Contents

[Cài đặt: 5](#_Toc104625117)

[kubectl 5](#_Toc104625118)

[kubecolor (đã có ansible) 6](#_Toc104625119)

[bash-completion 6](#_Toc104625120)

[minikube 6](#_Toc104625121)

[Helm(đã có ansible) 6](#_Toc104625122)

[Cài đặt từ Repo 6](#_Toc104625123)

[Tool hay cho k8s: 11](#_Toc104625124)

[Kompose = Convert docker-compose to K8s 11](#_Toc104625125)

[Kubecolor 11](#_Toc104625126)

[Indent-rainbow 11](#_Toc104625127)

[THUẬT NGỮ: 11](#_Toc104625128)

[File YAML Mô tả cấu trúc 12](#_Toc104625129)

[Debug lỗi: 14](#_Toc104625130)

[POD 14](#_Toc104625131)

[GET pod 14](#_Toc104625132)

[Run Pod 15](#_Toc104625133)

[APPLY pod 15](#_Toc104625134)

[EXEC Để chạy command trong pod 15](#_Toc104625135)

[EXEC -it Truy cập vào Pod: 16](#_Toc104625136)

[Port-Froward 16](#_Toc104625137)

[DELETE pod 16](#_Toc104625138)

[LOG pod 16](#_Toc104625139)

[Describe pod 16](#_Toc104625140)

[Job and CronJob 16](#_Toc104625141)

[LABEL 16](#_Toc104625142)

[NameSpace 17](#_Toc104625143)

[Tạo namespace 17](#_Toc104625144)

[Services 17](#_Toc104625145)

[VD1: ClusterIP: 17](#_Toc104625146)

[NodePort 18](#_Toc104625147)

[LoadBalancer 19](#_Toc104625148)

[Ingress: 19](#_Toc104625149)

[Services VD2 20](#_Toc104625150)

[Describe service 22](#_Toc104625151)

[Replicaset (rs) 23](#_Toc104625152)

[DELETE Replicaset 25](#_Toc104625153)

[DaemonSET 25](#_Toc104625154)

[JOB 25](#_Toc104625155)

[NETWORKING 26](#_Toc104625156)

[VOLUME 26](#_Toc104625157)

[emptyDir 26](#_Toc104625158)

[hostPath 27](#_Toc104625159)

[Storage Class / PV / PVC 28](#_Toc104625160)

[Storage 28](#_Toc104625161)

[VD1: Storage Class as NFS a Trung 30](#_Toc104625162)

[Recycling PersistentVolume 30](#_Toc104625163)

[Resource Limit 31](#_Toc104625164)

[RAM: 31](#_Toc104625165)

[CPU: 32](#_Toc104625166)

[LimitRange namespace 33](#_Toc104625167)

[Readiness & Liveness 33](#_Toc104625168)

[CONFIG MAP – SECRET 34](#_Toc104625169)

[ConfigMap 34](#_Toc104625170)

[ENV alone 34](#_Toc104625171)

[Create config map 35](#_Toc104625172)

[Configmap as ENV 35](#_Toc104625173)

[Configmap as Volume 36](#_Toc104625174)

[Secret 36](#_Toc104625175)

[Secret as ENV 37](#_Toc104625176)

[Secret as file in folder 38](#_Toc104625177)

[Secret as configfile 38](#_Toc104625178)

[DEPLOYMENT 39](#_Toc104625179)

[VD\_1 về đổi image trong deployment 41](#_Toc104625180)

[VD2 Zero downtime deployment: 41](#_Toc104625181)

[Statefull Set 42](#_Toc104625182)

[RBAC / Kubernetes API 44](#_Toc104625183)

[VD1: Role binding / service account – gọi trong pod 45](#_Toc104625184)

[VD2: Role binding / service account – gọi từ ngoài 46](#_Toc104625185)

[VD3: Kube Config 46](#_Toc104625186)

[Taints and tolerations 47](#_Toc104625187)

[Taints 47](#_Toc104625188)

[podAntiAffinity 47](#_Toc104625189)

[TIP-TRICK 48](#_Toc104625190)

[Install telnet in docker apk 48](#_Toc104625191)

[Sample Prod 48](#_Toc104625192)

[Kube Dashboard 48](#_Toc104625193)

[K8s Registry: 49](#_Toc104625194)

[Ingress nginx: 53](#_Toc104625195)

