Master/node

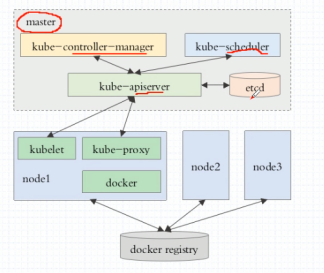
Master核心组件： API server，Scheduler，Controller-Manager etcd（存储组件）

Node核心组件： kubelet（核心组件）， docke（容器引擎（支持的引擎不止docker一个））， kube-proxy

Pod Label， label selector

Label： key=value

Label Selector：



Pod：

自主式Pod

控制器管理的Pod

ReplictionController(老版本控制器)

ReplicaSet（新版本控制器）

Deployment #通过控制replicaset来控制pod，最应该掌握的控制器之一

StatefuSet

Job，Ctonjob

AddOns： 附加组件

环境准备：、

Master， etcd： 172.18.0.70

Node1： 172.18.0.67

Node2： 172.18.0.68

前提：

1. 基于主机名通信： /etc/hosts
2. 时间同步 要不然会出一大推乱七八糟的报错
3. 关闭firewall和iptables.service

OS： Centos 7.3.1611 Rxtras仓库中

安装配置步骤：

1. Etcd cluster， 近master节点
2. Flannel，集群的所有节点
3. 配置k8s的master： 仅master节点

Kubernetes-master

启动的服务：

Kube-apiserver， kube-scheduler， kube-controller-manager

1. 配置k8s的各node节点：

Kubernetes-node

先设定启动docker服务

启动的k8s的服务

Kube-proxy， kubelet

k8s

先准备两台机器：

172.18.0.70 master #

172.18.0.67 node1

172.18.0.68 node2

注意： 一定要做好时间同步 要不然会有一大推莫名其妙的报错

然后两台机器都要配置docker和k8s的yum源

#docker18.09版本和k8s1.15.3版本

wget https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo -O /etc/yum.repos.d/docker-ce.repo

https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64/

直接都使用阿里的镜像源

[root@master ~]# yum list docker-ce --showduplicates | sort -r

#如果对版本有要求可以使用这个查看 18.09，安装对应的包

所有机器都安装

[root@master ~]# yum install docker-ce-18.09.3-3.el7

yum -y install kubelet kubeadm kubectl（api命令行工具）

然后准备启动docker，在启动docker之前建议先定义一个环境变量 随便另起一行写

[root@master ~]# vim /usr/lib/systemd/system/docker.service

Environment="HTTPS\_PROXY=http://www.ik8s.io:10080""

#意思是我们访问docker服务的时候通过代理下载相关的镜像文件

Environment="NO\_PROXY=127.0.0.8,172.18.0.0/16" #不用代理 本机

：wq

docker国内加速

mkdir -p /etc/docker

vim /etc/docker/daemon.json

{

  "registry-mirrors": ["https://lvb4p7mn.mirror.aliyuncs.com"]

}

上面这个貌似不太好用，

#!/bin/bash

# download k8s 1.15.3 images

# get image-list by 'kubeadm config images list --kubernetes-version=v1.15.3'

# gcr.azk8s.cn/google-containers == k8s.gcr.io

images=(

kube-apiserver:v1.15.3

kube-controller-manager:v1.15.3

kube-scheduler:v1.15.3

kube-proxy:v1.15.3

pause:3.1

etcd:3.3.10

coredns:1.3.1

)

for imageName in ${images[@]};do

docker pull gcr.azk8s.cn/google-containers/$imageName

docker tag gcr.azk8s.cn/google-containers/$imageName k8s.gcr.io/$imageName

docker rmi gcr.azk8s.cn/google-containers/$imageName

Done

#使用此脚本可以避免因为镜像源的问题！！！非常重要！！！

加载环境变量

[root@master ~]# systemctl daemon-reload

启动docker并使用docker info 查看

[root@master ~]# systemctl start docker && docker info

[root@master ~]#

#默认是1就不用改

[root@master ~]# rpm -ql kubelet

/etc/kubernetes/manifests #清单目录

/etc/sysconfig/kubelet #配置文件

/usr/bin/kubelet

/usr/lib/systemd/system/kubelet.service

[root@master ~]# systemctl enable kubelet.service #设置成开机自启动就可以了

[root@master ~]# systemctl enable docker

[root@master ~]# kubelet init --help

#准备初始化

要把swap交换分区关了！！！！！！！！

初始化：

[root@master ~]# kubeadm init --kubernetes-version=v1.15.3 --pod-network-cidr=10.244.0.0/16 --service-cidr=10.96.0.0/12

