docker编排工具

（一）

①docker compose：单机编排

②docker swarm：面向集群，将多个docker host整合为同一资源池，随后compoase只需要面对swarm整合出的资源池进行编排

③docker machine：初始化、预处理工具：将一个主机迅速初始化，使之能够加入swarm

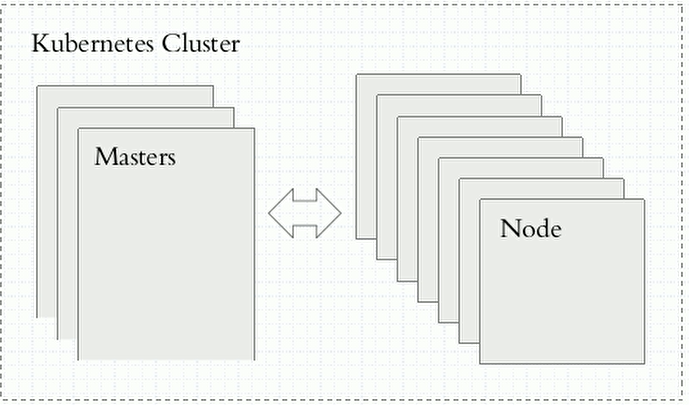
1. mesos，marathon
2. kubernates



Kubernetes

特点：自动装箱、自我修复、水平扩展、服务发现和负载均衡、自动发布和回滚、密钥和配置管理、存储编排、任务的批量处理执行

一、Kubernetes基础架构



三个master节点（高可用）； 多个Node节点：运行容器的工作节点

创建启动容器的请求都发给master 通过API Server接受、解析、处理请求；

调度器Scheduler分析各node现有的可用资源状态，找一个最佳适配的node，把容器调度到上面（两级调度，先筛选再优选），node上由kubelet接受请求然后由docker运行；

从镜像仓库拉取镜像（镜像仓库也可以以容器的形式部署在本身的这个k8s集群中，自托管）；

控制器负责监控pod的运行状态，pod挂了就向调度器发请求重建容器；控制器管理器controller-manager负责监控控制器的状态

一个pod内可以跑多个容器，一个pod内的容器共享网络名称空间、存储卷，

一个Pod中只有一个主容器，其他的称为边车容器（如filebeat）

Selector标签选择器，用标签（键值对）来标识Pod（元数据），用于pod的分类和管理

1. 控制器

无状态应用控制器：Deployment + HPA（对workload副本数进行自动水平扩缩容）

有状态应用控制器：StatefulSet

每个node上只运行一个副本：DaemonSet

作业：job 定时作业：Cronjob

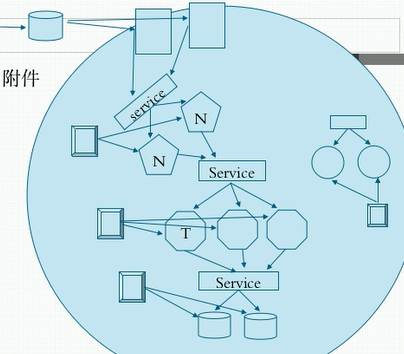
2、AddOns：附加组件

（1）service解析服务：由于pod对应的IP容易变动，因为pod的重启，其IP就会和原来不同，Service 提供了一个稳定访问的机制，提供稳定的IP。 Service 是一组逻辑上关联的 pod，利用selector，将label为app:MyApp的pod聚合在一起，形成个组，我们可以称为Endpoints，这也是一种k8s资源。

