# [K8S从入门到放弃系列-(1)环境初始化](https://www.cnblogs.com/tchua/p/10749143.html)

核心概念

**Kubernetes**的核心概念

**1.Pod**

Pod直译是豆荚，可以把容器想像成豆荚里的豆子，把一个或多个关系紧密的豆子包在一

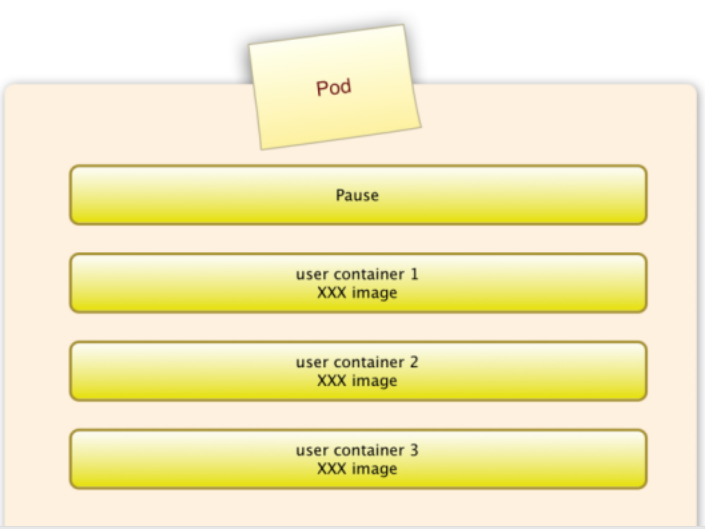
起就是豆荚（一个Pod）。在k8s中我们不会直接操作容器，而是把容器包装成Pod再进行管

理，运行于Node节点上，若干相关容器的组合。Pod内包含的容器运行在同一宿主机上，使

用相同的网络命名空间、IP地址和端口，能够通过localhost进行通信。Pod是Kubernetes进

行创建、调度和管理的最小单位，它提供了比容器更高层次的抽象，使得部署和管 理更加

灵活。一个Pod可以包含一个容器或者多个相关容器。



• 每个Pod都有一个特殊的被称为“根容器”的Pause容器，还包含一个或多个紧密相关的用

户业务容器；

• 一个Pod里的容器与另外主机上的Pod容器能够直接通信；

• 如果Pod所在的Node宕机，会将这个Node上的所有Pod重新调度到其他节点上；

• 普通Pod及静态Pod，前者存放在etcd(存储主管)中，后者存放在具体Node上的一个具体

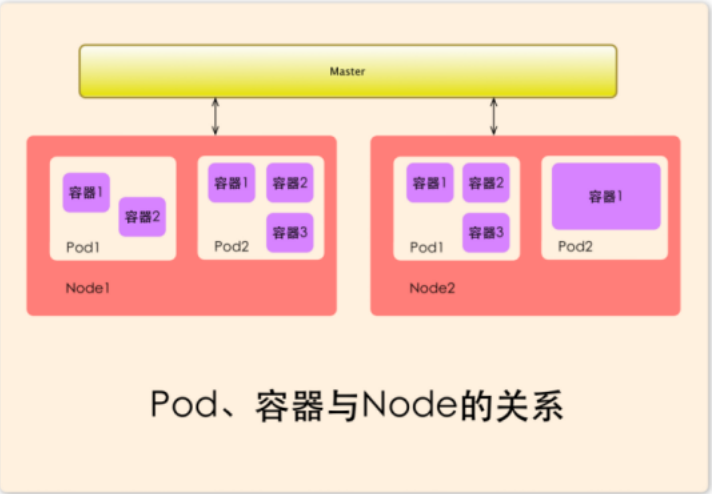
文件中，并且只能在此Node上启动运行；

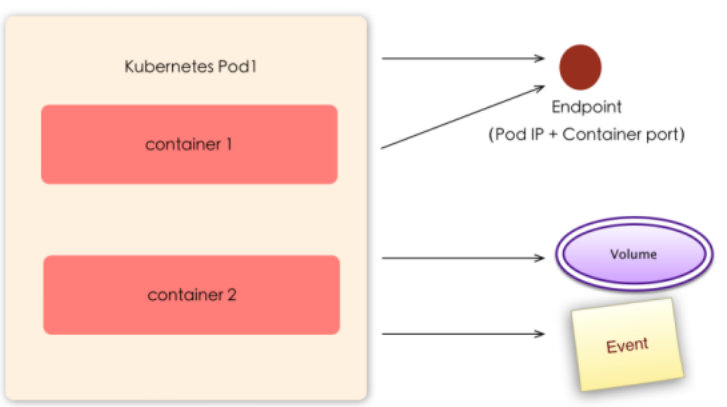
• Docker Volume对应Kubernetes中的Pod Volume；

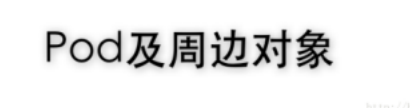
• 每个Pod可以设置限额的计算机资源有CPU和Memory； docker也行

