**Dokcer-k8s---概念**

**K8s核心功能：应用部署、访问、Scale Up/Down、RollingUpdate**

**Cluster：**集群，计算、存储网络资源的一个整合，Kubelet利用这些资源和各种各样的内容，都把他基于容器来产生应用

**Master：**是Cluster大脑，职责就是调度，将所有的内容合理的调度分配，保证要有多个安全可靠

**Node：**运行容器，以及容器内的应用，是由master统一调度的，会像master汇报自己的状态,node的运行时间由master说了算

**Pod：**是Kubelet中的最小的工作单元，pod内包含一个或者多个容器，pod中的容器作为一个整体被master调度到一个node上运行，与service是一个概念，把容器放到pod内，然后在运行pod，**pod的特点**：1、可管理性，将所有的数据封锁在一个单元中2、通讯和资源共享，pod的中的所有容器都使用同一个空间，可以直接使用localhost进行通信，容器可以共享所有的存储

**pod模式：** **Pod单一运行容器**：只能将单个容器封装成一个容器，

**Pod多容器**：可以将相同操作需要协调的容器放到同一个pod中，不需要要协调的放到不同的pod中，精密相连，并且直接共享资源

**Controller：**coredns不会直接创建一个pod，通过controller管理pod，controlle中存放了pod部署的特性和属性，Deployment经常使用的创建副本，statefuleset：保证副本的名字不会改变，service：给pod提供一个负载均衡操作，内外部的访问，namespace：完全隔离资源空间