所以现在的关系是 Service（Cluster IP）–> Endpoints --> 一组Pod（Pod ip）

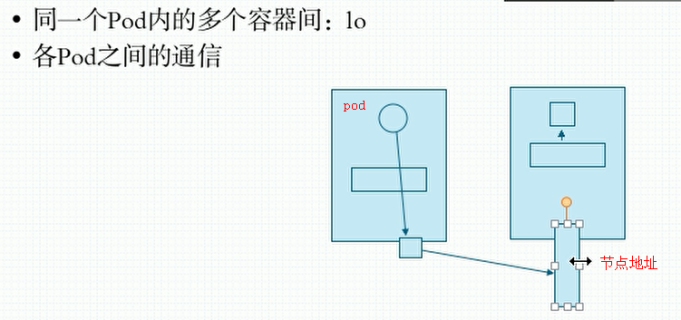
1. NMT

Nginx; M=数据库； Tomcat



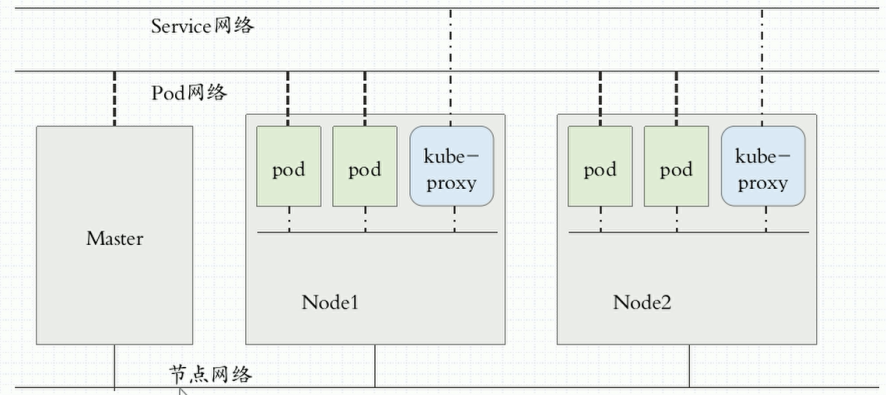
1. 网络

集群网络（service网络） -> pod网络 -> 节点网络



pod间直接通信： 叠加网络（Overlay Network）

service的创建和修改：kube-proxy



容器网络接口CNI：

flannel：网络配置（叠加网络）

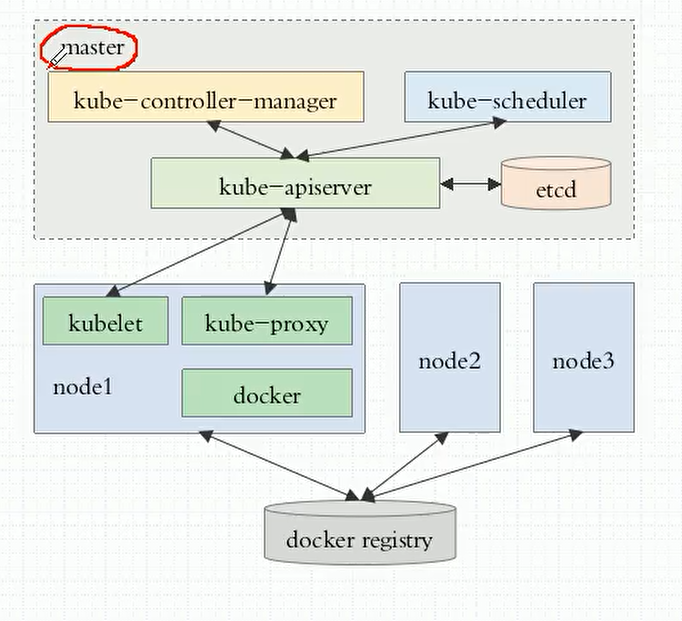
calico：网络配置+网络策略（BGP协议实现网络路由通信、三层隧道网络）

canel：网络配置（叠加网络）+ 网络策略（三层隧道）

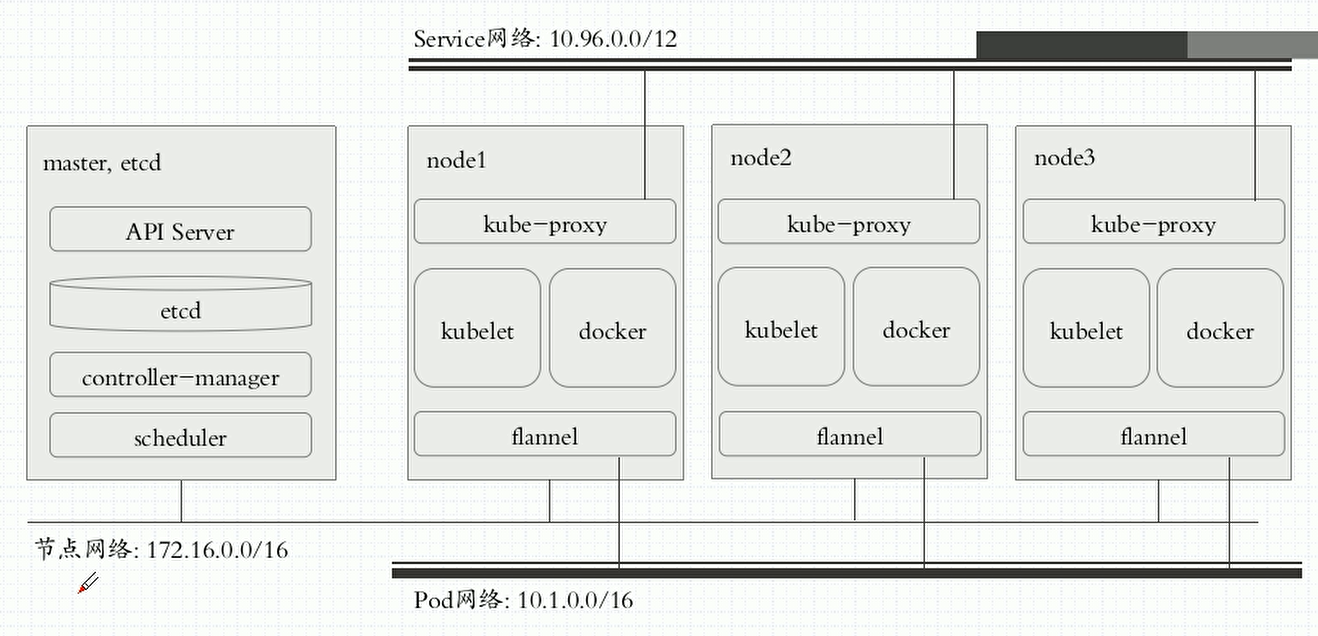
1. 名称空间 namespace :

