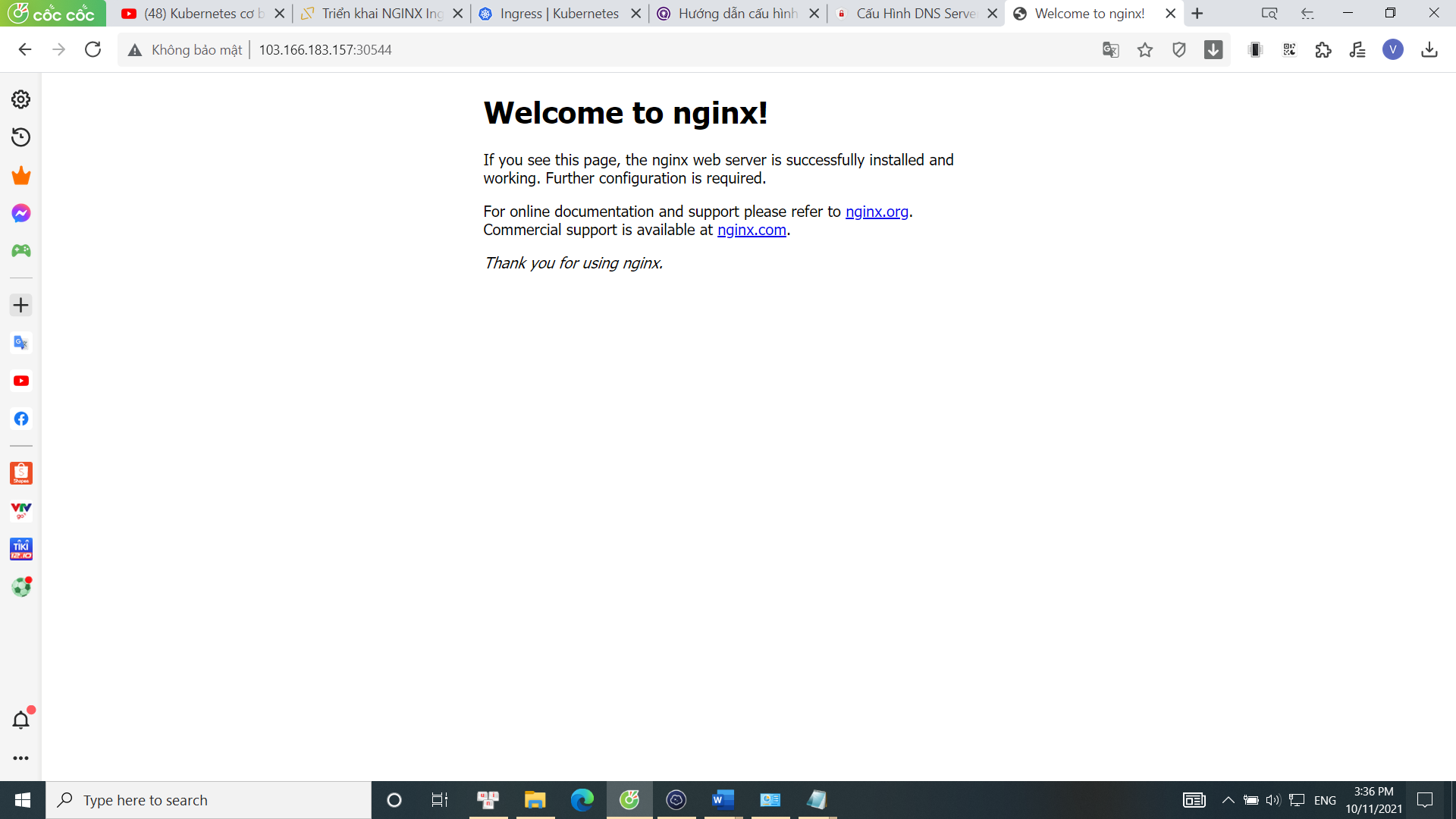
ở phần sau t sẽ thêm 1 cách truy cập bằng ip của chính node với kiểu Type là NodePort

# Phần 3: nodePort

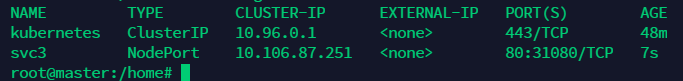
## Service

Expose 1 trang web ra public có thể sử dụng service NodePort



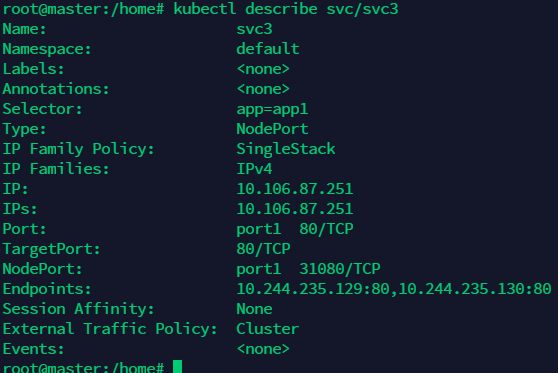


Tạo svc3



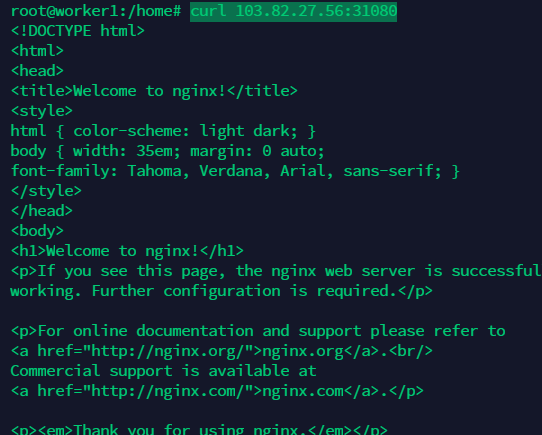
Kiểu nodePort

thiết lập kiểu với **type: NodePort**, lúc này Service tạo ra có thể truy cập từ các IP của **Node** với một cổng nó ngẫu nhiên sinh ra trong khoảng **30000-32767**.



Và 2 pod đó đang chạy trên worker1 nên ta có thể curl đến pod này thông qua ip worker1 và port là 31080

Trên worker1: curl 103.82.27.56:31080

 103.82.27.56 là ip của node worker1

**Thêm**

apiVersion: v1

kind: Service

metadata:

  name: svc3

spec:

  selector:

     app: app1       # labels để map với 2 thằng container ở trên

  type: NodePort

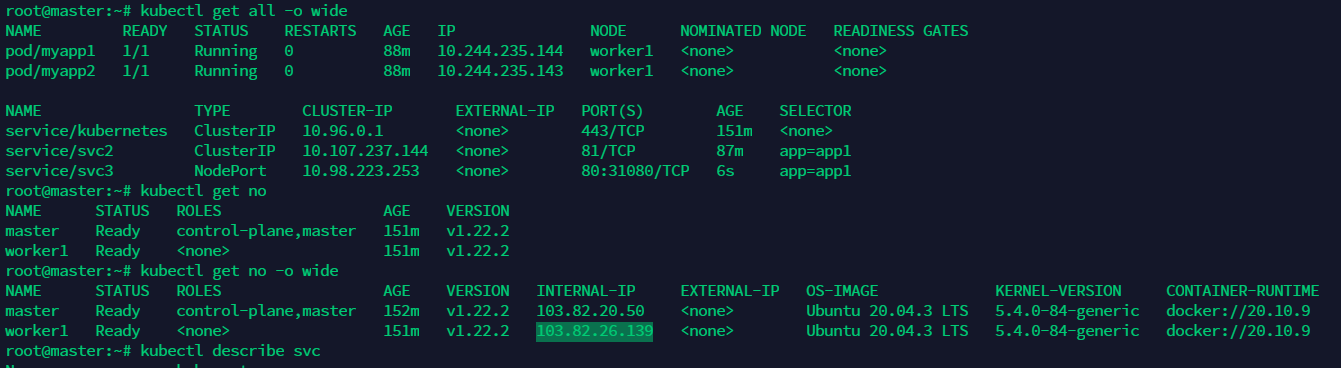
  ports:

    - name: port1

      port: 80

      targetPort: 80

      nodePort: 31080



Trên service/svc3 ko có hiện ip của worker1.

Svc3 type là NodePort lên tức là n sẽ có expose IP của worker

Curl đến cluster-ip của svc3 là port 80, curl đến ip node worker là port 31080



# Phần 4: Service, Deployment, Secret

Trong ví dụ này, sẽ thực hành triển khai chạy máy chủ nginx với mức độ áp dụng phức tạp hơn đó là:

* Xây dựng một image mới từ image cơ sở nginx rồi đưa lên registry Hub Docker đặt tên là ichte/swarmtest:nginx
* Tạo Secret chứa xác thực SSL sử dụng bởi ichte/swarmtest:nginx
* Tạo deployment chạy/quản lý các POD có chạy ichte/swarmtest:nginx
* Tạo Service kiểu NodePort để truy cập đến các POD trên

1. Xây dựng image ichte/swarmtest:nginx

cấu hình để nó nhận các yêu cầu http (cổng 80) và https (cổng 443).