• Requests，资源的最小申请量；

• Limits







**Event**

是一个事件记录(类似于日志)，记录了事件最早产生的时间、最后重复时间、重复次数、

发起者、类型，以及导致此事件的原因等信息。Event通常关联到具体资源对象上，是排查

故障的重要参考信息；

**Pod IP**

Pod的IP地址，是Docker Engine（docker引擎）根据docker0网桥的IP地址段进行分配的，

通常是一个虚拟的二层网络，Node上的Pod能够彼此通信，需要通过Pod IP所在的虚拟二层

网络进行通信（内部通信），而真实的TCP流量（对外通信流量）则是通过Node IP所在的

物理网卡流出的；

**Cluster IP**

Service的IP地址。特性如下：

• 仅仅作用于Kubernetes Servcie这个对象，并由Kubernetes管理和分配IP地址；

• 无法被Ping，因为没有一个“实体网络对象”来响应；

• 只能结合Service Port组成一个具体的通信端口；

• Node IP、Pod IP与Cluster IP之间的通信，采用的是Kubernetes自己设计的一种编程方

式的特殊的路由规则，与IP路由有很大的不同；

**Node IP**

Node节点的IP地址，是Kubernetes集群中每个节点的物理网卡的IP地址，是真是存在的物

理网络，所有属于这个网络的服务器（各个node）之间都能通过这个网络直接通信；

4/19**2.Replication Controller**（复制控制器）

Replication Controller用来管理Pod的副本，保证集群中存在指定数量的Pod副本。集群

中副本的数量大于指定数量，则会停止指定数量之外的多余容器数量，反之，则会启动少

于指定数量个数的容器，保证数量不变。Replication Controller是实现弹性伸缩、动态扩容

和滚动升级的核心。

部署和升级Pod，声明某种Pod的副本数量在任意时刻都符合某个预期值；

• Pod期待的副本数；

• 用于筛选目标Pod的Label Selector（标签选择器）；

• 当Pod副本数量小于预期数量的时候，用于创建新Pod的Pod模板（template）；

**3.Service**

Service定义了Pod的逻辑集合和访问该集合的策略，是真实服务的抽象。Service提供了

一个统一的服务访问入口以及服务代理和发现机制，用户不需要了解后台Pod是如何运行。

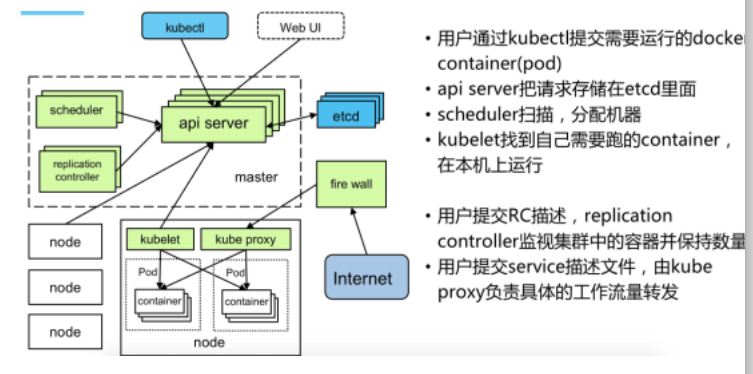
一个service定义了访问pod的方式，就像单个固定的IP地址和与其相对应的DNS名之间的

关系。

Service其实就是我们经常提起的微服务架构中的一个“微服务”，通过分析、识别并建模

系统中的所有服务为微服务——Kubernetes Service，最终我们的系统由多个提供不同业务

能力而又彼此独立的微服务单元所组成，服务之间通过TCP/IP进行通信，从而形成了我们



**kubernetes**部署

生产环境都是二进制安装（预编译安装）

yum无法自定义安装

编译安装可以自定义安装

预编译介于两者之间

准备三台服务器：

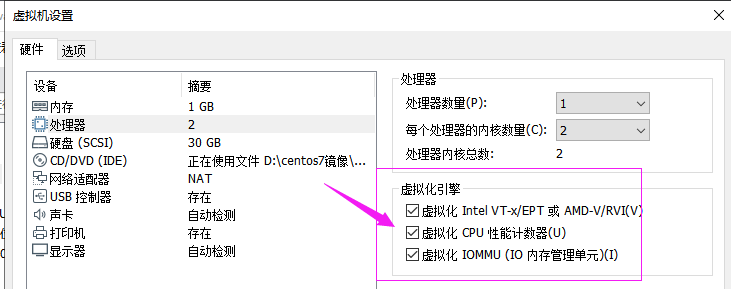
1、一台master

配置： IP：192.168.224.10 ，2H2G

2、两台node

配置： IP：192.168.224.11，192.168.224.12 ， 1H1G

三台虚拟机需要打开虚拟化：



### **一、系统规划**

|  |  |  |
| --- | --- | --- |
| 主机名 | IP | 组件 |
| k8smaster | 192.168.224.10 | etcd、kube-apiserver、kube-controller-manager、kube-scheduler |
| k8snode1 | 192.168.224.11 | kubelet、kube-proxy、docker、dns、calico |
| k8snode2 | 192.168.224.12 | kubelet、kube-proxy、docker、dns、calico |

### **初始化系统基础环境**

* 操作系统：CentOS-7.4-64Bit
* Docker版本：1.18.3
* Kubernetes版本：1.13.3

系统初始化时由于3台机器大部分操作都相同，我这里在配置过程中，在一台主机上进行配置文件创建，然后使用ansible进行分发，当然你也可以直接在对应主机上进行操作。