为pod提供管理的边界，本身不限制pod间的相互访问；不过可以通过iptables来隔离名称空间之间pod的相互访问；

03-初始化Kubernetes集群



部署架构：

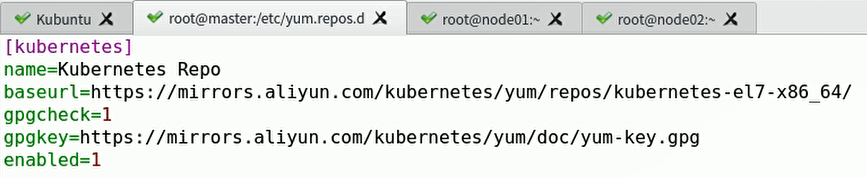


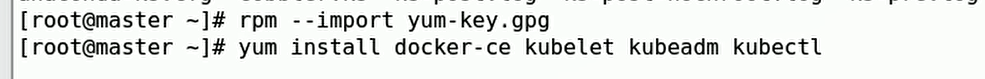
部署方式：

1. 手动部署到物理机节点上、手动创建各种CA认证
2. kubeadm：
   1. 所有节点上装好kubelet、kubeadm和docker并进行初始化
   2. 在作为master的节点上，kubeadm init ：用容器（pod，静态pod）运行api server、scheduler、controller-manager、etcd
   3. 余下的node节点上，kubeadm join： 用容器（pod，托管的动态pod）运行kube-proxy
   4. （所有节点都需要部署flannel）

部署：

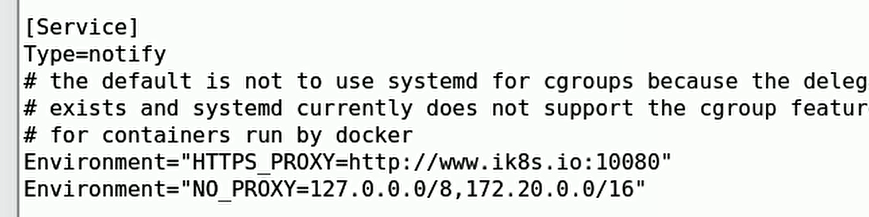
1、准备镜像

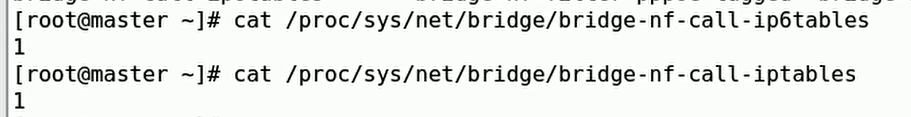






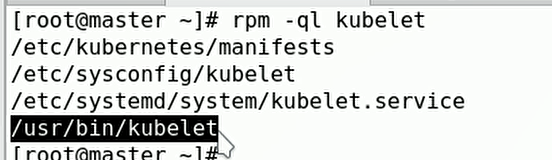
2、启动docker，做好环境设定





1. 设定kubelet

启动项：



设定kubelet配置忽略SWAP报错信息

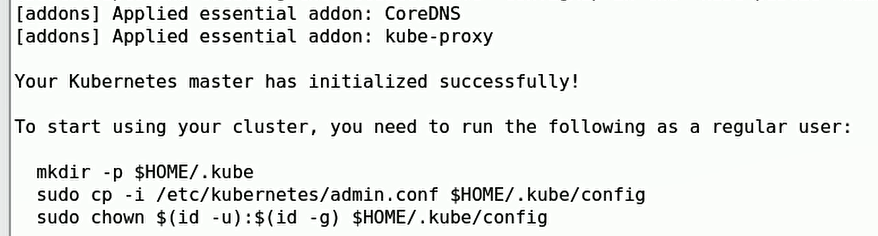
#vim /etc/sysconfig/kubelet

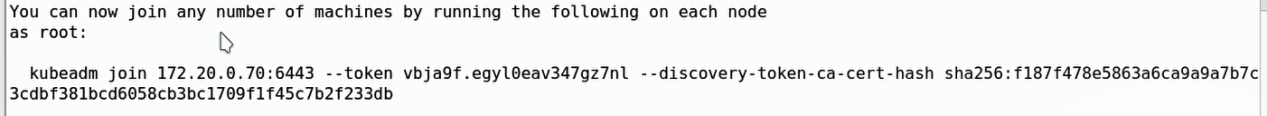
KUBELET\_EXTRA\_ARGS=”--fail-swap-on=false”