##ps： 如果因为镜像问题可以使用上面的脚本执行，直接拉取镜像

成功之后会有以下几条命令，执行一下 注意一定不要忘记执行!!

mkdir -p $HOME/.kube

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

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

初始化完成之后会有一段token哈希值，加入集群用的非常重要，可以把他先保存下来，如下：

kubeadm join 172.18.0.70:6443 --token nsezuu.f8xhql0lmz262tqv \

--discovery-token-ca-cert-hash sha256:c98f345addbb9a585d1e9edf3f584c9dcee4dbb847a63392dc2ba444e77ec0a7

[root@master ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

k8s.gcr.io/kube-proxy v1.15.3 232b5c793146 8 days ago 82.4MB

k8s.gcr.io/kube-apiserver v1.15.3 5eb2d3fc7a44 8 days ago 207MB

k8s.gcr.io/kube-controller-manager v1.15.3 e77c31de5547 8 days ago 159MB

k8s.gcr.io/kube-scheduler v1.15.3 703f9c69a5d5 8 days ago 81.1MB

k8s.gcr.io/coredns 1.3.1 eb516548c180 7 months ago 40.3MB

k8s.gcr.io/etcd 3.3.10 2c4adeb21b4f 9 months ago 258MB

k8s.gcr.io/pause 3.1 da86e6ba6ca1 20 months ago

成功之后在node1和node2上面都安装—

[root@node1 ~]# yum install docker-ce-18.09.7-3.el7 kubectl.x86\_64 kubelet.x86\_64 kubeadm.x86\_64

[root@node2 ~]# yum install docker-ce-18.09.7-3.el7 kubectl.x86\_64 kubelet.x86\_64 kubeadm.x86\_64

swap没关的话就忽略swap参数

vim /etc/sysconfig/kubelet

KUBELET\_EXTRA\_ARGS="--fail-swap-on=false"

KUBE\_PROXY\_MODE=ipvs

[root@master ~]# kubectl get cs #查看组件信息

NAME STATUS MESSAGE ERROR

scheduler Healthy ok

controller-manager Healthy ok

etcd-0 Healthy {"health":"true"}

[root@master ~]# kubectl get nodes

NAME STATUS ROLES AGE VERSION

master NotReady master 32m v1.15.3

#这里的状态信息显示还未准备好是因为缺少一个重要组件flannel

在git上查找flannel获取以下命令：

[root@master ~]# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

#这是一个在线的部署清单，基于此清单下载镜像flannel

[root@master ~]# kubectl get node

NAME STATUS ROLES AGE VERSION

master Ready master 41m v1.15.3

#现在就已经准备好了

[root@master ~]# kubectl get ns #查看名称空间

NAME STATUS AGE

default Active 43m

kube-node-lease Active 43m

kube-public Active 43m

kube-system Active 43m

把master的配置文件分别拷给node1和node2

[root@master ~] scp /usr/lib/systemd/system/docker.service node1:/usr/lib/systemd/system/docker.service

[root@master ~]# scp /usr/lib/systemd/system/docker.service node2:/usr/lib/systemd/system/docker.service

[root@master ~]# scp /etc/sysconfig/kubelet node1:/etc/sysconfig/kubelet

[root@master ~]# scp /etc/sysconfig/kubelet node2:/etc/sysconfig/kubelet

分别在node1和node2上面启动docker并设置成开启自启

[root@node1 ~]# systemctl start docker #启动docker

[root@node1 ~]# systemctl enable docker.service kubelet.service #设置成开启自启

[root@node2 ~]# systemctl start docker #启动docker

[root@node2 ~]# systemctl enable docker.service kubelet.service #设置成开启自启

[root@node1 ~]# kubeadm join 172.18.0.70:6443 --token k0q4vt.2ok77anvdhzip6s8 --discovery-token-ca-cert-hash sha256:77b7364b7fc2c886aa889488d3c06d6c1f3a8a1cfc48a8857354afa749c37630 --ignore-preflight-errors=Swap

[root@node2 ~]# kubeadm join 172.18.0.70:6443 --token k0q4vt.2ok77anvdhzip6s8 --discovery-token-ca-cert-hash sha256:77b7364b7fc2c886aa889488d3c06d6c1f3a8a1cfc48a8857354afa749c37630 --ignore-preflight-errors=Swap

