Contents

[Cài đặt: 5](#_Toc107031439)

[kubectl 5](#_Toc107031440)

[kubecolor (đã có ansible) 5](#_Toc107031441)

[bash-completion 6](#_Toc107031442)

[minikube 6](#_Toc107031443)

[Helm(đã có ansible) 6](#_Toc107031444)

[Cài đặt từ Repo 6](#_Toc107031445)

[Tool hay cho k8s: 11](#_Toc107031446)

[Kompose = Convert docker-compose to K8s 11](#_Toc107031447)

[Kubecolor 11](#_Toc107031448)

[Indent-rainbow 11](#_Toc107031449)

[THUẬT NGỮ: 11](#_Toc107031450)

[File YAML Mô tả cấu trúc 12](#_Toc107031451)

[Kiến trúc k8s 12](#_Toc107031452)

[Section1: Core concept k8s 13](#_Toc107031453)

[1.1 POD 13](#_Toc107031454)

[GET pod 13](#_Toc107031455)

[Run Pod 13](#_Toc107031456)

[APPLY pod 13](#_Toc107031457)

[EXEC Để chạy command trong pod 13](#_Toc107031458)

[EXEC -it Truy cập vào Pod: 13](#_Toc107031459)

[Port-Froward 14](#_Toc107031460)

[DELETE pod 14](#_Toc107031461)

[LOG pod 14](#_Toc107031462)

[Describe pod 14](#_Toc107031463)

[1.2 Job and CronJob 14](#_Toc107031464)

[1.3 LABEL 14](#_Toc107031465)

[1.4 NameSpace 15](#_Toc107031466)

[1.5 Services 15](#_Toc107031467)

[VD1: ClusterIP: 15](#_Toc107031468)

[NodePort 15](#_Toc107031469)

[LoadBalancer 15](#_Toc107031470)

[1.6 Ingress: 15](#_Toc107031471)

[1.7 DEPLOYMENT 15](#_Toc107031472)

[VD2 Zero downtime deployment: 16](#_Toc107031473)

[Replicaset (rs) 16](#_Toc107031474)

[DELETE Replicaset 18](#_Toc107031475)

[DaemonSET 18](#_Toc107031476)

[Statefull Set 18](#_Toc107031477)

[Section 3: Schedule 20](#_Toc107031478)

[3.1 nodeName 20](#_Toc107031479)

[3.2 nodeSelector 20](#_Toc107031480)

[3.3 Taints and Toleration 20](#_Toc107031481)

[3.4 nodeAffinity 21](#_Toc107031482)

[3.5 podAntiAffinity 21](#_Toc107031483)

[3.6 StaticPod 21](#_Toc107031484)

[3.7 MultiSchedule 21](#_Toc107031485)

[3.8 Resource Limit 21](#_Toc107031486)

[RAM: 21](#_Toc107031487)

[CPU: 23](#_Toc107031488)

[LimitRange namespace 23](#_Toc107031489)

[NETWORKING 24](#_Toc107031490)

[Readiness & Liveness 25](#_Toc107031491)

[Section5: Application Lifecycle Managerment 25](#_Toc107031492)

[1.8 Rolling update and rollback 25](#_Toc107031493)

[Command and Argument 25](#_Toc107031494)

[ENV - CONFIG MAP – SECRET 26](#_Toc107031495)

[ENV đơn 26](#_Toc107031496)

[ConfigMap 27](#_Toc107031497)

[Create config map 27](#_Toc107031498)

[Configmap as ENV 27](#_Toc107031499)

[Configmap as Volume 28](#_Toc107031500)

[Configmap as File in volume 28](#_Toc107031501)

[Secret 30](#_Toc107031502)

[Secret as ENV 30](#_Toc107031503)

[Secret as file in folder 31](#_Toc107031504)

[Secret as configfile 32](#_Toc107031505)

[MultiContainerPod 32](#_Toc107031506)

[InitContainer 32](#_Toc107031507)

[Section 6: Cluster Maintain 32](#_Toc107031508)

[1. OS upgrade (drain node) 32](#_Toc107031509)

[2. Cluster upgrade 33](#_Toc107031510)

[3. Backup / Restore ETCD 33](#_Toc107031511)

[Section 7: Security 34](#_Toc107031512)

[Api Groups 35](#_Toc107031513)

[ABAC Authorization 36](#_Toc107031514)

[154 RBAC / Kubernetes API 39](#_Toc107031515)

[Tạo Role/RoleBinding như sau: 39](#_Toc107031516)

[VD1: Role binding / service account – gọi trong pod 40](#_Toc107031517)

[VD2: Role binding / service account – gọi từ ngoài 41](#_Toc107031518)

[VD3: Kube Config 41](#_Toc107031519)

[159: Image security: with docker private registry 42](#_Toc107031520)

[162 Security Context 43](#_Toc107031521)

[165 Network Policies 44](#_Toc107031522)

[Section 8: Storage 44](#_Toc107031523)

[VOLUME 44](#_Toc107031524)

[emptyDir 44](#_Toc107031525)

[hostPath 45](#_Toc107031526)

[nfs 46](#_Toc107031527)

[Storage Class / PV / PVC 46](#_Toc107031528)

[PV 46](#_Toc107031529)

[PVC 46](#_Toc107031530)

[StorageClass 46](#_Toc107031531)

[VD1: Storage Class as NFS a Trung 48](#_Toc107031532)

[Recycling PersistentVolume 48](#_Toc107031533)

[Section 9: Networking 48](#_Toc107031534)

[187 CoreDNS 48](#_Toc107031535)

[Network namespace 49](#_Toc107031536)

[Cluster Networking in k8s 50](#_Toc107031537)

[196. Pod networking 51](#_Toc107031538)

[200. Weaver Network 51](#_Toc107031539)

[207 Services Networking 51](#_Toc107031540)

[217 Ingress 51](#_Toc107031541)

[Section 10: Install and design kubernetes cluster 52](#_Toc107031542)

[Section 11: Install kubernetes the kubeadm way 52](#_Toc107031543)

[TIP-TRICK 52](#_Toc107031544)

[Install telnet in docker apk 52](#_Toc107031545)

[Sample Prod 52](#_Toc107031546)

[Kube Dashboard 52](#_Toc107031547)

[K8s Registry: 53](#_Toc107031548)

[Ingress nginx: 57](#_Toc107031549)

[Nfs provisioning 58](#_Toc107031550)

[Debug image 60](#_Toc107031551)

---

# Cài đặt:

git clone <https://github.com/luksa/kubernetes-in-action.git>

### kubectl

cd /opt/; curl -LO [https://dl.k8s.io/release/**$(**curl -L -s https://dl.k8s.io/release/stable.txt**)**/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)

chmod +x kubectl

### kubecolor (giống với kubectl, thêm màu sắc dễ nhìn)

wget <https://github.com/hidetatz/kubecolor/releases/download/v0.0.20/kubecolor_0.0.20_Linux_x86_64.tar.gz>

tar -xvzf kubecolor\_0.0.20\_Linux\_x86\_64.tar.gz

mv kubecolor /usr/bin/ ; chmod 755 /usr/bin/kubecolor

### bash-completion

#apt-get install bash-completion

sudo yum install bash-completion -y

echo 'source <(kubectl completion bash)' >>~/.bashrc

echo 'alias k=kubectl' >>~/.bashrc

echo 'complete -F \_\_start\_kubectl k' >>~/.bashrc

### Helm(đã có ansible)

wget <https://get.helm.sh/helm-v3.8.1-linux-amd64.tar.gz>

(hoặc: curl -sSL <https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3> | bash )

sudo yum install bash-completion -y

helm completion bash > /etc/bash\_completion.d/helm

### Kustomize

curl -s "https://raw.githubusercontent.com/kubernetes-sigs/kustomize/master/hack/install\_kustomize.sh" **|** bash

### Minikube (dùng cho lab k8s, không phải dựng cụm cluster bằng tay)

(yêu cầu phải cài docker hoặc podman: curl -fsSL https://get.docker.com/ | sh )

curl -LO <https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64>

chmod +x minikube-linux-amd64

mv minikube-linux-amd64 minikube

sudo install ./minikube /usr/local/bin/minikube

su tuanda

minikube start

minikube stop

## Cài đặt từ Repo

<https://bikramat.medium.com/set-up-a-kubernetes-cluster-with-kubeadm-508db74028ce>

<https://phoenixnap.com/kb/how-to-install-kubernetes-on-centos>

<https://phoenixnap.com/kb/how-to-install-kubernetes-on-a-bare-metal-server>

<https://xuanthulab.net/gioi-thieu-va-cai-dat-kubernetes-cluster.html>

**B1: Đặt Hostname (run on Master + Worker node)**

hostnamectl set-hostname master-node

hostnamectl set-hostname worker-node1

hostnamectl set-hostname worker-node2

# cat << EOF >> /etc/hosts

192.168.88.12 master-node

192.168.88.13 worker-node1

192.168.88.14 worker-node2

EOF

**B2: Setting cơ bản (run on Master + Worker node)**

Disable Swap trên master và worker

# Tat swap

sed -i '/swap/d' /etc/fstab

swapoff -a

cat <<EOF > /etc/sysctl.d/k8s.conf

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

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

EOF

sysctl --system

**Disable Selinux: (run on Master + Worker node)**

sudo setenforce 0

sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

sudo sed -i 's/^SELINUX= permissive$/SELINUX=disabled/' /etc/selinux/config

**B3: Cài docker-ce (run on Master + Worker node)**

yum install epel-release -y ; curl -fsSL https://get.docker.com/ | sh

adduser tuanda ; usermod -aG docker tuanda

echo 'tuanda ALL=(ALL)NOPASSWD: ALL' > /etc/sudoers.d/tuanda

## Create /etc/docker directory.

mkdir /etc/docker

## Thay đổi cgroup theo từng loại OS (Centos/Ubuntu/...)

[tuanda@master-node ~]$ sudo docker info | grep -i cgroup

Cgroup Driver: systemd

Cgroup Version: 1

# Setup daemon theo cgroup ở trên.

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

{

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

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

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2",

"storage-opts": [

"overlay2.override\_kernel\_check=true"

