pods像一个k8s虚拟出来的小型虚拟机

node像是 k8s本身需要用到的物理机

用标签识别pod，举例：创建4个nginx，4个nginx的标签都是一样的。

查看仓库

yum repolist

确保所有节点值为1

cat /proc/sys/net/bridge/bridge-nf-call-ip6tables

kubeadm init --kubernetes-version=v1.14.1 --pod-network-cidr=10.244.0.0/16

10.244.0.0/16 是flannel网络

允许使用swap

vim /etc/sysconfig/kubelet

KUBELET\_EXTRA\_ARGS=”—fail-swap-on=false”

kubeadm init --apiserver-advertise-address=192.168.1.201 --kubernetes-version=v1.14.1 --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=Swap

执行一些初始化命令

mkdir –p $HOME/.kube

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

安装flannel

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

获取组件状态

kubectl get cs

部署容器

kubectl run nginx-deploy --image=nginx:1.14-alpine --port=80 --replicas=1

通过service暴露端口

kubectl expose deployment nginx-deploy --name=nginx-service --port=80 --target-port=80 --protocol=TCP

kubectl get service

nginx-service ClusterIP 10.102.223.254 <none> 80/TCP 104s

在集群中就可以通过 curl 10.102.223.254来访问

kubectl get pods -n kube-system -o wide

coredns-fb8b8dccf-gz6gs 1/1 Running 0 4h24m 10.244.0.3 k8s001 <none>

coredns-fb8b8dccf-jmzxj 1/1 Running 0 4h24m 10.244.0.2 k8s001 <none>

dns服务在pod网络中

kube-system下的服务

kubectl get svc -n kube-system

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

交互式进入busybox客户端

kubectl run client --image=busybox --replicas=1 -it --restart=Never

cat /etc/resolv.conf

nameserver 10.96.0.10

pod中的DNS就是kube-system命名空间下的kube-dns服务

master上

yum install -y bind-utils

让kube-system命名空间下的kube-dns服务来解析nginx-service.default.svc.cluster.local

dig -t A nginx-service.default.svc.cluster.local @10.96.0.10

解析出来的结果如下，域名nginx-service.default.svc.cluster.local对应pod的ip：10.102.223.254

nginx-service.default.svc.cluster.local. 5 IN A 10.102.223.254

**在busybox中执行wget -O - -q http://nginx-service 能访问到pod中的nginx，大致过程如下：**

在busybox容器中发送请求http://nginx-service，此请求需要DNS解析，busybox的DNS服务器是（kube-dns）10.96.0.10。kube-dns解析出来nginx-service.default.svc.cluster.local是ip是10.102.223.254，10.102.223.254就是nginx-service的ip。nginx-service的80端口已经映射到nginx-deploy-55d8d67cf-zt52t的80端口

展示pod的label

kubectl get pods --show-labels

nginx-deploy-55d8d67cf-zt52t 1/1 Running 0 3h18m pod-template-hash=55d8d67cf,run=nginx-deploy

kubectl describe svc nginx-service

Name: nginx-service

Namespace: default

Labels: run=nginx-deploy

Annotations: <none>

Selector: run=nginx-deploy

Type: ClusterIP

IP: 10.102.223.254

Port: <unset> 80/TCP

TargetPort: 80/TCP

Endpoints: 10.244.1.2:80

Session Affinity: None

Events: <none>

假设删除Endpoints: 10.244.1.2:80， k8s会重新再起一个新pod，新pod的ip会变化，但是label仍然是run=nginx-deploy，nginx-service会通过label选择器选中新pod。然后修改Endpoints:为新pod的ip