#加入集群

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

[root@master ~]# kubectl describe node master #查看节点的详细信息，比较常用

[root@master ~]# kubectl cluster-info #查看集群信息

使用k8s进行增删改查

测试：

[root@master ~]# kubectl run nginx-ceshi --image=nginx --port=80 --replicas=1

#启动一个nginx pod

[root@master ~]# kubectl get pod #查看pod

NAME READY STATUS RESTARTS AGE

nginx-ceshi-748587595b-975t7 0/1 ContainerCreating 0 8s

[root@master ~]# kubectl get svc #查看服务

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

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 93m

myapp ClusterIP 10.97.57.166 <none> 80/TCP 22s

nginx ClusterIP 10.108.78.248 <none> 80/TCP 72m

[root@master ~]# kubectl scale --replicas=5 deployment myapp

#扩容副本，后面数值多少就是几个副本

[root@master ~]# kubectl run client1 --image=busybox --replicas=1 -it --restart=Never

#加一个-it表示创建完直接以交互式方式进去，创建一个客户端进行测试

/ # wget -O - -q 10.244.1.5/hostname.html #查看是从哪个节点上运行的

myapp-84cd4b7f95-mj6k6

[root@master ~]# kubectl describe deployment nginx-ceshi

#查看选择器详细信息

[root@master ~]# kubectl get deployment -w

#-w实时查看

[root@master ~]# kubectl set image deployment myapp myapp=ikubernetes/myapp:v2

deployment.extensions/myapp image updated

#使用set image进行版本升级

[root@master ~]# kubectl rollout undo deployment myapp

#使用rollout out进行回滚版本

[root@master ~]# kubectl edit svc myapp

##修改svc内容

27： type: NodePort

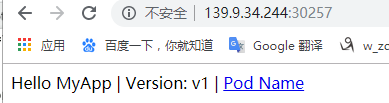
#大约27行，吧他修改为NodePort，就可以使用外网访问了，大小写不能错！

[root@master ~]# kubectl get svc #修改完之后使用get svc查看就会发现多一个端口

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

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 155m

myapp NodePort 10.97.57.166 <none> 80:30257/TCP 62m



删除node节点：

一共分两步：

1.排干此节点

kubectl drain 节点名称 --delete-local-data --force --ignore-daemonsets

2.删除此节点

kubectl delete node 节点名称

Yaml：

创建资源的方法：

apiserver仅接受JSON格式的资源定义；

yaml格式提供配置清单，apiserver可自动 将其转化为JSON格式，而后再提交。

大部分资源的配置清单：

[root@master ~]# kubectl explain pod #查看字段，相当于一个帮助

[root@master ~]# kubectl explain pods.status

##查看字段的详细帮助

apiversion： group/version

kind： 资源类别

metadata： 元数据

name

namespace

Latels

Annotations

每个资源的引用PATH

/api/GROUP/VERSION/namespeaces/NAMESPEACE/TYPE/NAME

Sepc: 期望的状态

Status： 当前状态，current state， 这五个一级字段由kubernetes集群维护；

下面是创建了一个pod里面有两个容器：格式注意下 yaml文件

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"

[root@master yamls]# kubectl create -f pod-damo.yaml

#根据清单创建pod

[root@master yamls]# kubectl delete -f pod-damo.yaml

[root@master yamls]# kubectl delete pods pod-demo

#删除pod

标签：

Key=value 键名和键值都必须小于63个字符

Key只能使用： 字母 数字 下划线 点号 ， 只能使用字母或数字开头

Value 可以为空，键值不能为空，只能以字母或数字开头及结尾，之间可使用

[root@master yamls]# kubectl get pod -l app #-l标签过滤

[root@master yamls]# kubectl label pods pod-demo release=ccc ##打标签

[root@master yamls]# kubectl get pods -l app --show-labels #查看

节点选择器：

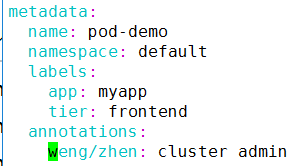
nodeSelector: yaml参数指定在哪个标签的node上运行，和containers平行

KEY=VALUE

NodeName :