1）设置主机名

在三台机器分别执行对应设置主机名的命令

hostnamectl set-hostname --static k8smaster

hostnamectl set-hostname --static k8snode1

hostnamectl set-hostname --static k8snode2

1. 配置免密钥登陆

以k8smaster为主机，对另外3台机器进行免密钥登陆

ssh-keygen ##一路回车进行公钥私钥创建

ssh-copy-id -i ~/.ssh/id\_rsa.pub -p 22 root@192.168.224.10

ssh-copy-id -i ~/.ssh/id\_rsa.pub -p 22 [root@192.168.224.11](mailto:root@192.168.224.11)

ssh-copy-id -i ~/.ssh/id\_rsa.pub -p 22 [root@192.168.224.12](mailto:root@192.168.224.12)

如果连接失败需要，开放防火墙，

iptables -A INPUT -p tcp -d 192.168.224.10/32 -m multiport --sports 22 -j ACCEPT

iptables -A INPUT -p tcp -d 192.168.224.11/32 -m multiport --sports 22 -j ACCEPT

iptables -A INPUT -p tcp -d 192.168.224.12/32 -m multiport --sports 22 -j ACCEPT

代表所有来源22端口访问本机的包都通过。

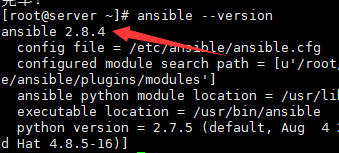
1. 安装ansible(可以不安装，把生成文件或者命令在各节点执行即可)

这里只需在master节点安装即可，后续一些操作均在此机器上执行，然后把生成的文件分发至对应节点

root@k8smaster01 ~]# yum install -y epel-release

[root@k8smaster01 ~]#  yum install ansible -y

[root@k8smaster01 ~]# ansible --version



定义主机组

vim /etc/ansible/hosts

#### [k8smaster] #master节点服务器组 k8smaster会报警告，

[k8smaster] #master服务器组

k8smaster ansible\_host=192.168.224.10 ansible\_ssh\_user=root

#别名 ip 用户

[k8snode] #node节点服务器组

k8snode1 ansible\_host=192.168.224.11 ansible\_ssh\_user=root

k8snode2 ansible\_host=192.168.224.12 ansible\_ssh\_user=root

[k8sall] #k8s集群服务器组

k8smaster ansible\_host=192.168.224.10 ansible\_ssh\_user=root

k8snode1 ansible\_host=192.168.224.11 ansible\_ssh\_user=root

k8snode2 ansible\_host=192.168.224.12 ansible\_ssh\_user=root

ansible k8sall -m ping  #测试ansible是否正常

4、关闭防火墙、selinux(3台机器都执行，我这里使用ansible)

[root@k8s\_masker ~]# ansible k8sall -m shell -a 'systemctl stop firewalld'

[root@k8s\_masker ~]# ansible k8sall -m shell -a 'systemctl disable firewalld'

下面关闭selinux 如果之前设置好了就不用了

[root@k8s\_masker ~]# ansible k8sall -m shell -a 'setenforce 0'

[root@k8s\_masker ~]# ansible k8sall -m replace -a 'path=/etc/selinux/config regexp="SELINUX=enforcing" replace=SELINUX=disabled'

[root@k8s\_masker ~]# ansible k8sall -m replace -a 'path=/etc/sysconfig/selinux regexp="SELINUX=enforcing" replace=SELINUX=disabled'

5 配置host主机域名解析

vim /etc/hosts

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.224.10 k8smaster

192.168.224.11 k8snode1

192.168.224.12 k8snode2

[root@k8s\_masker ~]# ansible k8sall -m copy -a "src=/etc/hosts dest=/etc/hosts" ##文件分发

ansible k8sall -m shell -a 'swapoff -a '

# swapoff -a

# free -m (确认关闭缓存)

6,设置内核

[root@k8smaster ~]# vim /etc/sysctl.d/k8s.conf

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

[root@k8s\_masker ~]# ansible k8sall -m copy -a "src=/etc/sysctl.d/k8s.conf dest=/etc/sysctl.d/k8s.conf"

[root@k8s\_masker ~]# ansible k8sall -m shell -a 'modprobe br\_netfilter'

[root@k8s\_masker ~]# ansible k8sall -m shell -a 'sysctl -p /etc/sysctl.d/k8s.conf'

7.时间同步

[root@k8s\_masker ~]# ansible k8sall -m yum -a "name=ntpdate state=latest"

[root@k8smaster ~]# ansible k8sall -m cron -a "name='k8s cluster crontab' minute=\*/30 hour=\* day=\* month=\* weekday=\* job='ntpdate time7.aliyun.com >/dev/null 2>&1'"

安装docker

**1.设置使用国内Yum源**

[root@linux-node1 ~]# cd /etc/yum.repos.d/

[root@k8smaster yum.repos.d]# wget <https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