]

}

EOF

#Chú ý cần có dòng này

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

# Restart Docker

systemctl enable docker.service

systemctl daemon-reload

systemctl restart docker

**Cài đặt kubelet/kubeadm/kubectl** **(run on Master + Worker node)**

cat <<EOF > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=0

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

yum install -y kubelet kubeadm kubectl

service kubelet start

systemctl enable kubelet.service

telnet localhost 10248

Mở port master node

sudo firewall-cmd --permanent --add-port=6443/tcp

sudo firewall-cmd --permanent --add-port=2379-2380/tcp

sudo firewall-cmd --permanent --add-port=10250/tcp

sudo firewall-cmd --permanent --add-port=10251/tcp

sudo firewall-cmd --permanent --add-port=10252/tcp

sudo firewall-cmd --permanent --add-port=10255/tcp

sudo firewall-cmd --reload

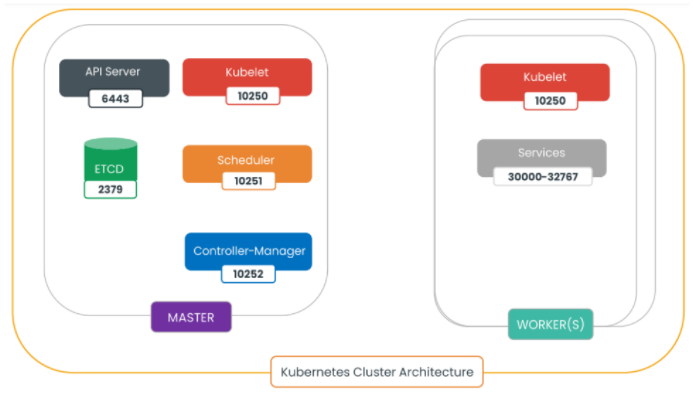
Mở port worker node

sudo firewall-cmd --permanent --add-port=10251/tcp

sudo firewall-cmd --permanent --add-port=10255/tcp

sudo firewall-cmd --reload

URL Port cần mở ở đây <https://kubernetes.io/docs/reference/ports-and-protocols/>



Bước 4 : Khởi tạo Master Node **(run on Master node)**

kubeadm init --apiserver-advertise-address=192.168.88.12 --pod-network-cidr=10.244.0.0/16

(trường hợp tạo lỗi, ta có thể gõ lệnh #kubeadm reset)

su - tuanda

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

Lệnh in lại command join: kubeadm token create --print-join-command HOẶC <https://ystatit.medium.com/regenerate-kubernetes-join-command-to-join-work-node-7eeb5d1f5787>

Bước 5: Pod Network trên Master-Node **(run on Master node)**

Ta có thể dùng nhiều addon như Flanel, cacilo, weaver. Và network chỉ apply được sau khi init master-node.

**Hoặc Calico**

Tham khảo cách cài calico 50node, 100node, etcd: <https://projectcalico.docs.tigera.io/getting-started/kubernetes/self-managed-onprem/onpremises>

# curl https://projectcalico.docs.tigera.io/manifests/calico.yaml -O

# kubectl apply -f calico.yaml

Bước 6: Join Worker node: **(run on Worker node)**

kubeadm join 192.168.88.12:6443 --token h46n34.uq80d4pro1qjyvk0 --discovery-token-ca-cert-hash xxxxxxxxxxx

Bước 7: Kiểm tra **(run on Master node)**

[tuanda@master-node ~]$ kubectl get node

[tuanda@master-node ~]$ kubectl cluster-info

[tuanda@master-node ~]$ kubectl get pod -A

Other: remove node:

kubectl drain Ten\_Node

# Tool hay cho k8s:

## Kompose (Convert docker-compose sang K8s)

<https://kompose.io/> (cài đặt)

$ kompose convert -f docker-compose.yaml

# THUẬT NGỮ:

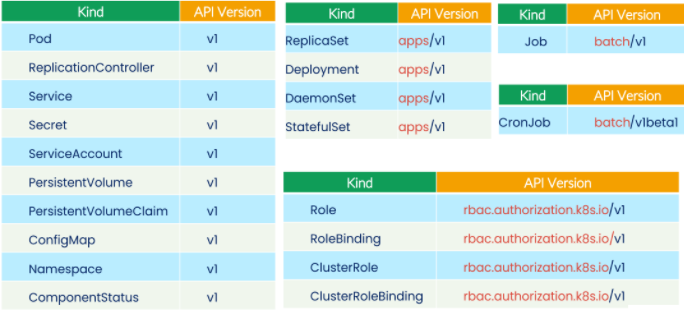
**Replicaset**: tạo ra multi-pod chạy cùng trên 1 image

**Depoyment**: Quản lý các replicaset, phục vụ cho việc thay đổi image hoặc cấu hình.

**Statefullset**: Khi tạo statefullset, Pod bị xóa đi thì pod mớ được sinh ra sẽ kế thừa network + volume từ pod cũ. mặc định sẽ tự tạo thêm PVC để giữ PV luôn cố định, dữ liệu sẽ không thay đổi, thích hợp cho sử dụng DB

**DaemonSet**:

# File YAML Mô tả cấu trúc



# KIẾN TRÚC K8S

* ETCD: etcd lưu toàn bộ cấu hình của k8s dưới dạng key-value (pod, services, deployment ….)
* kube-apiserver: cổng trao đổi toàn bộ tông tin của ks
* kube-controler-manager: có chức năng theo dõi health toàn bộ cluster: pod, node, job, replication, pv, pvc…
* kube-scheduler: quyết định pod/job mới sẽ đặt trên node nào (thỏa mãn taint, affinity)
* kubelet: tiến trình tại worker-node. cầu nối giao tiếp với master
* kube-proxy: quản lý network và port toàn bộ các node

# Section1: Core concept k8s

## 1.1 POD

Mỗi 1 microserver sẽ đặt trên 1 pod.

Để kiểm tra pod, ta sử dụng những lệnh sau:

### GET pod

kubectl get all

kubectl get pods

kubectl get pod --show-labels

kubectl explain pods

kubectl get pod nginxtest -o yaml (hoặc json)

kubectl get all -o wide (hiển thị rộng hơn)

### Run Pod

# kubectl run kubia --image=nginx:alpine --port=80

# kubectl run -i --tty busybox --image=busybox --restart=Never –- sh

# kubectl run busybox --image=busybox --restart=Never -o yaml --dry-run=client -- /bin/sh -c 'echo hello;sleep 3600'

### APPLY pod

Apply khác với create là vừa tạo, nếu đã có sẽ update.

# kubectl apply -f kubia-manual.yaml

### EXEC truy cập vào pod

kubectl exec [POD] -- [COMMAND]

# **kubectl exec kubia-manual -- ls**

# **kubectl -it exec webapp -- sh**

### Port-Froward

# **kubectl port-forward kubia-manual 8888:8080**

### DELETE pod

# **kubectl delete pod nginx**

# **kubectl delete pod --all** (xóa tất cả các pod trong cùng NS)

# kubectl delete pods <pod name> --grace-period=0 –force (Xóa pod bị stuck)

# kubectl patch pod <pod> -p '{"metadata":{"finalizers":null}}' (xóa pod nếu bị null)

<https://kubernetes.io/docs/tasks/run-application/force-delete-stateful-set-pod/#force-deletion>

### LOG pod

# kubectl logs -f nginx

### Describe pod

# kubectl describe pod nginx

## 1.2 Job and CronJob (QUEUE)

<https://vocon-it.com/2019/07/28/cka-labs-7-kubernetes-jobs-and-cronjobs/>

## 1.3 LABEL

kubectl get node --show-labels

kubectl get pod --show-labels

Add thêm labels:

kubectl label pod kubia-gg5t5 type=special

Thay label (tách pod ra khỏi replicaset/control)

[tuanda@localhost Chapter04]$ kubectl label pod kubia-gg5t5 app=ahihi --overwrite

## 1.4 NameSpace

# kubectl get ns

# k get all –A

## 1.5 Services (QUEUE)

### VD1: ClusterIP:

Đây là loại truy cập nội bộ các pod với nhau, không truy cập từ bên ngoài vào đc.

### NodePort

Nodeport có thể cho client bên ngoài gọi đc. Bằng IP của các node cluster. Nodeport có range từ 30000 đến 32767

### LoadBalancer

Hỗ trợ cả bên ngoài client và trong pod đều gọi vào đc.

