Kubernetes学习笔记

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 修订者 | 修订描述 |
| 2017.12.08 | v1 | 林 清 | 初稿 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[1. 介绍 3](#_Toc502071273)

[2. 核心概念 3](#_Toc502071274)

[2.1 API对象 3](#_Toc502071275)

[2.1.1 POD 3](#_Toc502071276)

[2.1.2 复制控制器（Replication Controller，RC） 4](#_Toc502071277)

[2.1.3 副本集（Replica Set，RS） 4](#_Toc502071278)

[2.1.4 部署(Deployment) 4](#_Toc502071279)

[2.1.5 服务（Service） 4](#_Toc502071280)

[2.1.6 任务（Job） 4](#_Toc502071281)

[2.1.7 后台支撑服务集（DaemonSet） 5](#_Toc502071282)

[2.1.8 有状态服务集（PetSet） 5](#_Toc502071283)

[2.1.9 集群联邦（Federation） 6](#_Toc502071284)

[2.1.10 存储卷（Volume） 6](#_Toc502071285)

[2.1.11 持久存储卷（Persistent Volume，PV）和持久存储卷声明（Persistent Volume Claim，PVC） 7](#_Toc502071286)

[2.1.12 节点（Node） 7](#_Toc502071287)

[2.1.13 密钥对象（Secret） 7](#_Toc502071288)

[2.1.14 用户帐户（User Account）和服务帐户（Service Account） 7](#_Toc502071289)

[2.1.15 名字空间（Namespace） 8](#_Toc502071290)

[2.1.16 RBAC访问授权 8](#_Toc502071291)

[3. 基于Docker本地运行Kubernetes 8](#_Toc502071292)

[4. minikube 9](#_Toc502071293)

[5. kubeadm 9](#_Toc502071294)

[5.1 配置Master 9](#_Toc502071295)

[5.2 添加node 11](#_Toc502071296)

[6. 读书笔记 11](#_Toc502071297)

[6.1 知识点 11](#_Toc502071298)

[6.1.1 资源分配 11](#_Toc502071299)

[6.1.2 Label 和 Label Selector 12](#_Toc502071300)

[6.1.3 Deployment 扩容 12](#_Toc502071301)

[6.1.4 几个IP 12](#_Toc502071302)

[6.1.5 使用PV和PVC 12](#_Toc502071303)

[6.2 关于Volume 14](#_Toc502071304)

[6.3 关于Labels and Selectors 15](#_Toc502071305)

[6.4 关于命名空间 16](#_Toc502071306)

[6.5 DNS服务 16](#_Toc502071307)

[6.6 常用kubectl命令 16](#_Toc502071308)

[6.7 反向代理 17](#_Toc502071309)

# 介绍

Kubernetes是一个开源的，用于管理云平台中多个主机上的容器，Kubernetes的目标是让部署容器化的应用简单并且高效（powerful）。Kubernetes提供了应用部署，规划，更新，维护的一种机制。

# 核心概念

## API对象

API对象是K8s集群中的管理操作单元。K8s集群系统每支持一项新功能，引入一项新技术，一定会新引入对应的API对象，支持对该功能的管理操作。

每个API对象都有3大类属性：元数据metadata、规范spec和状态status。

* **元数据：**用来标识API对象的，每个对象都至少有3个元数据：namespace，name和uid，除此以外还有各种各样的标签labels用来标识和匹配不同的对象。
* **规范：**描述了用户期望K8s集群中的分布式系统达到的理想状态（Desired State）。例如用户可以通过复制控制器Replication Controller设置期望的Pod副本数为3。
* **status**：描述了系统实际当前达到的状态（Status），例如系统当前实际的Pod副本数为2，那么复制控制器当前的程序逻辑就是自动启动新的Pod，争取达到副本数为3。

k8s中所有的配置都是通过API对象的spec去设置的，也就是用户通过配置系统的理想状态来改变系统，这是k8s重要设计理念之一，即所有的操作都是声明式（Declarative）的而不是命令式（Imperative）的。

### POD

Pod是在K8s集群中运行部署应用或服务的最小单元，它是可以支持多容器的。Pod的设计理念是支持多个容器在一个Pod中共享网络地址和文件系统，可以通过进程间通信和文件共享这种简单高效的方式组合完成服务。

