Kubernetes Lab

Contents

[1.](#_heading=h.30j0zll) Pod 2

[2.](#_heading=h.1fob9te) Label 3

[3.](#_heading=h.3znysh7) Namespace 5

[4.](#_heading=h.2et92p0) ReplicationControllers 6

[5.](#_heading=h.tyjcwt) ReplicaSets 7

[6.](#_heading=h.3dy6vkm) DaemonSet 8

[7.](#_heading=h.1t3h5sf) Deployment 10

[8.](#_heading=h.4d34og8) StatefulSet 11

[9.](#_heading=h.2s8eyo1) Volume 12

[a)](#_heading=h.17dp8vu) emptyDir 13

[b)](#_heading=h.3rdcrjn) gitRepo 15

[c)](#_heading=h.26in1rg) hostPath 16

[10.](#_heading=h.lnxbz9) Persistent Volume 17

# Pod

Pod là thành phần cơ bản nhất để deploy và chạy một ứng dụng, được tạo và quản lý bởi kubernetes. Pod được dùng để nhóm (group) và chạy một hoặc nhiều container lại với nhau trên cùng một worker node, những container trong một pod sẽ chia sẻ chung tài nguyên với nhau.

Ta sẽ tạo Dockerfile để build image sử dụng trong suốt quá trình.

**kind create cluster --name=management**

//Dockerfile

FROM nginx:1.13.9-alpine

COPY index.html /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

//index.html

<h1>DevOps Training 2022</h1>

Thực hiện build image và push lên docker-hub:

docker build –t [docker-hub]/nginx:latest .

docker push [docker-hub]/nginx:latest

Tạo file nginx.yaml

apiVersion: v1 # Descriptor conforms to version v1 of Kubernetes API

kind: Pod # Select Pod resource

metadata:

  name: nginx # The name of the pod

spec:

  containers:

    - image: [docker-hub]/nginx # Image to create the container

      name: nginx # The name of the container

      ports:

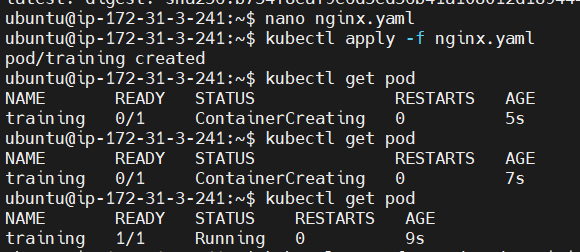
        - containerPort: 80 # The port the app is listening on

          protocol: TCP

Run file config:

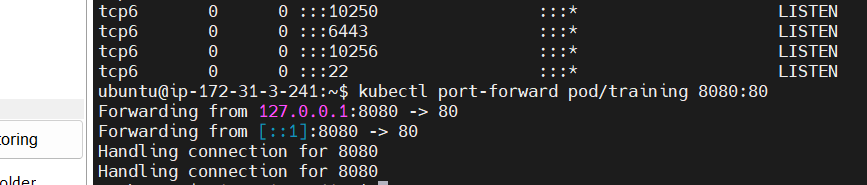
kubectl apply –f nginx.yaml

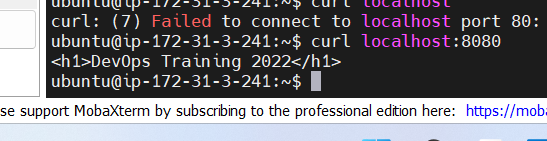
Kiểm tra pod đã chạy hay chưa:



Để kiểm tra chính xác pod đã chạy đúng hay chưa ta cần expose port của pod ra ngoài để kiểm tra. Có 2 cách để expose port của pod ra ngoài, dùng Service resource (mình sẽ nói về Service ở bài sau) hoặc dùng kubectl port-forward. Ở bài này chúng ta sẽ dùng port-forward, chạy câu lệnh sau để expose port của pod:

kubectl port-forward pod/training 8080:80





Để xóa pod ta dùng 1 trong 2 lệnh sau:

kubectl delete pod training

kubectl delete –f nginx.yaml

# Label

Dùng label là cách để chúng ta có thể phân chia các pod khác nhau tùy thuộc vào dự án hoặc môi trường. Ví dụ công ty của chúng ta có 3 môi trường là testing, staging, production, nếu chạy pod mà không có đánh label thì chúng ta rất khó để biết pod nào thuộc môi trường nào.