# kubectl expose deploy/nginx --type=LoadBalancer --port=80

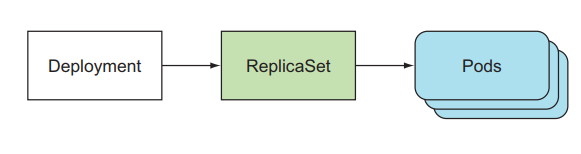
## 1.6 Ingress: (QUEUE)

<https://github.com/nguyenan122/k8s-collection/tree/main/sample_final/003.ingress-nginx>

## 1.7 DEPLOYMENT

kubectl create —-record=true -f nginx-deployment.yaml

Lệnh sẽ lưu lịch sử của deployment, để kiểm tra: kubectl rollout history deploy/nginx



Để đổi trực tiếp image mới ta có thể làm như sau:

[tuanda@localhost Chapter09]$ kubectl edit deployments.apps kubia

Hoặc

[tuanda@localhost Chapter09]$ kubectl set image deployment kubia nodejs=luksa/kubia:v2 (hoặc kubia:v3, v4)

Để khôi phục lại version trước đó, ta có các lệnh sau:

[tuanda@localhost Chapter09]$ kubectl rollout undo deployment kubia

Để theo dõi real-time undo đang làm gì, ta có lệnh status sau

[tuanda@localhost Chapter09]$ kubectl rollout status deployment kubia

Để show các version rollout, ta có lệnh sau:

[tuanda@localhost Chapter09]$ kubectl rollout history deployment kubia

Để đổi về 1 version history có chỉ định

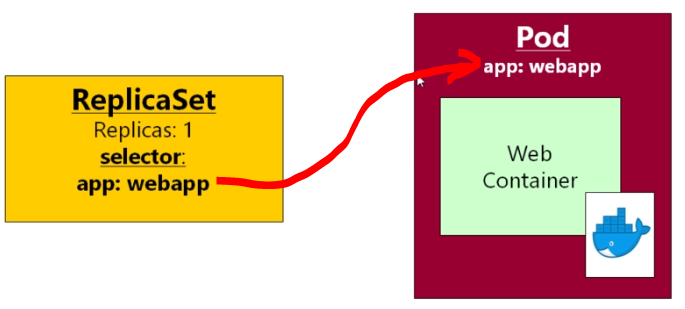
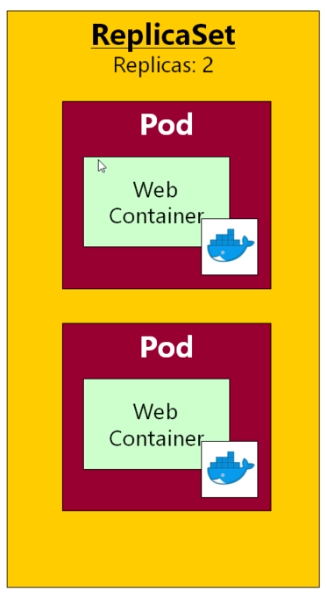
[tuanda@localhost Chapter09]$ kubectl rollout undo deployment kubia --to-revision=4

### VD2 Zero downtime deployment: (QUEUE)

<https://viblo.asia/p/zero-downtime-voi-kubernetes-p1-truly-stateless-application-ORNZqjErl0n>

## Replicaset (rs)

Được quản lý bởi deployment.

 >>>>chuyển sang>>>>>>>

[tuanda@localhost Chapter04]$ cat kubia-replicaset.yaml

apiVersion: apps/v1beta2

kind: ReplicaSet

metadata:

  name: kubia

spec:

  replicas: 3

  selector:

    matchLabels:

      app: kubia

  template:

    metadata:

      labels:

        app: kubia

    spec:

      containers:

      - name: kubia

        image: luksa/kubia

[tuanda@localhost Chapter04]$ kubectl describe rs kubia

# kubectl get rs

# kubectl edit rs Tên\_RS

## DaemonSET (QUEUE)

Giúp đảm bảo mỗi node đều được deploy pod. Dùng để tăng tính dự phòng của hệ thống khi node chết, thì pod ở node khác vẫn chạy

## Statefull Set (QUEUE)

Khi tạo statefullset, mặc định sẽ tự tạo thêm PVC để giữ PV luôn cố định, dữ liệu sẽ không thay đổi, thích hợp cho sử dụng DB

[tuanda@localhost Chapter10]$ cat persistent-volumes-hostpath.yaml

kind: List

apiVersion: v1

items:

- apiVersion: v1

  kind: PersistentVolume

  metadata:

    name: pv-a

  spec:

    capacity:

      storage: 1Mi

    accessModes:

      - ReadWriteOnce

    persistentVolumeReclaimPolicy: Recycle

    hostPath:

      path: /tmp/pv-a

- apiVersion: v1

  kind: PersistentVolume

  metadata:

    name: pv-b

  spec:

    capacity:

      storage: 1Mi

    accessModes:

      - ReadWriteOnce

    persistentVolumeReclaimPolicy: Recycle

    hostPath:

      path: /tmp/pv-b

- apiVersion: v1

  kind: PersistentVolume

  metadata:

    name: pv-c

  spec:

    capacity:

      storage: 1Mi

    accessModes:

      - ReadWriteOnce

    persistentVolumeReclaimPolicy: Recycle

    hostPath:

      path: /tmp/pv-c

[tuanda@localhost Chapter10]$ cat kubia-statefulset.yaml

apiVersion: apps/v1

kind: StatefulSet

metadata:

  name: kubia

spec:

  serviceName: kubia

  replicas: 2

  selector:

    matchLabels:

      app: kubia # has to match .spec.template.metadata.labels

  template:

    metadata:

      labels:

        app: kubia

    spec:

      containers:

      - name: kubia

        image: luksa/kubia-pet

        ports:

        - name: http

          containerPort: 8080

        volumeMounts:

        - name: data

          mountPath: /var/data

  volumeClaimTemplates:

  - metadata:

      name: data

    spec:

      resources:

        requests:

          storage: 1Mi

      accessModes:

      - ReadWriteOnce

# Section 3: Schedule

## 3.1 nodeName (QUEUE)

nodeName là chọn trực tiếp tên node để gán pod vào (không dùng label như nodeSelector). Ngoài ra nodeName giúp cài pod lên node mà không cần kube-system scheduler đang chạy.

## 3.2 nodeSelector (QUEUE)

## 3.3 Taints and Toleration

Đầu tiên ta kiểm tra Taint giữa master và worker có gì khác nhau:

[root@master-node ~]# k describe node master-node | grep -i taint

Taints: node-role.kubernetes.io/master:NoSchedule

[root@master-node ~]# k describe node worker-node1 | grep -i taint

Taints: <none>

Điều kiện 1 pod có thể deploy là có Toleration giống với Taint của node.

[root@master-node ~]# k describe pod calico-kube-controllers-566dc76669-mvtn2 -n kube-system | grep master

Tolerations:

node-role.kubernetes.io/master:NoSchedule

## 3.4 nodeAffinity (QUEUE)

## 3.5 podAntiAffinity (QUEUE)

## 3.6 StaticPod (QUEUE)

## 3.7 MultiSchedule (QUEUE)

## 3.8 Resource Limit (QUEUE)

<https://vocon-it.com/2019/07/19/certified-kubernetes-administrator-labs-4-resource-management/>

kubectl create deployment stress --image vish/stress --dry-run=client -o yaml > stress.yaml

k apply -f stress.yaml

### RAM:

Ta thực hiện thêm limit/request vào file stress.yaml

[tuanda@master-node k8s-resource-limit]$ cat stress.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  creationTimestamp: null

  labels:

    app: stress

  name: stress

spec:

  replicas: 1

  selector:

    matchLabels:

      app: stress

  strategy: {}

  template:

    metadata:

      creationTimestamp: null

      labels:

        app: stress

    spec:

      containers:

      - image: vish/stress

        name: stress

        resources:

          limits:

            memory: "500Mi"

          requests:

            memory: "250Mi"

args:

- -cpus

- "2"

- -mem-total

- "400Mi"

- -mem-alloc-size

- "100Mi"

- -mem-alloc-sleep

- "1s"

        terminationMessagePolicy: FallbackToLogsOnError

status: {}

(lệnh thực thi trong pod: /stress -logtostderr -cpus 2 -mem-total 400Mi -mem-alloc-size 100Mi -mem-alloc-sleep 1s)

1. Khi apply config trên. CPU sẽ sử dụng 2 core và ram là 400Mi ở ức cho phép < 500 Mi. Nên Pod vẫn chạy BT

2. Khi sửa lại mem-total=600Mi. Pod sẽ xuất hiện OOMKilled > CrashLoopBackOff > Pod sẽ bị xóa

### CPU:

        resources:

          limits:

            cpu: "0.4"

            memory: "500Mi"

          requests:

            cpu: "0.1"

            memory: "250Mi"

        args:

        - -cpus

        - "2"

        - -mem-total

        - "100Mi"

        - -mem-alloc-size

        - "100Mi"

        - -mem-alloc-sleep

        - "1s"

        terminationMessagePolicy: FallbackToLogsOnError

kiểm tra top-c ta sẽ thấy tiến trình chỉ sử dụng 40% của 1 core

Ta thử sửa deployment thành 5 pod. Thì mỗi pod sẽ chiếm 40% của 1 core. Tổng là 200%

### LimitRange namespace

Ta có thể limit resource theo toàn bộ NS như sau:

#vim limitrange.yaml

apiVersion: v1

kind: LimitRange

metadata:

  name: low-resource-range

spec:

  limits:

  - type: Container

    default:

      cpu: 0.2

    defaultRequest:

      cpu: 0.1

Tạo ns, apply LitmitRage vào NS, sau đó deploy pod và xem spec

# k create namespace low-resource-range

# k create -f limitrange.yaml --namespace low-resource-range

# k create deployment nginx --image=nginx --namespace low-resource-range

# kubectl get pod -n low-resource-range nginx-85b98978db-bfqcn -o yaml

    name: nginx

    resources:

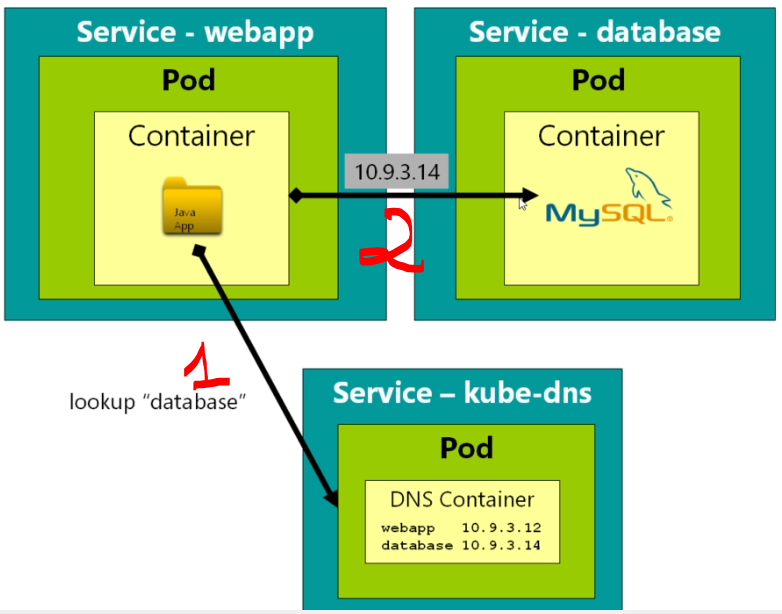
      limits:

        cpu: 200m

      requests:

        cpu: 100m

# NETWORKING



Ta sẽ thực hiện trỏ bằng dns, sau đó kube-dns sẽ phân giải ra IP cần tìm.

Kiểm tra dns

# kubectl get svc kube-dns -n kube-system

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kube-dns ClusterIP 10.96.0.10 <none> 53/UDP,53/TCP,9153/TCP 24h

# kubectl describe svc kube-dns -n kube-system

# Readiness & Liveness (QUEUE)

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

# Section5: Application Lifecycle Managerment

## 1.8 Rolling update and rollback

Để biết tỉ lệ pod rollingupdate là bao nhiêu, ta có thể describe pod, nhìn mục StrategyType và RollingUpdateStrategy = 25%

Có các loại StrategyType sau:

* Recreate: xóa all pod sau đó launch pod mới
* Rolling update: Tạo pod mới, sau đó xóa theo tỉ lệ phần trăm đặt sẵn, chèn pod mới vào.
* Rename Rollout:

## Command and Argument

kubectl run nginx --image=nginx --dry-run=client -o yaml --command -- sleep 10

## ENV - CONFIG MAP – SECRET

### ENV đơn

apiVersion: v1

kind: Pod

metadata:

  name: fortune-env

spec:

  containers:

  - image: luksa/fortune:env

    env:

    - name: INTERVAL

      value: "30"

    - name: TUANDA

      value: "kaka"

    name: html-generator

    volumeMounts:

    - name: html

      mountPath: /var/htdocs

  - image: nginx:alpine

    name: web-server

    volumeMounts:

    - name: html

      mountPath: /usr/share/nginx/html

      readOnly: true

    ports:

    - containerPort: 80

      protocol: TCP

  volumes:

  - name: html

    emptyDir: {}

[tuanda@localhost Chapter07]$ kubectl exec -it fortune-env -- printenv

INTERVAL=30

TUANDA=kaka

## ConfigMap

<https://kubernetes.io/docs/concepts/configuration/configmap/>

### Create config map

**Tạo config map từ command-line**

[tuanda@localhost Chapter07]$ kubectl create configmap fortune-config --from-literal=sleep-interval=25

**Hoặc từ file config hoặc yaml, json (thích hợp với import file dài, khó)**

[tuanda@localhost configmap-files]$ kubectl create configmap tuanda-config --from-file=customkey=my-nginx-config.conf

### Configmap as ENV

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/>

apiVersion: v1

kind: Pod

metadata:

  name: fortune-env-from-configmap

spec:

  containers:

  - image: luksa/fortune:env

    env:

    - name: INTERVAL

      valueFrom:

        configMapKeyRef:

          name: fortune-config

          key: sleep-interval

    name: html-generator

    volumeMounts:

    - name: html

      mountPath: /var/htdocs

  - image: nginx:alpine

    name: web-server

    volumeMounts:

    - name: html

      mountPath: /usr/share/nginx/html

      readOnly: true

    ports:

    - containerPort: 80

      protocol: TCP

  volumes:

  - name: html

    emptyDir: {}

khi vào pod, ta sẽ thấy container có biến môi trường là : INTERVAL=25

### Configmap as Volume

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/>

apiVersion: v1

kind: Pod

metadata:

  name: dapi-test-pod

spec:

  containers:

    - name: test-container

      image: busybox

      command: [ "/bin/sh", "-c", "ls /etc/config/" ]

      volumeMounts:

      - name: config-volume

        mountPath: /etc/folder

  volumes:

    - name: config-volume

      configMap:

        name: special-config

  restartPolicy: Never

### Configmap as File in volume

# echo tuanda=123 > config.properties

# k create configmap configmap-test111 --from-file=config.properties

# k run podtest --image=busybox --dry-run=client -o yaml --command -- sleep 100000 > pod.yaml

Sửa file pod thêm volume như sau:

apiVersion: v1

kind: Pod

metadata:

  creationTimestamp: null

  labels:

    run: podtest

  name: podtest

spec:

  containers:

  - command:

    - sleep

    - "100000"

    image: busybox

    name: podtest

    volumeMounts:

    - name: configmap-as-file

      mountPath: /etc/config.properties

      subPath: config.properties

  volumes:

    - name: configmap-as-file

      configMap:

        name: configmap-test111

Note1: Trường hợp configmap là from-literal. Khi mount vào file yaml trên, sẽ có kết quả như sau

    volumeMounts:

    - name: configmap-as-file

      mountPath: /etc/tuanda

  volumes:

    - name: configmap-as-file

      configMap:

        name: configmap-test112

# k create configmap configmap-test112 --from-literal=key1=value1

> (pod)# cat /etc/tuanda/key1

value1

## Secret

<https://kubernetes.io/docs/concepts/configuration/secret/>

Ngoài generic, Secret hỗ trợ các loại:

|  |  |
| --- | --- |
| Opaque | arbitrary user-defined data |
| kubernetes.io/service-account-token | service account token |
| kubernetes.io/dockercfg | serialized ~/.dockercfg file |
| kubernetes.io/dockerconfigjson | serialized ~/.docker/config.json file |
| kubernetes.io/basic-auth | credentials for basic authentication |
| kubernetes.io/ssh-auth | credentials for SSH authentication |
| kubernetes.io/tls | data for a TLS client or server |
| bootstrap.kubernetes.io/token | bootstrap token data |

Secret được dùng cho file mout vào file trong pod, env cho pod.

Tạo secret bằng command line

# kubectl create secret generic prod-db-secret --from-literal=username=produser --from-literal=password=Y4nys7f11

VD về Opaque

**apiVersion**: v1

**kind**: Secret

**metadata**:

**name**: mysecret

**type**: Opaque

**data**:

**USER\_NAME**: YWRtaW4=

**PASSWORD**: MWYyZDFlMmU2N2Rm

### Secret as ENV

apiVersion: v1

kind: Pod

metadata:

  name: secret-env-pod

spec:

  containers:

  - name: mycontainer

    image: redis

    env:

      - name: SECRET\_USERNAME

        valueFrom:

          secretKeyRef:

            name: mysecret

            key: username

            optional: false

      - name: SECRET\_PASSWORD

        valueFrom:

          secretKeyRef:

            name: mysecret

            key: password

            optional: false

  restartPolicy: Never

### Secret as file in folder

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: mypod

**spec**:

**containers**:

- **name**: mypod

**image**: redis

**volumeMounts**:

- **name**: foo

**mountPath**: "/etc/foo"

**readOnly**: **true**

**volumes**:

- **name**: foo

**secret**:

**secretName**: mysecret

### Secret as configfile

## MultiContainerPod (QUEUE)

## InitContainer (QUEUE)

# Section 6: Cluster Maintain

## 1. OS upgrade (drain node)

Hướng: Tách từng node ra để upgrade, sau đó join lại bằng các lệnh drain, conrdon.

Drain: gỡ evic pod ra khỏi cụm node, chuyển pod sang node khác, đồng thời gán nhãn unschedule cordon cho node đang tác động. (**Chú ý:** nếu POD single không được quản lý bởi deployment/replicaset… thì khi drain sẽ bị xóa – **Cẩn thận )**

Uncordon: mark node chuyển thành Schedulable (cho pod deploy)

Cordon: mark node chuyển thành Un-Schedulable (chặn pod deploy)

Khác biệt giữa Cordon và draitn là: cordon không chuyển pod đang có trên node. Còn Drain thì có, sẽ chuyển pod sang node khác.