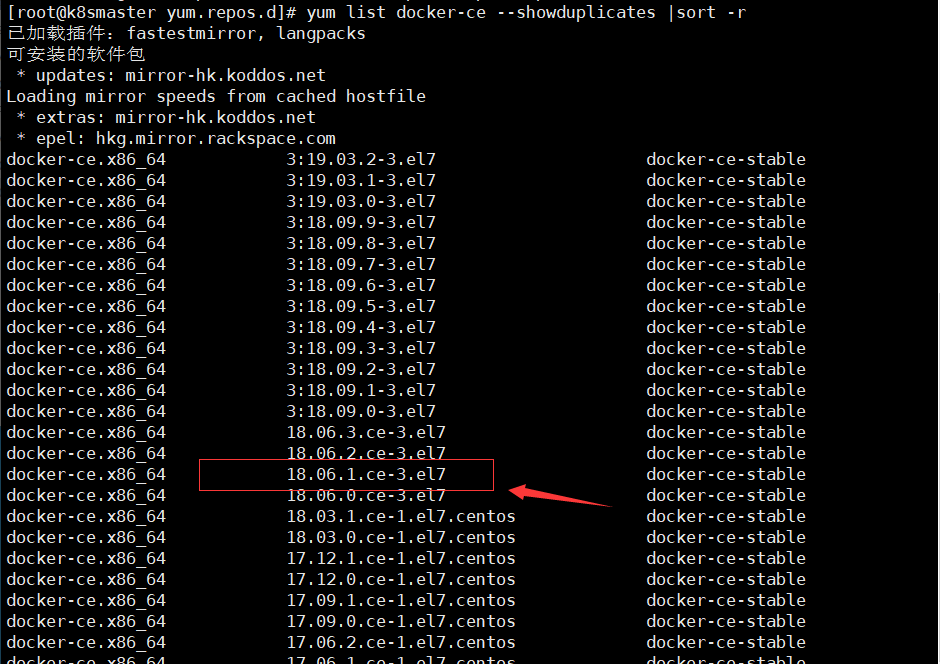
文件分发：

[root@k8smaster yum.repos.d]# ansible k8sall -m copy -a 'src=/etc/yum.repos.d/docker-ce.repo dest=/etc/yum.repos.d/docker-ce.repo'

**2.安装指定的Docker版本**

由于kubeadm对Docker的版本是有要求的，需要安装与kubeadm匹配的版本。

[root@k8smaster yum.repos.d]# yum list docker-ce --showduplicates |sort -r



安装Docker18.06版本

[root@k8smaster yum.repos.d]# yum install -y docker-ce-18.06.1.ce\*

其他也安装：[root@k8smaster yum.repos.d]# ansible k8sall -m shell -a 'yum install -y docker-ce-18.06.1.ce\*'

查看Docker版本

[root@k8smaster yum.repos.d]# docker --version



解压相关镜像文件

ansible k8sall -m shell -a 'tar -xjvf /root/k8s\_images.tar.bz2'

###K8S源

vim /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

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

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg

yum makecache fast

### **文件分发**

ansible k8sall -m copy -a 'src=/etc/yum.repos.d/kubernetes.repo dest=/etc/yum.repos.d/kubernetes.repo'

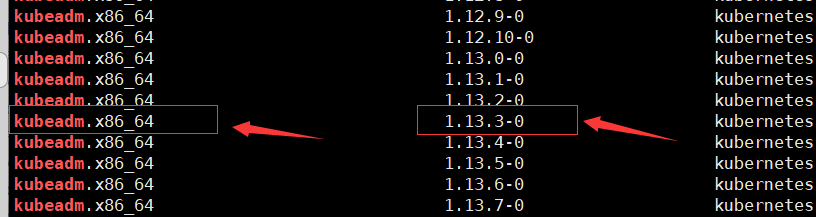
更新Yum仓库

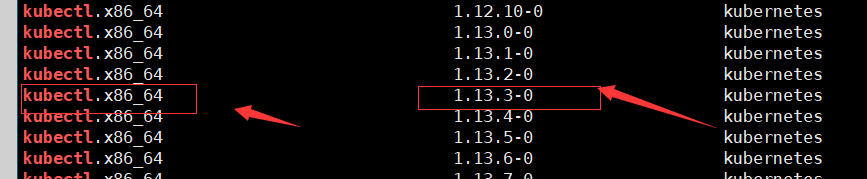
ansible k8sall -m shell -a 'yum makecache fast -y '

**5.安装软件包**

由于版本更新频繁，请指定对应的版本号，本文采用1.13.3版本，其它版本未经测试。

### **[root@k8smaster yum.repos.d]# yum list --showduplicates | grep 'kubeadm\|kubectl\|kubelet'**





### 

Ansible 安装

[root@k8smaster yum.repos.d]# ansible k8sall -m shell -a 'yum install -y kubeadm-1.13.3 kubectl-1.13.3 kubelet-1.13.3 kubernetes-cni-0.6.0-0'

**手动安装kubernetes rpm安装包 手动安装kubeadm以及相关插件服务**

ansible k8sall -m shell -a 'rpm -ivh /root/k8s\_images/socat-1.7.3.2-2.el7.x86\_64.rpm'

ansible k8sall -m shell -a 'rpm -ivh /root/k8s\_images/kubernetes-cni-0.6.0-0.x86\_64.rpm /root/k8s\_images/kubelet-1.9.9-9.x86\_64.rpm'

ansible k8sall -m shell -a 'rpm -ivh /root/k8s\_images/kubectl-1.9.0-0.x86\_64.rpm'

ansible k8sall -m shell -a 'rpm -ivh /root/k8s\_images/kubeadm-1.9.0-0.x86\_64.rpm'

**重新加载守护进程和启动docker kubelet**

**ansible k8sall -m shell -a 'systemctl daemon-reload'**

**ansible k8sall -m shell -a 'systemctl enable docker kubelet'**

**ansible k8sall -m shell -a 'systemctl restart docker kubelet'**

本地拉取镜像

ansible k8sall -m shell -a 'docker pull mirrorgooglecontainers/kube-apiserver:v1.14.3'

ansible k8sall -m shell -a 'docker pull mirrorgooglecontainers/kube-controller-manager:v1.14.3'

ansible k8sall -m shell -a 'docker pull mirrorgooglecontainers/kube-scheduler:v1.14.3'