启动： systemctl start kubelet

1. kubeadm init

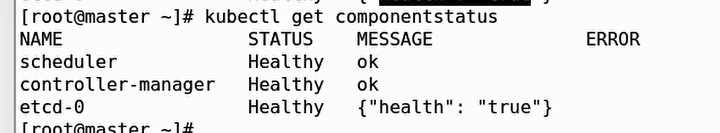






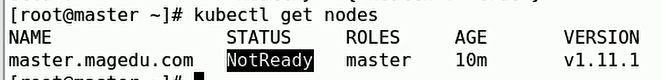
保存好tocken 和ca-cert-hash

查看组件状态： kubectl get component status (kubectl get cs)



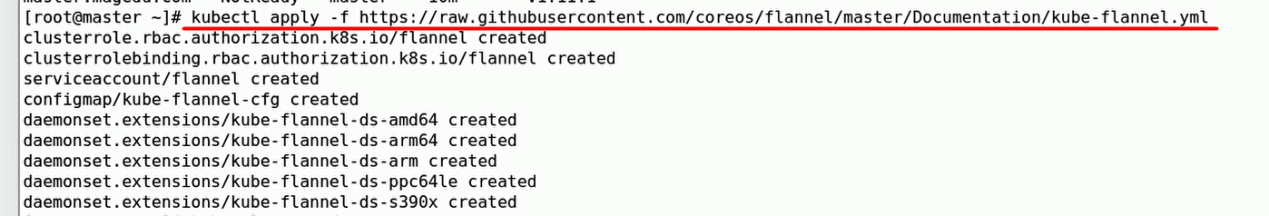
查看集群节点信息 kubectl get nodes

如果没装好flannel或其他网络组件的话 节点会显示未就绪状态

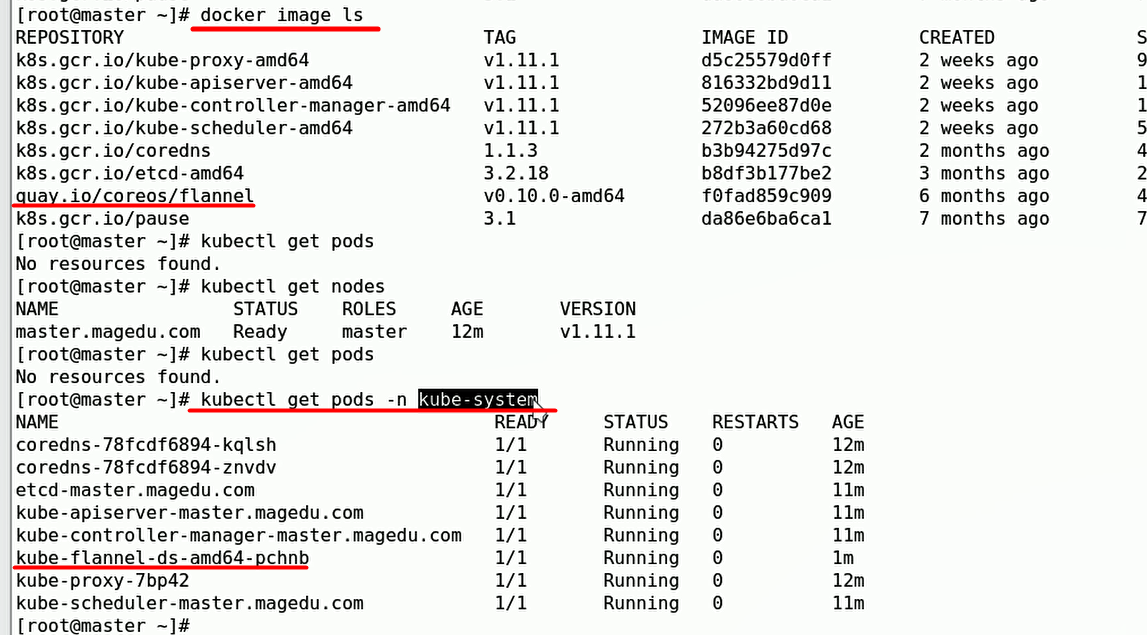


1. flannel

自动获取清单并下载镜像：



docker image ls 能看到flannel镜像后

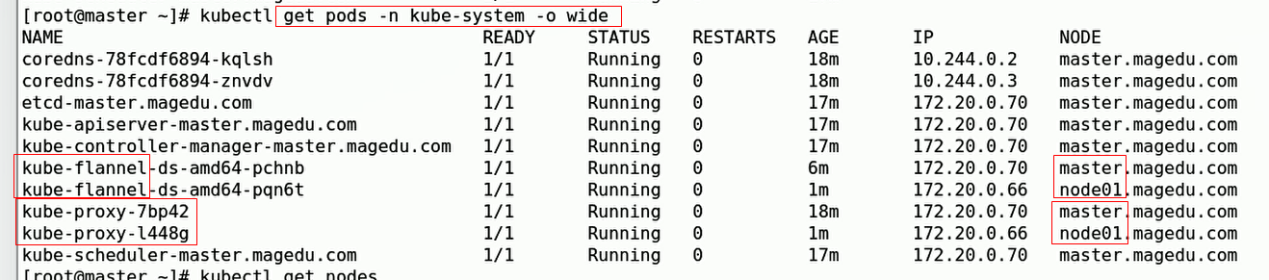


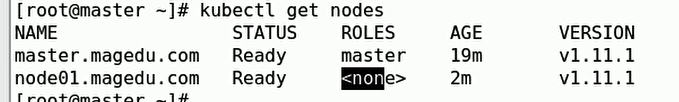
1. node节点加入集群