Labels là một thuộc tính cặp key-value mà chúng ta gán vào resource ở phần metadata, ta có thể đặt tên key và value với tên bất kì. Ví dụ:

#nginx.yaml

apiVersion: v1

kind: Pod

metadata:

  name: nginx-testing

  labels:

    environment: testing # label

spec:

  containers:

    - image: ddthuong/nginx

      name: nginx

      ports:

        - containerPort: 80

          protocol: TCP

---

apiVersion: v1

kind: Pod

metadata:

  name: nginx-production

  labels:

    environment: production # label

spec:

  containers:

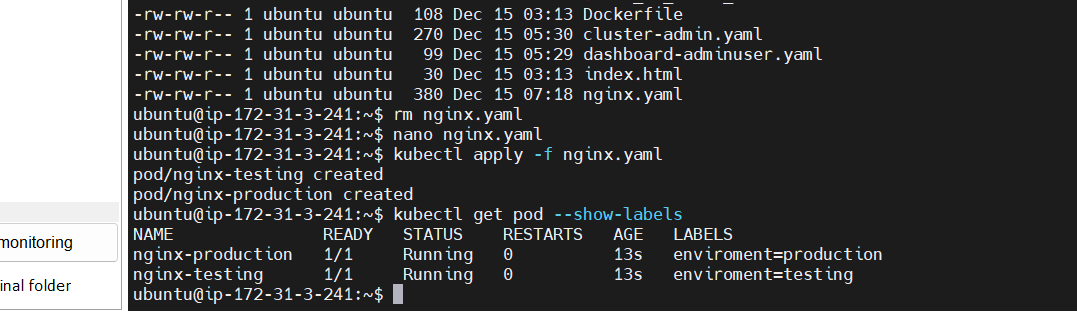
    - image: ddthuong/nginx

      name: nginx

      ports:

        - containerPort: 80

          protocol: TCP



Ta có thể list pod với các option sau:

kubectl get pod -L environment

kubectl get pod -l environment=production

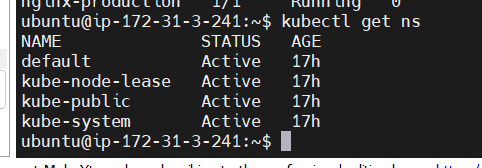
Xóa resource:

kubectl delete -f hello-kube.yaml

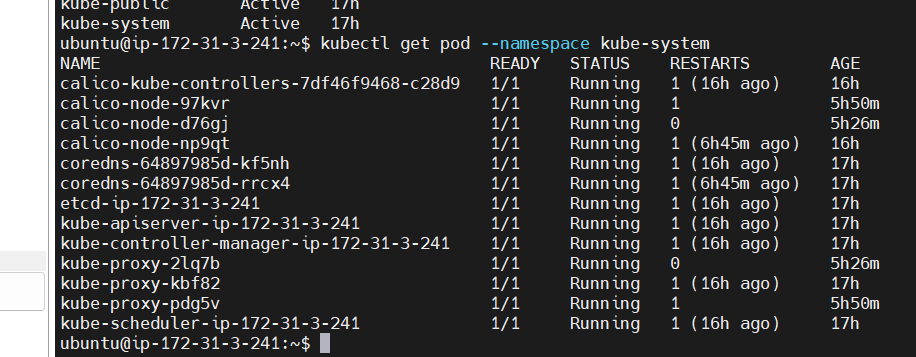
# Namespace

Namespace là cách để ta chia tài nguyên của cluster, và nhóm tất cả những resource liên quan lại với nhau, bạn có thể hiểu namespace như là một sub-cluster.

List toàn bộ namespace:



Ta có thể chỉ định resource của namespace chúng ta muốn bằng cách thêm option --namespace vào.