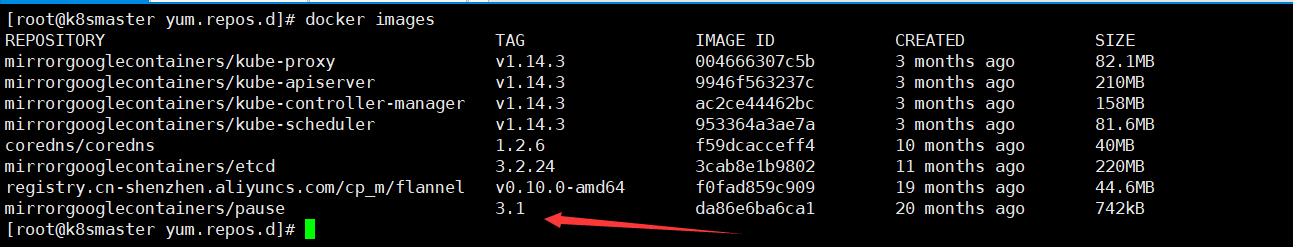
ansible k8sall -m shell -a 'docker pull mirrorgooglecontainers/kube-proxy:v1.14.3'

ansible k8sall -m shell -a 'docker pull mirrorgooglecontainers/pause:3.1'

ansible k8sall -m shell -a 'docker pull mirrorgooglecontainers/etcd:3.2.24'

ansible k8sall -m shell -a 'docker pull coredns/coredns:1.2.6'

ansible k8sall -m shell -a 'docker pull registry.cn-shenzhen.aliyuncs.com/cp\_m/flannel:v0.10.0-amd64'



**d**ocker tag :**标记本地镜像，将其归入某一仓库。**

ansible k8sall -m shell -a 'docker tag mirrorgooglecontainers/kube-apiserver:v1.14.3 k8s.gcr.io/kube-apiserver:v1.14.3'

ansible k8sall -m shell -a 'docker tag mirrorgooglecontainers/kube-controller-manager:v1.14.3 k8s.gcr.io/kube-controller-manager:v1.14.3'

ansible k8sall -m shell -a 'docker tag mirrorgooglecontainers/kube-scheduler:v1.14.3 k8s.gcr.io/kube-scheduler:v1.14.3'

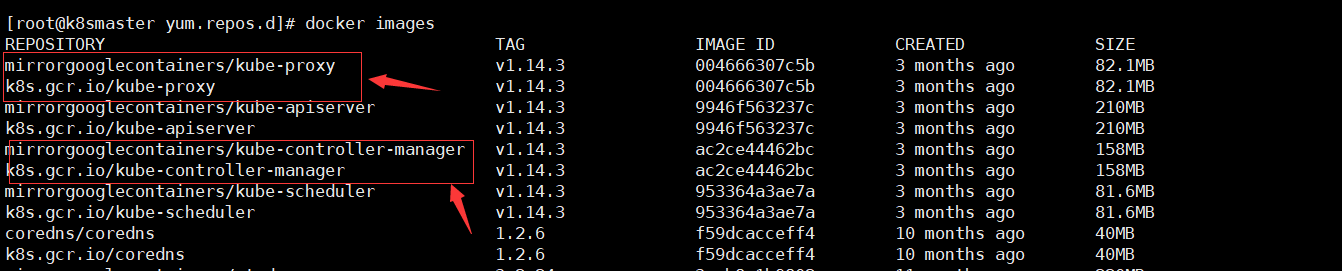
ansible k8sall -m shell -a 'docker tag mirrorgooglecontainers/kube-proxy:v1.14.3 k8s.gcr.io/kube-proxy:v1.14.3'

ansible k8sall -m shell -a 'docker tag mirrorgooglecontainers/pause:3.1 k8s.gcr.io/pause:3.1'

ansible k8sall -m shell -a 'docker tag mirrorgooglecontainers/etcd:3.2.24 k8s.gcr.io/etcd:3.2.24'

ansible k8sall -m shell -a 'docker tag coredns/coredns:1.2.6 k8s.gcr.io/coredns:1.2.6'

ansible k8sall -m shell -a 'docker tag registry.cn-shenzhen.aliyuncs.com/cp\_m/flannel:v0.10.0-amd64 quay.io/coreos/flannel:v0.10.0-amd64'



然后把之前的镜像删除掉

ansible k8sall -m shell -a 'docker rmi mirrorgooglecontainers/kube-apiserver:v1.14.3 '

ansible k8sall -m shell -a 'docker rmi mirrorgooglecontainers/kube-controller-manager:v1.14.3'

ansible k8sall -m shell -a 'docker rmi mirrorgooglecontainers/kube-scheduler:v1.14.3 '