先用和master相同的配置启动docker kubelet

kubeadm join:





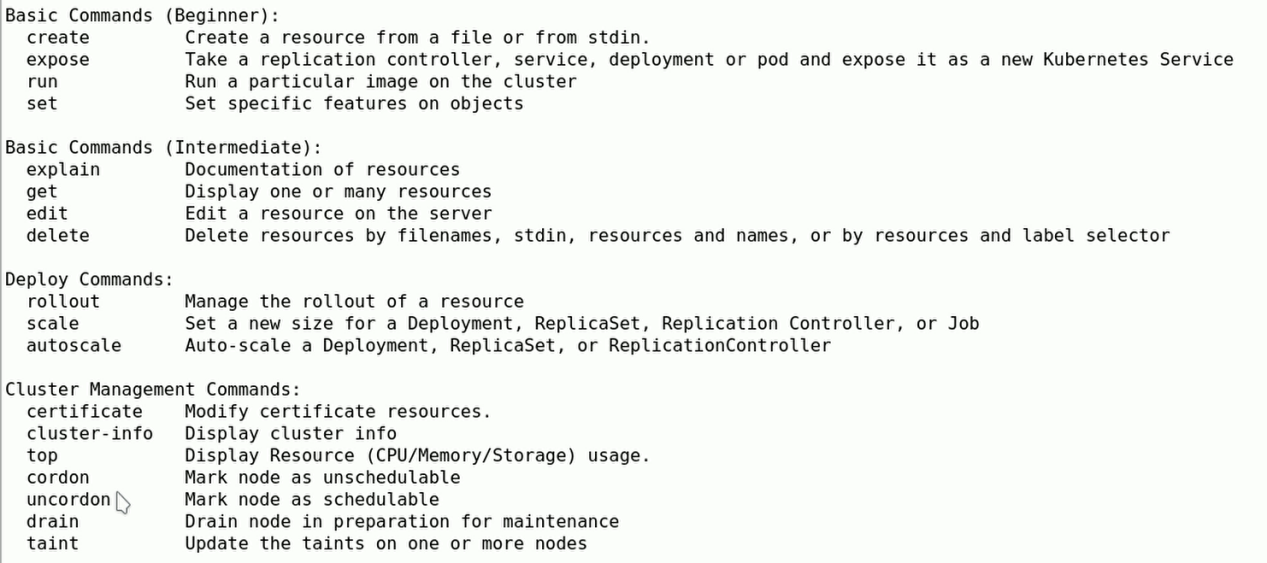


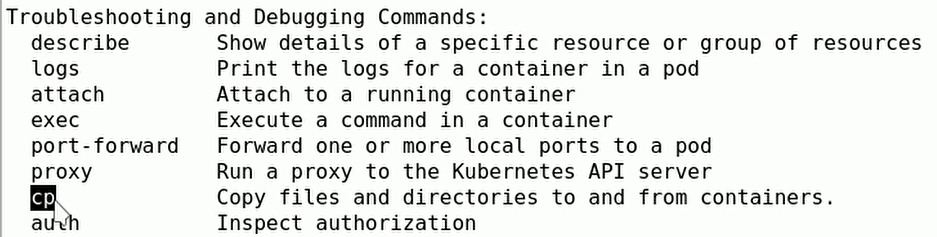
04、应用入门

1、kubctl

k8s集群管理入口的客户端，连接api server实现对象资源的增删改查（对象包括pod,service,replicaset,deployment,statefulset,daemonset,job,cronjob,node等）

kubectl后面可带的参数：





### **查看所有 pod 列表**

查看所有的pod 列表，-n 后跟 namespace， 查看指定的命名空间

kubectl get pod  
kubectl get pod -n kube-system #查看指定的命名空间的pod  
​  
kubectl get pod -o wide #查看更详细的信息，比如pod 所在的节点  
kubectl get pod --show-labels  #获取pod 并查看pod 的标签