Tạo một namespace và tạo pod trong namespace đó:

kubectl create ns testing

#nginx.yaml

apiVersion: v1

kind: Pod

metadata:

  name: nginx

  namespace: testing # namespace name

spec:

  containers:

    - image: ddthuong/nginx

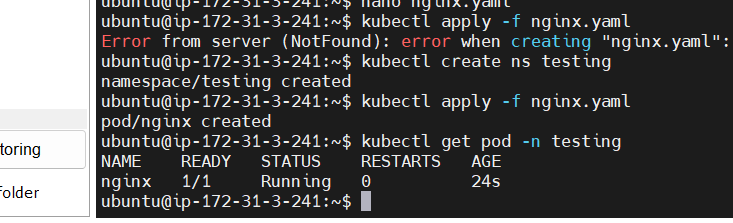
      name: nginx

      ports:

        - containerPort: 80

          protocol: TCP

Tạo pod và list pod theo namespace:



Delete pod trong namespace:

kubectl delete pod nginx -n testing

Delete namespace:

kubectl delete ns testing

# ReplicationControllers

ReplicationControllers là một resource mà sẽ tạo và quản lý pod, và chắc chắn là số lượng pod nó quản lý không thay đổi và kept running. ReplicationControllers sẽ tạo số lượng pod bằng với số ta chỉ định ở thuộc tính replicas và quản lý pod thông qua labels của pod.

Tạo một ReplicationControllers:

#replica.yaml

apiVersion: v1

kind: ReplicationController

metadata:

  name: training

spec:

  replicas: 2 # number of the pod

  selector: # The pod selector determining what pods the RC is operating on

    app: nginx # label value

  template: # pod template

    metadata:

      labels:

        app: nginx # label value

    spec:

      containers:

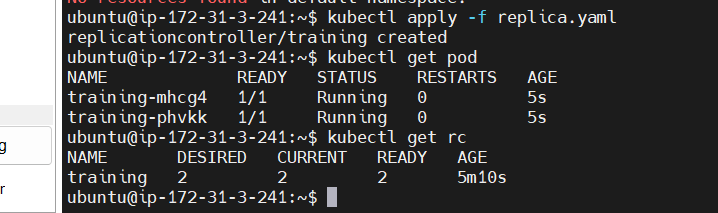
      - image: ddthuong/nginx # image used to run container

        name: nginx # name of the container

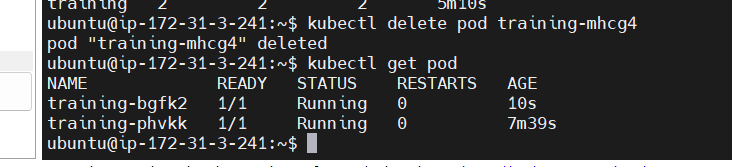
        ports:

          - containerPort: 80

Chạy thành công, ta đã tạo được đúng số lượng pod trong file config.



Ta thử xóa đi 1 pod và nó đã tự tạo ra pod mới



Xóa RC ta dùng lệnh:

kubectl delete -f replica.yaml

# ReplicaSets

Đây là một resource tương tự như RC, nhưng nó là một phiên bản mới hơn của RC và sẽ được sử dụng để thay thế RC.

Tạo file tên là replicaset.yaml, copy config sau vào:

apiVersion: apps/v1 # change version API

kind: ReplicaSet # change resource name

metadata:

  name: training

spec:

  replicas: 2

  selector:

    matchLabels: # change here

      app: training

  template:

    metadata:

      labels:

        app: training

    spec:

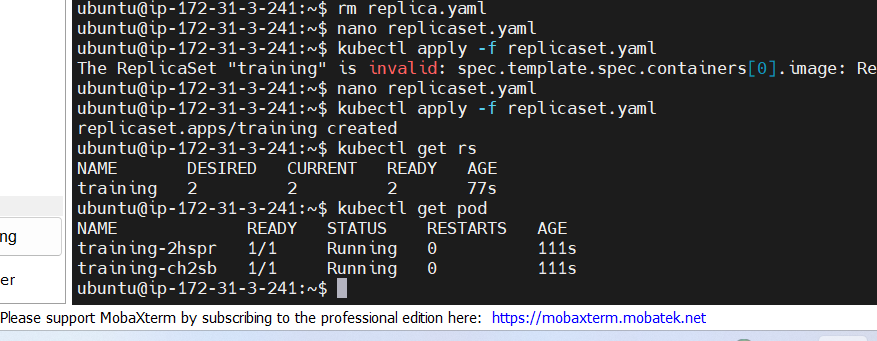
      containers:

      - image: ddthuong/nginx

        name: training

        ports:

          - containerPort: 80



Có 2 pod tạo ra là chúng ta đã chạy RS thành công. Đề xóa RS ta dùng câu lệnh:

kubectl delete rs training

# DaemonSet

Đây là một resource khác của kube, giống như RS, nó cũng sẽ giám xác và quản lý pod theo labels. Nhưng RS thì pod có thể deploy ở bất cứ node nào, và trong một node có thể chạy mấy pod cũng được. Còn DaemonSets này sẽ deploy tới mỗi node một pod duy nhất, và chắc chắn có bao nhiêu node sẽ có mấy nhiêu pod, nó sẽ không có thuộc tính replicas.

Tạo file daemonset.yaml:

apiVersion: apps/v1

kind: DaemonSet

metadata:

  name: ssd-monitor

spec:

  selector:

    matchLabels:

      app: ssd-monitor

  template:

    metadata:

      labels:

        app: ssd-monitor

    spec:

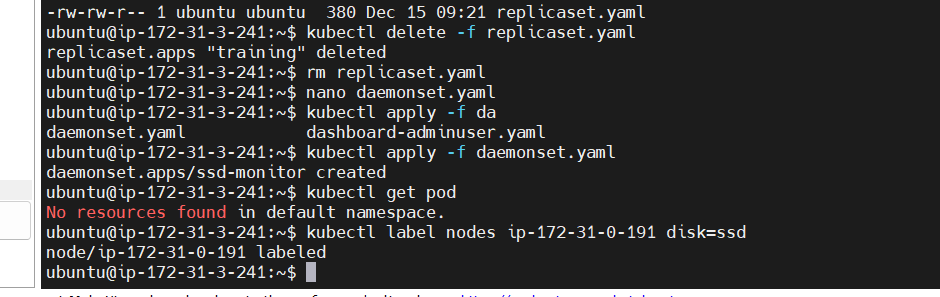
      nodeSelector:

        disk: ssd

      containers:

        - name: main

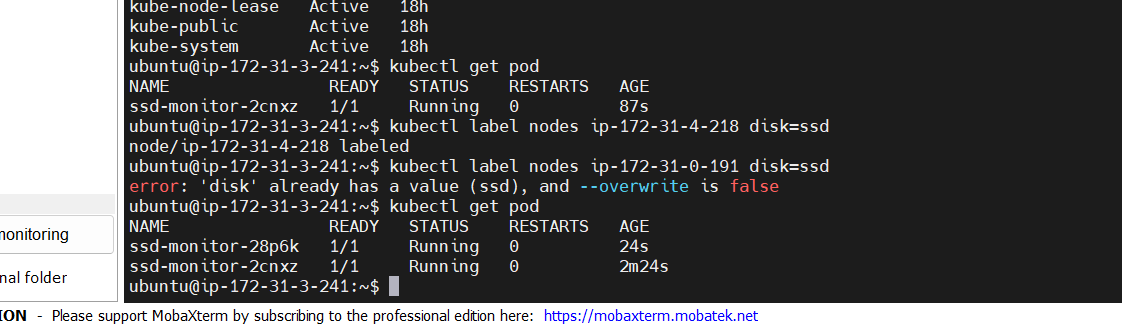
          image: luksa/ssd-monitor



Chỉ định DaemonSets ở cột nodeSelector với disk=ssd. Chỉ deploy thằng pod trên node có ổ đĩa ssd. Ta thấy chưa pod nào được tạo.

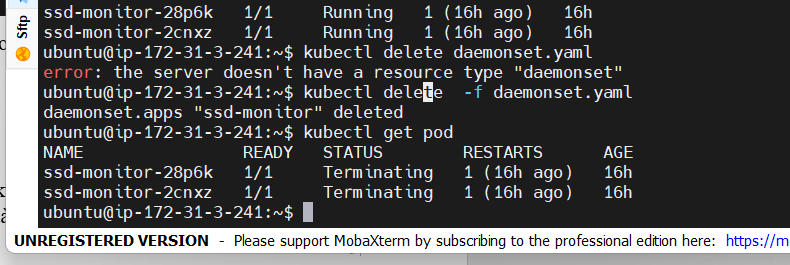
Đánh label cho worker node:

kubectl label nodes <your-node-name> disk=ssd



Daemonset đã chạy 2 pod trên e worker node đánh label.