VD1:

# k drain node01 --ignore-daemonsets

# k get node

NAME STATUS ROLES AGE VERSION

controlplane Ready control-plane,master 16m v1.23.0

node01 Ready,SchedulingDisabled <none> 15m v1.23.0

## 2. Cluster upgrade



Phải đảm bảo version như ảnh trên. Ví dụ kube-apiserver là 1.10 thì kubelet bét nhất phải là >=1.8

Có thể dùng 2 lệnh này để kiểm tra

# kubeadm upgrade plan

# kubeadm upgrade apply

**Upgrade master-node**

# apt-get update

## 3. Backup / Restore ETCD

Khi backup ta cần thực hiện lại những bước sau:

* Backup resource: kubectl get all –A –o yaml > file.yaml
* Backup ETCD: như ở dưới
* Restore ETCD: như ở dưới

Toàn bộ quá trình backup / restore như sau:

Bước 1: Backup

root@controlplane:~# ETCDCTL\_API=3 etcdctl --endpoints=https://[127.0.0.1]:2379 \

--cacert=/etc/kubernetes/pki/etcd/ca.crt \

--cert=/etc/kubernetes/pki/etcd/server.crt \

--key=/etc/kubernetes/pki/etcd/server.key \

snapshot save /opt/snapshot-pre-boot.db

Bước 2: Restore

root@controlplane:~# ETCDCTL\_API=3 etcdctl --data-dir /var/lib/etcd-from-backup \

snapshot restore /opt/snapshot-pre-boot.db

Bước 3: đổi config etcd trỏ vào thư mục backup

# vim /etc/kubernetes/manifests/etcd.yaml

Tìm đến cuối cùng chỗ volume và sửa

volumes:

- hostPath:

path: /var/lib/etcd-from-backup

type: DirectoryOrCreate

name: etcd-data

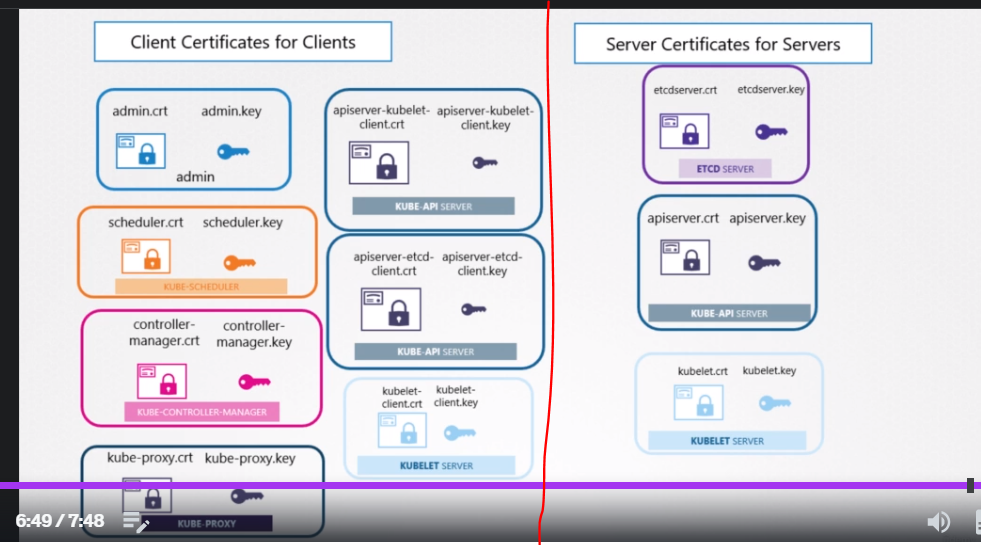
Bước 4 dự phòng

Mặc định khi đổi etcd sẽ tự reload lại, nếu ko ta có thể thêm bước xóa pod . và đợi 3-5p kiểm tra lại

# kubectl delete pod -n kube-system etcd-controlplane

# Section 7: Security

Các cặp key trong k8s



**Cách tạo key: /etc/kubernetes/pki/ca.crt vaf /etc/kubernetes/pki/ca.key**

# openssl genrsa -out ca.key 2048

# openssl req -new -key ca.key -subj "/CN=KUBERNETES-CA" -out ca.csr

# openssl x509 -req -in ca.csr –signkey ca.key -out ca.crt

Đây là file list API <https://github.com/mmumshad/kubernetes-the-hard-way/tree/master/tools>

## Api Groups

Link API dựa theo link này: <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.24/>

/metrics

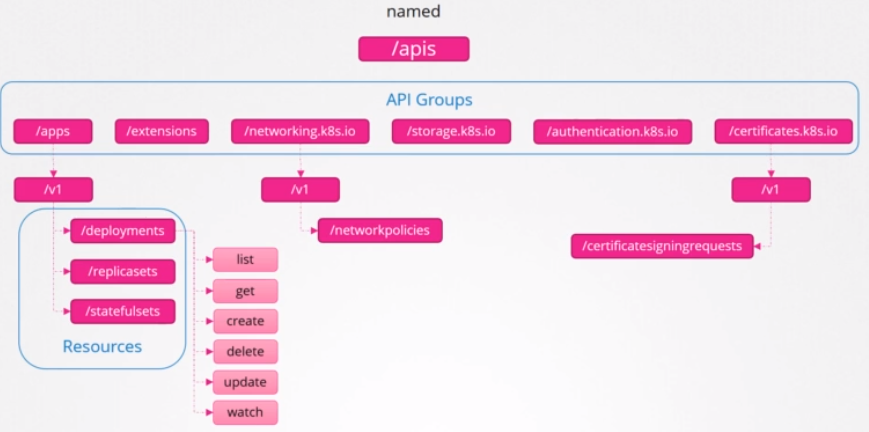
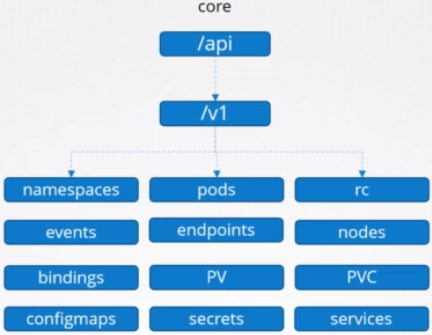
/healthz

/version

/api

/apis

/logs



## ABAC Authorization

Kubeconfig file

Trong kubeconfig file được chia làm 3 phần chính

* Clusters
* Contexts
* Users

**apiVersion**: v1

**kind**: Config

**proxy-url**: https://proxy.host:3128

**clusters**:

- **cluster**:

**name**: development

**users**:

- **name**: developer

**contexts**:

- **context**:

**name**: development

Ví dụ về config file nhiều user/clusster/context

apiVersion: v1

kind: Config

clusters:

- name: production

  cluster:

    certificate-authority: /etc/kubernetes/pki/ca.crt

    server: https://controlplane:6443

- name: development

  cluster:

    certificate-authority: /etc/kubernetes/pki/ca.crt

    server: https://controlplane:6443

- name: kubernetes-on-aws

  cluster:

    certificate-authority: /etc/kubernetes/pki/ca.crt

    server: https://controlplane:6443

- name: test-cluster-1

  cluster:

    certificate-authority: /etc/kubernetes/pki/ca.crt

    server: https://controlplane:6443

contexts:

- name: test-user@development

  context:

    cluster: development

    user: test-user

- name: aws-user@kubernetes-on-aws

  context:

    cluster: kubernetes-on-aws

    user: aws-user

- name: test-user@production

  context:

    cluster: production

    user: test-user

- name: research

  context:

    cluster: test-cluster-1

    user: dev-user

users:

- name: test-user

  user:

    client-certificate: /etc/kubernetes/pki/users/test-user/test-user.crt

    client-key: /etc/kubernetes/pki/users/test-user/test-user.key

- name: dev-user

  user:

    client-certificate: /etc/kubernetes/pki/users/dev-user/developer-user.crt

    client-key: /etc/kubernetes/pki/users/dev-user/dev-user.key

- name: aws-user

  user:

    client-certificate: /etc/kubernetes/pki/users/aws-user/aws-user.crt

    client-key: /etc/kubernetes/pki/users/aws-user/aws-user.key

current-context: test-user@development

preferences: {}

**Một số lệnh sử dụng:**

# kubectl config view

# kubectl config use-context prod-user@prodution

# kubectl config current-context --kubeconfig my-kube-config

**Chuyển current context từ file:**

# kubectl config --kubeconfig=/root/my-kube-config use-context research

# kubectl config --kubeconfig=/root/my-kube-config current-context (kiểm tra current context từ file)

Chạy lệnh với user đã có trong kubeconfig

# kubectl get pods --as dev-user

## 154 RBAC / Kubernetes API

Có các cách để giọ API. Ta chia 3 ví dụ dưới đây

* Gọi từ pod với role binding (RBAC)
* Gọi từ ~/.kube/config (xóa từ bản 1.24???)
* Public Port service “kubernetes” trong ns default