标识：

annotations: #与label不同的地方在于，它不能用于挑选资源对象，仅用于为对象提供“元数据”，，写在metadata里面，如下



使用kubectl describe pods pod名字 查看

Pod的生命周期：

状态： Pending（挂起）, Running，Failed，Succeeded，Unknown

Pod生命周期中的重要行为：

初始化容器

容器探测

Liveness（存活性探测）

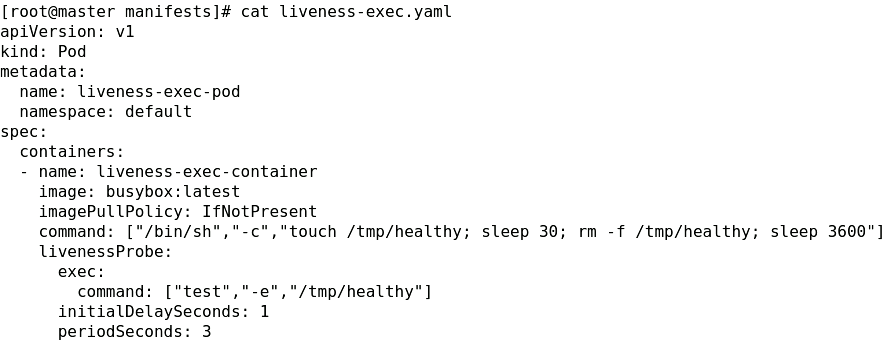
Readinessi(就绪与否)

--必须做的

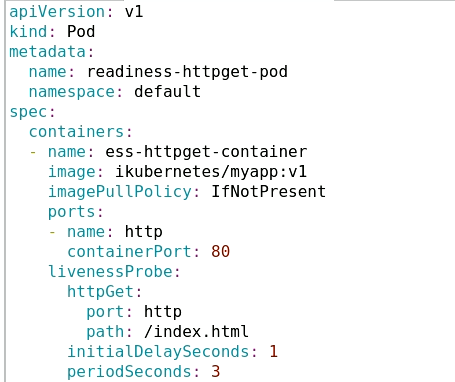
探针类型有三种：

ExecAction TCPSocketAction HTTPGetAction

示例：



HTTPGetAction：



[root@master ym]# kubectl explain pods.spec.container.lifecycle #查看pod生命周期帮助

有点晕这里

回顾： Pod

apiVersion, kind, metadata, status(只读)

spec:

containers:

nodeSelector

nodeName

restartPolicy: #重启策略

Always, Never, OnFailure

containers:

name

image

imagePullPolicy: Always Never IfNotPresent

ports:

name

contarnerPort #暴露端口

livenessProbe #存活性探测

readlinessProbe #就绪状态探测

Liftcycle

ExecAction： exec

TCPocketAction: tcpSocket

HTTPGetAction: httpGet

Pod 控制器

ReplictionController(老版本控制器)

ReplicaSet（新版本控制器） #支持扩容，直接编辑yaml实时文件replicas

[root@master ~]# kubectl edit rs myapp 保存即生效

Deployment #通过控制replicaset来控制pod，最应该掌握的控制器之一

DaemoSet #控制每个node上都有一个pod副本

Job #

Cronjob

StatefulSet

[root@master ~]# kubectl explain rs #查看新版本控制器的手册

使用ReplicaSet控制器创建的yaml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: myapp

namespace: default

spec:

replicas: 2

selector:

matchLabels:

app: myapp

release: canary

template:

metadata:

name: myapp-pod

labels:

app: myapp

release: canary

spec:

containers:

- name: myapp-container

image: ikubernetes/myapp:v1

ports:

- name: http

containerPort: 80

使用Deployment控制器创建pod

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

[root@master ~]# kubectl apply -f deplo.yaml

[root@master ~]# kubectl get deploy

[root@master ~]# kubectl get rs

如果想要扩容副本数，直接用vim编辑yaml文件 然后再执行apply -f 创建，apply可以重复创建

使用DamoSet控制器 实例：

[root@master ~]# kubectl explain ds ds缩写

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

--- #用三横杠隔开就可以写在同一个yaml文件里

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: myapp-ds

namespace: default

spec:

selector:

matchLabels:

release: stable

app: filebeat

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

value: info+

两个pod之前联动靠svc

Service：

工作模式： userspace， iptables, ipvs

Userspace: 1.1

iptables: 1.10

Ipvs: 1.11+

类型：

ExternalName，ClusterIP（集群内部）， NodePort， and LoadBalancer

NodePort：

过程：Client-->NodeIP:NodePort-->ClusterIP：ServicePort-->PodIP:contarinersPort

资源记录：

SVC\_NAME. NS\_NAME DAMAIN.LTD

Svc.Cluster.Local

Redis.default.svc.cluster.local

这张不太懂

存储卷：

SAN(本地存储): ISCSI

NAS（网络存储）: nfs, cifs

分布式存储： glusterfs， rbd， cephfs

云存储：EBS（亚马逊的），Azure Disk（微软的）

# Kubectl explain pods.spec.volumes 这里显示它支持哪些存储

[root@master ~]# kubectl explain pods.spec.volumes.emptyDir

[root@master ~]# kubectl explain pods.spec.containers.volumeMounts

apiVersion: v1

kind: Pod

metadata:

name: pod-demo

namespace: default

labels:

app: myapp

tier: frontend

annotations:

magedu.com/created-by: "cluster admin"