Clear resource:



# Deployment

Deployment là một resource của kubernetes giúp ta trong việc cập cập một version mới của ứng dụng một cách dễ dàng, nó cung cấp sẵn 2 strategy để deploy là Recreate và RollingUpdate, tất cả đều được thực hiện tự động bên dưới, và các version được deploy sẽ có một history ở đằng sau, ta có thể rollback and rollout giữa các phiên bản bất cứ lúc nào mà không cần chạy lại CI/CD.

Tạo file deployment.yaml

apiVersion: apps/v1

kind: Deployment # change here

metadata:

  name: nginx

spec:

  replicas: 2

  strategy: # change here

    type: RollingUpdate

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - image: ddthuong/nginx

        name: nginx

        imagePullPolicy: Always

        ports:

          - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

  name: nginx

spec:

  type: NodePort

  selector:

    app: nginx

  ports:

    - port: 80

      targetPort: 80

      nodePort: 31000

Tạo deployment, khi ta tạo deployment nó sẽ tạo một thằng ReplicaSet ở bên dưới, và thằng ReplicaSet đó sẽ tạo Pod.

kubectl apply –f deployment.yaml --record

Để update ta sử dụng lệnh sau:

kubectl set image deployment nginx nginx=ddthuong/nginx:v2

Cấu trúc câu lệnh

kubectl set image deployment <deployment-name> <container-name>=<new-image>

Kiểm tra qua trình update đã xong chưa:

kubectl rollout status deploy nginx

Kiểm tra lịch sử cập nhật:

kubectl rollout history deploy nginx

Quay lại bản cập nhật trước:

kubectl rollout undo deployment hello-app --to-revision=2

# StatefulSet

Giống như ReplicaSet, StatefulSet là một resource giúp chúng ta chạy nhiều Pod mà cùng một template bằng cách set thuộc tính replicas, nhưng khác với ReplicaSet ở chỗ là Pod của StatefulSet sẽ được định danh chính xác và mỗi thằng sẽ có một stable network identity của riêng nó.

Mỗi Pod được tạo ra bởi StatefulSet sẽ được gán với một index, index này sẽ được sử dụng để định danh cho mỗi Pod. Và tên của Pod sẽ được đặt theo kiểu <statefulset name>-<index>, chứ không phải random như của ReplicaSet.

Tạo file stateful.yaml

apiVersion: v1

kind: Service

metadata:

  name: training

spec:

  clusterIP: None

  selector:

    app: training

  ports:

    - name: http

      port: 80

---

apiVersion: apps/v1

kind: StatefulSet

metadata:

  name: training

spec:

  serviceName: training # the name of service

  replicas: 2

  selector:

    matchLabels:

      app: training

  template: # pod template

    metadata:

      labels:

        app: training

    spec:

      containers:

        - name: training

          image: ddthuong/nginx

          ports:

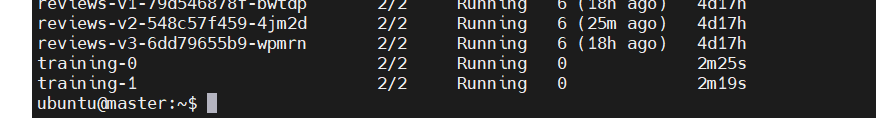
            - name: http

              containerPort: 80

Tạo StatefulSet:

kubectl apply –f stateful.yaml

Get list pod:



Ta thấy Pod chúng ta được tạo ra sẽ có tên được gán theo index.

# Volume

Volume hiểu đơn giản chỉ là một mount point từ hệ thống file của server vào bên trong container.

Tại sao ta cần volume thì đối với container, những thứ ta ghi vào filesystem của nó thì chỉ tồn tại khi container còn chạy. Khi một thằng Pod bị xóa và tạo lại, container mới sẽ được tao ra, lúc này thì những thứ ta ghi ở container trước sẽ bị mất đi. Nếu ta muốn giữ lại những dữ liệu đó thì ta phải sử dụng volume.