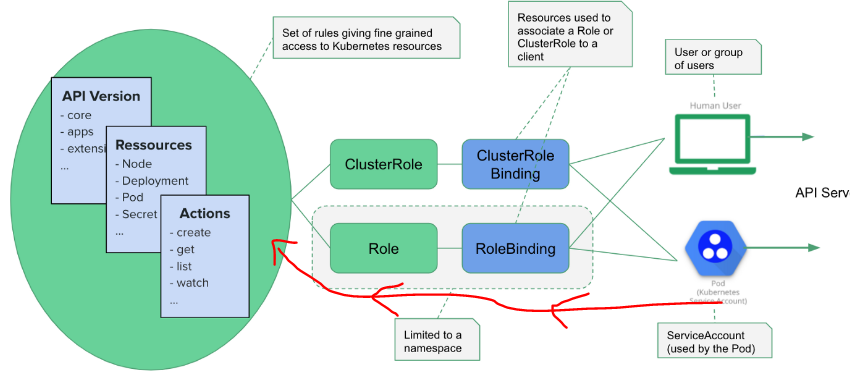
Rolebinding có thể gắn cho: User, Group , ServicesAccount

### Tạo Role/RoleBinding như sau:

kubectl create role developer --verb=list,create,delete --resource=pods --dry-run=client -o yaml > role.yaml

k create rolebinding dev-user-binding --role=developer --user=dev-user

### VD1: Role binding / service account – gọi trong pod



Mối liên hệ: [Pod < serviceaccount|secret < rolebinding < role]

# kubectl create **serviceaccount** demo-sa -o yaml --dry-run=client

# kubectl create **role** list-pods --verb=list --resource=pods --dry-run=client -o yaml

# kubectl create rolebinding list-pods\_demo-sa --role=list-pods --serviceaccount=default:demo-sa --dry-run=client -o yaml

# kubectl run debugpod --image=nicolaka/netshoot --dry-run=client -o yaml

Sửa pod gắn thêm service-account:

apiVersion: v1

kind: Pod

metadata:

  creationTimestamp: null

  labels:

    run: debugpod

  name: debugpod

spec:

  serviceAccountName: demo-sa

  containers:

  - image: nicolaka/netshoot

    name: debugpod

    command: ["/bin/bash"]

    args: ["-c", "sleep 100000"]

Ta vào trong POD, kiểm tra serviceaccout được mount vào file:

> cat /var/run/secrets/kubernetes.io/serviceaccount/token

> TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)

> curl -H "Authorization: Bearer $TOKEN" https://kubernetes/api/v1/ --insecure

### VD2: Role binding / service account – gọi từ ngoài

<https://nieldw.medium.com/curling-the-kubernetes-api-server-d7675cfc398c>

<https://iximiuz.com/en/posts/kubernetes-api-call-simple-http-client/>

kubectl create serviceaccount api-explorer --dry-run=client -o yaml

kubectl create clusterrole log-reader --verb=get,list,watch --resource=pods,pods/log --dry-run=client -o yaml

kubectl create rolebinding api-explorer:log-reader --clusterrole=log-reader --serviceaccount=default:api-explorer

SERVICE\_ACCOUNT=api-explorer

SECRET=$(kubectl get serviceaccount ${SERVICE\_ACCOUNT} -o json | jq -Mr '.secrets[].name | select(contains("token"))')

TOKEN=$(kubectl get secret ${SECRET} -o json | jq -Mr '.data.token' | base64 -d)

kubectl get secret ${SECRET} -o json | jq -Mr '.data["ca.crt"]' | base64 -d > /tmp/ca.crt

APISERVER=https://$(kubectl -n default get endpoints kubernetes --no-headers | awk '{ print $2 }')

curl -s $APISERVER/openapi/v2 -H "Authorization: Bearer $TOKEN" --cacert /tmp/ca.crt | jq .

curl -s $APISERVER/api/v1/namespaces/default/pods/ -H "Authorization: Bearer $TOKEN" --cacert /tmp/ca.crt

( Note: để lấy toàn bộ thông tin k8s, ta có thể lấy ca.crt và token của kube-system: k get serviceaccounts -A | grep default | grep kube-system)

### VD3: Kube Config

<https://vocon-it.com/2019/07/24/certified-kubernetes-administrator-labs-6-kubernetes-api/>

Thực hiện cat kube/config để lấy ra 3 token đã mã hóa

CLIENT\_CERT=$(grep client-cert ~/.kube/config | awk '{print $2}')

CLIENT\_KEY\_DATA=$(grep client-key-data ~/.kube/config | awk '{print $2}')

CERTIFICATE\_AUTHORITY\_DATA=$(grep certificate-authority-data ~/.kube/config | awk '{print $2}')

echo $CLIENT\_CERT | base64 -d > client-cert.pem

echo $CLIENT\_KEY\_DATA | base64 -d > client-key-data.pem

echo $CERTIFICATE\_AUTHORITY\_DATA | base64 -d > certificate-authority-data.pem

for item in client-cert client-key-data certificate-authority-data;

do

  echo $item;

  grep $item ~/.kube/config | awk '{print $2}' | base64 -d > $item.pem;

done

curl -k -s --cert ./client-cert.pem \

           --key ./client-key-data.pem \

           --cacert ./certificate-authority-data.pem https://127.0.0.1:6443/api/v1 | grep \"name\"

## 159: Image security: with docker private registry

<https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/>

Tạo seret registry:

# kubectl create secret docker-registry private-reg-cred --docker-username=tuanda --docker-password=123 --docker-server=registry.tuan.name.vn:31320 --docker-email=ahihi@registry.tuan.name.vn

Tạo pod sử dụng secret trên

apiVersion: apps/v1

kind: Deployment

metadata:

  name: hello-kubernetes

  namespace: tuanda

spec:

  replicas: 1

  selector:

    matchLabels:

      app: hello-kubernetes

  template:

    metadata:

      labels:

        app: hello-kubernetes

    spec:

      imagePullSecrets:

      - name: private-reg-cred

      containers:

      - name: hello-kubernetes-nginx

        image: registry.tuan.name.vn:31320/nginx:alpine

        ports:

        - containerPort: 80

## 162 Security Context

Security Context as POD

<https://kubernetes.io/docs/tasks/configure-pod-container/security-context/>

Security Context as Deployment

apiVersion: apps/v1

kind: Deployment

metadata:

  creationTimestamp: null

  labels:

    app: ahihi

  name: ahihi

spec:

  replicas: 3

  selector:

    matchLabels:

      app: ahihi

  strategy: {}

  template:

    metadata:

      creationTimestamp: null

      labels:

        app: ahihi

    spec:

      containers:

      - command:

        - sleep

        - "1000"

        image: busybox

        name: busybox

        securityContext:

          runAsUser: 1000

          runAsGroup: 3000

## 165 Network Policies

# Section 8: Storage

## VOLUME

Các loại Volume : <https://kubernetes.io/docs/concepts/storage/volumes/>

### emptyDir

[tuanda@localhost Chapter06]$ cat fortune-pod.yaml

apiVersion: v1

kind: Pod

metadata:

  name: fortune

spec:

  containers:

  - image: luksa/fortune

    name: html-generator

    volumeMounts:

    - name: html

      mountPath: /var/htdocs

  - image: nginx:alpine

    name: web-server

    volumeMounts:

    - name: html

      mountPath: /usr/share/nginx/html

      readOnly: true

    ports:

    - containerPort: 80

      protocol: TCP

  volumes:

  - name: html

    emptyDir: {}

[tuanda@localhost Chapter06]$ kubectl port-forward fortune 8080:80

[tuanda@localhost Chapter06]$ curl localhost:8080

### hostPath

<https://kubernetes.io/docs/concepts/storage/volumes/#hostpath>

VD1

Lưu vào phân vùng của node/minikube

[tuanda@localhost Chapter06]$ cat mongodb-pod-hostpath.yaml

apiVersion: v1

kind: Pod

metadata:

name: mongodb

spec:

containers:

- image: mongo

name: mongodb

volumeMounts:

- name: mongodb-data

mountPath: /data/db

ports:

- containerPort: 27017

protocol: TCP

volumes:

- name: mongodb-data

hostPath:

path: /tmp/mongodb

### nfs

## Storage Class / PV / PVC

### PV

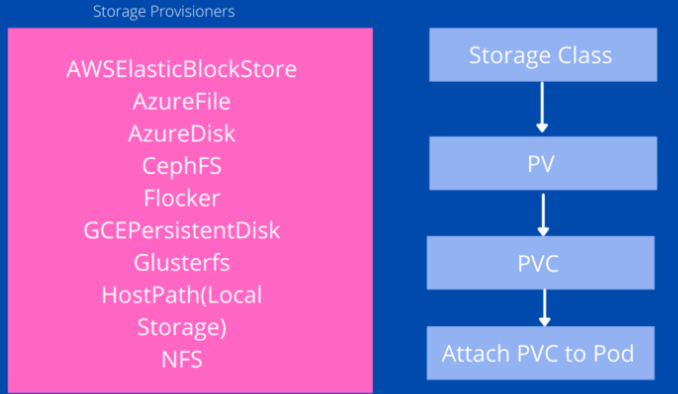
### PVC

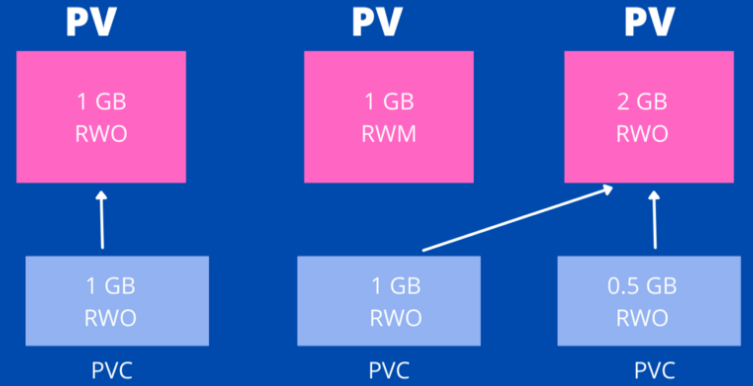
### StorageClass

<https://kubernetes.io/docs/concepts/storage/storage-classes/>

<https://medium.com/codex/kubernetes-persistent-volume-explained-fb27df29c393>

Kiến trúc:





Các loại storage có thể lên trang chủ để tìm.