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

ports:

- name: http

containerPort: 80

- name: https

containerPort: 443

volumeMounts:

- name: html

mountPath: /data/web/html

- name: busybox

image: busybox:latest

imagePullPolicy: IfNotPresent

command:

- "/bin/sh"

- "-c"

- "sleep 7200"

volumeMounts:

- name: html

mountPath: /data/

volumes:

- name: html

emptyDir: {}

hostPath： 节点级存储 缺点节点坏了，数据也就没了

示例：

apiVersion: v1

kind: Pod

metadata:

name: pod-vol

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

NFS: 网络持久化存储



PVC：

Pvc也是kubernetes里标准的资源

[root@master ~]# kubectl explain pvc.spec

accessModes #访问模式：

ReadWriteOnce： 单路读写

ReadOnlyMany： 多路只读

ReadWriteMany： 多路读写

resources #资源限制

Selector #标签选择器

storageClassName #存储类

volumeMode #存储卷的模式

volumeName #存储卷的名字

环境准备：

在所有机器上都安装nfs-utils

以node2作为nfs服务器

在node2上创建五个挂载目录 ： [root@node2 ~]# mkdir v{1..5}

[root@node2 ~]# vim /etc/exports

/root/v1 172.18.0.0/16(rw)

/root/v2 172.18.0.0/16(rw)

/root/v3 172.18.0.0/16(rw)

/root/v4 172.18.0.0/16(rw)

/root/v5 172.18.0.0/16(rw)

创建pv： 如下

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv001

labels:

name: pv001

spec:

nfs:

path: /root/v1

server: node2

accessModes: ["ReadWriteMany","ReadWriteOnce"] #设置访问类型。必须要写

capacity:

storage: 1Gi #大小

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv002

labels:

name: pv002

spec:

nfs:

path: /root/v2

server: node2

accessModes: ["ReadWriteMany","ReadWriteOnce"]

capacity:

storage: 2Gi

[root@master vl]# kubectl apply -f pv.yaml

persistentvolume/pv001 created

persistentvolume/pv002 created

。。。

[root@master vl]# kubectl get pv

NAME CAPACITY ACCESS MODES RECLAIM POLICY（回收策略） STATUS CLAIM STORAGECLASS REASON AGE

pv001 1Gi RWO,RWX Retain Available 5s

pv002 2Gi RWO,RWX Retain Available 5s

。。。。

创建pvc

[root@master vl]# vim pvc.yaml 如下

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mypvc

namespace: default #pvc有命名空间，pv没有

spec:

accessModes: ["ReadWriteMany"]

resources:

requests:

storage: 1Gi

---

apiVersion: v1

kind: Pod

metadata:

name: pod-pvc

namespace: default

labels:

app: myapp

spec:

containers:

- name: mypvc

image: ikubernetes/myapp:v1

volumeMounts:

- name: mydisk

mountPath: /data/wengwengweng

volumes:

- name: mydisk

persistentVolumeClaim:

claimName: mypvc

[root@master vl]# kubectl get pvc

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

mypvc Bound绑定 pv001 1Gi RWO,RWX 2m58s

Pvc是存储在etcd中,就算pod死机，数据也不会丢失

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

configmap

配置容器化应用的方式：

1 自定义命令行参数

Args：

2 把配置文件直接焙进镜像

3 环境变量

4 存储卷

[root@master volume]# kubectl create configmap --help #查看configmap帮助信息

[root@master ~]# kubectl create configmap nginx.cconf --from-file=./nginx.cconf

#创建文件式的cm

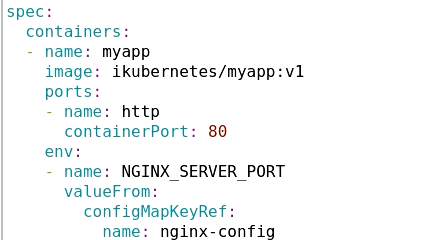
[root@master ~]# kubectl create configmap nginxport --from-literal=nginx\_pory=80