Trong kubernetes thì sẽ có những loại volume như sau:

* emptyDir
* hostPath
* gitRepo
* nfs
* gcePersistentDisk, awsElasticBlockStore, azureDisk **(cloud storage)**
* cinder, cephfs, iscsi, flocker, glusterfs, quobyte, rbd, flexVolume, vsphereVolume, photonPersistentDisk, scaleIO
* configMap, secret, downwardAPI
* PersistentVolumeClaim

Những loại volume trên được phân chia thành 3 dạng chính:

* Volume dùng để chia sẻ dữ liệu giữa các container trong Pod
* Volume đính kèm vào trong filesystem một node
* Volume đính kèm vào cluster và các node khác nhau có thể truy cập

Chúng ta không cần phải nhớ hết các loại volume, xài tới đâu thì ta google tìm kiếm tới đó. Ta chỉ cần nhớ một vài loại hay sử dụng nhất là **emptyDir, hostPath, cloud storage, PersistentVolumeClaim**.

## emptyDir

emptyDir là loại volume đơn giản nhất, nó sẽ tạo ra một empty directory bên trong Pod, các container trong một Pod có thể ghi dữ liệu vào bên trong nó. Volume chỉ tồn tại trong một lifecycle của Pod, dữ liệu trong loại volume này chỉ được lưu trữ tạm thời và sẽ mất đi khi Pod bị xóa. Ta dùng loại volume này khi ta chỉ muốn các container có thể chia sẻ dữ liệu lẫn nhau và không cần lưu trữ dữ liệu lại. Ví dụ là dữ liệu log từ một thằng container chạy web API, và ta có một thằng container khác sẽ truy cập vào log đó để xử lý log.

Ta sẽ làm một ví dụ đơn giản cho dễ hiểu hơn, tạo một file emptydir.yaml với config như sau:

apiVersion: v1

kind: Pod

metadata:

  name: fortune

spec:

  containers:

    - name: html-generator

      image: luksa/fortune

      volumeMounts:

        - name: html # The volume called html is mounted at /var/htdocs in the container

          mountPath: /var/htdocs

    - name: web-server

      image: nginx:alpine

      ports:

        - containerPort: 80

          protocol: TCP

      volumeMounts:

        - name: html # The volume called html is mounted at /usr/share/nginx/html in the container

          mountPath: /usr/share/nginx/html

          readOnly: true

  volumes: # define volumes

    - name: html # name of the volumes

      emptyDir: {} # define type is emptyDir

Code của script trong container **luksa/fortune**.

#!/bin/bash

trap "exit" SIGINT

mkdir /var/htdocs

while :

do

  echo $(date) Writing fortune to /var/htdocs/index.html

  /usr/games/fortune > /var/htdocs/index.html

  sleep 10

done

Container html-generator này sẽ cứ mỗi 10 giây sẽ tạo ra một nội dung bất kì và lưu nó vào file index.html. Và ta sẽ có một container khác, tên là web-server, sẽ start một server và hosting nội dung ở folder /usr/share/nginx/html (folder mặc định của nginx).

Ở đây ta có một emptyDir volume tên là html, được mount vào container html-generator ở folder /var/htdocs và container html-generator sẽ tạo một file html index.html ở trong emptyDir volume này. Và emptyDir volume này được mount tới container web-server, ở folder /usr/share/nginx/html. Nên khi ta truy cập container web thì ta sẽ thấy được những nội dung mà container html-generator đã tạo ra.

Test xem nó có hoạt động đúng không.

kubectl apply –f emptydir.yaml

kubectl port-forward fortune 8080:80

curl http://localhost:8080

Lưu ý là chỉ dùng emptyDir để chia sẻ dữ liệu giữa những container chứ không dùng để lưu persistent data.

## gitRepo

gitRepo là loại volume cũng giống emptyDir là sẽ tạo một empty folder, và sau đó nó sẽ clone code của git repository vào folder này.