[Nfs provisioning 54](#_Toc104625196)

[Debug image 55](#_Toc104625197)

[**ARGO-CD -- ARGO-CD -- ARGO-CD-- ARGO-CD -- ARGO-CD --** 56](#_Toc104625198)

[CÀI ĐẶT 56](#_Toc104625199)

[3. Hello World Workflow 57](#_Toc104625200)

[TEMPLATE TRONG ARGO 57](#_Toc104625201)

[Container template 57](#_Toc104625202)

[Script Template 58](#_Toc104625203)

[Resource Template 59](#_Toc104625204)

[Suspend 60](#_Toc104625205)

[Template INVOCATORS 60](#_Toc104625206)

[STEP 60](#_Toc104625207)

[Serial Step (Step nối tiếp nhau) 60](#_Toc104625208)

[Step Parabel (Step song song) 62](#_Toc104625209)

[Suspend Step Template 63](#_Toc104625210)

[DAG 65](#_Toc104625211)

[Bài tập 1 về Step và Dag 67](#_Toc104625212)

[WORKFLOW FUNCTION ARGO 69](#_Toc104625213)

[1.MinIO 69](#_Toc104625214)

[2.Cài đặt Argo-Cli 70](#_Toc104625215)

[3.InputParameter: 70](#_Toc104625216)

[4.Scripts Result 74](#_Toc104625217)

[5.Output parameter 77](#_Toc104625218)

[6.Output Parameter File 80](#_Toc104625219)

[7. Artifact 82](#_Toc104625220)

[8. Secrets as environment variables 85](#_Toc104625221)

[9. Secrets as mounted volumes 85](#_Toc104625222)

[10. Loops 85](#_Toc104625223)

[11. Loops with sets 87](#_Toc104625224)

[12. Loops with sets as input parameters 89](#_Toc104625225)

[13. Dynamic Loops 92](#_Toc104625226)

[14. Conditionals 94](#_Toc104625227)

[15. Depends 96](#_Toc104625228)

[16. Depends theorie 97](#_Toc104625229)

[17. Retry strategy 97](#_Toc104625230)

[18. Recursion 97](#_Toc104625231)

[19. Exercise 2 - task introduction 97](#_Toc104625232)

[20. Exercise 2 - solution 97](#_Toc104625233)

---

# 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

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

### kubecolor (đã có ansible)

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

### minikube

(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

Kiểm tra minikube và clusster

#kubectl cluster-info

#

### 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

## 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

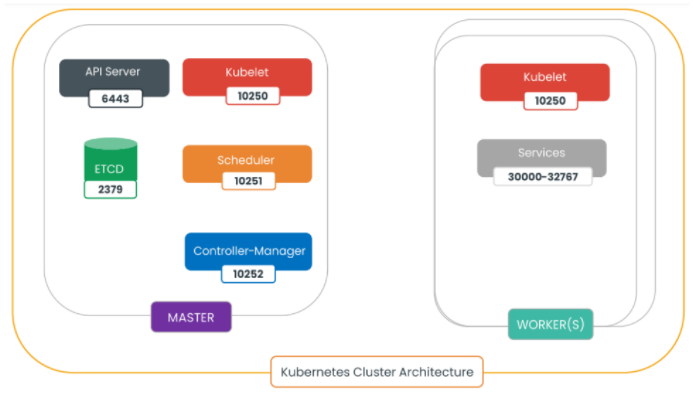
Mở port worker node

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 to K8s

<https://kompose.io/> (cài đặt)

$ kompose convert -f docker-compose.yaml

## Kubecolor

<https://github.com/hidetatz/kubecolor/releases>

## Indent-rainbow

<https://marketplace.visualstudio.com/items?itemName=oderwat.indent-rainbow>

# THUẬT NGỮ:

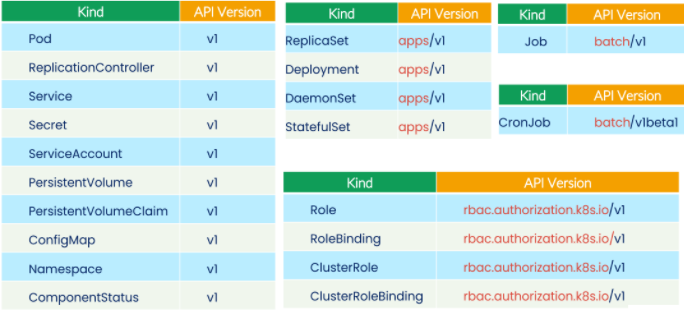
Replicaset: tạo ra multi-pod chạy cùng trên 1 image