#简单的cm

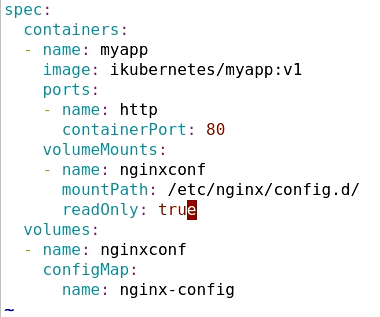
configmap是名称空间级别的资源 直接搜索：

[root@master ~]# kubectl explain cm

以环境变量的方式注入到容器里：



以挂载的方式：



支持edit动态修改

[root@master ~]# kubectl edit cm cm名

secret

[root@master ~]# kubectl create secret --help

docker-registry #保存认证信息

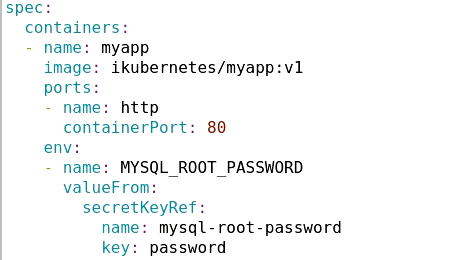
generic #通用的

tls #私钥类型

secret

和configmap很相似

以环境变量方式注入：



注意以这种方式注入的密码都是解码之后的，安全性不高

如果支持dry-run的方式 可以使用-o yaml 来生成一个yaml框架！

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

Statefulset控制器

1. 稳定且唯一的网络标识符
2. 稳定且持久的存储
3. 有序，平滑地部署和扩展
4. 有序的滚动更新

一个典型的statefulset应该由三个组件组成: headless(必须是一个无头服务)

Statefulset

volumeclaimTemplate

[root@master ~]# kubectl explain sts #查看控制器的帮助，简称

apiVersion: v1

kind: Service

metadata:

name: myapp

labels:

app: myapp

spec:

ports:

- port: 80

name: web

clusterIP: None

selector:

app: myapp-pod

---

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: myapp

spec:

serviceName: myapp

replicas: 3

selector:

matchLabels:

app: myapp-pod

template:

metadata:

labels:

app: myapp-pod

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

ports:

- containerPort: 80

name: web

volumeMounts:

- name: myappdata

mountPath: /usr/share/nginx/html

volumeClaimTemplates:

- metadata:

name: myappdata

spec:

accessModes: ["ReadWriteOnce"]

resources:

requests:

storage: 10Gi

sts也支持动态更新，动态扩容

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

[root@master ~]# kubectl proxy --port=8080

[root@master ~]# curl <http://localhost:8080/> #查看各种资源

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

认证

认证 ----> 授权-----> 准入控制

授权用户：

ServiceaccessountName 授权名 写在spec下面

创建sa： kubectl create serviceaccount admin



Dashboard：

1. 部署：
   1. [root@master ~]# Kubectl apply -f <https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml>
   2. 将service改为Nodeport

[root@master ~]# kubectl patch svc kubernetes-dashboard -p '{"spec": {"type": Nodeport}}' -n kube-system

* 1. 认证：

认证时的账号必须为serviceaccount，被dashboard pod拿来由kubernetes进行认证

token:

（1）创建ServiceAccount，根据其管理目标，使用rolebinding或clusterrolebinding绑定至合理role或

clusterrole;

（2）获取到此ServiceAccount的secret，查看secret的详细信息，其中就有token；

kubeconfig: 把ServiceAccount的token封装为kubeconfig文件

（1）创建ServiceAccount，根据其管理目标，使用rolebinding或clusterrolebinding绑定至合理role或

clusterrole;

（2）kubectl get secret | awk '/^ServiceAccount/{print $1}'

KUBE\_TOKEN=$(kubectl get secret SERVCIEACCOUNT\_SERRET\_NAME -o jsonpath={.data.token} |

base64 -d)

（3）生成kubeconfig文件

kubectl config set-cluster --kubeconfig=/PATH/TO/SOMEFILE

kubectl config set-credentials NAME --token=$KUBE\_TOKEN --kubeconfig=/PATH/TO/SOMEFILE

kubectl config set-context

kubectl config use-context

令牌认证：

[root@master ~]# kubectl create serviceaccount dashboard-admin -n kube-system

#先创建一个用户

[root@master ~]# kubectl create clusterrolebinding dashboard-cluster-admin --clusterrole=clu

ster-admin --serviceaccount=kube-system:dashboard-admin #与集群角色进行绑定，左侧名称空间，右侧账号

[root@master ~]# kubectl get secret #找到刚创建角色的token

[root@master ~]# kubectl describe secret dashboard-admin-token-ctjgn -n kube-system