Service：定义外界访问一组特定的pod的方式，服务发现

Namespace：把一个物理集群，逻辑上分成多个集群

Default：默认资源所在namespace

Kube-system：k8s自己创建的系统资源（不要放置其它资源）

**#集群中的概念**

**Kubelet:agent：**所有的框架结构在某个特定的node上运行pod以后，会将pod的配置信息发送给节点中的Kubelet，根据这些信息创建容器，在向master汇报，kubelet是采集客户端中的agent代理

**Kueproxy:balace：**每个node节点上都会运行Kueproxy服务它负责将service的所发来的tcp或udp的数据流转发到后端的容器上，有多个副本的话负Kueproxy就会产生负载均衡

**Pod网络：**pod可以相互通信，flannel就是主要的应用

**#Master运行的内容组件：**

Apl server：接口服务器，管理接口

Scheduler：形成

Contriller manager：用来管理Contriller

Kubelet：用来接收客户端kubelet发送来的消息

Kube-proxy：让数据可以访问到service内

Flannel：协调访问，master与node节点都启动

Coredns：dns服务

Etcd：键值对的数据库

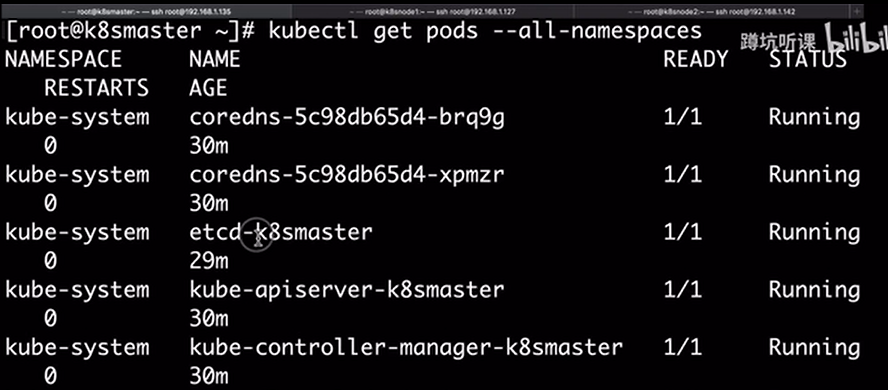
Pause：快速挂起容器

**#Node节点运行的组件：**

Kubelet、Kube-proxy

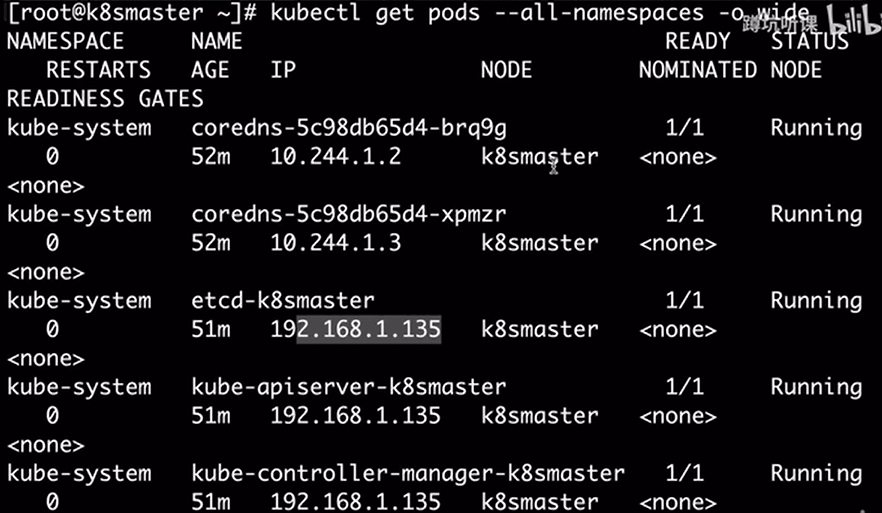
**#查看coredns的所有组件**

kubectl get pods –add-namespaces



**#查看kubectl的详细信息**

kubectl get pods --add-namespaces -o wide

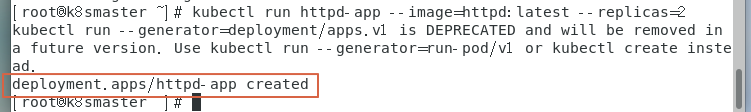


**三、Dokcer-k8s---组件理解**

**#小实验理解kubectl的组件是怎么协同工作的**

**#部署一个组件**

kubectl run httpd-app --image=httpd:latest --replicas=2



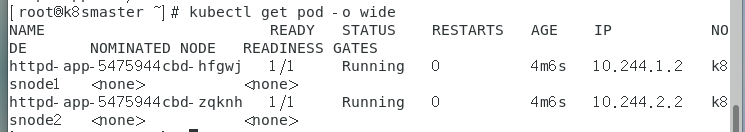
**#查看**deployments

kubectl get deployments.



**#查看副本在哪里运行**pod

kubectl get pod -o wide



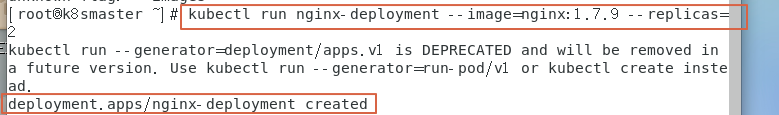
**#创建的过程**

1. kubectl发送部署请求给API server提供http/https的完整访问
2. API server 🡪controller-manager进行创建资源管理各种资源确保处于期望状态以及生命周期🡪创建一个deployments资源
3. Etcd：保存所有得信息
4. 通过kube-scheduler组件，将pod的两个副本给node1一个，node2一个
5. node1上运行分别运行pod的副本，node2上运行分别运行pod的副本
6. node1 kubelet：当作采集器，将自身得配置信息提交给scheduler
7. node1 kude-proxy：把service得数据转发到pod得后端容器
8. flannel会个每一个容器分配一个IP地址

**四、Deployment：命令部署**

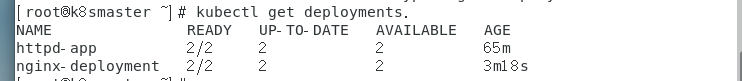
**#运行nginx的容器副本，如果本地没有nginx的镜像时会自己去拉取镜像**