service给pod提供固定访问端点

kubectl run myapp --image=ikubernetes/myapp:v1 --replicas=2

暴露端口

kubectl expose deployment myapp --name=myapp-service --port=80

kubectl get svc

获取到的hostname会变化，从结果来看，是随机调度算法

curl 10.99.67.114/hostname.html

kubectl scale --replicas=5 deployment myapp

通过set image来改变deployment的版本，k8s一个个pod更新版本（灰度升级）

kubectl set image deployment myapp myapp=ikubernetes/myapp:v2

busybox中执行，查看版本

while true;do wget -O - -q myapp-service; sleep 1;done

deployment版本回滚

kubectl rollout undo deployment myapp

service是iptables中的规则地址

kubectl get svc

ping 10.99.67.114 无法ping通

curl 10.99.67.114 但可以访问，因为这iptables规则会转发到其他ip

service暴露端口

kubectl edit svc myapp-service

type: NodePort

<http://192.168.4.155:30586/>

yaml字段说明

kubectl explain pod

kubectl explain pod.spec

创建第一个pod，一个pod运行两个容器

vim pod-demo.yaml

apiVersion: v1

kind: Pod

metadata:

name: pod-demo

namespace: default

labels:

app: myapp

tier: frontend

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

- name: busybox

image: busybox:latest

command:

- "bin/sh"

- "-c"

- "echo $(date) >> /usr/share/nginx/html/index.html; sleep 5"

通过yaml配置文件创建pod

kubectl create -f pod-demo.yaml

进入容器中

kubectl exec -it pod-demo -c myapp -- /bin/sh

删除资源清单中定义的资源

kubectl delete -f pod-demo.yaml

kubectl run 是通过控制器创建pod，删除pod后，控制器还在，pod会自动重建

通过yaml创建pod未指定控制器，删除pod后不会自动创建新pod

vim pod-demo.yaml

apiVersion: v1

kind: Pod

metadata:

name: pod-demo

namespace: default

labels:

app: myapp

tier: frontend

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

#显式暴露pod的端口。pod都在pod网络中，不暴露端口也能在pod网络中相互访问，显示暴露端口只是让人看得更明白

#使用name可以在外部通过name替代实际端口号

ports:

#

- name: http

containerPort: 80

- name: https

containerPort: 443

- name: busybox

image: busybox:latest

#如果不填写command，则会执行容器中的entrypoint

#如果指定了args，则entrypoint将使用k8s的args而不使用cmd

command:

- "/bin/sh"

- "-c"

- "echo $(date) >> /usr/share/nginx/html/index.html; sleep 5"

有command无args，则只有command被执行，容器中的entrypoint、cmd被忽略

有args无command, 则args会替换镜像中的cmd

有command和args，则运行command，参数是args

command也可以写成[“/bin/sh”, “-c”, “echo $(date) >> /usr/share/nginx/html/index.html; sleep 5”]

kubectl get pods -l app --show-labels 显示有app标签的pod

kubectl label pods pod-demo release=canary 打标签

kubectl label pods pod-demo release=release –overwrite 改变标签

获取node标签

kubectl get node --show-labels

node设置标签

kubectl label nodes k8s002 disktype=ssd

通过nodeSelector指定pod在标签为disktype=ssd的node上创建

spec:

nodeSelector:

disktype: ssd

加注解

annnotations:

k8s001/create-by: "cluster admin"

restartPolicy:

Always 默认，容器会重启

OnFailure 容器出错会重启，正常停止不重启

Never 不重启

**livenessProbe 存活探针**

vim liveness-exec.yaml

apiVersion: v1

kind: Pod

metadata:

name: liveness-exec-pod

namespace: default

spec:

containers:

- name: liveness-exec-container

image: busybox:latest

imagePullPolicy: IfNotPresent

command: ["/bin/sh", "-c", "touch /tmp/healthy; sleep 30; rm -f /tmp/healthy; sleep 3600"]

livenessProbe:

exec:

command: ["test", "-e", "/tmp/healthy"]

initialDelaySeconds: 1

periodSeconds: 3

在容器中创建文件/tmp/healthy，30秒后删除，3600秒后再次创建。

容器创建成功，1秒后执行检测，每隔3秒检测一次

检测不到文件，当test –e /tmp/healthy 返回false，容器会重启

kubectl get pods pod重启次数很多

http方式的存活探针

vim liveness-httpget.yaml

apiVersion: v1

kind: Pod

metadata:

name: liveness-httpget-pod

namespace: default

spec:

containers:

- name: liveness-httpget-container

image: ikubernetes/myapp:v1

imagePullPolicy: IfNotPresent

ports:

- name: http

containerPort: 80

livenessProbe:

httpGet:

port: http

path: /index.html

initialDelaySeconds: 1

periodSeconds: 3

kubectl get pods

kubectl exec -it liveness-httpget-pod -- /bin/sh

rm -f /usr/share/nginx/html/index.html

退出，容器会重启一次，因为重启后index.html会被重新生成

**容器服务就绪探针 readinessProbe**

vim readiness-httpget.yaml

apiVersion: v1

kind: Pod

metadata:

name: readiness-httpget-pod

namespace: default

spec:

containers:

- name: readiness-httpget-container

image: ikubernetes/myapp:v1

imagePullPolicy: IfNotPresent

ports:

- name: http

containerPort: 80

readinessProbe:

httpGet:

port: http

path: /index.html

initialDelaySeconds: 1

periodSeconds: 3

kubectl exec -it readiness-httpget-pod -- /bin/sh

rm -f /usr/share/nginx/html/index.html

容器还在运行，但是未就绪

kubectl get pods

readiness-httpget-pod 0/1 Running

就绪探针很重要，因为容器启动成功后，还需要一些时间才能对外提供服务，只有当容器处于ready时，pod才会被加入到service中对外提供服务，如果不做就绪探针，容器一启动就对外提供服务，容器内的服务还没启动完成就被加入到service中，访问服务时就报错了

**启动后钩子postStart、终止前钩子preStop shell脚本有问题**

vim poststart.yaml

apiVersion: v1

kind: Pod

metadata:

name: poststart-pod

namespace: default

spec:

containers:

- name: busybox-httpd

image: busybox:latest

imagePullPolicy: IfNotPresent

lifecycle:

postStart:

exec:

command: ["/bin/sh", "-c", "mkdir -p /data/web/html; touch /data/web/html/index.html"]

command: ["/bin/sh", "-c", "echo 这个command比lifecycle中的command先执行"]

**kubernetes控制器相当于状态机，用于控制pod的具体状态和行为，控制器有 ReplicaSet、Deployment、DaemonSet等**

**ReplicaSet，确保pod以指定的副本数运行**

vim rs-demo.yaml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: myapp

namespace: default

#控制器的spec,通过控制器定义replicas副本数

spec:

replicas: 2

selector:

matchLabels:

app: myapp

release: canary

template:

metadata:

name: myapp-pod

labels:

app: myapp

release: canary

environment: qa

#pod的spec

spec:

containers:

- name: myapp-container

image: ikubernetes/myapp:v1

ports:

- name: http

containerPort: 80

将pod-demo的label也设置有app=myapp,release=canar，由于控制器ReplicaSet会控制含有app=myapp,release=canary的pod只能有2个，所以会随机删除一个含有app=myapp,release=canary标签的pod