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

  name: local-storage

provisioner: kubernetes.io/no-provisioner

volumeBindingMode: WaitForFirstConsumer

allowVolumeExpansion: true

reclaimPolicy: Delete

### VD1: Storage Class as NFS a Trung

<https://github.com/nguyenan122/k8s-collection/tree/main/sample_final/001.nfs-provisioning>

## Recycling PersistentVolume

Có 3 loại:

* Retain: khi xóa PVC thì PV vẫn còn- dữ liệu trong PV không bị xóa.
* Recycle: khi xóa PVC thì PV vẫn còn, nhưng dữ liệu trong PV sẽ được xóa đi để tái sử dụng
* Delete: khi xóa PVC thì PV sẽ bị xóa luôn.

Subpath khi sử dụng chung 1 PVC khá hay <https://kubernetes.io/docs/concepts/storage/volumes/#using-subpath>

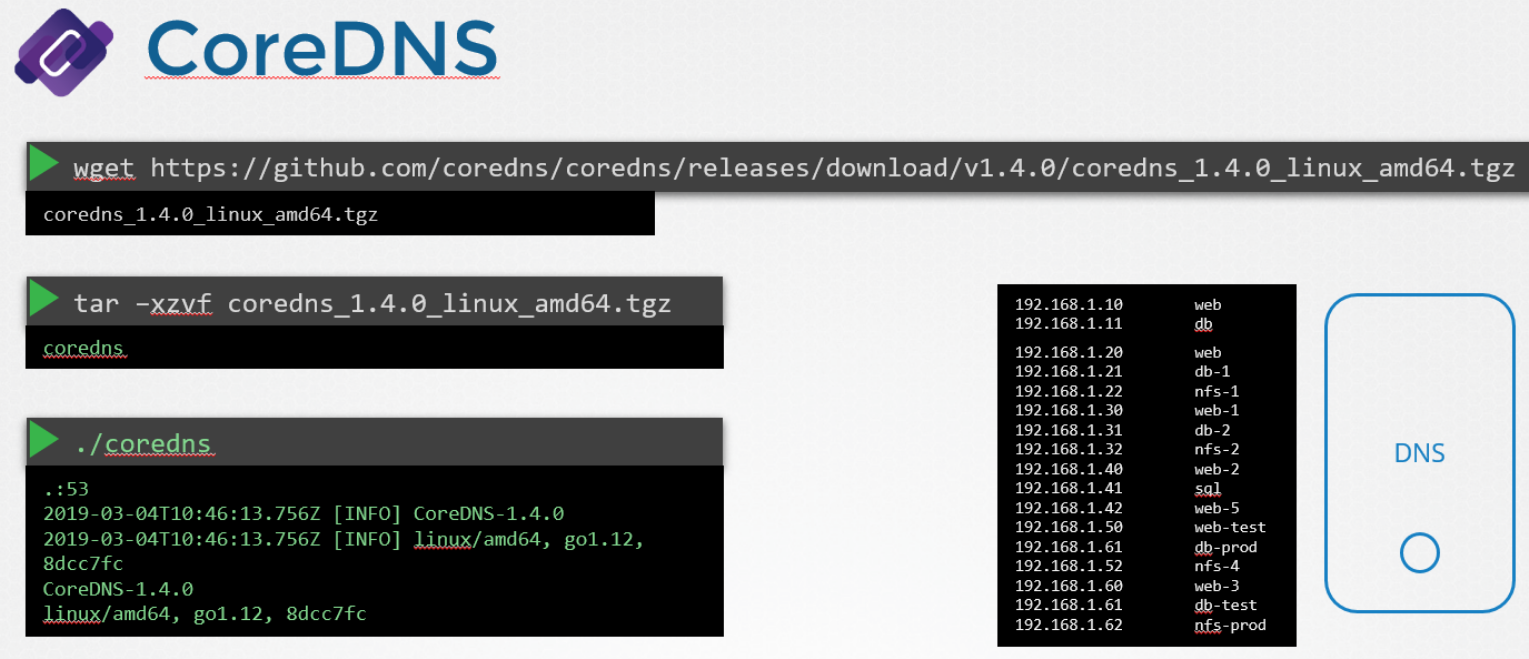
# Section 9: Networking

# ip link

# echo 1 /proc/sys/net/ipv4/ip\_forward

## 187 CoreDNS

Các bản ghi <https://github.com/kubernetes/dns/blob/master/docs/specification.md> và <https://coredns.io/plugins/kubernetes/>



Sau khi cài đặt, ta setting dns ở /etc/hosts, sau đó ta import vào coredns như sau



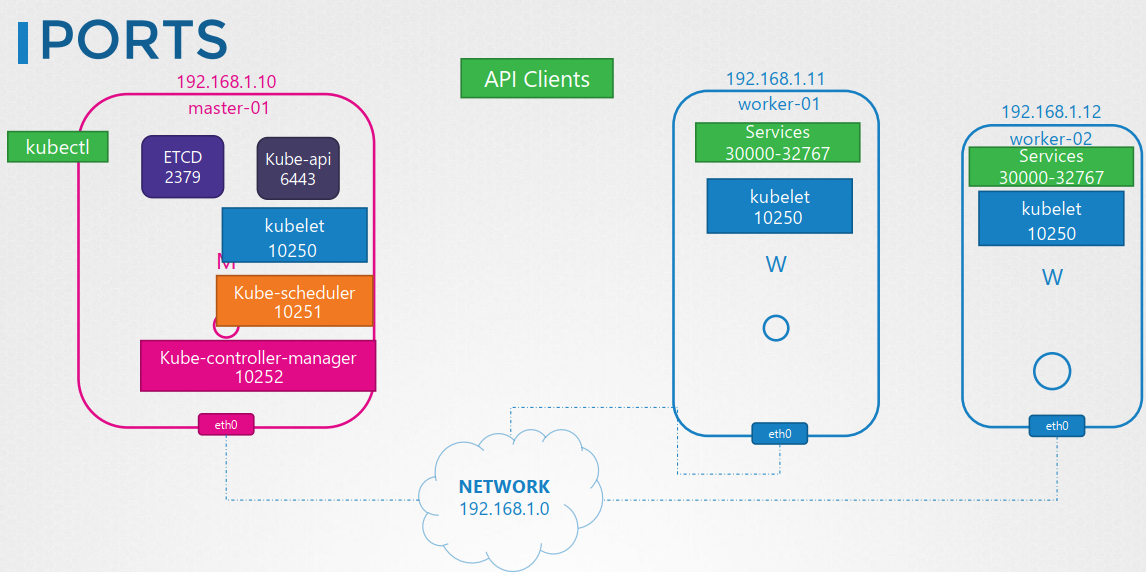
## Network namespace

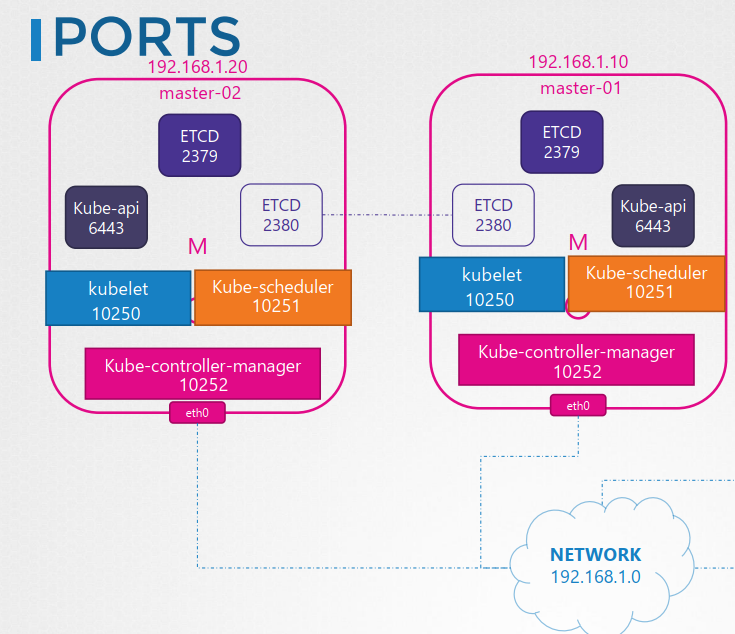
# ip netns add red

# ip netns add blue

# ip netns

## Cluster Networking in k8s





## 196. Pod networking

## 200. Weaver Network

## 207 Services Networking

## 217 Ingress

# Section 10: Install and design kubernetes cluster

# Section 11: Install kubernetes the kubeadm way

# TIP-TRICK

### Install telnet in docker apk

$ apk update

$ apk add busybox-extras

$ busybox-extras telnet localhost 6900

# Sample Prod

## Kube Dashboard

Hướng dẫn: <https://www.replex.io/blog/how-to-install-access-and-add-heapster-metrics-to-the-kubernetes-dashboard>

curl -O [https://raw.githubusercontent.com/kubernetes/dashboard/v2.4.0/aio/deploy/recommended.yaml](https://raw.githubusercontent.com/kubernetes/dashboard/v2.4.0/aio/deploy/recommended.yaml%20-o%20kubernetes-dashboard.yaml)

Ta sửa lại Services về NodePort

kubectl apply -f recommended.yaml

truy cập vào https://IP:Nodeport

Tạo Token:

kubectl create serviceaccount dashboard-admin-sa

kubectl create clusterrolebinding dashboard-admin-sa --clusterrole=cluster-admin --serviceaccount=default:dashboard-admin-sa

kubectl get secrets | grep dashboard-admin-sa-token

k get secrets dashboard-admin-sa-toke.... –o yaml | grep token

Ta lấy token từ secret, sau đó **base64** decode là có token.