kubectl run nginx-deployment --images=nginx --replicas=2



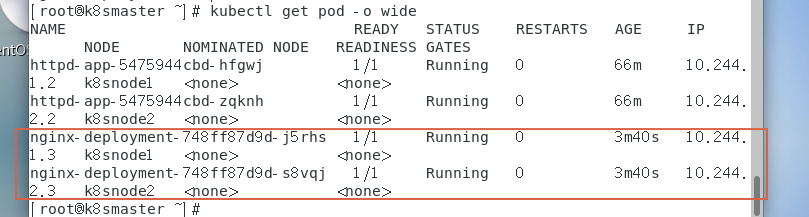
**#查看副本**

kubectl get deployment



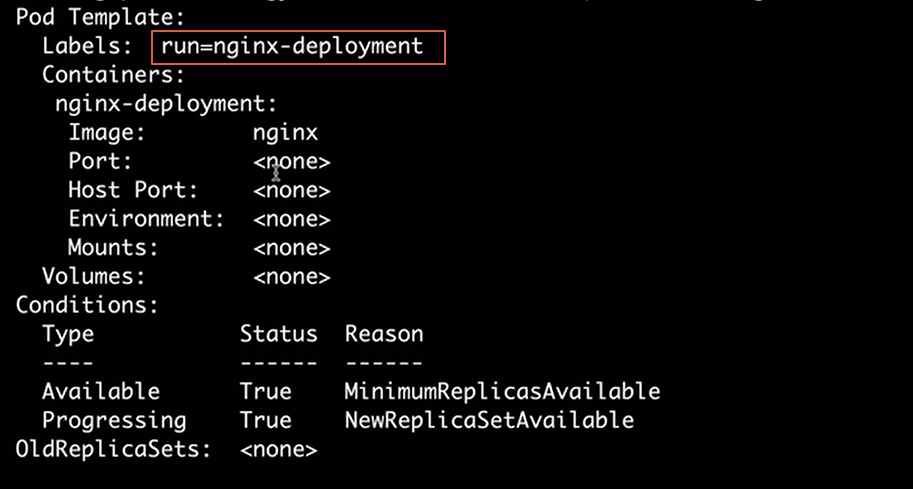
**#查看nginx单个副本的运行状态**

kubectl get feployment nginx-deployment



**#查看单个容器的详细的信息**

kubectl describe nginx-deployments.

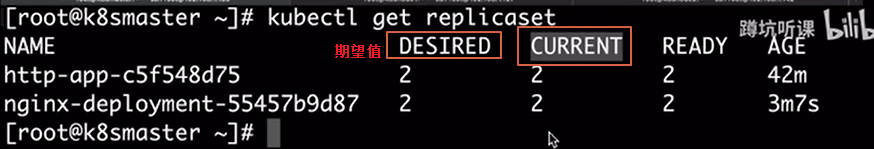


**#查看replicasets.的详细信息**

kubectl describe replicasets.