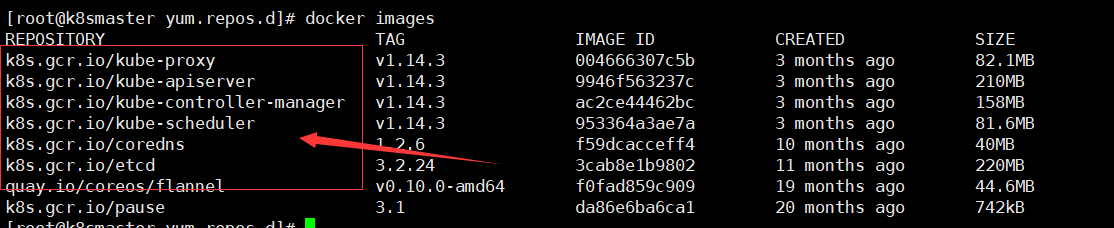
ansible k8sall -m shell -a 'docker rmi mirrorgooglecontainers/kube-proxy:v1.14.3 '

ansible k8sall -m shell -a 'docker rmi mirrorgooglecontainers/pause:3.1 '

ansible k8sall -m shell -a 'docker rmi mirrorgooglecontainers/etcd:3.2.24 '

ansible k8sall -m shell -a 'docker rmi coredns/coredns:1.2.6'

ansible k8sall -m shell -a 'docker rmi registry.cn-shenzhen.aliyuncs.com/cp\_m/flannel:v0.10.0-amd64'



**下载相关镜像，直接导入**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/etcd-amd64\_v3.1.10.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/flannel\_v0.9.1-amd64.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/k8s-dns-dnsmasq-nanny-amd64\_v1.14.7.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/k8s-dns-kube-dns-amd64\_1.14.7.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/k8s-dns-sidecar-amd64\_1.14.7.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/kube-apiserver-amd64\_v1.9.0.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/kube-controller-manager-amd64\_v1.9.0.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/kube-scheduler-amd64\_v1.9.0.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/kube-proxy-amd64\_v1.9.0.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/docker\_images/pause-amd64\_3.0.tar'**

**ansible k8sall -m shell -a 'docker load </root/k8s\_images/kubernetes-dashboard\_v1.8.1.tar'**

**直接安装镜像**

**ansible k8sall -m shell -a 'docker pull lxy940112/etcd-adm64:3.2.18'**

**ansible k8sall -m shell -a 'docker pull jmgao1983/flannel'**

**ansible k8sall -m shell -a 'docker pull ist0ne/k8s-dns-dnsmasq-nanny-amd64'**

**ansible k8sall -m shell -a 'docker pull ist0ne/k8s-dns-kube-dns-amd64'**

**ansible k8sall -m shell -a 'docker pull ist0ne/k8s-dns-sidecar-amd64'**

**ansible k8sall -m shell -a 'docker pull kubernets/kube-apiserver-amd64'**

**ansible k8sall -m shell -a 'docker pull champly/kube-controller-manager-amd64'**

**ansible k8sall -m shell -a 'docker pull pigletfly/kube-proxy-amd64'**

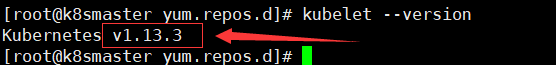
**ansible k8sall -m shell -a 'docker pull ist0ne/kube-scheduler-amd64 '**

**ansible k8sall -m shell -a 'docker pull ist0ne/pause-amd64'**

**查看镜像**

**在主节点执行：**

**安装k8s 先查看版本，然后初始化网络服务**

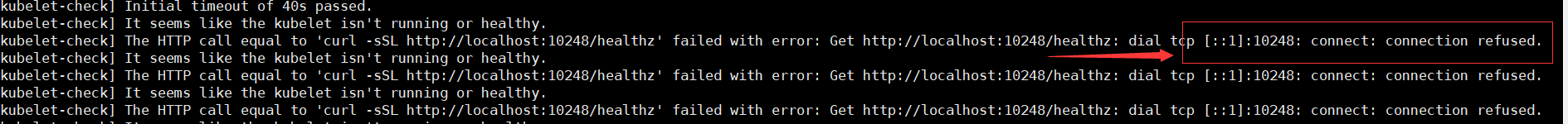


kubeadm init --kubernetes-version=v1.13.3 --apiserver-advertise-address 192.168.224.10 --pod-network-cidr=10.244.0.0/16

先忽略报错，我们来看一下，初始化选项的意义：

* --apiserver-advertise-address：指定用 Master 的哪个IP地址与 Cluster的其他节点通信。
* --service-cidr：指定Service网络的范围，即负载均衡VIP使用的IP地址段。
* --pod-network-cidr：指定Pod网络的范围，即Pod的IP地址段。
* --image-repository：Kubenetes默认Registries地址是 k8s.gcr.io，在国内并不能访问 gcr.io，在1.13版本中我们可以增加-image-repository参数，默认值是 k8s.gcr.io，将其指定为阿里云镜像地址：registry.aliyuncs.com/google\_containers。
* --kubernetes-version=v1.14.3：指定要安装的版本号。
* --ignore-preflight-errors=：忽略运行时的错误，例如上面目前存在[ERROR NumCPU]和[ERROR Swap]，忽略这两个报错就是增加--ignore-preflight-errors=NumCPU 和--ignore-preflight-errors=Swap的配置即可。