把查到的token信息复制到网页上完成登入

Config认证：

[root@master ~]# kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/pki/ca.crt --server="https://172.18.0.70:6443" --embed-certs=true --kubeconfig=/root/def-ns-admin.conf

[root@master ~]# kubectl config view --kubeconfig=/root/def-ns-admin.conf

[root@master ~]# DEF\_NS\_ADMIN\_TOKEN=$(kubectl get secret def-ns-admin-token-v4wxn -o jsonpath={.data.token} | base64 -d) #知道它的token并解码赋值给变量

[root@master ~]# kubectl config set-credentials def-ns-admin --token=$DEF\_NS\_ADMIN\_TOKEN --kubeconfig=/root/def-ns-admin.conf #把token传进去

[root@master ~]# kubectl config set-context def-ns-admin@kubernetes --cluster=kubernetes --user=def-ns-admin --kubeconfig=/root/def-ns-admin.conf

[root@master ~]# kubectl config use-context def-ns-admin@kubernetes --kubeconfig=/root/def-ns-admin.conf

完成！ 现在的配置文件可以下载下去，传到网页上就可以用了

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

授权插件：

Node， ABAC ， RBAC， Webhook

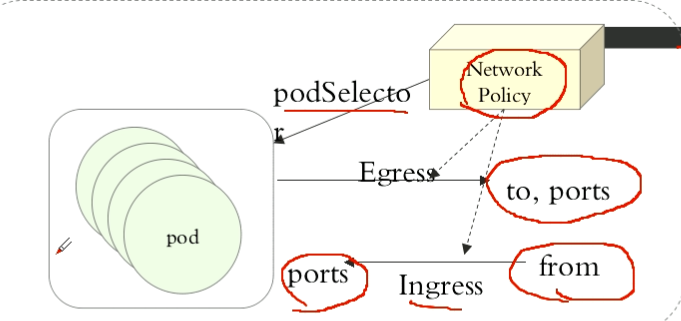
Kubernetes集群的管理方式：

1. 命令式： create， run， expose， delete。。。
2. 命令式配置文件： create -f create -f /PATH/TO/RESOURCE\_CONFIGURATION\_FILE, delete -f, replace -f
3. 声明式配置文件: apply -f, patch,

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

网络插件calico（网络策略）

Calico作为网络插件使用工作于192.168.0.0/16网段



官网文档：

<https://docs.projectcalico.org/v3.9/getting-started/kubernetes/installation/flannel>

选择使用calico用于网络策略，flannel用于网络

部署：

[root@master ~]# kubectl apply -f https://docs.projectcalico.org/v3.9/manifests/canal.yaml

[root@master ~]# kubectl get pods -n kube-system #查看是否安装

使用：

[root@master ~]# kubectl explain networkpolicy.spec

egress 出站

ingress 入站

potSelector 选择哪个pod

policyTypes 策略

基于名称空间的网络策略

拒绝所有入站：

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: dey-all-ingress

spec:

podSelector: {}

policyTypes:

- Ingress #拒绝所有入站

[root@master ~]# kubectl apply -f ingress.yaml -n dev -n 代表指定名称空间

[root@master ~]# kubectl get netpol -n dev #查看策略

放行所有进站：

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: dey-all-ingress

spec:

podSelector: {}

ingress:

- {} #写上之后默认放行所有

policyTypes:

- Ingress

放行特定的网络：

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-all-ingress

spec:

podSelector:

matchLabels:

app: myapp #放行有这个标签的pod

ingress:

- from:

- ipBlock:

cidr: 10.244.0.0/16

except: #指定拒绝特定的网络

- 10.244.1.2/32

ports: #指定放行的端口

- protocol: TCP

port: 80

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

节点选择器：nodeSelector， nodeName

节点亲和调度：nodeAffinity

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

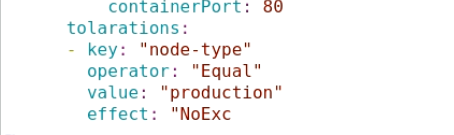
Taint污点：

taint的effect定义对Pod排斥效果：

NoSchedule：仅影响调度过程，对现存的Pod对象不产生影响；

NoExecute：既影响调度过程，也影响现在的Pod对象；不容忍的Pod对象将被驱逐；

PreferNoSchedule：



资源需求与限制：

容器的资源需求，资源限制

requests: 需求，最低保障；

limits： 限制，硬限制

CPU：

1颗逻辑cpu

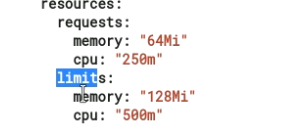
1=1000，millicores

500=0.5CPU

内存：

E, P, T, G, M, K

Ei，Pi

limits 代表上限

Qos： #服务质量

Guranteed： 当资源不够的时候，优先运行#同时设置CPU和内存的requests和limit

cpu.limits = cpu.requests

memory.limits = memory.request #当条件满足时，自动归类为 guranteed

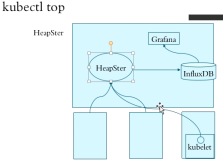
Burstable： #至少有一个容器设置CPU或内存资源的requests属性

BestEffort: #没有任何一个容器设置了requests或limit属性；最低优先级

#当pod资源不够时，自动杀死BestEffort属性的容器

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

Kubectl top：



Helm：

核心术语：

Chart：一个helm程序包；

Repository：Charts仓库，https/http服务器；

Release：特定的Chart部署于目标集群上的一个实例；

Chart -> Config -> Release

程序架构：

helm：客户端，管理本地的Chart仓库，管理Chart, 与Tiller服务器交互，发送Chart，实例安装、查询、卸载等 操作

Tiller：服务端，接收helm发来的Charts与Config，合并生成relase；

安装步骤：

现在git上找到相应的helm包，下载下来

地址：<https://github.com/helm/helm/releases>

解压下载的文件：

[root@master linux-386]# tar -xf tar -xf helm-v2.9.1-linux-amd64.tar.gz

#建议使用旧版本的，要不然会有很多莫名其妙的错误

把helm文件直接mv到/usr/bin 就可以直接使用

因为helm运行要使用管理员权限所以还要绑定cluster-admin集群角色上：

RBAC示例：

<https://github.com/helm/helm/blob/master/docs/rbac.md>

把示例里的yaml文件复制下来，使用apply部署

helm init --service-account tiller --upgrade -i registry.cn-hangzhou.aliyuncs.com/google\_containers/tiller:v2.9.1 --stable-repo-url <https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts>

#因为网络问题，这里使用阿里云的helm镜像

[root@master helm]# kubectl get pods -n kube-system #查看pod是否启动

[root@master helm]# helm version #查看helm和tiller版本

[root@master helm]# helm repo update

[root@master helm]# helm repo list #查看可用的正在使用的仓库

官方可用的chart列表：

<https://hub.kubeapps.com/>

Stable代表稳定版 incubator代表测试版，最好还是使用稳定版

示例：

部署一个memcache：

[root@master ~]# helm search stable/memcachecd

[root@master ~]# helm inspect table/memcached

[root@master ~]# helm install --name mem1 stable/memcached

#部署完会有一个反馈信息在下面

[root@master ~]# helm delete mem1 #卸载

helm常用命令：

release管理：

install #安装

delete #删除

upgrade/rollback #更新/回滚

list #列出

history：release的历史信息；

status：获取release状态信息；

chart管理：

create

fetch

get

inspect

package

Verify

[root@master ~]# helm fetch stable/redis 下载包

[root@master redis]# tree ./

./

├── Chart.yaml #记录chart的元数据。。

├── README.md #说明

├── templates #各种模板文件

│   ├── deployment.yaml

│   ├── \_helpers.tpl

│   ├── networkpolicy.yaml

│   ├── NOTES.txt

│   ├── pvc.yaml

│   ├── secrets.yaml

│   └── svc.yaml

└── values.yaml #自定义属性设置默认值

创建chart：

[root@master ~]# helm create myapp 它会自动生成配置文件

[root@master ~]# tree myapp/

myapp/

├── charts

├── Chart.yaml

├── templates

│   ├── deployment.yaml

│   ├── \_helpers.tpl

│   ├── ingress.yaml

│   ├── NOTES.txt

│   └── service.yaml

└── values.yaml

修改完yaml文件后使用

[root@master redis]# helm lint ../redis #语法检测

EFK： #类似于ELK

E： elasticsearch 和kibana版本一定要一致，错一个小版本号都不行

L: logstash

K: Fluentd 日志收集代理工具

部署ELK：

[root@master ~]# helm repo add incubator <https://kubernetes-charts-incubator.storage.googleapis.com/> #添加测试版仓库

[root@master elasticsearch]# helm install --name els1 --namespace=efk -f values.yaml incubator/elasticsearch