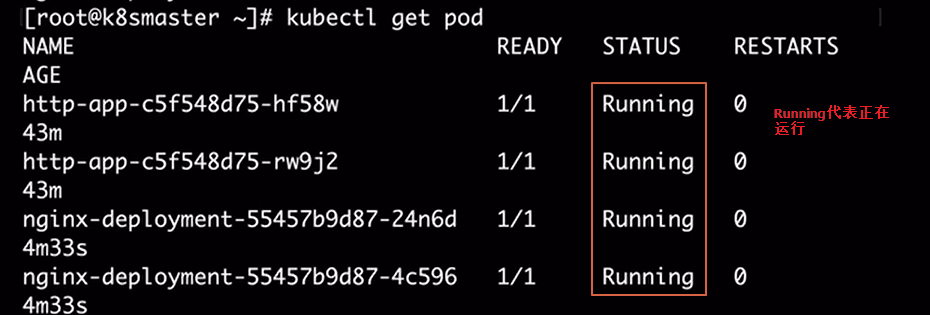
**#查看replicasets的作用简短显示**

kubectl get replicaset



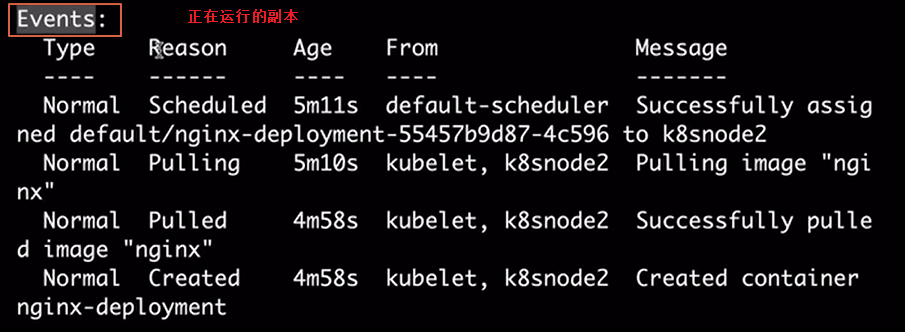
**#查看正在运行的容器副本**

kubectl get pod



**#查看pod的详细信息，Events记录了pod 创建过程的步骤**

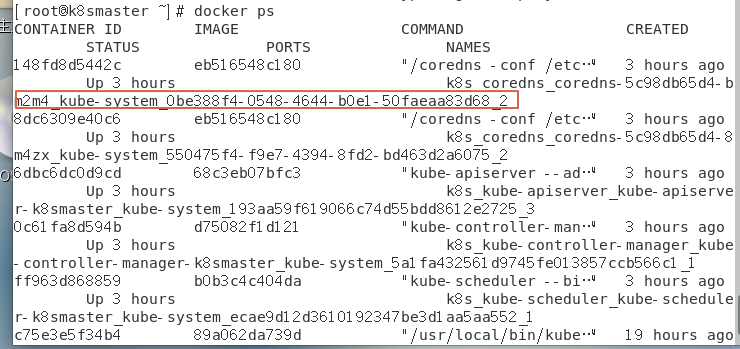
kubectl describe pod





**#镜像的名字组成**

NAME:[Deployment-name]-[ReplicaSet-code]-[Container-code]



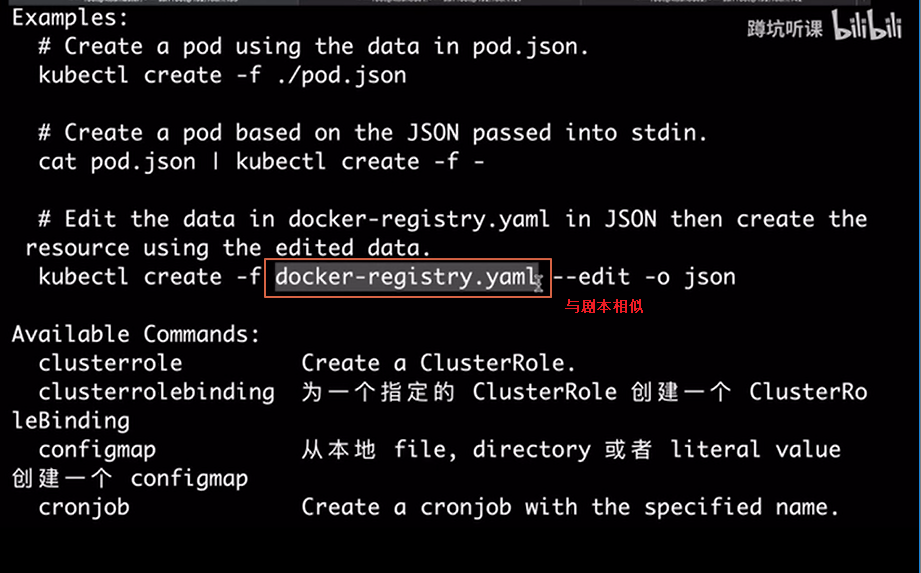
**五、deployment脚本创建**

**Kubectl run 创建容器副本时有一个让替代的命令，Kubectl run会被替换，**

**kubectl apply特点：可以对资源进行调整、整合**

**#使用替换的命令创建,发现是不能创建的，会给一大堆的提示**

Kubectl create http-test --images=httpd --replicas=2



**#创建yml文件部署nginx**

vim nginx.yml

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 2

template:

metadata:

labels:

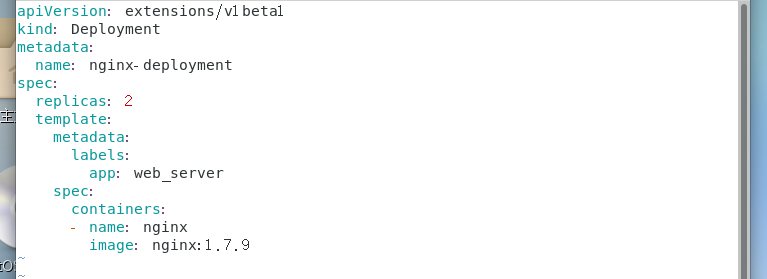
app: web\_server

spec:

containers:

- name: nginx

image: nginx:1.7.9



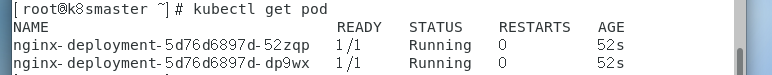
**#写完后执行创建**

kubectl apply -f nginx.yml



**#查看是否创建**

kubectl get pod



**六、yaml简单解读**

vim nginx.yml

apiVersion: extensions/v1beta1 //固定的版本号，版本可以改变

kind: Deployment //创建的类型

metadata: //该资源的元数据

name: nginx-deployment2 //元数据中的项，给元数据起名

spec: // Deployment的说明书，规格

replicas: 2 //当前的期望副本数量2个，不写默认是1个

template: //定义pod的模板，

metadata: //定义pod的元数据，最少要定义一个labels

labels: //用来控制pod所运行的容器在哪一个节点上和不要在那个节点上

app: web\_server //定义好的labels

spec: //pod的规格pod中的每个容器都根据这个pod来运行

containers: //定义必要的name和images

* name: nginx //名字

image: nginx //镜像名

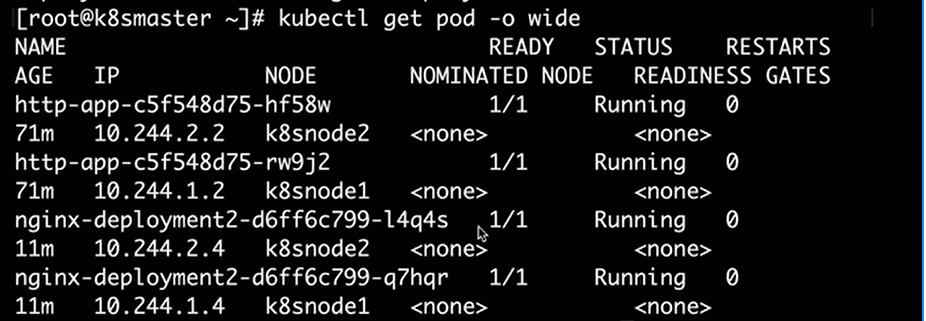
**#卸载run命令创建的容器文件**

kubectl delete deployment nginx-deployment



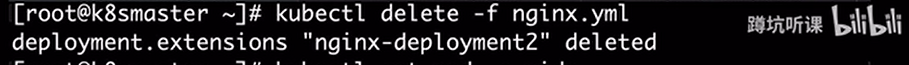
**#查看是否卸载**

kudectl get pod -o wide



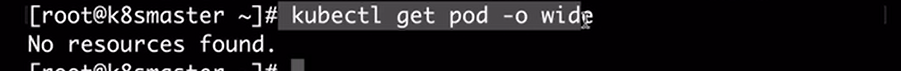
**#卸载使用yml文件创建的容器**

Kubectl delete -f nginx.yml



**#查看是否卸载,**

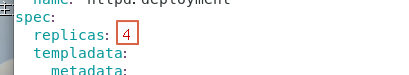
kudectl get pod -o wide



**小实验：部署使用脚本httpd**

vim nginx.yml

**#弹性伸缩只要改脚本里面要创建的数量就行**



**#让k8smaster上也能分配pod容器（默认是不分配的）**

kubectl taint node k8smaster node-role.kubernetes.io/master-