kubectl label pods pod-demo release=canary

动态修改副本数为5

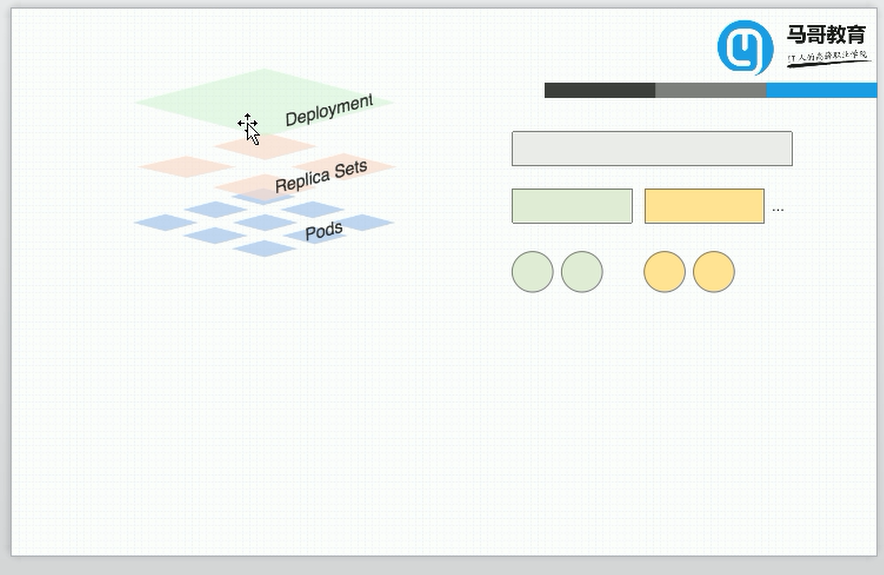
kubectl edit rs myapp

replicas: 5

改版本号，需要删除pod，pod重建后版本才会是v2

image: ikubernetes/myapp:v2

**Deployment控制Replica Sets控制Pods**



vim deploy-demo.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deploy

namespace: default

spec:

replicas: 2

selector:

matchLabels:

app: myapp

release: canary

template:

metadata:

labels:

app: myapp

release: canary

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

ports:

- name: http

containerPort: 80

kubectl apply -f deploy-demo.yaml

**apply通过文件名或标准输入流(stdin)对资源进行配置，即可用于创建也可用于更新**

kubectl get deployment

myapp-deploy 2/2 2 2 2m30s

创建deployment会自动创建replicaSets

kubectl get rs

myapp-deploy-67b6dfcd8 2 2 2 3m2s

67b6dfcd8是template模板的hash

kubectl get pod

myapp-deploy-67b6dfcd8-98gxh 1/1 Running 0 179m

myapp-deploy-67b6dfcd8-ksqnh 1/1 Running 0 179m

pod名称是 deployment名称加上hash

滚动更新

vim deploy-demo.yaml

image: ikubernetes/myapp:v2

**deployment已经创建，kubectl apply会更新deployment**

kubectl apply -f deploy-demo.yaml

kubectl get pods -l app=myapp –w

pod会滚动更新

kubectl get rs

myapp-deploy-67b6dfcd8 0 0 0 4h56m

旧的deployment还保留着，以便回滚时使用

打补丁

kubectl patch deployment myapp-deploy -p '{"spec":{"replicas":5}}'

rollingUpdate滚动更新

kubectl explain deploy.spec.strategy.rollingUpdate.maxSurge

kubectl explain deploy.spec.strategy.rollingUpdate.maxUnavailable

设置更新策略，更新期间允许多一个pod

kubectl patch deployment myapp-deploy -p '{"spec":{"strategy":{"rollingUpdate":{"maxSurge":1,"maxUnavailable":0}}}}'

更新一个pod的版本，并且暂停更新过程，此时有六个pod，金丝雀更新

kubectl set image deployment myapp-deploy myapp=ikubernetes/myapp:v3 && kubectl rollout pause deployment myapp-deploy

kubectl get pods -l app=myapp -w

kubectl rollout status deployment myapp-deploy

用了一端时间后没有问题，重新更新

kubectl rollout resume deployment myapp-deploy

已经升级到v3

kubectl describe pod myapp-deploy-7f577979c8-4hxkx

查看deploy的历史版本

kubectl rollout history deployment myapp-deploy

回滚到第一个版本

kubectl rollout undo deployment myapp-deploy --to-revision=1

**DaemonSet**

vim ds-demo.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: redis

namespace: default

spec:

replicas: 1

selector:

matchLabels:

app: redis

role: logstor

template:

metadata:

labels:

app: redis

role: logstor

spec:

containers:

- name: redis

image: redis:4.0-alpine

ports:

- name: redis

containerPort: 6379

---

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: filebeat-ds

namespace: default

spec:

selector:

matchLabels:

app: filebeat

release: stable

template:

metadata:

labels:

app: filebeat

release: stable

spec:

containers:

- name: filebeat

image: ikubernetes/filebeat:5.6.5-alpine

env:

- name: REDIS\_HOST

value: redis.default.svc.cluster.local

- name: REDIS\_LOG\_LEVEL

value: info

kubectl apply -f ds-demo.yaml

使用---在yaml中分割两个资源

没定义replica副本数，但pod任然有两份，因为从节点有两个，每个节点部署一个pod

创建service暴露端口

kubectl expose deployment redis --port=6379

服务之间调用通过service，yaml配置环境变量

env:

- name: REDIS\_HOST

value: redis.default.svc.cluster.local

DaemonSet更新不支持maxSuger，因为每个节点只能有一个，不能多

**Service**

ClusterIP类型的svc

vim redis-svc.yaml

apiVersion: v1

kind: Service

metadata:

name: redis

namespace: default

spec:

selector:

app: redis

role: logstor

clusterIP: 10.97.97.97

type: ClusterIP

ports:

# service端口，给外部使用

- port: 6379

# pod的端口

targetPort: 6379

kubectl apply -f redis-svc.yaml

kubectl describe svc redis

Selector: app=redis,role=logstor

Type: ClusterIP

IP: 10.97.97.97

Port: <unset> 6379/TCP

TargetPort: 6379/TCP

Endpoints: 10.244.2.34:6379

新创建的svc匹配到一个pod redis-58b9f5776-pmgd7 ，pod ip是10.244.2.34

nodePort类型的SVC

vim myapp-svc.yaml

apiVersion: v1

kind: Service

metadata:

name: myapp

namespace: default

spec:

selector:

app: myapp

release: canary

clusterIP: 10.99.99.99

type: NodePort

ports:

# service端口，给外部使用

- port: 80

# pod的端口

targetPort: 80

#节点端口，不指定则动态分配。手动指定，在节点中，此端口不能被占用

nodePort: 30080

<http://192.168.4.155:30080/hostname.html>

<http://192.168.4.156:30080/hostname.html>

<http://192.168.4.157:30080/hostname.html>

集群节点都能访问服务

无头svc

vim myapp-svc-headless.yaml

apiVersion: v1

kind: Service

metadata:

name: myapp-svc

namespace: default

spec:

selector:

app: myapp

release: canary

#无头service，请求不经过service，而是直接访问pod，所以也就不需要写clusterIP

clusterIP: None

ports:

- port: 80

targetPort: 80

kubectl apply -f myapp-svc-headless.yaml

获取DNS服务器

kubectl get svc -n kube-system

kube-dns ClusterIP 10.96.0.10 <none>

DNS解析myapp-svc服务域名

dig -t A myapp-svc.default.svc.cluster.local. @10.96.0.10

myapp-svc.default.svc.cluster.local. 5 IN A 10.244.1.39

myapp-svc.default.svc.cluster.local. 5 IN A 10.244.1.42

myapp-svc.default.svc.cluster.local. 5 IN A 10.244.2.33

myapp-svc.default.svc.cluster.local. 5 IN A 10.244.1.41

myapp-svc.default.svc.cluster.local. 5 IN A 10.244.2.36

myapp-svc对应的ip正好是myapp这5个pod的ip

kubectl get pods -o wide -l app=myapp

myapp-deploy-67b6dfcd8-4cm4s 1/1 Running 2 21h 10.244.1.42

myapp-deploy-67b6dfcd8-6ncv7 1/1 Running 2 21h 10.244.2.33

myapp-deploy-67b6dfcd8-b4jds 1/1 Running 2 21h 10.244.2.36

myapp-deploy-67b6dfcd8-g9vk6 1/1 Running 2 21h 10.244.1.39

myapp-deploy-67b6dfcd8-q4jx4 1/1 Running 2 21h 10.244.1.41

进入一个pod中

kubectl exec -it myapp-deploy-67b6dfcd8-4cm4s /bin/sh

通过域名发送请求

wget -O - -q <http://myapp-svc>

**ingress**

kubectl explain ingress

mkdir ingress-nginx

先创建namespace

kubectl apply -f namespace.yaml

批量创建文件夹中声明的资源

kubectl apply -f ./

default-backend.yaml的镜像无法下载，需要在3个节点上执行

docker pull mirrorgooglecontainers/defaultbackend:1.4

docker tag mirrorgooglecontainers/defaultbackend:1.4 gcr.io/google\_containers/defaultbackend:1.4