### **查看 RC 和 service 列表**

# 查看 RC 和 service 列表， -o wide 查看详细信息

kubectl get rc,svc  
kubectl get pod,svc -o wide  
kubectl get pod <pod-name> -o yaml

### **显示 Node 的详细信息**

kubectl describe node 192.168.0.212 #可以跟Node 或者IP或者主机名

### **显示 Pod 的详细信息, 特别是查看 pod 无法创建的时候的日志**

kubectl describe pod <pod-name>

### **根据 yaml 创建资源, apply 可以重复执行，create 不行**

kubectl create -f pod.yaml  
kubectl apply -f pod.yaml

### **基于 pod.yaml 定义的名称删除指定资源**

kubectl delete -f pod.yaml

### **删除所有包含某个 label 的pod 和 service**

kubectl delete pod,svc -l name=<label-name>

### **删除默认命名空间下的所有 Pod**

kubectl delete pod --all

### **执行 pod 命令**

kubectl exec <pod-name> -- date  
kubectl exec <pod-name> -- bash  
kubectl exec <pod-name> -- ping 10.24.51.9

### **通过bash获得 pod 中某个容器的TTY，相当于登录容器**

kubectl exec -it <pod-name> -c <container-name> -- bash  
eg:  
kubectl exec -it redis-master-cln81 -- bash

### **查看容器日志**

kubectl logs <pod-name>  
kubectl logs -f <pod-name> # 实时查看日志  
kubectl log <pod-name>  -c <container\_name> # 若 pod 只有一个容器，可以不加 -c   
​  
kubectl logs -l app=frontend # 返回所有标记为 app=frontend 的 pod 的合并日志。

### **查看节点 labels**

kubectl get node --show-lables

### **重启pod**

kubectl get pod <POD名称> -n <NAMESPACE名称> -o yaml | kubectl replace --force -f -

## **2、命令自动补全**

source <(kubectl completion bash) # 在 bash 中设置当前 shell 的自动补全，要先安装 bash-completion 包。  
echo "source <(kubectl completion bash)" >> ~/.bashrc

## **3、创建命令**

kubectl apply -f ./my-manifest.yaml         # 创建资源  
kubectl apply -f ./my1.yaml -f ./my2.yaml     # 使用多个文件创建  
kubectl apply -f ./dir             # 基于目录下的所有清单文件创建资源  
kubectl apply -f https://git.io/vPieo       # 从 URL 中创建资源  
kubectl create deployment nginx --image=nginx # 启动单实例 nginx  
kubectl explain pods,svc                # 获取 pod 清单的文档说明  
​

# 从标准输入创建多个 YAML 对象  
cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: Pod  
metadata:  
 name: busybox-sleep  
spec:  
 containers:  
  - name: busybox  
   image: busybox  
   args:  
    - sleep  
    - "1000000"  
---  
apiVersion: v1  
kind: Pod  
metadata:  
 name: busybox-sleep-less  
spec:  
 containers:  
  - name: busybox  
   image: busybox  
   args:  
    - sleep  
    - "1000"  
EOF  
​  
# 创建有多个 key 的 Secret  
cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: Secret  
metadata:  
 name: mysecret  
type: Opaque  
data:  
 password: $(echo -n "s33msi4" | base64 -w0)  
 username: $(echo -n "jane" | base64 -w0)  
EOF