**#不让k8s分配任务**

kubectl taint node k8smaster node-role.kubernetes.io/master="":NoSchedule

**七、failover功能**

**#部署理解**

kubectl apply -f nginx.yml

**#查看创建的容器位置**

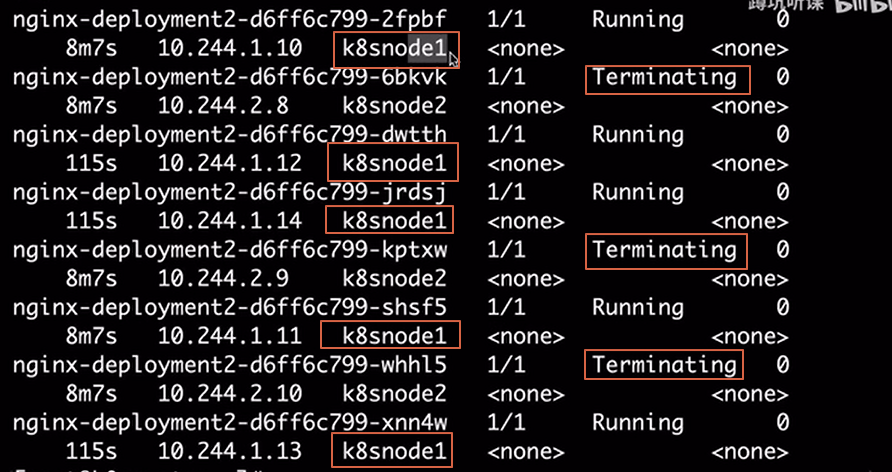
kubectl get pod -o wide

**#将node2强制关机**

**当node2宕机后会将有期望值的容器进行迁移到node1上，而node2节点上的容器会关闭不可用，在将node2开启后容器是不会在迁移回去的，在node2节点上的容器会彻底的关闭消失，再次开启后的节点会自动加入到集群中**

**#再次查看创建的容器位置，**

kubectl get pod -o wide

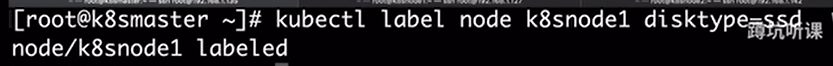


**八、用label控制pod的位置**

**根据容器的需求不同通过label来控制pod的硬件资源的合理分配，=是运算符，==是真的=**

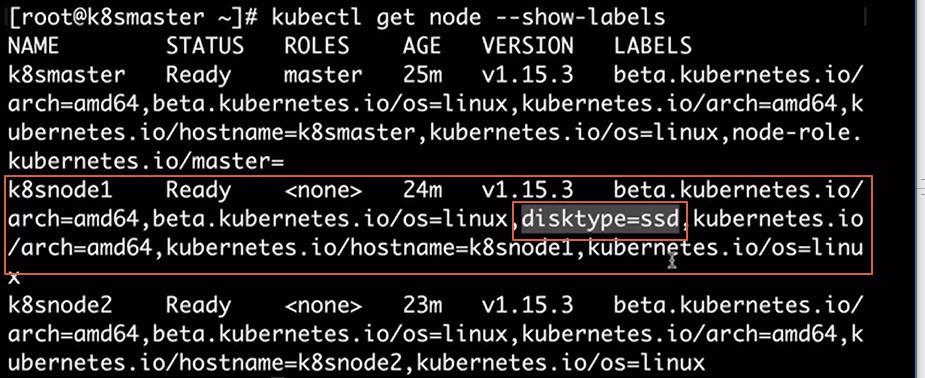
**#在node1节点上使用label打标签，硬盘位ssd**

kubectl label node k8snode1 disktype=ssd



**#查看节点是否有标签**

kubectl get node --show-labels



#将有需求的pod部署到ssd的节点上

vim nginx.yml

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name:nginx-deployment

spec:

replicas: 6

template:

metadata:

labels:

app: web\_server

spec:

containers:

* name: nginx

image: nginx

nodeSelector:

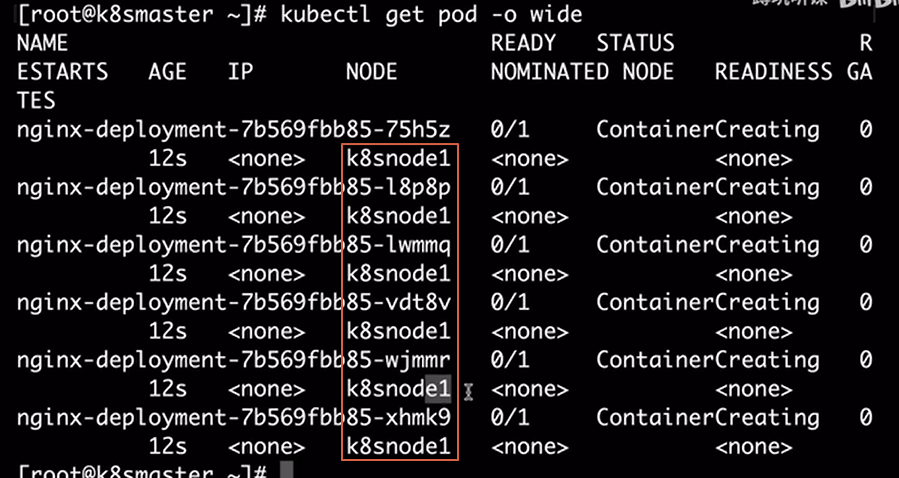
disktypeL ssd //指定标签

**#部署节点**

kubectl apply -f nginx.yml

**#查看部署的容器，都在node1上**

kubectl get pod -o wide



**#删除节点的标签，删除标签后部署好的容器不会迁移，也不会删除**

Kubectl label node k8snode1 disktype-