Tạo ra thư mục nginx để chứa các file dữ liệu, đầu tiên là tạo ra file cấu hình nginx.conf, file cấu hình này được copy vào image ở đường dẫn /etc/nginx/nginx.conf khi build image (đường dẫn bên trong images - container)

**file cấu hình nginx.conf**

Để ý file cấu hình này, thiết lập nginx lắng nghe yêu cầu gửi đến cổng 80 và 443 (tương ứng với 2 server), thư mục gốc làm việc mặc định của chúng là /usr/share/nginx/html, tại đây sẽ copy và một file index.html

**2) Chuẩn bị file index.html**

Index.html

**3) Xây dựng image mới từ dockerfile**

**Tạo docker file để build image có sẵn 1 trang web và cấu hình khác, cấu hình xác thực nginx ở trên**

# build image từ Dockerfile, đặt tên image mới là ichte/swarmtest:nginx

docker build -t ichte/swarmtest:nginx -f dockerfile .

# đẩy image lên hub docker

docker push ichte/swarmtest:nginx

giải thích: xác thực ssl trong https port 443

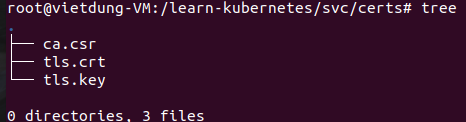
tạo xác thực ssl, những file xác thực này sẽ được lưu vào hệ thống thông qua 1 secret

tạo thư mục certs để lưu file xác thực

Thực hiện lệnh sau để sinh file tự xác thực

openssl req -nodes -newkey rsa:2048 -keyout tls.key -out ca.csr -subj "/CN=vietdungvl.net"

openssl x509 -req -sha256 -days 365 -in ca.csr -signkey tls.key -out tls.crt



2 file tls này là thông tin nhạy cảm

Cho nên kubernetes khuyến khích trúng ta sử dụng đối tượng lưu trữ là secret

ở đây chúng ta sử dụng Secret này kiểu tls, tức chứa xác thực SSL chuyên để lưu file xác thực

# kubectl create secret tls secret-nginx-cert --cert=certs/tls.crt --key=certs/tls.key

secret-nginx-cert: tên secret

certs/tls.crt : là đường dẫn đến file

* Kiểm tra trên hệ thống có những secret nào

# kubectl describe secret

# kubectl describe secret/secret-nginx-cert (hoặc)

4) Tạo Deployment triển khai các Pod chạy bằng images trên

- 6.nginx.yaml

**Lưu ý:**

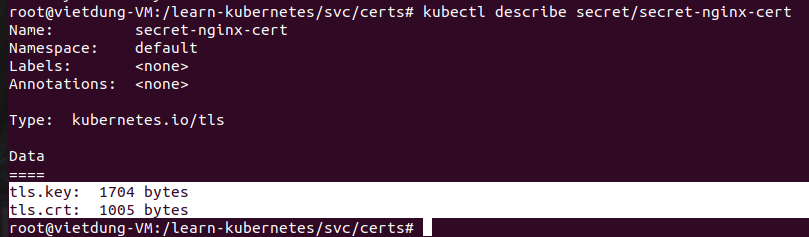
trong mục spec có 2 loại volume được tạo ra

* name volume

1. volume được tạo ra được lưu trên file vật lý của máy host gọi là **hostpath**
2. volume được tạo ra lưu trong file của container gọi là **mountPath** trong volumeMounts

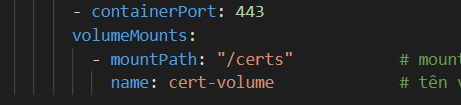
ở phần deploy nginx.yaml khai báo ở trên ta có thể thấy volume được tạo ra là

 volume này có dữ liệu của thằng secret trên hệ thống kubernetes chứa 2 file tls. Secret là thư mục lưu trữ để lưu file xác thực được ht kubernetes khuyến khích sử dụng



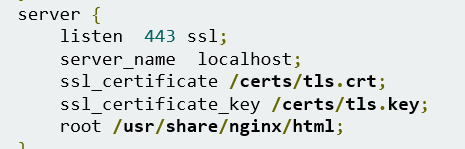
Trong secret của kubernetes có 2 file này

Sau khi có 2 file đó lưu trong volume, ta lại mount volume đó vào file /certs của thằng container bằng mountPath, lưu trong thư mục /certs của container



Trong thư mục /certs trên container lúc này đã có 2 file xác thực tls

Và thư mục /certs này phải tạo đúng để thằng nginx thì trỏ đến file xác thực nó tìm thấy

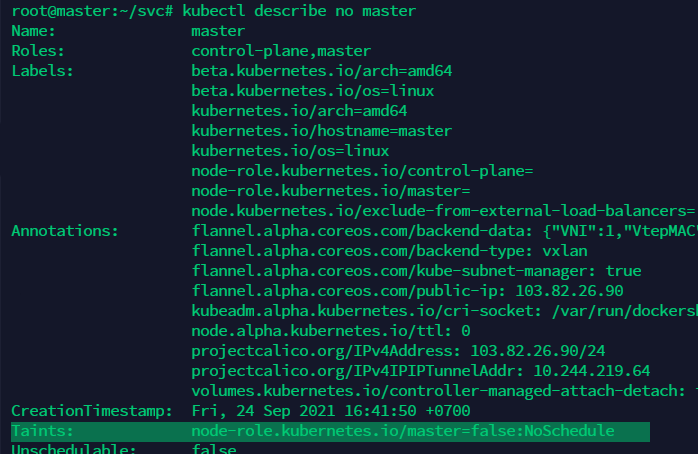
 file deployment chạy trước, file nginx chạy sau

# DaemonSet Job và CronJob trong Kubernetes

(ds, )

daemonSet hoạt động tương tự như replicaset, tức n có khả năng tạo và quản lý các pod, nhưng điểm khác biệt là daemonSet n chỉ tạo ra trên mỗi node chỉ có 1 pod

Mục đích: Triển khai DaemonSet khi cần ở mỗi máy (Node) một POD, thường dùng cho các ứng dụng như thu thập log, tạo ổ đĩa trên mỗi Node ..



Dòng này trên node master thể hiện là trên node này ko được phép triển khai các pod chạy trên node này

Nhưng nếu muốn node master này cũng được phép triển khai chạy các pod thì chỉ cần xóa đi cái dòng này

xóa taint trên node master.xtl cho phép tạo Pod

kubectl taint node master node-role.kubernetes.io/master-

# thêm taint trên node master.xtl ngăn tạo Pod trên nó

kubectl taint nodes master node-role.kubernetes.io/master=false:NoSchedule

master: là tên node (cú pháp đơn giản)

# Liệt kê các DaemonSet

kubectl get ds -o wide

# Liệt kê các POD theo nhãn

kubectl get pod -o wide -l "app=ds-nginx"

# Chi tiết về ds

kubectl describe ds/dsapp

# Xóa DaemonSet

kubectl delete ds/dsapp

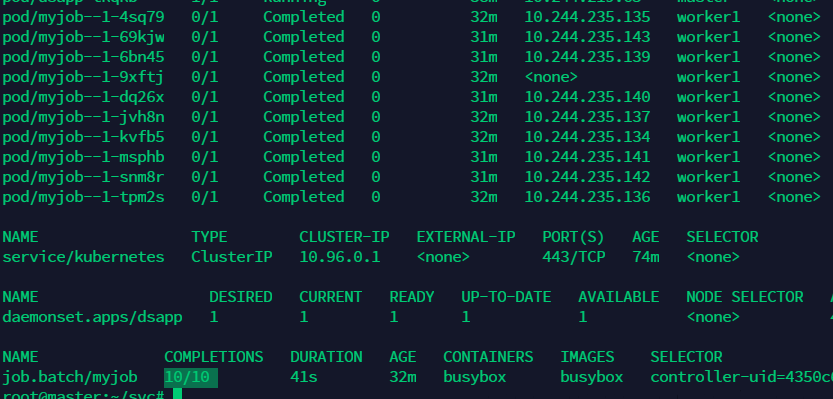
## 2.Job trong Kubernetes

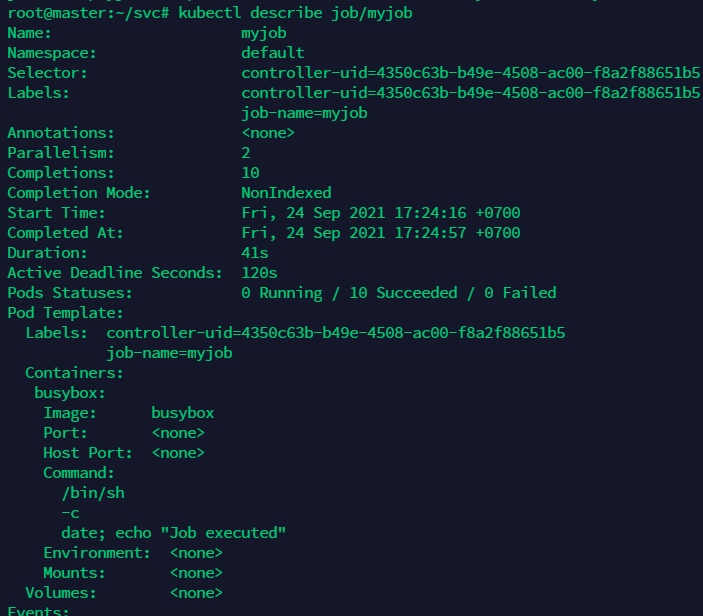
**Job** (jobs) có chức năng tạo, chạy POD nào đó, pod đó chạy và thi hành 1 tác vụ và phải hoàn thành trong khoảng tgian. Khi hoàn thành thì pod đó kết thúc và dừng