静态POD的配置被写在具体的文件中，比如master节点上的pod。普遍的pod的配置被存到etcd中。

### 复制控制器（Replication Controller，RC）

RC是K8s集群中最早的保证Pod高可用的API对象。通过监控运行中的Pod来保证集群中运行指定数目的Pod副本。RC是K8s较早期的技术概念，只适用于长期伺服型的业务类型。

### 副本集（Replica Set，RS）

RS是新一代RC，提供同样的高可用能力，区别主要在于RS后来居上，能支持更多种类的匹配模式。副本集对象一般不单独使用，而是作为Deployment的理想状态参数使用。

### 部署(Deployment)

部署表示用户对K8s集群的一次更新操作。部署是一个比RS应用模式更广的API对象，可以是创建一个新的服务，更新一个新的服务，也可以是滚动升级一个服务。滚动升级一个服务，实际是创建一个新的RS，然后逐渐将新RS中副本数增加到理想状态，将旧RS中的副本数减小到0的复合操作。以K8s的发展方向，未来对所有长期伺服型的的业务的管理，都会通过Deployment来管理。

### 服务（Service）

RC、RS和Deployment只是保证了支撑服务的微服务Pod的数量，但是没有解决如何访问这些服务的问题。一个Pod只是一个运行服务的实例，随时可能在一个节点上停止，在另一个节点以一个新的IP启动一个新的Pod，因此不能以确定的IP和端口号提供服务。要稳定地提供服务需要服务发现和负载均衡能力。服务发现完成的工作，是针对客户端访问的服务，找到对应的的后端服务实例。在K8s集群中，客户端需要访问的服务就是Service对象。每个Service会对应一个集群内部有效的虚拟IP，集群内部通过虚拟IP访问一个服务。在K8s集群中微服务的负载均衡是由Kube-proxy实现的。Kube-proxy是K8s集群内部的负载均衡器。它是一个分布式代理服务器，在K8s的每个节点上都有一个；这一设计体现了它的伸缩性优势，需要访问服务的节点越多，提供负载均衡能力的Kube-proxy就越多，高可用节点也随之增多。

### 任务（Job）

Job是K8s用来控制批处理型任务的API对象。批处理业务与长期伺服业务的主要区别是批处理业务的运行有头有尾，而长期伺服业务在用户不停止的情况下永远运行。Job管理的Pod根据用户的设置把任务成功完成就自动退出了。成功完成的标志根据不同的spec.completions策略而不同：单Pod型任务有一个Pod成功就标志完成；定数成功型任务保证有N个任务全部成功；工作队列型任务根据应用确认的全局成功而标志成功。

### 后台支撑服务集（DaemonSet）

长期伺服型和批处理型服务的核心在业务应用，可能有些节点运行多个同类业务的Pod，有些节点上又没有这类Pod运行；而后台支撑型服务的核心关注点在K8s集群中的节点（物理机或虚拟机），要保证每个节点上都有一个此类Pod运行。节点可能是所有集群节点也可能是通过nodeSelector选定的一些特定节点。典型的后台支撑型服务包括，存储，日志和监控等在每个节点上支持K8s集群运行的服务。

### 有状态服务集（PetSet）

K8s在1.3版本里发布了Alpha版的PetSet功能。在云原生应用的体系里，有下面两组近义词；第一组是无状态（stateless）、牲畜（cattle）、无名（nameless）、可丢弃（disposable）；第二组是有状态（stateful）、宠物（pet）、有名（having name）、不可丢弃（non-disposable）。RC和RS主要是控制提供无状态服务的，其所控制的Pod的名字是随机设置的，一个Pod出故障了就被丢弃掉，在另一个地方重启一个新的Pod，名字变了、名字和启动在哪儿都不重要，重要的只是Pod总数；而PetSet是用来控制有状态服务，PetSet中的每个Pod的名字都是事先确定的，不能更改。PetSet中Pod的名字的作用，并不是《千与千寻》的人性原因，而是关联与该Pod对应的状态。

对于RC和RS中的Pod，一般不挂载存储或者挂载共享存储，保存的是所有Pod共享的状态，Pod像牲畜一样没有分别（这似乎也确实意味着失去了人性特征）；对于PetSet中的Pod，每个Pod挂载自己独立的存储，如果一个Pod出现故障，从其他节点启动一个同样名字的Pod，要挂载上原来Pod的存储继续以它的状态提供服务。