kubectl get pods -n ingress-nginx

创建ingress资源文件目录

mkdir ingress && cd ingress

vim deploy-demo.yaml

apiVersion: v1

kind: Service

metadata:

name: myapp

namespace: default

spec:

selector:

app: myapp

release: canary

ports:

- name: http

port: 80

targetPort: 80

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deploy

namespace: default

spec:

replicas: 2

selector:

matchLabels:

app: myapp

release: canary

template:

metadata:

labels:

app: myapp

release: canary

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v2

ports:

- name: http

containerPort: 80

kubectl apply -f deploy-demo.yaml

安装service-nodeport，将ingress-nginx的端口映射到宿主机

wget <https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/provider/baremetal/service-nodeport.yaml>

vim service-nodeport.yaml 指定nodePort方便记忆

- name: http

port: 80

targetPort: 80

protocol: TCP

nodePort: 30080

- name: https

port: 443

targetPort: 443

protocol: TCP

nodePort: 30443

kubectl apply -f service-nodeport.yaml

service 10.107.200.114:80对应着多个pod10.244.1.45:80,10.244.2.39:80

kubectl describe svc myapp

IP: 10.107.200.114

Port: http 80/TCP

TargetPort: 80/TCP

Endpoints: 10.244.1.45:80,10.244.2.39:80

添加DNS转发

vim /etc/hosts

127.0.0.1 myapp.k8s001

创建ingress-myapp

cd ingress-nginx/ingress/

vim ingress-myapp.yaml

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: ingress-myapp

namespace: default

annotations:

kubernetes.io/ingress.class: "nginx"

spec:

rules:

- host: myapp.k8s001

http:

paths:

- path:

backend:

serviceName: myapp

servicePort: 80

kubectl apply -f ingress-myapp.yaml

通过域名访问

curl http://myapp.k8s001:30080

kubectl describe ingress ingress-myapp

Rules:

Host Path Backends

---- ---- --------

k8s001

myapp:80 (10.244.1.45:80,10.244.2.39:80)

kubectl exec -n ingress-nginx -it nginx-ingress-controller-84c5d78fd5-dfwwl -- /bin/sh

cat nginx.conf

## start server myapp.k8s001

server {

server\_name myapp.k8s001 ;

listen 80;

ingress-myapp.yaml中的配置注入到了容器的nginx中

定义volume

# vim pod-vol-demo.yaml

apiVersion: v1

kind: Pod

metadata:

name: pod-demo

namespace: default

labels:

app: myapp

tier: frontend

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

ports:

- name: http

containerPort: 80

#使用名称为html的volumes

volumeMounts:

- name: html

mountPath: /data/web/html/

- name: busybox

image: busybox:latest

imagePullPolicy: IfNotPresent

volumeMounts:

- name: html

mountPath: /data/

command:

- "/bin/sh"

- "-c"

- "sleep 7200"

# 定义一个volume，名称必填

volumes:

- name: html

emptyDir: {}

#emptyDir: {} 表示数据存储在pod中，pod没了，数据就没了

kubectl apply -f pod-vol-demo.yaml

kubectl get pod

kubectl exec -it pod-demo -c busybox -- /bin/sh

在/data 下生成 index.html

echo $(data) >> /data/index.html

myapp与busybox共用一个volume，busybox映射到容器内的/data下，myapp映射到容器内的/data/web/html下

kubectl exec -it pod-demo -c myapp -- /bin/sh

cat /data/web/html/index.html busybox中生成的index.html已经在myapp中了

kubectl delete -f pod-vol-demo.yaml

vim pod-vol-demo.yaml

apiVersion: v1

kind: Pod

metadata:

name: pod-demo

namespace: default

labels:

app: myapp

tier: frontend

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

imagePullPolicy: IfNotPresent

ports:

- name: http

containerPort: 80

volumeMounts:

- name: html

mountPath: /usr/share/nginx/html/

- name: busybox

image: busybox:latest

imagePullPolicy: IfNotPresent

volumeMounts:

- name: html

mountPath: /data/

command: ["/bin/sh"]

args: ["-c", "while true; do echo $(date) >> /data/index.html; sleep 2; done"]

volumes:

- name: html

emptyDir: {}

shell脚本一定要写成 while true; do echo $(date) >> /data/index.html; sleep 2; done

kubectl apply -f pod-vol-demo.yaml

kubectl get pod -o wide

curl 10.244.2.27

得到一大堆日期数据

vim pod-hostpath-vol.yaml

apiVersion: v1

kind: Pod

metadata:

name: pod-vol-hostpath

namespace: default

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

volumeMounts:

- name: html

mountPath: /usr/share/nginx/html/

volumes:

- name: html

hostPath:

path: /data/pod/volume1

type: DirectoryOrCreate

type: DirectoryOrCreate 表示数据存储在节点上，无法跨节点存储

kubectl apply -f pod-hostpath-vol.yaml

kubectl get pod -o wide

curl 10.244.2.40

pod在k8s003上，获取到k8s003上的数据

删除pod数据还存在，因为数据在节点上

多节点，使用nfs共享存储方案

在一台机器上安装nfs

yum -y install nfs-utils

mkdir /data/volumes –pv

vim /etc/exports

/data/volumes 192.168.4.\* (rw, no\_root\_squash)

# \*和()之间有空格

systemctl start nfs

ss –tnl # nfs监听2049端口

/data/volumes新建index.html

vim index.html

<h1>使用nfs共享存储，index主页</h1>

k8s002、k8s003安装nfs

yum -y install nfs-utils

mount -t nfs 192.168.4.159:/data/volumes /mnt

#卸载/mnt，之前仅做测试用

umount /mnt

vim pod-vol-nfs.yaml

apiVersion: v1

kind: Pod

metadata:

name: pod-vol-nfs

namespace: default

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

volumeMounts:

- name: html

mountPath: /usr/share/nginx/html/

volumes:

- name: html

nfs:

path: /data/volumes

server: 192.168.4.159

kubectl apply -f pod-vol-nfs.yaml

即可看到nfs上的主页

curl 10.244.2.41

**使用pv、pvc**

nfs机器上执行，设置多个存储卷

mkdir v{1,2,3,4,5}

vim /etc/exports

/data/volumes/v1 192.168.4.0/24(rw,sync,no\_root\_squash)

/data/volumes/v2 192.168.4.0/24(rw,sync,no\_root\_squash)

/data/volumes/v3 192.168.4.0/24(rw,sync,no\_root\_squash)

/data/volumes/v4 192.168.4.0/24(rw,sync,no\_root\_squash)

/data/volumes/v5 192.168.4.0/24(rw,sync,no\_root\_squash)

systemctl restart nfs

exportfs –arv

showmount -e

主节点创建5个pv

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv001

labels:

name: pv001

spec:

nfs:

path: /data/volumes/v1

server: 192.168.4.159

accessModes: ["ReadWriteMany", "ReadWriteOnce"]

capacity:

storage: 2Gi

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv002

labels:

name: pv002

spec:

nfs:

path: /data/volumes/v2

server: 192.168.4.159

accessModes: ["ReadWriteMany"]

capacity:

storage: 5Gi

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv003

labels:

name: pv003

spec:

nfs:

path: /data/volumes/v3

server: 192.168.4.159

accessModes: ["ReadWriteMany", "ReadWriteOnce"]