## K8s Registry:

Bước 1. Chỉ định hosts:

echo 192.168.88.12 registry.tuanda.vn >> /etc/hosts

Bước 2: Import basic-auth và ssl vào configmap

# mkdir /opt/certs /opt/registry

# cd /opt

# openssl req -x509 -out ca.crt -keyout ca.key -days 1825 \

-newkey rsa:2048 -nodes -sha256 \

-subj '/CN=registry.tuanda.vn' -extensions EXT -config <( \

printf "[dn]\nCN=registry.tuanda.vn\n[req]\ndistinguished\_name = dn\n[EXT]\nsubjectAltName=DNS:registry.tuanda.vn\nkeyUsage=digitalSignature\nextendedKeyUsage=serverAuth")

# cd /opt/certs/

# kubectl create configmap registry-cert --from-file=ca.crt --from-file=ca.key

# yum install httpd-tools -y ; htpasswd -Bbn tuanda 123 > htpasswd

# kubectl create configmap registry-basic-auth --from-file=htpasswd

# kubectl get configmaps

Bước 4: Tạo deployment và service NodePort

apiVersion: apps/v1

kind: Deployment

metadata:

name: private-repository-k8s

labels:

app: private-repository-k8s

spec:

replicas: 1

selector:

matchLabels:

app: private-repository-k8s

template:

metadata:

labels:

app: private-repository-k8s

spec:

volumes:

- name: certs-vol

configMap:

name: registry-cert

- name: auth-vol

configMap:

name: registry-basic-auth

- name: registry-vol

hostPath:

path: /opt/registry

type: Directory

containers:

- image: registry:2

name: private-repository-k8s

imagePullPolicy: IfNotPresent

env:

- name: REGISTRY\_AUTH

value: htpasswd

- name: REGISTRY\_AUTH\_HTPASSWD\_PATH

value: "/auth/htpasswd"

- name: REGISTRY\_AUTH\_HTPASSWD\_REALM

value: Registry Realm

- name: REGISTRY\_HTTP\_TLS\_CERTIFICATE

value: "/certs/ca.crt"

- name: REGISTRY\_HTTP\_TLS\_KEY

value: "/certs/ca.key"

ports:

- containerPort: 5000

volumeMounts:

- name: certs-vol

mountPath: /certs

- name: registry-vol

mountPath: /var/lib/registry

- name: auth-vol

mountPath: /auth

---

apiVersion: v1

kind: Service

metadata:

labels:

app: private-repository-k8s

name: private-repository-k8s

spec:

ports:

- port: 5000

nodePort: 31320

protocol: TCP

targetPort: 5000

selector:

app: private-repository-k8s

type: NodePort

Bước 4: Trust CA

sudo cp -rp /opt/certs/ca.crt /etc/pki/ca-trust/source/anchors/

sudo update-ca-trust

sudo service docker restart

Bước 5: Đẩy cert vào tất cả các node docker, để permit self-certificate gọi pull. (all node)

mkdir -p /etc/docker/certs.d/registry.tuanda.vn:31320

cp -rp /opt/certs/ca.crt /etc/docker/certs.d/registry.tuanda.vn\:31320/

Bước 6: docker login đẩy config registry client sang các node:

# curl -v --user tuanda:123 https://registry.tuanda.vn:31320/v2/

# docker login registry.tuanda.vn:31320 -u tuanda -p 123

cat ~/.docker/config.json

{

"auths": {

"registry.tuanda.vn:31320": {

"auth": "dHVhbmRhOjEyMw=="

}

}

}

mkdir -p /home/tuanda/.docker ; chown -R tuanda.tuanda /home/tuanda/.docker

Ta copy file config.json ở trên sang các worker node trong cluster. (/home/tuanda/.docker/config.json)

Bước 6: đẩy image lên registry:

# docker pull nginx:alpine

# docker tag nginx:alpine registry.tuanda.vn:31320/nginx:alpine

# docker push registry.tuanda.vn:31320/nginx:alpine

Bước 7: Launch pod với option registry

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello-kubernetes

namespace: tuanda

spec:

replicas: 1

selector:

matchLabels:

app: hello-kubernetes

template:

metadata:

labels:

app: hello-kubernetes

spec:

containers:

- name: hello-kubernetes-debug

image: admin.tuan.name.vn:31320/debug-tools:1.0.0

ports:

- containerPort: 8080

- name: hello-kubernetes-nginx

image: admin.tuan.name.vn:31320/nginx:alpine

ports:

- containerPort: 80

imagePullSecrets:

- name: regcred

## Ingress nginx:

<https://kubernetes.github.io/ingress-nginx/deploy/>

curl -O <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.1/deploy/static/provider/cloud/deploy.yaml>

# kubectl apply -f deploy.yaml

kubectl create deployment demo --image=httpd --port=80

kubectl expose deployment demo

kubectl create ingress demo-localhost --class=nginx --rule=demo.localdev.me/\*=demo:80

[tuanda@master-node ~]$ k get ingress -A

NAMESPACE NAME CLASS HOSTS ADDRESS PORTS AGE

default demo-localhost nginx demo.localdev.me 80 10m

[tuanda@master-node ~]$ k get ingress demo-localhost -o yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: demo-localhost

namespace: default

spec:

ingressClassName: nginx

rules:

- host: demo.localdev.me

http:

paths:

- backend:

service:

name: demo

port:

number: 80

path: /

pathType: Prefix

status:

loadBalancer: {}

## Nfs provisioning

<https://fabianlee.org/2022/01/12/kubernetes-nfs-mount-using-dynamic-volume-and-storage-class/?msclkid=a5ca54e2ae7111eca259ced5d2223a6f>

<https://artifacthub.io/packages/helm/nfs-subdir-external-provisioner/nfs-subdir-external-provisioner>

B1: Cài Nfs server:

yum install nfs-utils nfs-utils-lib -y

chkconfig rpcbind on

chkconfig nfs on

service rpcbind restart

service nfs restart

mkdir -p /data/nfs-k8s/ ; vim /etc/exports

/data/nfs-k8s/ 192.168.88.0/24(rw,sync,subtree\_check,no\_root\_squash)

exportfs -a

showmount -e 127.0.0.1

B2: Cài nfs client trên mỗi worker node, nếu không sẽ lỗi không mout đc vào pod.

yum install nfs-utils nfs-utils-lib -y

chkconfig nfs off

chkconfig rpcbind off

B3: Helm install

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

helm pull nfs-subdir-external-provisioner/nfs-subdir-external-provisioner

helm template nfs-subdir-external-provisioner . --set nfs.server=192.168.88.12 \

  --set nfs.path=/data/nfs-k8s/ \

  --set storageClass.name=nfs-provisioner \

  --set storageClass.onDelete=retain \

  --set storageClass.accessModes=ReadWriteMany

B4: Test

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: sc-nfs-pvc

spec:

  accessModes:

    - ReadWriteMany

  storageClassName: nfs-provisioner

  resources:

    requests:

      storage: 2Gi

apiVersion: v1

kind: Pod

metadata:

  name: busybox

spec:

  volumes:

  - name: myvol

    persistentVolumeClaim:

      claimName: sc-nfs-pvc

  containers:

  - image: busybox

    name: busybox

    command: ["/bin/sh"]

    args: ["-c", "sleep 600000"]

    volumeMounts:

    - name: myvol

      mountPath: /data

## Debug image

<https://cloudogu.com/en/blog/k8s-app-ops-part-2>

apiVersion: v1

kind: Pod

metadata:

  creationTimestamp: null

  labels:

    run: debugpod

  name: debugpod

spec:

  containers:

  - image: nicolaka/netshoot

    name: debugpod

    command: ["/bin/bash"]

    args: ["-c", "sleep 100000"]

hoặc

**Dockerfile.app**

FROM alpine:latestRUN apk update && \  
 apk --no-cache add \  
 bash \  
 curl

**Dockerfile.debug**

FROM alpine:latestRUN apk update && \  
 apk --no-cache add \  
 bash \  
 tcpdumpCMD exec /bin/bash -c "trap : TERM INT; sleep infinity & wait"

and build them:

> docker build --no-cache --progress=plain -f docker\Dockerfile.app -t do-wget:1.0.0 docker\

> docker build --no-cache --progress=plain -f docker\Dockerfile.debug -t debug-tools:1.0.0 docker\