Nhiệm vụ: để update dữ liệu, dọn dẹp,…

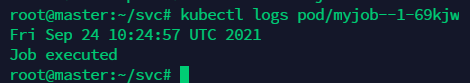
File 02-job.yaml

Vd chạy container nào đấy có nhiệm vụ nào đấy và hoàn thành nó



 kubectl describe job/myjob

Để xem logs của 1 pod đó



## 3.CronJob trong Kubernetes

Mục đích CronJob là vd 10h sáng thì update dữ liệu, hay 9h sáng tự động gửi mail cho khách hàng

# Danh sách các CronJob

kubectl get cj -o wide

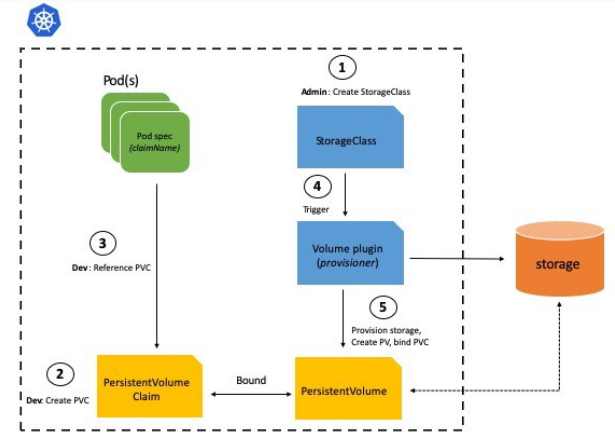
# Danh sách các Job

kubectl get jobs -o wide

# Sử dụng Persistent Volume (pv) và Persistent Volume Claim (pvc) trong Kubernetes

**PersistentVolume** (pv) là một phần không gian lưu trữ dữ liệu tronnng cluster, các PersistentVolume giống với Volume bình thường tuy nhiên nó tồn tại độc lập với POD (pod bị xóa PV vẫn tồn tại), có nhiều loại PersistentVolume có thể triển khai như NFS, Clusterfs ... (xem tại [Các kiểu PersistentVolume](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#types-of-persistent-volumes))

**PersistentVolumeClaim** (pvc) là yêu cầu sử dụng không gian lưu trữ (sử dụng PV). Hình dung PV giống như Node, PVC giống như POD. POD chạy nó sử dụng các tài nguyên của NODE, PVC hoạt động nó sử dụng tài nguyên của PV.



Chúng ta có 1 ổ đĩa lưu trữ file là storage. Để các pod truy cập vào ổ đĩa thì

Đầu tiên phải có persistentVolume (pvc). Pvc truy cập đến dịch vụ lưu trữ file – storage

1pv chỉ có 1 PVC. PVC mount vào pod

**PersistentVolume** **Type**

* NFS
* Azurefile
* CSI
* HostPath

I.

* Tạo pv với bộ nhớ 5G trên local host và n độc lập vs pod
* Tạo pvc lấy tài nguyên của pv và được attach với pod

1. Tạo Persistent Volume trong Kubernetes

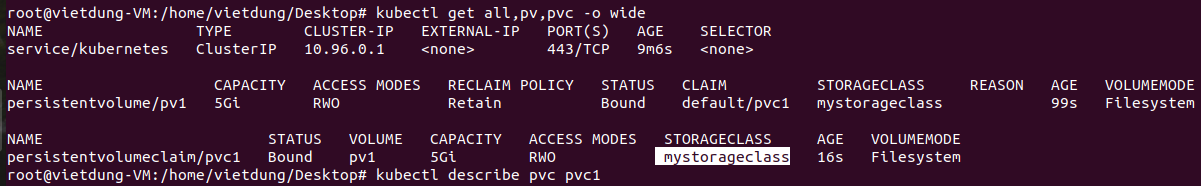
1.persistent-vol.yaml

# kubectl get all,pvc,pv -o wide

Ta đã có 1 Persistent Volume tên là pv1

Để các pod truy cập được vào Persistent Volume này ta phải tạo ra các PersistentVolumeClaim để yêu cầu truy cập, mỗi pv chỉ có 1 pvc

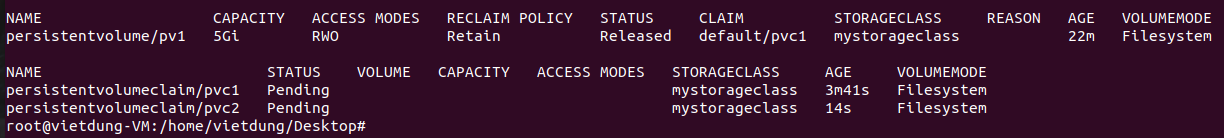
2.Tạo pvc để yêu cầu xin truy cập pv



2 volume đã ở trạng thái **mount** với nhau STATUS: Bound

Mount với nhau thông qua key: StorageClass

**Lưu Ý:**

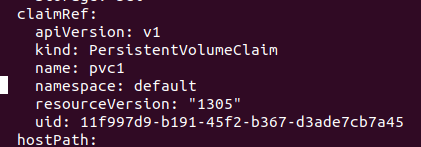
****

Khi xóa pvc1 đi, nhưng khi chạy lại thì ko thể attach pvc-pv được.

Bởi vì thằng pv n đang attach (claim) với pv1 rồi nên bgio khi khởi tạo con khác hay tạo lại chính con cũ thì cũng ko attach được với pv**. 1 pv chỉ mount vs 1 pvc và ngược lại**

Muốn để pv này cho phép thằng khác mount lại thì t phải xóa toàn bộ mục ClaimRef của pv đi

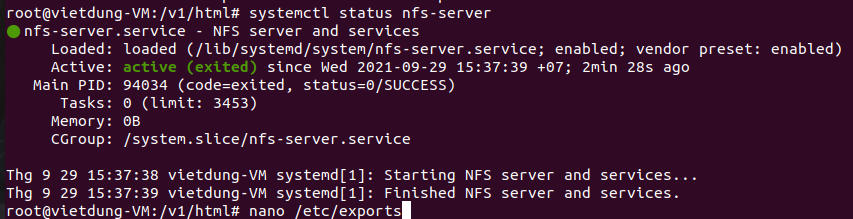
# kubectl edit pv pv1



# Sử dụng PersistentVolume NFS trên Kubernetes

Giúp chia sẻ dữ liệu tới các pod chạy trên các pod khác nhau

1.Cài NFS server (NFS trong thư viện): libnfs-utils/focal 4.0.0-1 amd64



2.Cấu hình thư mục chia sẻ /data/mydata/

# nano /etc/exports : thư mục cấu hình NFS

Thêm: /data/mydata \*(rw,sync,no\_subtree\_check,insecure)

# Tạo thư mục

mkdir -p /data/mydata

chmod -R 777 /data/mydata

# export và kiểm tra cấu hình chia sẻ

exportfs -rav

exportfs -v

showmount -e

# Khởi động lại và kiểm tra dịch vụ

systemctl stop nfs-server

systemctl start nfs-server

systemctl status nfs-server

3.Cài tiếp NFS trên worker1.