## **4、查看和查找资源**

# get 命令的基本输出  
kubectl get services                          # 列出当前命名空间下的所有 services  
kubectl get pods --all-namespaces             # 列出所有命名空间下的全部的 Pods  
kubectl get pods -o wide                      # 列出当前命名空间下的全部 Pods，并显示更详细的信息  
kubectl get deployment my-dep                 # 列出某个特定的 Deployment  
kubectl get pods                              # 列出当前命名空间下的全部 Pods  
kubectl get pod my-pod -o yaml                # 获取一个 pod 的 YAML  
​  
# describe 命令的详细输出  
kubectl describe nodes my-node  
kubectl describe pods my-pod  
​  
# 列出当前名字空间下所有 Services，按名称排序  
kubectl get services --sort-by=.metadata.name  
​  
# 列出 Pods，按重启次数排序  
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'  
​  
# 列举所有 PV 持久卷，按容量排序  
kubectl get pv --sort-by=.spec.capacity.storage  
​  
# 获取包含 app=cassandra 标签的所有 Pods 的 version 标签  
kubectl get pods --selector=app=cassandra -o \  
  jsonpath='{.items[\*].metadata.labels.version}'  
​  
# 获取所有工作节点（使用选择器以排除标签名称为 'node-role.kubernetes.io/master' 的结果）  
kubectl get node --selector='!node-role.kubernetes.io/master'  
​  
# 获取当前命名空间中正在运行的 Pods  
kubectl get pods --field-selector=status.phase=Running  
​  
# 获取全部节点的 ExternalIP 地址  
kubectl get nodes -o jsonpath='{.items[\*].status.addresses[?(@.type=="ExternalIP")].address}'  
​  
# 列出属于某个特定 RC 的 Pods 的名称  
# 在转换对于 jsonpath 过于复杂的场合，"jq" 命令很有用；可以在 https://stedolan.github.io/jq/ 找到它。  
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to\_entries | .[] | "\(.key)=\(.value),"')%?}  
echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.name})  
​  
# 显示所有 Pods 的标签（或任何其他支持标签的 Kubernetes 对象）  
kubectl get pods --show-labels  
​  
# 检查哪些节点处于就绪状态  
JSONPATH='{range .items[\*]}{@.metadata.name}:{range @.status.conditions[\*]}{@.type}={@.status};{end}{end}' \  
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"  
​  
# 列出被一个 Pod 使用的全部 Secret  
kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v null | sort | uniq  
​  
# 列举所有 Pods 中初始化容器的容器 ID（containerID）  
# Helpful when cleaning up stopped containers, while avoiding removal of initContainers.  
kubectl get pods --all-namespaces -o jsonpath='{range .items[\*].status.initContainerStatuses[\*]}{.containerID}{"\n"}{end}' | cut -d/ -f3  
​  
# 列出事件（Events），按时间戳排序  
kubectl get events --sort-by=.metadata.creationTimestamp  
​  
# 比较当前的集群状态和假定某清单被应用之后的集群状态  
kubectl diff -f ./my-manifest.yaml

## **5、更新资源**

kubectl set image deployment/frontend www=image:v2               # 滚动更新 "frontend" Deployment 的 "www" 容器镜像  
kubectl rollout history deployment/frontend                      # 检查 Deployment 的历史记录，包括版本   
kubectl rollout undo deployment/frontend                         # 回滚到上次部署版本  
kubectl rollout undo deployment/frontend --to-revision=2         # 回滚到特定部署版本  
kubectl rollout status -w deployment/frontend                    # 监视 "frontend" Deployment 的滚动升级状态直到完成  
kubectl rollout restart deployment/frontend                      # 轮替重启 "frontend" Deployment  
​  
cat pod.json | kubectl replace -f -                              # 通过传入到标准输入的 JSON 来替换 Pod  
​  
# 强制替换，删除后重建资源。会导致服务不可用。  
kubectl replace --force -f ./pod.json  
​  
# 为多副本的 nginx 创建服务，使用 80 端口提供服务，连接到容器的 8000 端口。  
kubectl expose rc nginx --port=80 --target-port=8000  
# 将某单容器 Pod 的镜像版本（标签）更新到 v4  
kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.\*$/\1:v4/' | kubectl replace -f -  
​  
kubectl label pods my-pod new-label=awesome                      # 添加标签  
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq       # 添加注解  
kubectl autoscale deployment foo --min=2 --max=10                # 对 "foo" Deployment 自动伸缩容

## **6、部分更新资源**

# 部分更新某节点  
kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'   
​  
# 更新容器的镜像；spec.containers[\*].name 是必须的。因为它是一个合并性质的主键。  
kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-hostname","image":"new image"}]}}'  
​  
# 使用带位置数组的 JSON patch 更新容器的镜像  
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/containers/0/image", "value":"new image"}]'  
​  
# 使用带位置数组的 JSON patch 禁用某 Deployment 的 livenessProbe  
kubectl patch deployment valid-deployment  --type json   -p='[{"op": "remove", "path": "/spec/template/spec/containers/0/livenessProbe"}]'  
​  
# 在带位置数组中添加元素   
kubectl patch sa default --type='json' -p='[{"op": "add", "path": "/secrets/1", "value": {"name": "whatever" } }]'

## **7、删除资源**

kubectl delete -f ./pod.json                                      # 删除在 pod.json 中指定的类型和名称的 Pod  
kubectl delete pod,service baz foo                                # 删除名称为 "baz" 和 "foo" 的 Pod 和服务  
kubectl delete pods,services -l name=myLabel                      # 删除包含 name=myLabel 标签的 pods 和服务  
kubectl delete pods,services -l name=myLabel --include-uninitialized      # 删除包含 label name=myLabel 标签的 Pods 和服务  
kubectl -n my-ns delete po,svc --all                              # 删除在 my-ns 名字空间中全部的 Pods 和服务  
# 删除所有与 pattern1 或 pattern2 awk 模式匹配的 Pods  
kubectl get pods  -n mynamespace --no-headers=true | awk '/pattern1|pattern2/{print $1}' | xargs kubectl delete -n mynamespace pod