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 kubia-manual -o yaml (hoặc json)

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

### Run Pod

# kubectl run kubia --image=luksa/kubia --port=8080

# 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 Để chạy command trong pod

kubectl exec [POD] -- [COMMAND]

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

bin

dev

etc

### EXEC -it Truy cập vào Pod:

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

/ # whoami

root

### Port-Froward

tuanda@localhost Chapter03]$ **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 kubia-manual

### Describe pod

# kubectl describe pod webapp

# kubectl describe pod nginx

## 1.2 Job and CronJob

<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

Có thể tạo từ yaml hoặc command:

# kubectl create namespace tuanda -o yaml --dry-run=client

## 1.5 Services

### 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

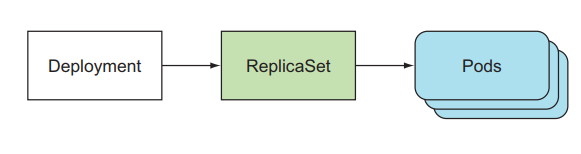
## 1.6 Ingress:

<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:

## 1.8 Rolling update and rollback

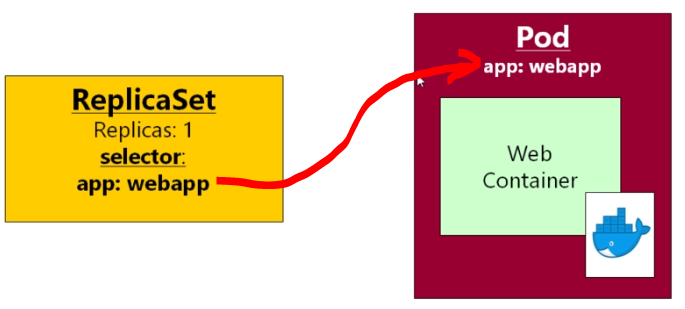
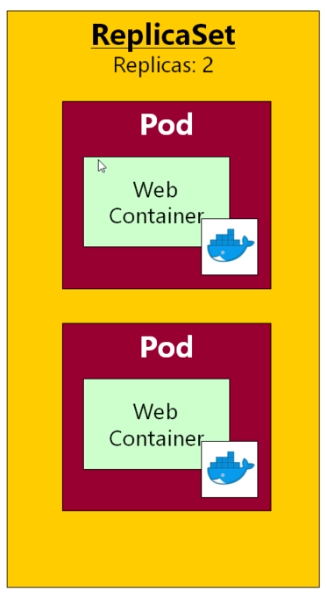
Để biết tỉ lệ pod rollingupdate là bao nhiêu, ta có thể describe pod, nhìn mục StrategyType và RollingUpdateStrategy = 25%

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:

# Replicaset (rs)

Ta thực hiện chỉnh sửa file pods.yaml, còn services.yaml vẫn giữ nguyên

 >>>>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

### DELETE Replicaset

# kubectl delete rs webapp

# DaemonSET

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

# Statefull Set

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

## 3.2 nodeSelector

## 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

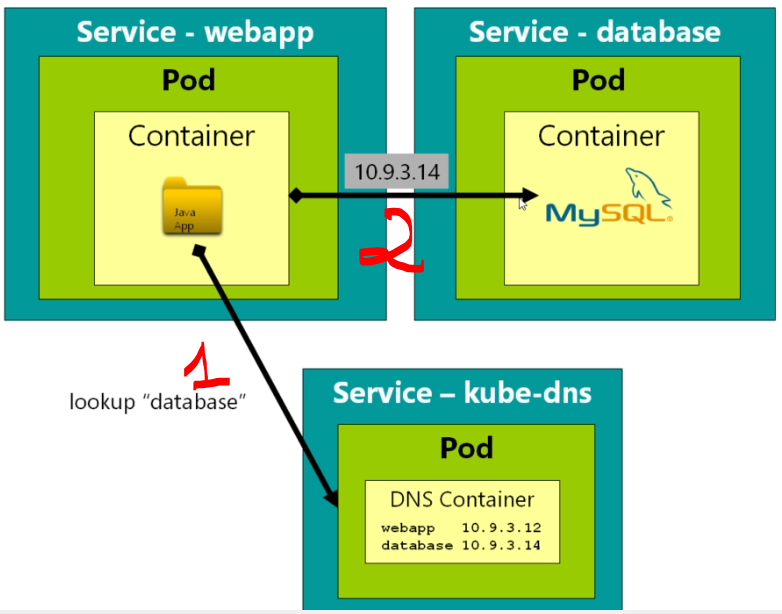
## 3.4 nodeAffinity

## 3.5 podAntiAffinity

# Command and Argument

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

# 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

# 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

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

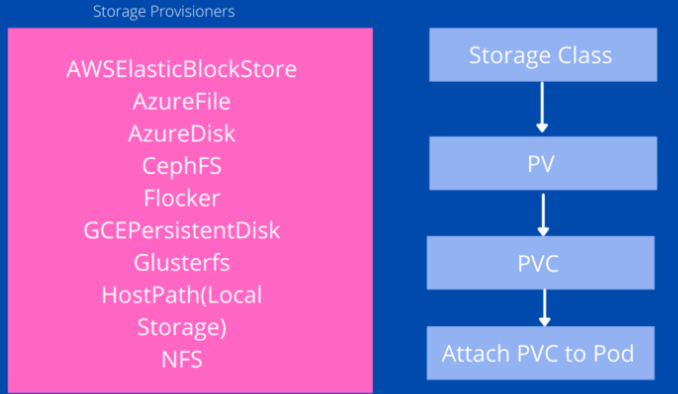
# Storage Class / PV / PVC

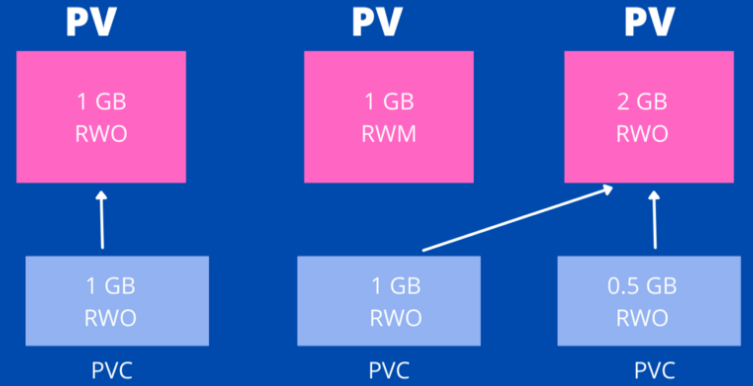
## Storage

<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>

# Resource Limit

<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

# Readiness & Liveness

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

# 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: k8s.gcr.io/busybox

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

      volumeMounts:

      - name: config-volume

        mountPath: /etc/config

  volumes:

    - name: config-volume

      configMap:

        name: special-config

  restartPolicy: Never

## 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

**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

**items**:

- **key**: username

**path**: my-group/my-username

# Section 6: Cluster Maintain

## 1. OS upgrade

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

Uncordon: mark node chuyển thành Schedulable

Cordon: mark node chuyển thành Un-Schedulable

## 2. Cluster upgrade

## 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

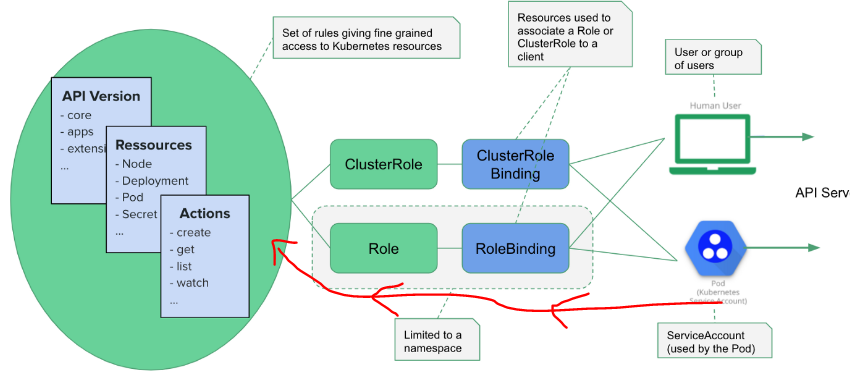
# kubectl delete pod -n kube-system etcd-controlplane

# 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
* Public Port service “kubernetes” trong ns default

## 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\"

# 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\