适合于PetSet的业务包括数据库服务MySQL和PostgreSQL，集群化管理服务Zookeeper、etcd等有状态服务。PetSet的另一种典型应用场景是作为一种比普通容器更稳定可靠的模拟虚拟机的机制。传统的虚拟机正是一种有状态的宠物，运维人员需要不断地维护它，容器刚开始流行时，我们用容器来模拟虚拟机使用，所有状态都保存在容器里，而这已被证明是非常不安全、不可靠的。使用PetSet，Pod仍然可以通过漂移到不同节点提供高可用，而存储也可以通过外挂的存储来提供高可靠性，PetSet做的只是将确定的Pod与确定的存储关联起来保证状态的连续性。PetSet还只在Alpha阶段，后面的设计如何演变，我们还要继续观察。

### 集群联邦（Federation）

K8s在1.3版本里发布了beta版的Federation功能。在云计算环境中，服务的作用距离范围从近到远一般可以有：同主机（Host，Node）、跨主机同可用区（Available Zone）、跨可用区同地区（Region）、跨地区同服务商（Cloud Service Provider）、跨云平台。K8s的设计定位是单一集群在同一个地域内（单个Region内部），因为同一个地区的网络性能才能满足K8s的调度和计算存储连接要求。而联合集群服务就是为提供跨Region跨服务商K8s集群服务而设计的。

每个K8s Federation有自己的分布式存储、API Server和Controller Manager。用户可以通过Federation的API Server注册该Federation的成员K8s Cluster。当用户通过Federation的API Server创建、更改API对象时，Federation API Server会在自己所有注册的子K8s Cluster都创建一份对应的API对象。在提供业务请求服务时，K8s Federation会先在自己的各个子Cluster之间做负载均衡，而对于发送到某个具体K8s Cluster的业务请求，会依照这个K8s Cluster独立提供服务时一样的调度模式去做K8s Cluster内部的负载均衡。而Cluster之间的负载均衡是通过域名服务的负载均衡来实现的。

所有的设计都尽量不影响K8s Cluster现有的工作机制，这样对于每个子K8s集群来说，并不需要更外层的有一个K8s Federation，也就是意味着所有现有的K8s代码和机制不需要因为Federation功能有任何变化。

### 存储卷（Volume）

K8s集群中的存储卷跟Docker的存储卷有些类似，只不过Docker的存储卷作用范围为一个容器，而K8s的存储卷的生命周期和作用范围是一个Pod。每个Pod中声明的存储卷由Pod中的所有容器共享。K8s支持非常多的存储卷类型，特别的，支持多种公有云平台的存储，包括AWS，Google和Azure云；支持多种分布式存储包括GlusterFS和Ceph；也支持较容易使用的主机本地目录hostPath和NFS。K8s还支持使用Persistent Volume Claim即PVC这种逻辑存储，使用这种存储，使得存储的使用者可以忽略后台的实际存储技术（例如AWS，Google或GlusterFS和Ceph），而将有关存储实际技术的配置交给存储管理员通过Persistent Volume来配置。

### 持久存储卷（Persistent Volume，PV）和持久存储卷声明（Persistent Volume Claim，PVC）

PV和PVC使得K8s集群具备了存储的逻辑抽象能力，使得在配置Pod的逻辑里可以忽略对实际后台存储技术的配置，而把这项配置的工作交给PV的配置者，即集群的管理者。存储的PV和PVC的这种关系，跟计算的Node和Pod的关系是非常类似的；PV和Node是资源的提供者，根据集群的基础设施变化而变化，由K8s集群管理员配置；而PVC和Pod是资源的使用者，根据业务服务的需求变化而变化，有K8s集群的使用者即服务的管理员来配置。

### 节点（Node）

K8s集群中的计算能力由Node提供，最初Node称为服务节点Minion，后来改名为Node。K8s集群中的Node也就等同于Mesos集群中的Slave节点，是所有Pod运行所在的工作主机，可以是物理机也可以是虚拟机。不论是物理机还是虚拟机，工作主机的统一特征是上面要运行kubelet管理节点上运行的容器。