## **8、pod常用操作**

kubectl logs my-pod                                 # 获取 pod 日志（标准输出）  
kubectl logs -l name=myLabel                        # 获取含 name=myLabel 标签的 Pods 的日志（标准输出）  
kubectl logs my-pod --previous                      # 获取上个容器实例的 pod 日志（标准输出）  
kubectl logs my-pod -c my-container                 # 获取 Pod 容器的日志（标准输出, 多容器场景）  
kubectl logs -l name=myLabel -c my-container        # 获取含 name=myLabel 标签的 Pod 容器日志（标准输出, 多容器场景）  
kubectl logs my-pod -c my-container --previous      # 获取 Pod 中某容器的上个实例的日志（标准输出, 多容器场景）  
kubectl logs -f my-pod                              # 流式输出 Pod 的日志（标准输出）  
kubectl logs -f my-pod -c my-container              # 流式输出 Pod 容器的日志（标准输出, 多容器场景）  
kubectl logs -f -l name=myLabel --all-containers    # 流式输出含 name=myLabel 标签的 Pod 的所有日志（标准输出）  
kubectl run -i --tty busybox --image=busybox -- sh  # 以交互式 Shell 运行 Pod  
kubectl run nginx --image=nginx -n mynamespace      # 在指定名字空间中运行 nginx Pod  
kubectl run nginx --image=nginx --dry-run=client -o yaml > pod.yaml    # 运行 ngins Pod 并将其规约写入到名为 pod.yaml 的文件     
​  
kubectl attach my-pod -i                            # 挂接到一个运行的容器中  
kubectl port-forward my-pod 5000:6000               # 在本地计算机上侦听端口 5000 并转发到 my-pod 上的端口 6000  
kubectl exec my-pod -- ls /                         # 在已有的 Pod 中运行命令（单容器场景）  
kubectl exec my-pod -c my-container -- ls /         # 在已有的 Pod 中运行命令（多容器场景）  
kubectl top pod POD\_NAME --containers               # 显示给定 Pod 和其中容器的监控数据

## **9、节点操作**

kubectl cordon my-node                                                # 标记 my-node 节点为不可调度  
kubectl drain my-node                                                 # 对 my-node 节点进行清空操作，为节点维护做准备  
kubectl uncordon my-node                                              # 标记 my-node 节点为可以调度  
kubectl top node my-node                                              # 显示给定节点的度量值  
kubectl cluster-info                                                  # 显示主控节点和服务的地址  
kubectl cluster-info dump                                             # 将当前集群状态转储到标准输出  
kubectl cluster-info dump --output-directory=/path/to/cluster-state   # 将当前集群状态输出到 /path/to/cluster-state  
​  
# 如果已存在具有指定键和效果的污点，则替换其值为指定值  
kubectl taint nodes foo dedicated=special-user:NoSchedule

## **10、格式化输出**

要以特定格式将详细信息输出到终端窗口，可以将 -o 或 --output 参数添加到支持的 kubectl 命令。

| **输出格式** | **描述** |
| --- | --- |
| -o=custom-columns=<spec> | 使用逗号分隔的自定义列来打印表格 |
| -o=custom-columns-file=<filename> | 使用 <filename> 文件中的自定义列模板打印表格 |
| -o=json | 输出 JSON 格式的 API 对象 |
| -o=jsonpath=<template> | 打印 [jsonpath](https://kubernetes.io/docs/reference/kubectl/jsonpath) 表达式中定义的字段 |
| -o=jsonpath-file=<filename> | 打印在 <filename> 文件中定义的 [jsonpath](https://kubernetes.io/docs/reference/kubectl/jsonpath) 表达式所指定的字段。 |
| -o=name | 仅打印资源名称而不打印其他内容 |
| -o=wide | 以纯文本格式输出额外信息，对于 Pod 来说，输出中包含了节点名称 |
| -o=yaml | 输出 YAML 格式的 API 对象 |

使用 -o=custom-columns 的示例：

# 集群中运行着的所有镜像  
kubectl get pods -A -o=custom-columns='DATA:spec.containers[\*].image'  
​  
# 除 "k8s.gcr.io/coredns:1.6.2" 之外的所有镜像  
kubectl get pods -A -o=custom-columns='DATA:spec.containers[?(@.image!="k8s.gcr.io/coredns:1.6.2")].image'  
​  
# 输出 metadata 下面的所有字段，无论 Pod 名字为何  
kubectl get pods -A -o=custom-columns='DATA:metadata.\*'