**初始化报错**



把防火墙打开，重新初始化就可以了 kubeadm reset

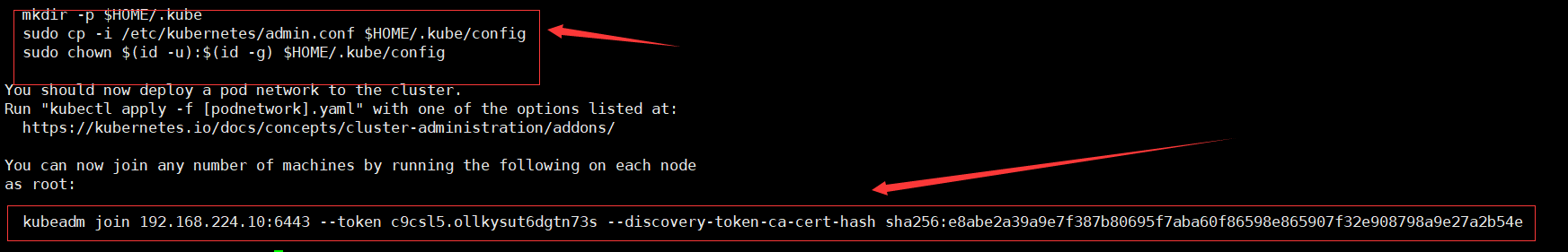
**按提示创建文件，需要记住两行重要信息**

**mkdir -p $HOME/.kube**

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

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

kubeadm join 192.168.224.10:6443 --token c9csl5.ollkysut6dgtn73s --discovery-token-ca-cert-hash sha256:e8abe2a39a9e7f387b80695f7aba60f86598e865907f32e908798a9e27a2b54e



以上为kubeadm初始化命令的输出信息，记录输出结果的最后2行。这2行为在集群成员上执行的命令，用于将成员加入集群中。

#### **9.2 初始化失败解决办法**

$ kubeadm reset

// 或者删除相关文件和images

$ rm -rf /etc/kubernetes/\*.conf

$ rm -rf /etc/kubernetes/manifests/\*.yaml

$ docker ps -a |awk '{print $1}' |xargs docker rm -f

$ systemctl stop kubelet

再次初始化前需要执行清除etcd所有数据的操作

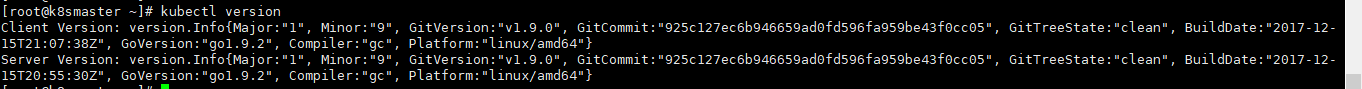
设置环境变量

# echo "export KUBECONFIG=/etc/kubernetes/admin.conf" >> ~/.bash\_profile

# source ~/.bash\_profile

确认版本信息

# kubectl version



**安装Pod网络**

安装 Pod网络是 Pod之间进行通信的必要条件，k8s支持众多网络方案，这里我们依然选用经典的 flannel方案

* 首先设置系统参数：

sysctl net.bridge.bridge-nf-call-iptables=1

* 然后在 Master节点上执行如下命令：

kubectl apply -f kube-flannel.yml

安装flannel插件

没有可以去本地上传kube-flananel.yml文件

cd /root/k8s\_images

kubectl create -f kube-flannel.yml

执行了上面的kubectl apply 就可以不用执行 kubectl create

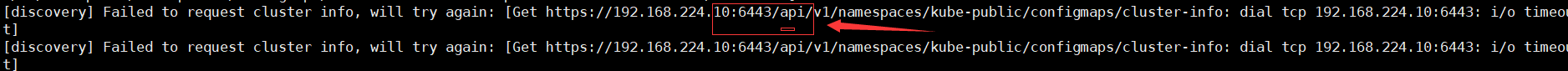
！！！在集群成员节点执行！！！

执行token命令，参考主节点上执行kubeadm init命令的输出结果。

样例（注意不能复制，一定要复制主节点kubeadm init命令的输出结果的最后2行）

kubeadm join 192.168.224.10:6443 --token c9csl5.ollkysut6dgtn73s --discovery-token-ca-cert-hash sha256:e8abe2a39a9e7f387b80695f7aba60f86598e865907f32e908798a9e27a2b54e

如果加入报错误



需要开放防火墙6443端口

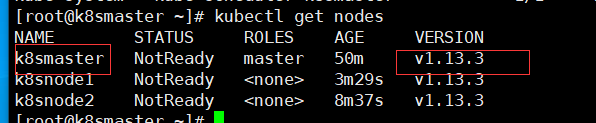
iptables -A INPUT -p tcp -m tcp --dport 6443 -j ACCEPT

iptables -A INPUT -p tcp -d 192.168.224.11/32 -m tcp --sport 6443 -j ACCEPT

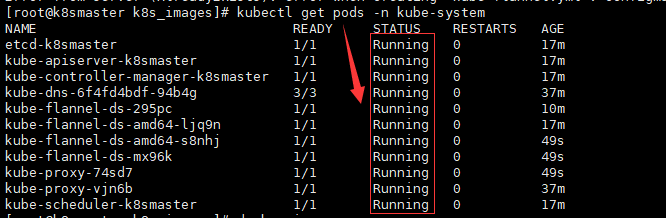
在master操作

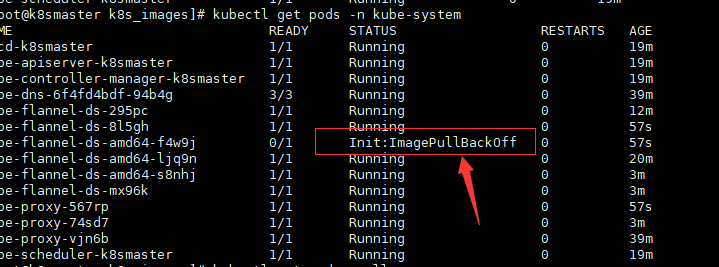
确认所有节点都加入到集群中，注意：下面的命令在成员加入后要等一段时间才会生效。最快10秒