### 密钥对象（Secret）

Secret是用来保存和传递密码、密钥、认证凭证这些敏感信息的对象。使用Secret的好处是可以避免把敏感信息明文写在配置文件里。在K8s集群中配置和使用服务不可避免的要用到各种敏感信息实现登录、认证等功能，例如访问AWS存储的用户名密码。为了避免将类似的敏感信息明文写在所有需要使用的配置文件中，可以将这些信息存入一个Secret对象，而在配置文件中通过Secret对象引用这些敏感信息。这种方式的好处包括：意图明确，避免重复，减少暴漏机会。

### 用户帐户（User Account）和服务帐户（Service Account）

顾名思义，用户帐户为人提供账户标识，而服务账户为计算机进程和K8s集群中运行的Pod提供账户标识。用户帐户和服务帐户的一个区别是作用范围；用户帐户对应的是人的身份，人的身份与服务的namespace无关，所以用户账户是跨namespace的；而服务帐户对应的是一个运行中程序的身份，与特定namespace是相关的。

### 名字空间（Namespace）

名字空间为K8s集群提供虚拟的隔离作用，K8s集群初始有两个名字空间，分别是默认名字空间default和系统名字空间kube-system，除此以外，管理员可以可以创建新的名字空间满足需要。

### RBAC访问授权

K8s在1.3版本中发布了alpha版的基于角色的访问控制（Role-based Access Control，RBAC）的授权模式。相对于基于属性的访问控制（Attribute-based Access Control，ABAC），RBAC主要是引入了角色（Role）和角色绑定（RoleBinding）的抽象概念。在ABAC中，K8s集群中的访问策略只能跟用户直接关联；而在RBAC中，访问策略可以跟某个角色关联，具体的用户在跟一个或多个角色相关联。显然，RBAC像其他新功能一样，每次引入新功能，都会引入新的API对象，从而引入新的概念抽象，而这一新的概念抽象一定会使集群服务管理和使用更容易扩展和重用。

# 基于Docker本地运行Kubernetes

# 确认HOST支持memory and swap accounting；

# 安装etcd；

docker run --net=host -d gcr.io/google\_containers/etcd:2.0.12 /usr/local/bin/etcd --addr=127.0.0.1:4001 --bind-addr=0.0.0.0:4001 --data-dir=/var/etcd/data

# 运行master

docker run \

--volume=/:/rootfs:ro \

--volume=/sys:/sys:ro \

--volume=/dev:/dev \

--volume=/var/lib/docker/:/var/lib/docker:ro \

--volume=/var/lib/kubelet/:/var/lib/kubelet:rw \

--volume=/var/run:/var/run:rw \

--net=host \

--pid=host \

--privileged=true \

-d \

daocloud.io/power5/hyperkube:v1.0.1 \

/hyperkube kubelet --containerized --hostname-override="127.0.0.1" --address="0.0.0.0" --api-servers=http://localhost:8080 --config=/etc/kubernetes/manifests

# 运行**service proxy**

docker run -d --net=host --privileged gcr.io/google\_containers/hyperkube:v1.0.1 /hyperkube proxy --master=http://127.0.0.1:8080 --v=2

# 下载kubectl工具

<https://storage.googleapis.com/kubernetes-release/release/v1.8.5/bin/linux/amd64/kubectl>

# minikube

minikube用于在一个单节点的快速部署Kubernetes，他的目标主要是为了让开发者快速搭建本地开发环境。

部署时几个需要注意的地方：

1. 是一个独立的二进制文件，下下来就能用；
2. 需要KVM和一个KVM插件docker-machine-driver-kvm2支持

sudo yum install libvirt-daemon-kvm qemu-kvm

sudo usermod -a -G libvirt $(whoami)

newgrp libvirt

curl -LO https://storage.googleapis.com/minikube/releases/latest/docker-machine-driver-kvm2 && chmod +x docker-machine-driver-kvm2 && sudo mv docker-machine-driver-kvm2 /usr/bin/

1. 通过minikube start --vm-driver=kvm2创建Kubernetes环境。

镜像的存放目录：/root/.minikube/cache/iso

# kubeadm

使用kubeadm安装——先装Mastor，在一个一个添加工作节点

## ;配置Master

添加阿里的镜像源

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

[kubernetes]

name=Kubernetes

baseurl=http://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=0

repo\_gpgcheck=0

gpgkey=http://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg

http://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg

EOF

setenforce 0

yum -y install lrzsz docker epel-release python-pip kubelet kubeadm kubectl

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

# 给docker配个代理，连接gcr.io

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

cat <<EOF > /etc/systemd/system/docker.service.d/http\_proxy.conf

[Service]