Ví dụ như sau, tạo một file tên là gitRepo.yaml với config như sau:

apiVersion: v1

kind: Pod

metadata:

  name: gitrepo-volume-pod

spec:

  containers:

    - image: nginx:alpine

      name: web-server

      volumeMounts:

        - name: html

          mountPath: /usr/share/nginx/html

          readOnly: true

      ports:

        - containerPort: 80

          protocol: TCP

  volumes:

    - name: html

      gitRepo: # gitRepo volume

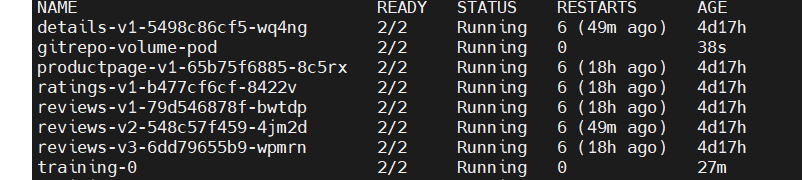
        repository: https://github.com/Thuong1234/Training.git # The volume will clone this Git repository

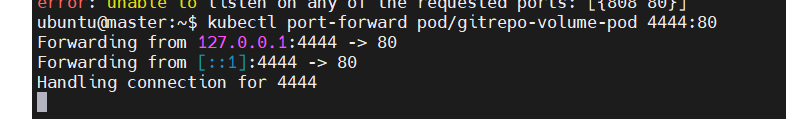
        revision: main # master branch

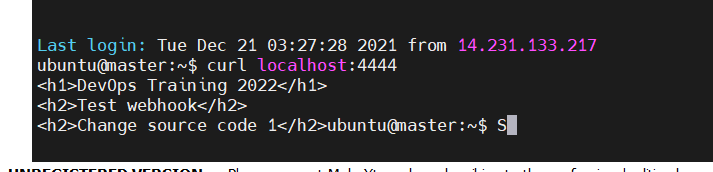
        directory: . # cloned into the root dir of the volume.

Ta xem nó có hoạt động không:

kubectl apply –f gitRepo.yaml







## hostPath

hostPath là loại volume sẽ tạo một mount point từ Pod ra ngoài filesystem của node. Đây là loại volume đầu tiên ta nói mà có thể dùng để lưu trữ persistent data. Dữ liệu lưu trong volume này chỉ tồn tại trên một worker node và sẽ không bị xóa đi khi Pod bị xóa.

Tạo file hostpath.yaml:

apiVersion: v1

kind: Pod

metadata:

  name: hostpath-volume

spec:

  containers:

    - image: ddthuong/nginx

      name: web-server

      volumeMounts:

        - name: log # log volume

          mountPath: /var/log/nginx # mounted at /var/log/nginx in the container

      ports:

        - containerPort: 80

          protocol: TCP

  volumes:

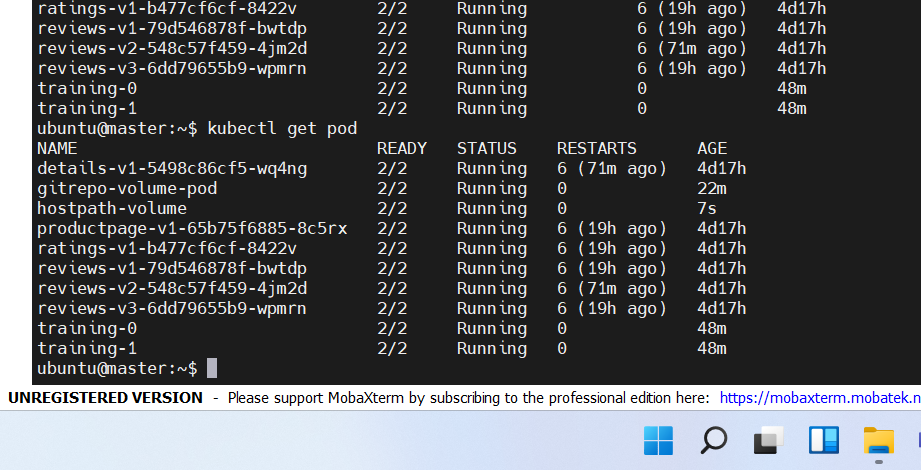
    - name: log

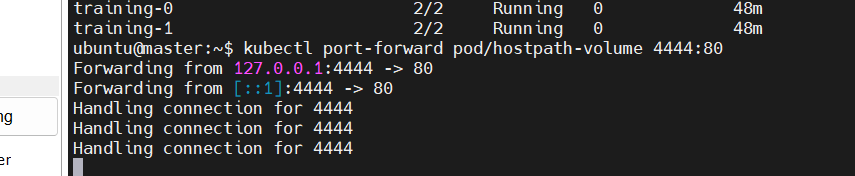
      hostPath: # hostPath volume

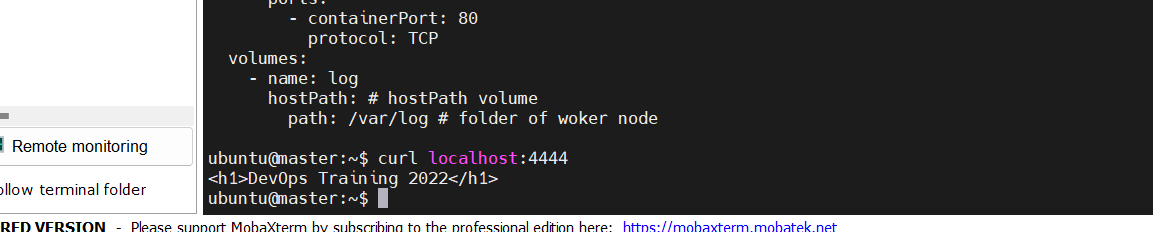
        path: /var/log # folder of worker node

Tạo hostpath:

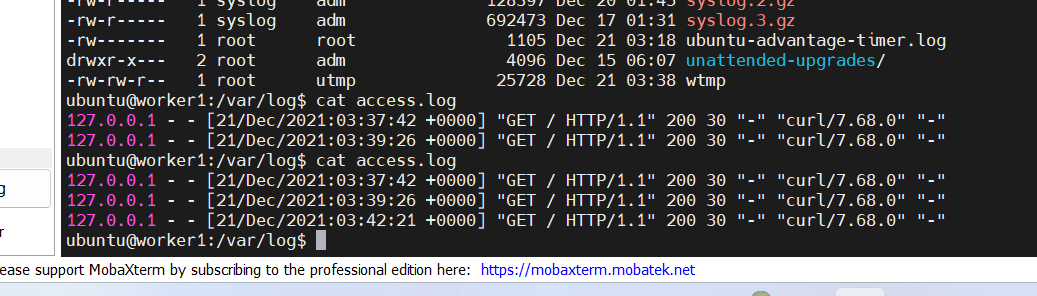
kubectl apply –f hostpath.yaml







Kiểm tra trong đường dẫn của hostpath thấy log đã được ghi.



# Persistent Volume

PersistentVolumes là resource sẽ tương tác với kiến trúc storage bên dưới, và PersistentVolumeClaims sẽ request storage từ PersistentVolumes, tương tự như Pod. **Pods tiêu thụ node resources và PersistentVolumeClaims tiêu thụ PersistentVolumes resources**.

Tạo PersistentVolume file pv.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

  name: aws-pv

  labels:

    type: aws-pv

spec:

  capacity:

    storage: 3Gi

  accessModes:

    - ReadWriteOnce

  awsElasticBlockStore:

    volumeID: vol-0ca0077fbc12216dc # <YOUR EBS VOLUME ID HERE>

    fsType: xfs

Tạo PersistentVolumeClaim file pvc.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: aws-pvc

  labels:

     type: aws-pvc

spec:

  accessModes:

    - ReadWriteOnce

  resources:

    requests:

      storage: 3Gi

  selector:

    matchLabels:

      type: aws-pv

Chạy và kiểm tra kết quả:



Ta đã claim được PV, để sử dụng ta mount volume vào pod với persistentVolumeClaim.

apiVersion: v1

kind: Pod

metadata:

  name: nginx

spec:

  containers:

    - name: web-server

      image: nginx:alpine

      ports:

        - containerPort: 80

          protocol: TCP

      volumeMounts:

        - name: html # The volume called html is mounted at /usr/share/nginx/html in the container

          mountPath: /usr/share/nginx/html

  volumes: # define volumes

    - name: html # name of the volumes

      persistentVolumeClaim:

        claimName: aws-pvc