Chỉ cài NFS thôi

# Gắn ổ đĩa

mount -t nfs 172.16.10.100:/data/mydata /home/data/

172.16.10.100: là ip NFS, ip của con master đang là nfs server

/data/mydata: là thư mục của master mount vào thư mục /home/data/ của worker

Oke 2 máy đã chia sẻ được dữ liệu cho nhau

1. Tạo pv, pvc

* Tạo PersistentVolume NFS

1-pv-nfs.yaml

* Tạo PersistentVolumeClaim NFS

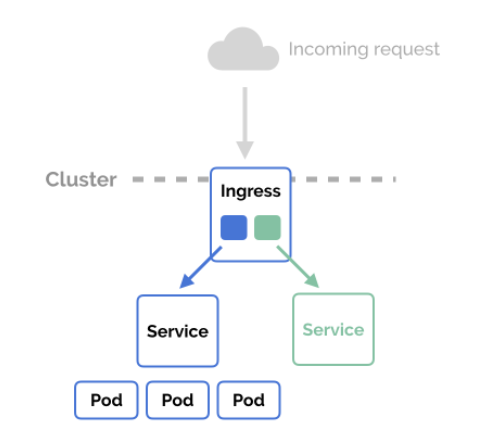
2-pvc-nfs.yaml

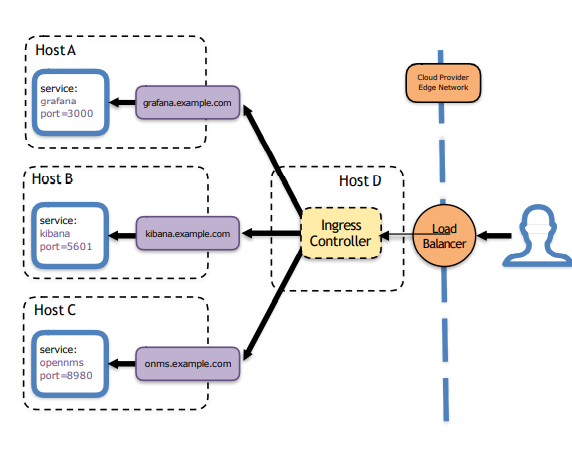
1. Khi có pvc rồi thì giờ tạo pod

Sau đó truy cập web khi đã được cấu hình thư mục làm việc map với thư mục pvc

# Ingress trong Kubernetes

**Ingress** là thành phần được dùng để điều hướng các yêu cầu traffic giao thức HTTP và HTTPS từ bên ngoài (interneet) vào các dịch vụ bên trong Cluster.



****

**Ingress:**

* Hoạt động như hệ thống cân bằng tải, proxy

Tiếp nhận yêu cầu từ internet dạng HTTP,HTTPS và chuyển hướng vào bên trong dịch vụ

Tham khảo cài ingress controller

https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/#additional-controllers

* Haproxy ingress controller

<https://www.haproxy.com/documentation/kubernetes/latest/installation/enterprise/kubernetes/>

B1: Cài đặt HAProxy Ingress Controller

#Tạo namespace có tên ingress-controller

kubectl create ns ingress-controller

kubectl get ns (liệt kê namespace)

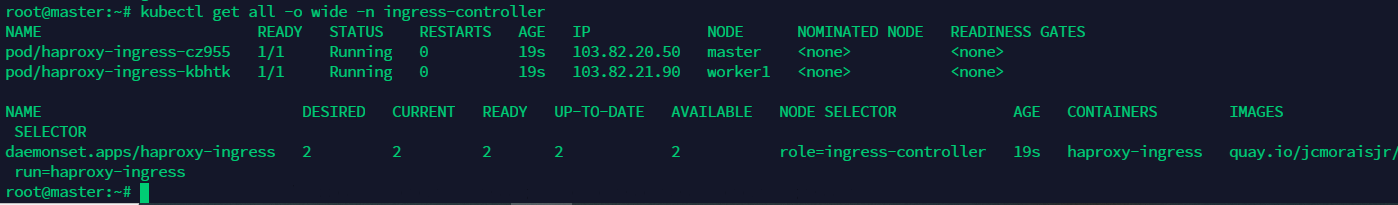
kubectl apply -f <https://haproxy-ingress.github.io/resources/haproxy-ingress.yaml>

# Gán thêm label cho các Node (ví dụ node worker2.xtl, worker1.xtl ...)

kubectl label node master role=ingress-controller

kubectl label node worker1 role=ingress-controller

sau khi có ingress thì ta phải gán thêm label cho từng node để ingress hđ trên đó



Tạo ra 2 pod, và 2 pod này vẫn chưa có ingress đâu

apiVersion: v1

kind: Service

metadata:

  name: http-test-svc

  namespace: ingress-controller  #namespace của n là đây

spec:

  ports:

  - port: 80

    protocol: TCP

    targetPort: 80      # chuyển hướng đến cổng 80 của các pod

  selector:

    run: http-test-app

  sessionAffinity: None

  type: ClusterIP

---

apiVersion: apps/v1

kind: Deployment

metadata:

  labels:

    run: http-test-svc

  name: http-test-svc

  # Deploy này tạo tại namespace có Haproxy Ingress

  replicas: 2

  selector:

    matchLabels:

      run: http-test-app

  template:

    metadata:

      labels:

        run: http-test-app

    spec:

      containers:

      - image: nginx

        imagePullPolicy: IfNotPresent

        name: http

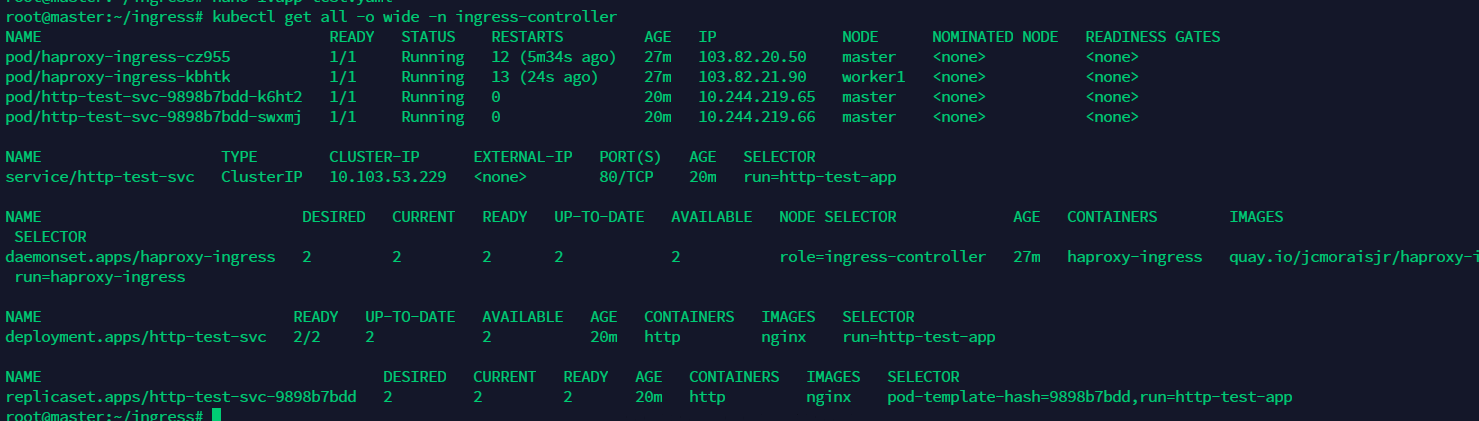
        ports:

        - containerPort: 80

          protocol: TCP

        resources: {}

* Kiểm tra các thành phần từ namespace có tên in ingress-controller



1.Triển khai một ứng dụng (http)

1.app-test.yaml

2.Tạo ingress

Tạo ingress để khi truy vấn đến cổng 80 của node bằng 1 domain name thì n sẽ chuyển hướng vào service http này

B1: thiết lập tên miền ảo

# nano /etc/hosts

172.16.10.102 vietdungvl.vnet

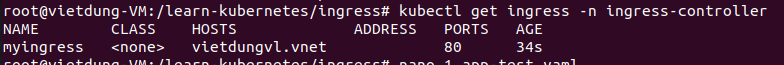
172.16.10.102 : ip của woker node

Tức là khi curl đến tên miền kia là curl đến ip của woker node. Mà container đó cho phép expose ip node

**B2: Tạo Ingress**

2.app-test-ingress.yaml

File ingress này nhận yêu cầu http

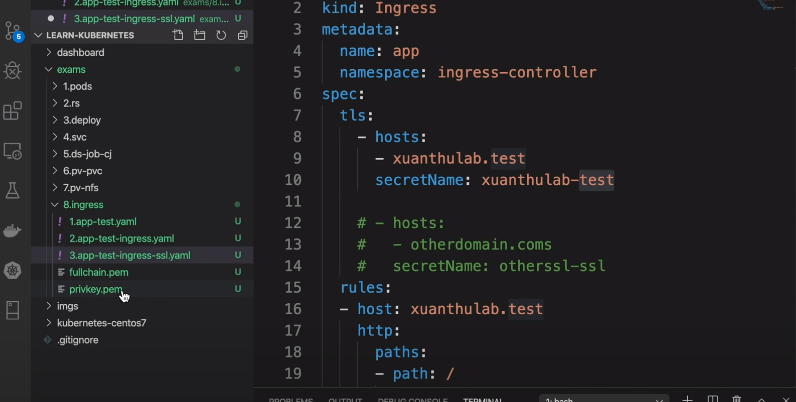


Ingress có tên là myingress nhận truy vấn có DNS là vietdungvl.net vào container http với cổng lắng nghe là 80

* Truy cập vào địa chỉ này cũng tương tự như vào ip node worker mà container expose ra

B3: tạo ingress truy cập giao thức HTTPS

* Xóa cái ingress http đi
* Tạo xác thực SSL



Chạy lệnh sau để sinh xác thực cho tên miền ảo xuanthulab.test

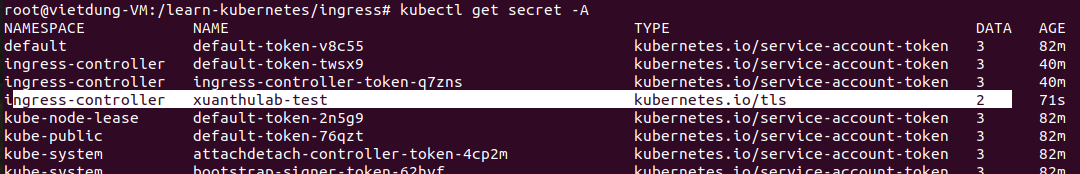
openssl req -x509 -newkey rsa:2048 -nodes -days 365 -keyout privkey.pem -out fullchain.pem -subj '/CN=xuanthulab.test'