Environment="HTTP\_PROXY=http://192.168.21.37:1080/"

EOF

sudo systemctl daemon-reload && systemctl show --property=Environment docker

systemctl enable docker && systemctl start docker

# 安装master节点

kubeadm init \

--kubernetes-version=v1.9.8 \

--pod-network-cidr=10.244.0.0/16 \

--apiserver-advertise-address=172.24.10.81

# 完成之后

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run (as a regular user):

mkdir -p $HOME/.kube

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

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

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

http://kubernetes.io/docs/admin/addons/

You can now join any number of machines by running the following on each node

as root:

kubeadm join --token 6a133f.8153aadd07710c09 172.24.10.81:6443

mkdir -p $HOME/.kube

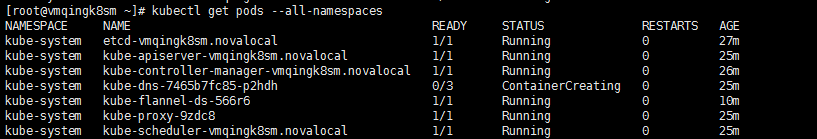
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

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

sudo wget <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

sudo kubectl apply -f kube-flannel.yml

配置完之后PODS的装态，kube-dns依然启动不了：



查看kubelet日志：



# 少启动文件“portmap”，原因是Flannel v0.9.1的镜像少代了这个工具

# 解决<https://github.com/coreos/flannel/issues/890>

# 解除master节点的隔离

kubectl taint nodes --all node-role.kubernetes.io/master-

## 添加node

node要装kubelet和kubeadm，node节点的docker也需要配置代理。

kubeadm join --token 60e3fd.92ce42e9f1bd7d58 172.24.10.81:6443

通过kubeadm命令可以查到这个Token：

kubeadm token list



# 删除Node

kubectl drain <node name> --delete-local-data --force --ignore-daemonsets

kubectl delete node <node name>

kubeadm reset

# 读书笔记

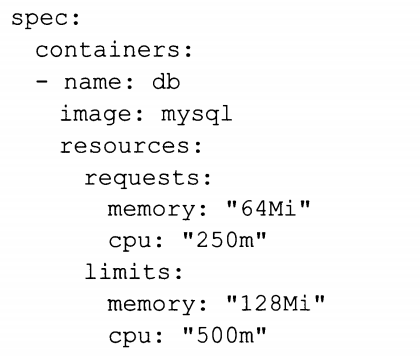
## 知识点

### 资源分配

包括CPU资源和内存，以绝对值来分配。

CPU资源，最小单位是千分之一个CPU；

内存资源，以字节单位分配；



### Label 和 Label Selector

任意资源都可以打上label，Label Selector有如下形式

key = value

key ！= value

key in （value1，value2）

key not in （value1，value2）

### Deployment 扩容

手工扩容：

kubectl scale deployment nginx-deployment --replicas 10

HPA自动扩容：原理是统计1分钟内的CPU利用率的平均值

kubectl autoscale deployment nginx-deployment --min=10 --max=15 --cpu-percent=80

deployment "nginx-deployment" autoscaled

比例扩容：

### 几个IP

Node IP

Pod IP -- 容器的IP，通过docker网桥实现的

Cluster IP – 对应到服务，和端口号（port）构成endpoint

NodeType Port – 当开放了一个NodeType Port之后，所有的Node节点的该端口都会被kube-poxy监听

Port -- service对应的端口号，只能在内部使用

target-port – 容器的端口后，不指定port时，port和他一致

### 使用PV和PVC

以NFS为例：

# 声明PV，NFS的实际空间和storage无关

apiVersion: v1

kind: PersistentVolume

metadata:

name: nfs-pv2

spec:

capacity:

storage: 200Gi

accessModes:

- ReadWriteOnce

nfs:

path: /minio2

server: 172.24.3.172

# 声明PVC，

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: minio-1-pv-claim

labels:

app: minio-1-storage-claim

spec:

accessModes:

- ReadWriteOnce

resources:

# This is the request for storage. Should be available in the cluster.

requests:

storage: 10Gi

# Uncomment and add storageClass specific to your requirements below. Read more https://kubernetes.io/docs/concepts/storage/persistent-volumes/#class-1

#storageClassName:

# 在POD使用声明好的PVC

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: minio-deployment

spec:

strategy:

type: Recreate

template:

metadata:

labels:

app: minio

spec:

# Refer to the PVC created earlier

volumes:

- name: storage

persistentVolumeClaim:

# Name of the PVC created earlier

claimName: minio-1-pv-claim

containers:

- name: minio

# Pulls the default Minio image from Docker Hub

image: minio/minio

args:

- server

- /storage

env:

# Minio access key and secret key

- name: MINIO\_ACCESS\_KEY

value: "minio"

- name: MINIO\_SECRET\_KEY

value: "ruijie123"

ports:

- containerPort: 9000

hostPort: 9000

# Mount the volume into the pod

volumeMounts:

- name: storage # must match the volume name, above

mountPath: "/storage"

关于PV的状态：Available、Bound（已经绑定到PVC）、Released（资源没有回收）、Failed（回收失败）

动态持久卷，PVC不使用storageClassName时表示省缺类型（缺省 StorageClass值，可以用户指定），指定storageClassName=“”表示关闭自动分配。PV的存储类型通过label指定。

规则：

* 已有的 “Available” 的属于该 storageClassName 的 PV 会被判定可以用于对应PVC；
* 没有给 PVC 指定 storageClassName，会使用缺省的 StorageClass。现有的没有指定缺省 Storage Class 标签 "Available" 状态的 PV，不会分配给 PVC。
* 如果 PVC 中给 storageClassName 赋值为空字符串（""），就不会使用 Storage Class（就是说，动态卷供给功能无效）。现有的 “Available” PV（没有指定 storageClassName）会被判定为可以帮到到这个 PVC。
* 当一个 PVC 释放的时候，对应的动态供给卷就会从存储供应者这里进行删除（有点像Nova用完系统盘，请求Cinder删除Volume）。<https://kubernetes.io/docs/user-guide/persistent-volumes/#reclaim-policy>

Persistent Volume Claim Protection：

active状态的PVC不能删除，pod 状态为pending、Running、或者assigned to a node是PVC的状态都是active的。

Expanding Persistent Volumes Claims：

部分PV的后端支持扩容特性，但是该扩容不能热生效，并且只能支持XFS、Ext3\4几种文件系统。

扩容可以通过编辑PVC来触发。需要重启deploy或者Pod才能生效。

Volume Mode：

默认情形下，volume plugin在PV创建时会在PV上创建文件系统。但是1.9之后通过volumeMode可以指定该PV是文件系统，还是Block。

## 关于Volume

由于在k8s中Volume支持多种介质的存储，因此本质上来看并非不能作为持久化存储（虽然生命周期有点短，但是数据是可以持久保存的。）

关于Volume的类型可以参考：<https://kubernetes.io/docs/concepts/storage/volumes/>

当前k8s有对接以下几家商业Volume解决方案：

portworx：提供分布式块存储层；

商业存储系统：quobyte、scaleIO、storageOS、vsphereVolume；

开源系统：Ceph（rbd、Cephfs）、OpenShift（downwardAPI）、glusterfs；

第三方插件：flocker；

emptyDir方式的存储介质，默认情形下是存储在/var/lib/kubelet目录下。emptyDir.medium通过参数可以指定成内存（Memory）。

FlexVolume plugin：

参考：

<https://github.com/kubernetes/community/blob/master/contributors/devel/flexvolume.md>

Mount propagation：

1.8的alpha特性，有可能被重新设计，或者被移除。[Linux kernel documentation](https://www.kernel.org/doc/Documentation/filesystems/sharedsubtree.txt)关于Mount传播特性的描述。

Resource：

当前emptyDir，hostPath均没有配额控制，后续会开发？？？

Sub-path：

apiVersion: v1

kind: Pod

metadata:

name: my-lamp-site

spec:

containers:

- name: mysql

image: mysql

env:

- name: MYSQL\_ROOT\_PASSWORD

value: "rootpasswd"

volumeMounts:

- mountPath: /var/lib/mysql

name: site-data

subPath: mysql

- name: php

image: php:7.0-apache

volumeMounts:

- mountPath: /var/www/html

name: site-data

subPath: html

volumes:

- name: site-data

persistentVolumeClaim:

claimName: my-lamp-site-data

## 关于Labels and Selectors

参考：

<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors>

## 关于命名空间

https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

$ kubectl config set-context $(kubectl config current-context) --namespace=<insert-namespace-name-here>

# Validate it

$ kubectl config view | grep namespace:

## DNS服务

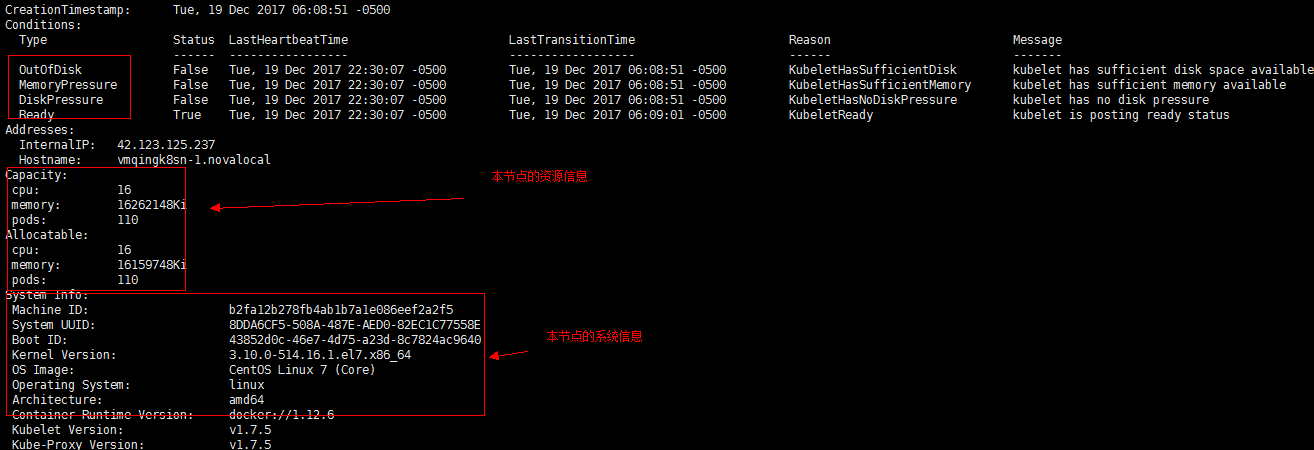
https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/

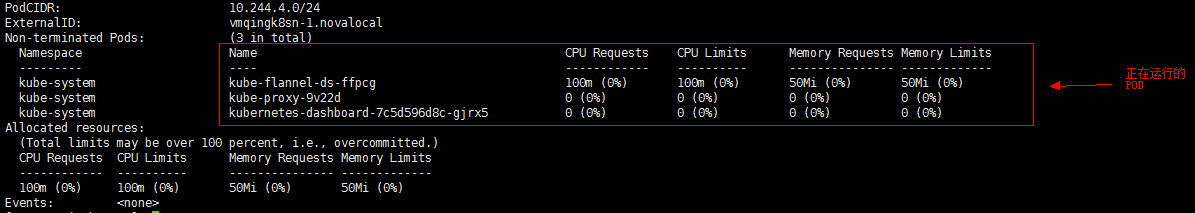
## 常用kubectl命令

# 查看资源的详情

kubectl describe <resource> <res-name> --namespace=<>

# 查看node时可以看到每个节点的内存、CPU、磁盘状态，以及POD信息





# 查看POD时，可以注意EVENT信息。

# 更新deployment

kubectl edit deployment/<deployment\_name>

kubectl rollout status deployment/<deployment\_name>

# 更新历史，注意要用—record记录才能展示

kubectl rollout history deployment/frontend

# 更新到某一个版本

kubectl rollout undo deployment/nginx-deployment --to-revision=2

# 暂停deploy

kubectl rollout pause deployment/nginx-deployment

# 暂停期间继续，deploy继续对外提供功能，更新不会影响deploy

# 恢复暂停的deploy

kubectl rollout resume deploy nginx

## 反向代理

一个代理服务器能够代理外部网络上的主机，访问内部网络时，这种代理服务的方式称为反向代理服务。在K8S中，kube-proxy的作用就是通过帮助外部网络访问节点内部POD ip等，所以可以认为是一个反向代理。