capacity:

storage: 10Gi

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv004

labels:

name: pv004

spec:

nfs:

path: /data/volumes/v4

server: 192.168.4.159

accessModes: ["ReadWriteMany", "ReadWriteOnce"]

capacity:

storage: 10Gi

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv005

labels:

name: pv005

spec:

nfs:

path: /data/volumes/v5

server: 192.168.4.159

accessModes: ["ReadWriteMany", "ReadWriteOnce"]

capacity:

storage: 10Gi

pv不需要写名称空间，pv属于集群级别。但PVC是要有名称空间namespace

kubectl apply -f pv-demo.yaml

kubectl get pv

创建PVC

vim pod-vol-pvc.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mypvc

namespace: default

spec:

accessModes: ["ReadWriteMany"]

resources:

requests:

storage: 6Gi

---

apiVersion: v1

kind: Pod

metadata:

name: pod-vol-pvc

namespace: default

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

volumeMounts:

- name: html

mountPath: /usr/share/nginx/html/

volumes:

- name: html

persistentVolumeClaim:

claimName: mypvc

pvc的accessModes必须pv的accessModes的子级

resource:

request:

storage: 6Gi

选择存储容量>=6Gi的pv

kubectl apply -f pod-vol-pvc.yaml

kubectl get pvc

kubectl get pv

pv003 10Gi RWO,RWX Retain Bound default/mypvc

pv003这个pv 和mypvc建立了 绑定关系

在v3中建立index.html

master上curl 10.244.2.42 就能获取到index.html

由于pv的RECLAIM POLICY是Retain，即便把pod、PVC都删了，pv003上的数据还是会保留

kubectl get pvc -o wide

kubectl get pv -o wide

PV、PVC都没有ip，即不存储在节点上，是存储在k8s的etcd中

configMap

kubectl create configmap nginx-config --from-literal=nginx\_port=80 --from-literal=server\_name=192.168.4.155

kubectl get cm

kubectl describe cm nginx-config

vim [www.conf](http://www.conf)

server{

server\_name 192.168.4.155;

listen 80;

root /data/web/html/

}

kubectl create configmap nginx-www --from-file=www=./www.conf www为键，值等于文件中的配置

kubectl create configmap nginx-www --from-file=./www.conf [www.conf](http://www.conf)文件名为键，值等于文件中的配置

kubectl get cm nginx-www -o yaml

从configMap中获取配置

vim pod-configmap.yaml

apiVersion: v1

kind: Pod

metadata:

name: pod-cm-1

namespace: default

labels:

app: myapp

tier: frontend

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

ports:

- name: http

containerPort: 80

env:

- name: NGINX\_SERVER\_PORT

valueFrom:

configMapKeyRef:

name: nginx-config

key: nginx\_port

- name: NGINX\_SERVER\_NAME

valueFrom:

configMapKeyRef:

name: nginx-config

key: server\_name

kubectl apply -f pod-configmap.yaml

kubectl exec -it pod-cm-1 -- /bin/sh

printenv

环境变量已经注入了

NGINX\_SERVER\_PORT=80

NGINX\_SERVER\_NAME=192.168.4.155

运行时编辑，把端口改成8080

kubectl edit cm nginx-config

nginx\_port: "8080"

cm nginx-config改了，但是容器里面的环境变量没改

通过volume实现动态替换配置文件

vim pod-configmap-2.yaml

apiVersion: v1

kind: Pod

metadata:

name: pod-cm-2

namespace: default

labels:

app: myapp

tier: frontend

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

ports:

- name: http

containerPort: 80

volumeMounts:

- name: nginxconf

mountPath: /etc/nginx/config.d/

readOnly: true

volumes:

- name: nginxconf

configMap:

name: nginx-config

kubectl apply -f pod-configmap-2.yaml

kubectl get pod

kubectl exec -it pod-cm-2 -- /bin/sh

kubectl edit cm nginx-config

nginx\_port: "8088"