Sau đó tạo một Secret (thuộc namespace chạy POD), đặt tên Secret này là xuanthulab-test

kubectl create secret tls xuanthulab-test --cert=fullchain.pem --key=privkey.pem -n ingress-controller

* Kiểm tra : lấy secret ở mọi NameSpace

# kubectl get secret -A



ở NAMESPACE ingress xuananthu có 2 file xác thực ssl

3.app-test-ingress-ssl.yaml

- kiểm tra lại ingress

# kubectl get ingress -A

# Triển khai NGINX Ingress Controller trong Kubernetes

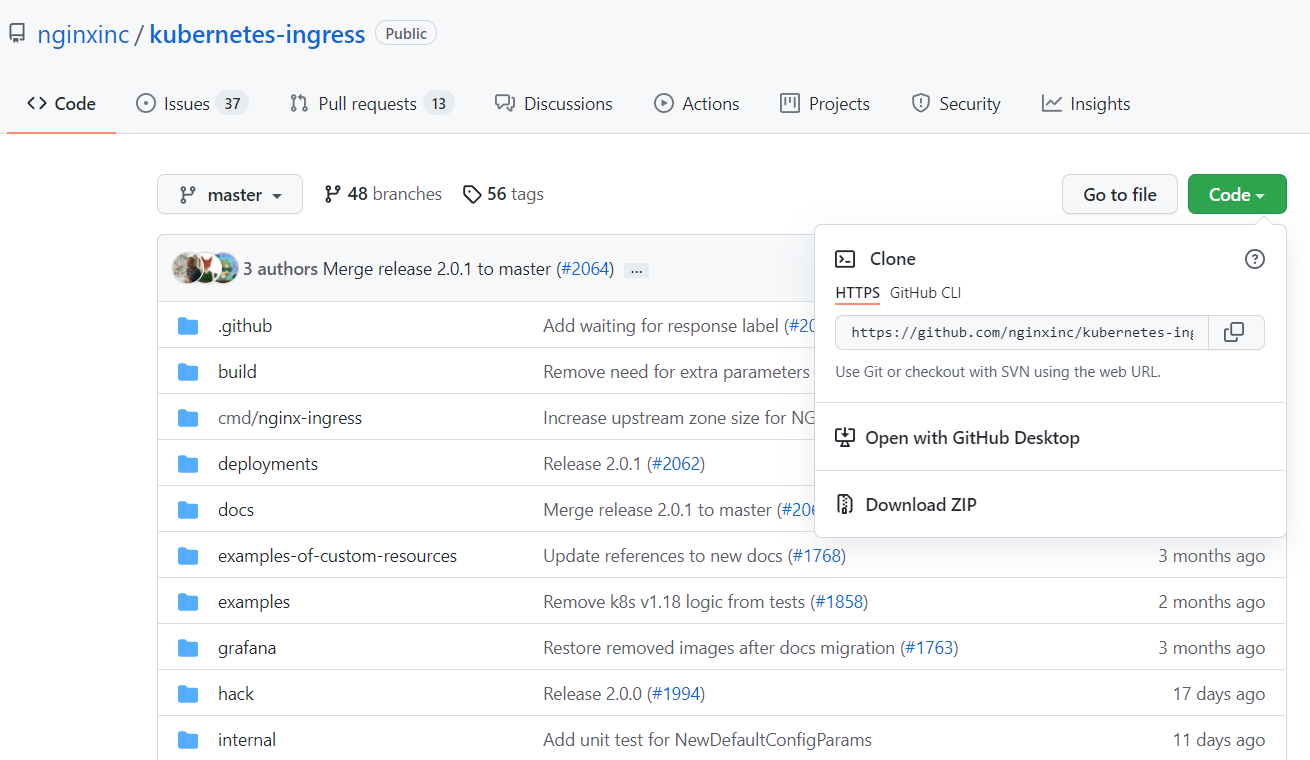
Tải nginx ingress controller

git clone git@github.com:nginxinc/kubernetes-ingress.git

khi tải về n có thư mục **kubernetes-ingress**

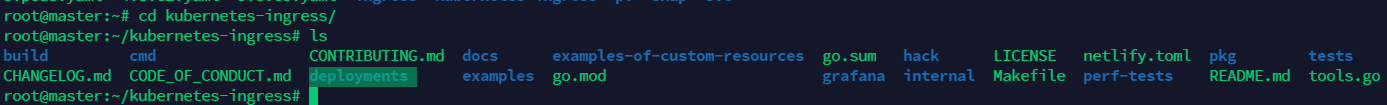
cd kubernetes-ingress

<https://github.com/nginxinc/kubernetes-ingress>



Git clone đống này về

git clone <https://github.com/nginxinc/kubernetes-ingress.git>



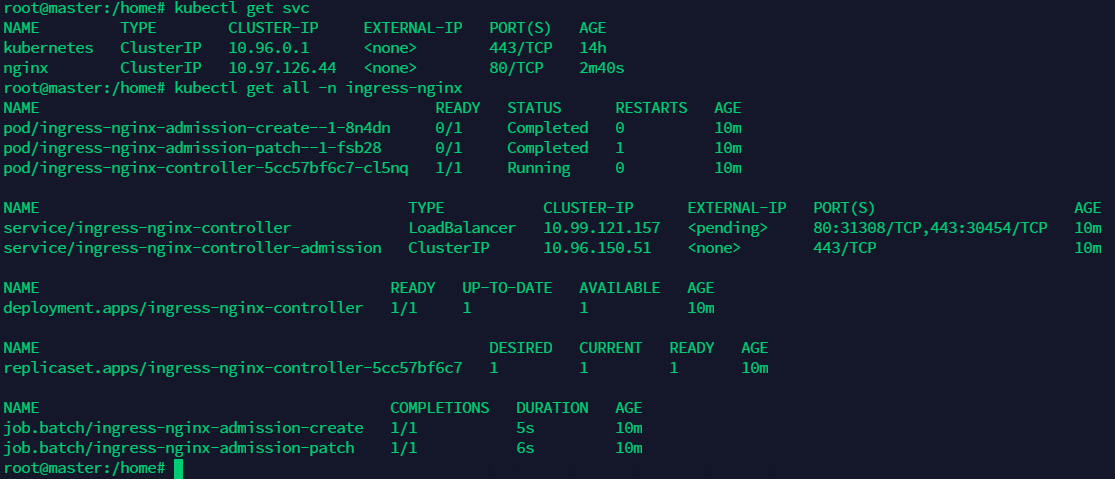
# cd kubernetes-ingress

# cd deployments

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.0.3/deploy/static/provider/cloud/deploy.yaml

kubectl get all -n ingress-nginx

kubectl get svc





Expose 1 trang web có thể sử dụng service NodePort

**Ingress** là thành phần được dùng để điều hướng các yêu cầu traffic giao thức HTTP và HTTPS từ bên ngoài (interneet) vào các dịch vụ bên trong Cluster.

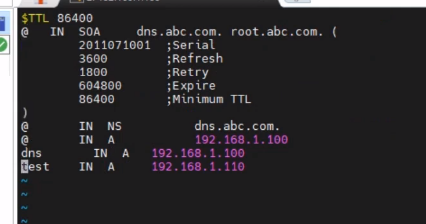
**Ingress** chỉ để phục vụ các cổng, yêu cầu HTTP, HTTPS còn các loại cổng khác, giao thức khác để truy cập được từ bên ngoài thì dùng Service với kiểu NodePort và LoadBalancer

Để Ingress hoặt động, hệ thồng cần một điều khiển ingress trước (Ingress controller), có nhiều loại để chọn sử dụng (tham khảo [Ingress Controller](https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/#additional-controllers))

Nếu chọn Ngix Ingress Controller thì cài đặt theo: [NGINX Ingress Controller for Kubernetes.](https://github.com/nginxinc/kubernetes-ingress)

kubectl run nginx --image=nginx:1.19 --port=80

kubectl expose pod nginx --name=nginx1



@ = abc.com

test.abc.com 192.168.1.110

dns.abc.com 192.168.1.100

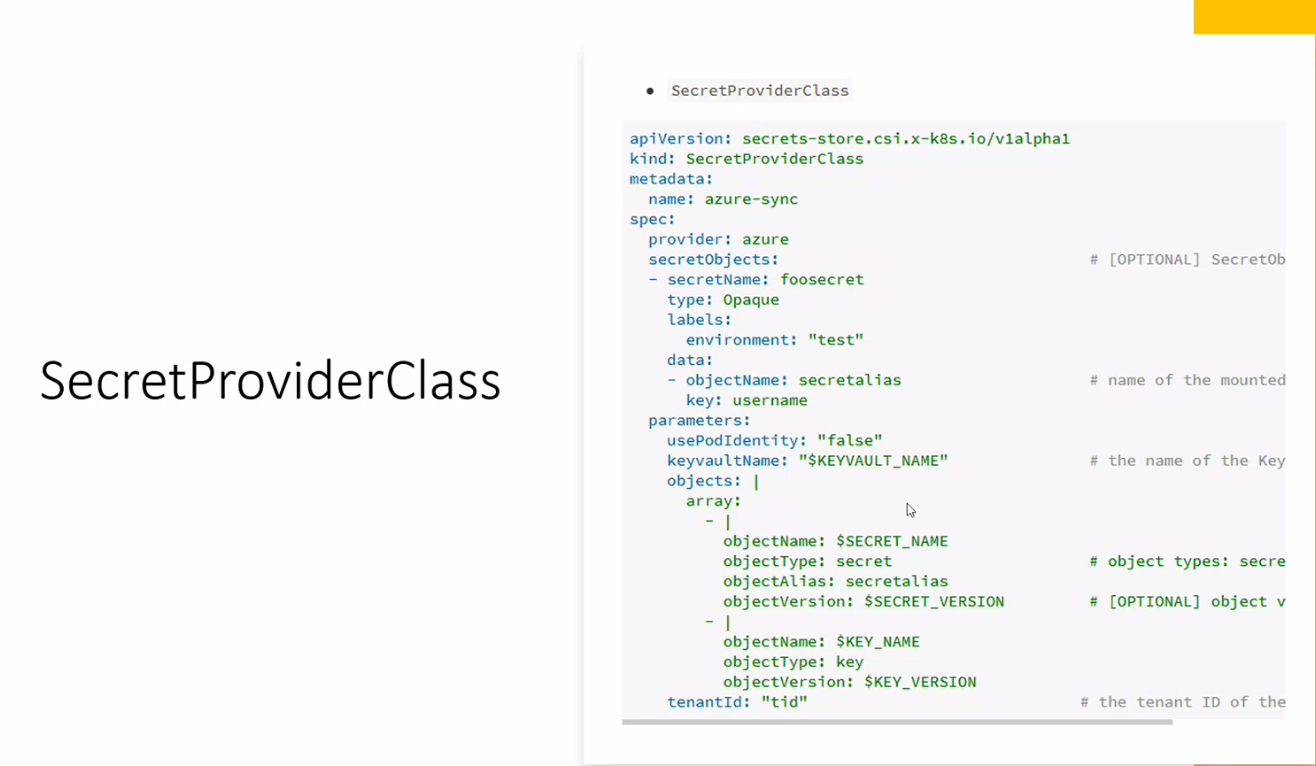
kubectl exec -it pod/nginx-74cd9f9b68-v6jlf -- /bin/bash



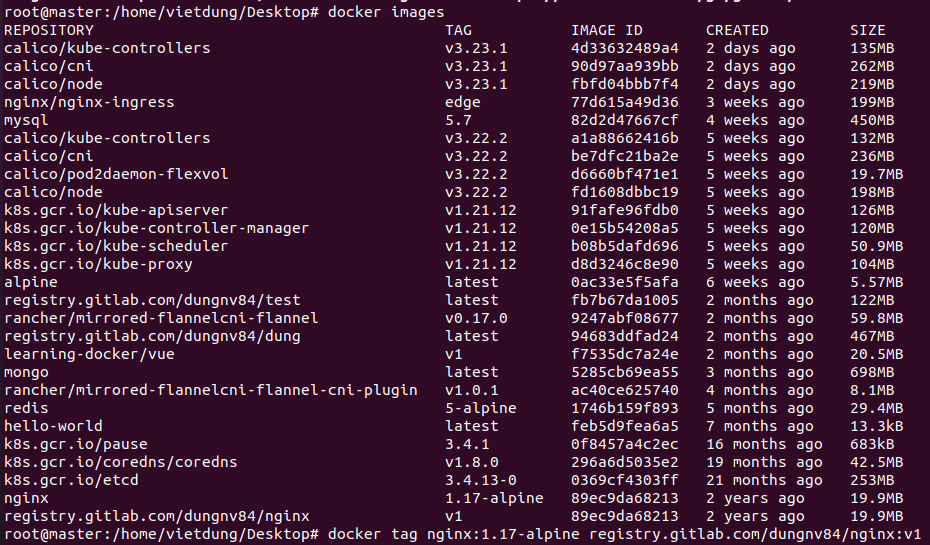
Cách 1 là đưa secret vào dưới dạng biến mtruong

Cách 2: tạo volume

Để lấy được secret trên KV, ta cần CSI, SPC và lấy thông qua MSI hoặc SP qua kvcred



objectName: tên lưu trên kv





Mount volume

# Image container registry

## Credential tới gitlab

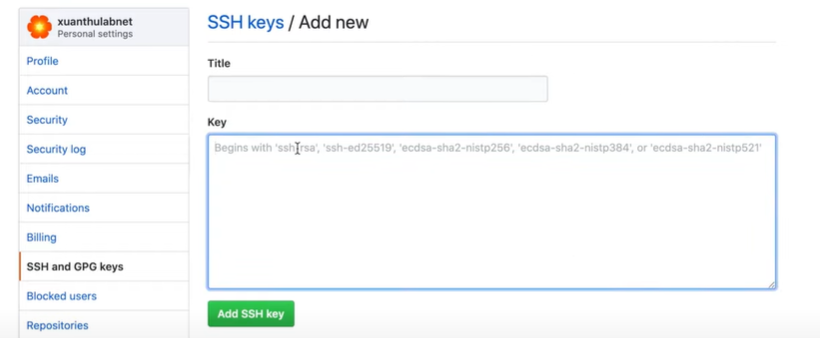
Có 2 kiểu credential tới gitlab là dùng user/pass và ssh

Cách 1: git init, git push

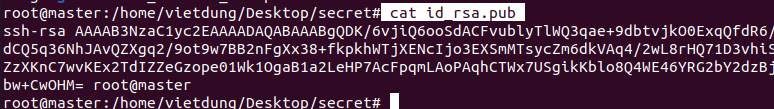
Cách 2: để credential với github, gitlab bằng ssh thì ta chỉ cần config trên githab

ssh-keygen -t rsa -f id\_rsa

b2: Vào github -> setting > developer setting > persional access token > new token > Tick all > và copy mã token paster thành password



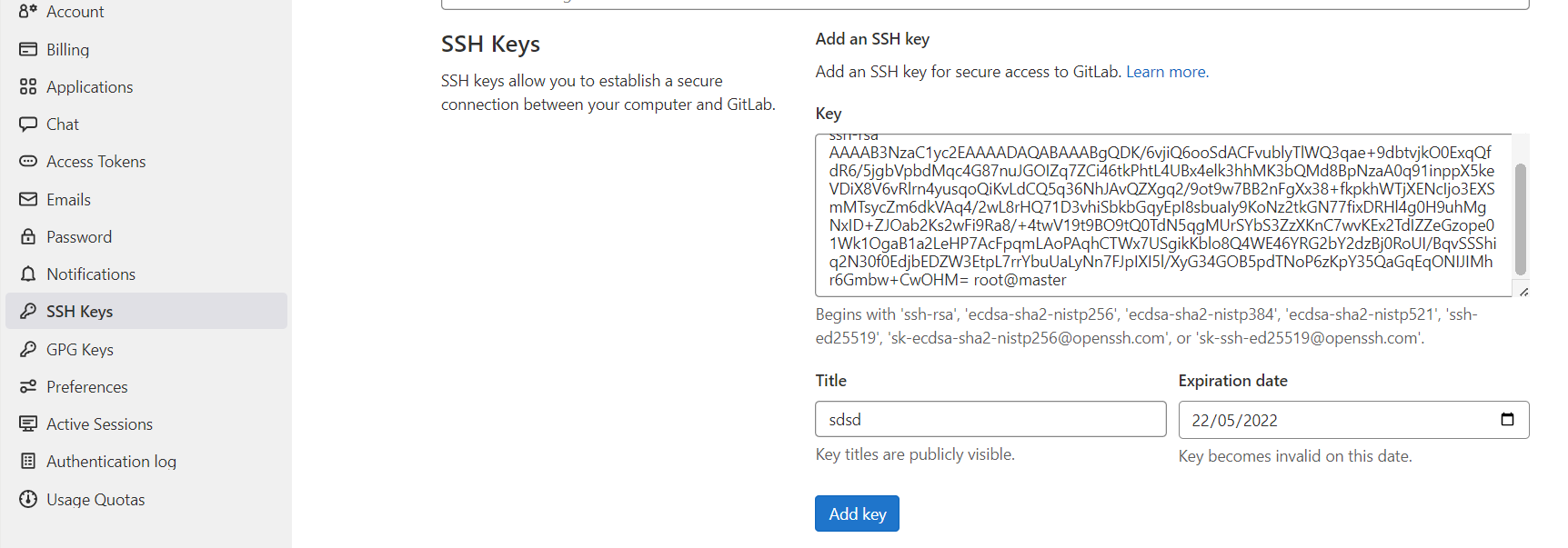
Và add public key vừa tạo vào github này



Add hết nội dung public key vào github.

Tương tự với gitlab, vào đường dẫn và add key vào

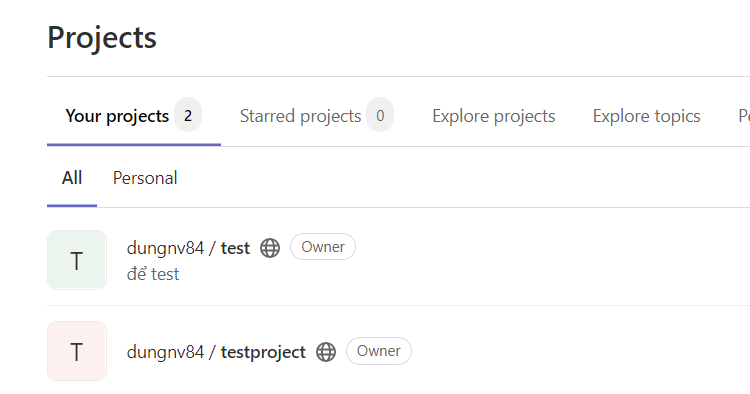
<https://gitlab.com/-/profile/keys>



B3: config khi credential vào gitlab bằng ssh key, tức kiểu publickey kia

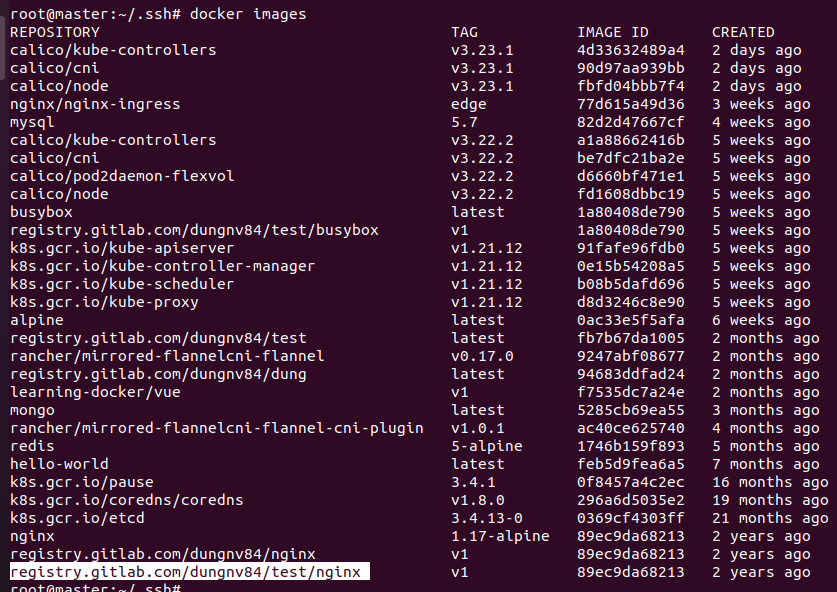


## Push image lên gitlab



Có 2 project rồi, ko tạo nữa

Tạo tag trên repository (kho chứa image trên local)

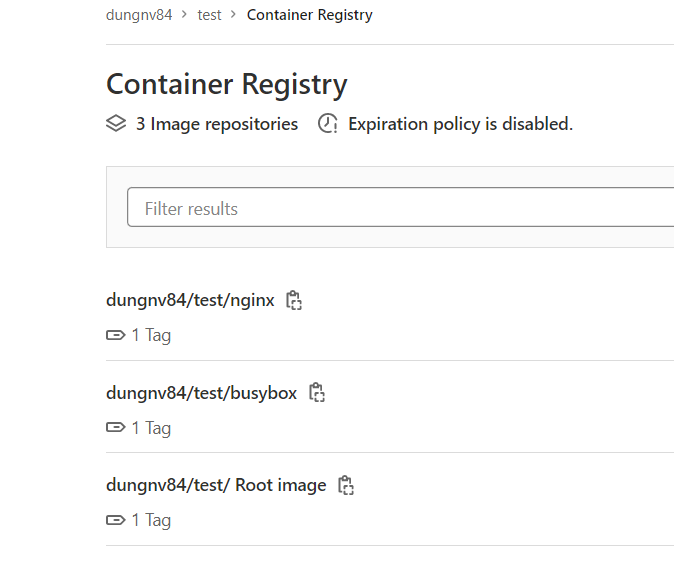


docker tag nginx:latest registry.gitlab.com/dungnv84/test/nginx:v1

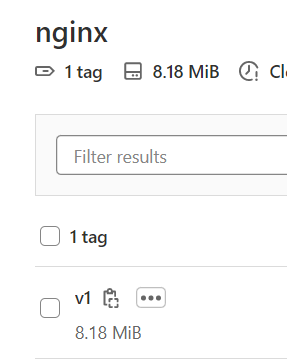
tạo 1 tag lấy image là nginx:latest rồi đánh thành image registry.gitlab.com/dungnv84/test/nginx:v1

image registry.gitlab.com/dungnv84/test/nginx:v1 cũng được lưu trên repository nhưng đặc điểm của image này là đã đồng bộ vs container registry của gitlab

docker push registry.gitlab.com/dungnv84/test/nginx:v1



Oke đã có rồi đây, bấm vô cái đầu

 oke tag v1

* Push image lên docker

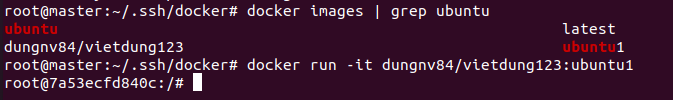
docker tag busybox:latest dungnv84/vietdung123:busybox

docker push dungnv84/vietdung123:busybox

lưu ý là lên dùng tạm busybox trước đã, dùng image khác run lỗi do config image

ak với trường hợp mà dùng dockerfile để build image, ta lên check image đó, chạy image đó bằng docker xem n có run dk ko đã, run dk thì chạy trên pod n mới run dk, nếu tạo sai image thì pull về chạy vs trên kuber n sẽ lỗi, vì kubernetes run pod, n chạy trên nền tảng docker runtime mà

* Check image



Đây, vậy nó chạy được rồi nhá



Image busybox đây

## Tạo pod dùng container registry private

Để tạo secret credential tới docker, github, gitlab ta có 2 cách

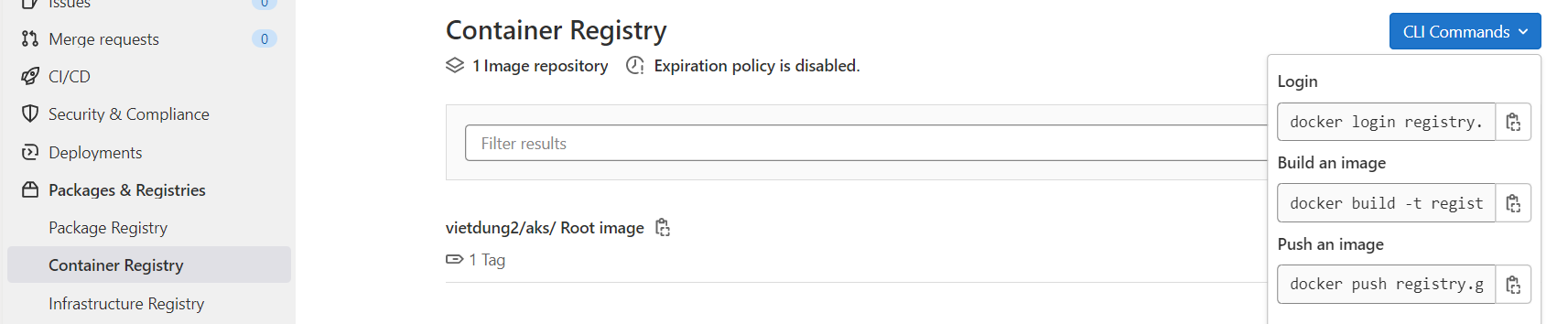
Cách 1: đó là ta access, login vào docker, gitlab trước

Thì với docker, ta dùng lệnh docker login

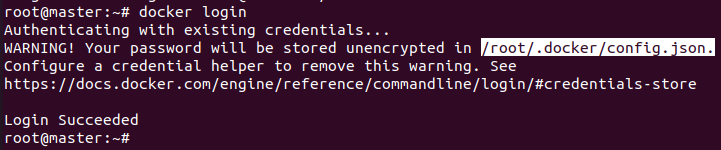
Với gitlab, ta dùng lệnh

docker login registry.gitlab.com

ta có thể check chính xác cách login ở đây (ở gitlab)



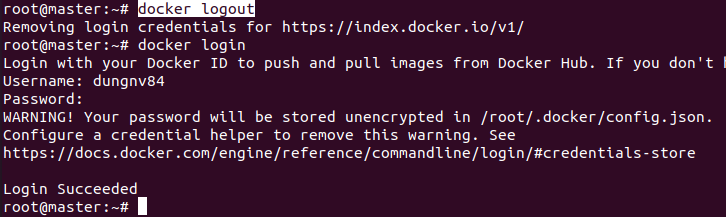
* Sau khi login xog, n đã lưu sync access vào file /root/.docker/config.json



Vì mình đã login trước đó, cho nên là n đã xác thực bằng file này rồi, n giống kiểu xác thực bằng cái file config của .kube ấy

* Và mình cũng có thể logout

docker logout (nếu muốn logout docker)

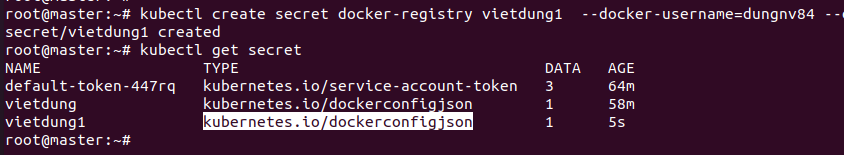


Đấy khi logout ra, thì mình có thể login lại nhưng sẽ nhập lại user/pass

* Oke, đi vào vấn đề chính là bgio tạo secret để access đến private container registry trên docker hub

Cách 1: lệnh

kubectl create secret docker-registry vietdung1 --docker-username=dungnv84 --docker-password=vietdungvl123 --docker-email=vietdungvl1997@gmail.com



Nhưng mình chả biết là cái lệnh này n có chính xác ko nữa,

Nhưng với cách 2 thì sẽ chính xác nhất

Cách 2:

Vì mình đã vừa login được đến docker hub rồi, và cái thông tin n nằm trong cái file config của **/root/.docker/config.json,** đấy, bgio mình lấy luôn file này để tạo secret

kubectl create secret generic vietdung --from-file=.dockerconfigjson=config.json --type kubernetes.io/dockerconfigjson

cái type secret của docker thì ảnh ở trên, dùng câu lệnh tạo n hiện ra ngay

ok secret vietdung đã dk tạo

* Run pod sử dụng image đó nào

apiVersion: v1

kind: Pod

metadata:

  creationTimestamp: null

  labels:

    run: static-busybox

  name: static-busybox

spec:

  imagePullSecrets:

  - name: vietdung

  containers:

  - command:

    - sleep

    - "1000"

    image: dungnv84/vietdung123:busybox

    name: static-busybox

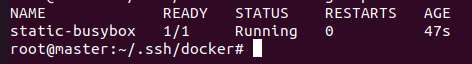
    resources: {}

  dnsPolicy: ClusterFirst

  restartPolicy: Never

status: {}

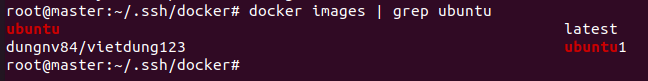
và…



Running rồi,

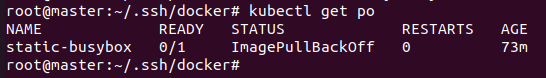
Bgio xóa pod đi, xóa secret trong file pod đi, thì khi apply lại, pod này có run ko

Câu trả lời là có, vì sao, vì image này đang lưu trên repository của local rồi, chứ n ko pull image trên private container registry của docker



**Đây, pull trên repository cần j secret**

Rồi, để demo pull secret ở private container registry mà ko có secret, ta phải xóa image đó trên Repository trước

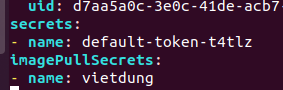


Hé, thế là ta đã hoàn thành phần demo

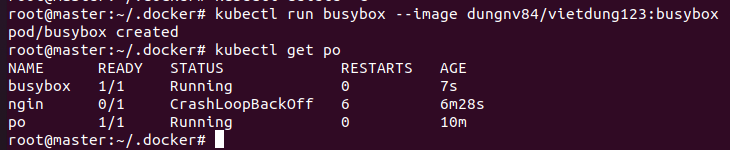
À thêm 1 lưu ý nữa,

Nếu ta ko dùng secret, vậy ta ủy quyền cho node pull image về, biến node đó thành 1 docker repository và pod chạy thì cứ lấy image trên Repository thôi

**Cách 3: nhồi secret vào Serviceaccount**

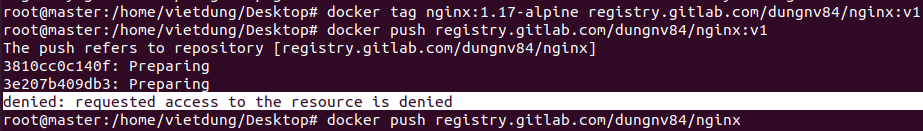
****

kubectl run busybox --image dungnv84/vietdung123:busybox --command sleep 1000 --restart=Never

****

## Tạo secret bằng public, private key, tạo pod kiểu mount secret

Lưu ý lỗi dưới đây



Ko phải gitlab chặn mà sai đường dẫn của container registry

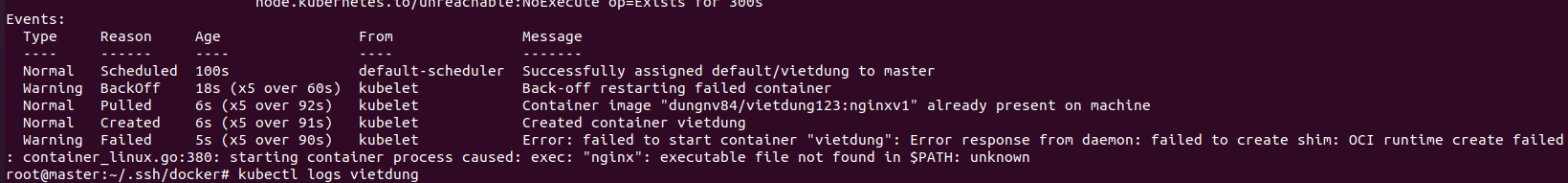
registry.gitlab.com/dungnv84 /nginx:v1

mà ta chỉ có 2 registry là gitlab.com/dungnv84/test và gitlab.com/dungnv84/testproject

docker image rm nginx:1.17-alpine –f

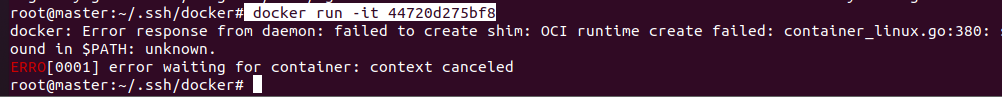
docker login registry.example.com -u <username> -p <token>





Pod ko run, có vẻ build Image lỗi

Chạy thử image vừa build trên container runtime



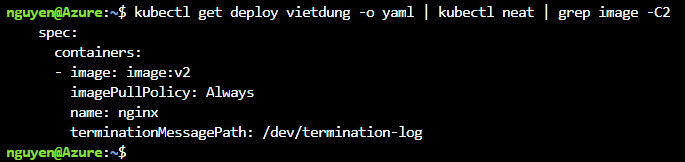
ừ image lỗi r, build lại từ image nhưng lỗi

# Check container của pod

*# Liệt kê tất cả các sự kiện được sắp xếp theo thời gian*

kubectl get events --sort-by=.metadata.creationTimestamp

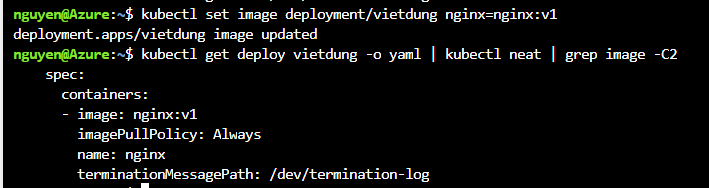
## thay đổi image của pod



Vì image là nằm trong container

kubectl set image deployment/vietdung nginx=nginx:v1

* vietdung: tên deploy
* nginx: tên container



Oke image đã đổi

## Với pod có nhiều container

Lấy log pod nhiều container

kubectl logs -f my-pod -c my-container

kubectl logs -f -l name=myLabel --all-containers *# lấy logs của tất cả các container của pod có nhãn name=myLabel (stdout)*

kubectl exec my-pod -c my-container -- ls / *# Chạy lệnh trong pod (trường hợp nhiều container)*

kubectl top pod POD\_NAME --containers *# Hiển thị số liệu của pod và container chạy trong nó*

*# Cập nhật một phần một node*

kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'

*# Cập nhật image của container; spec.containers[\*].name là bắt buộc vì đó là khóa hợp nhất*

kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-hostname","image":"new image"}]}}'

*# Cập nhật image của container sử dụng một bản vá json với các mảng vị trí*

kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/containers/0/image", "value":"new image"}]'

*# Vô hiệu hóa một deployment livenessProbe sử dụng một bản vá json với các mảng vị trí*

kubectl patch deployment valid-deployment --type json -p='[{"op": "remove", "path": "/spec/template/spec/containers/0/livenessProbe"}]'

*# Thêm một phần tử mới vào một mảng vị trí*

kubectl patch sa default --type='json' -p='[{"op": "add", "path": "/secrets/1", "value": {"name": "whatever" } }]'