kubectl get nodes



kubectl get pods -n kube-system





这种可以等几分钟再看看

kubectl get pods --all-namespaces

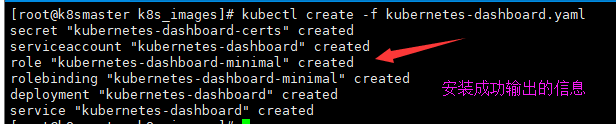
==========

安装控制台仪表盘

！！！在主节点执行！！！

# cd /root/k8s\_images

# kubectl create -f kubernetes-dashboard.yaml



创建账号密码文件

# cat <<EOF > /etc/kubernetes/pki/basic\_auth\_file

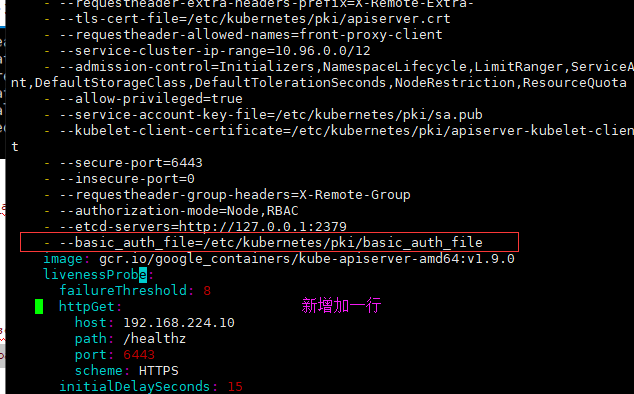
admin,admin,2

EOF

配置密码文件

# vim /etc/kubernetes/manifests/kube-apiserver.yaml

- --basic\_auth\_file=/etc/kubernetes/pki/basic\_auth\_file



重新启动kubelet服务，使密码配置生效

# systemctl restart kubelet

应用API服务器配置

# cd /etc/kubernetes/manifests

[root@server manifests]# kubectl apply -f kube-apiserver.yaml



(如果遇到错误“The connection to the server 192.168.224.10:6443 was refused - did you specify the right host or port?”，再次执行 kubectl apply -f kube-apiserver.yaml 和 systemctl restart kubelet)

配置Dashboard

# kubectl get clusterrole/cluster-admin -o yaml

# kubectl create clusterrolebinding login-on-dashboard-with-cluster-admin --clusterrole=cluster-admin --user=admin



# kubectl get clusterrolebinding/login-on-dashboard-with-cluster-admin -o yaml

确认端口开启

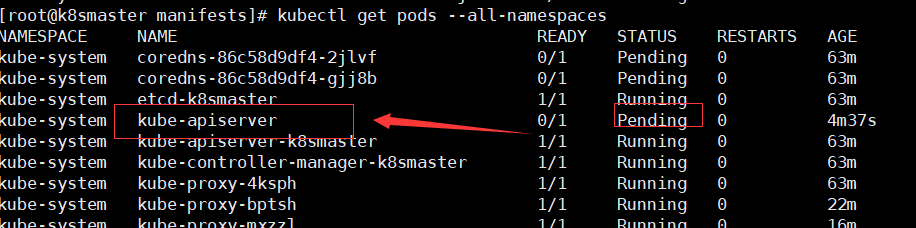
# lsof -i:32666

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

kube-prox 5979 root 9u IPv6 52713 0t0 TCP \*:32666 (LISTEN)

查看pods启动状态，发现每次都是api启动不了

kubectl get pods --all-namespaces



[root@server manifests]# free -m

total used free shared buff/cache available

Mem: 1823 719 76 10 1026 856

Swap: 0 0 0

访问 https://192.168.224.10:32666

在控制台首页：

选择：基本

账号/密码： admin/admin

获取token,通过令牌登陆

kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')

方法一：通过控制台部署。

部署Nginx到 K8S集群中：

进入 控制台-> 工作负载-> 点击链接“➕创建”位于网页左上角 。复制如下代码来从云服务器自动下载nginx镜像。

apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2

kind: Deployment

metadata:

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginx

replicas: 2 # tells deployment to run 2 pods matching the template

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.7.9

ports:

- containerPort: 80

方法二：通过命令行部署。

1. 将内容保存为 yml文件。

2. 执行命令：kubectl create -f deployment.yml --save-config

在Nginx部署完成后，在Master执行下面的命令，确保docker有部署到集群上。

[root@server manifests]# kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-deployment-6c54bd5869-7b655 1/1 Running 0 5m

nginx-deployment-6c54bd5869-8xc7p 1/1 Running 0 5m

查看应用部署情况

# kubectl get deployments

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

nginx-deployment 2 2 2 2 3m

查看节点IP

# kubectl get svc

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

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 22m

将Docker中的虚拟机暴露在网络中

# kubectl expose deployment nginx-deployment --type="LoadBalancer"

查看Docker镜像在主节点的映射端口。本例中：镜像为80端口，映射到主节点的32517端口上。

[root@server manifests]# kubectl get services

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

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 37m

nginx-deployment LoadBalancer 10.109.70.217 <pending> 80:32517/TCP 10s

查看资源详细信息：

kubectl describe pods nginx

查看资源运行情况

kubectl get pods -o wide

访问Docker中的Nginx端口：http://192.168.224.10:32517

======================================================

API部署例子：

我们使用kubectl run来运行我们的第一个应用 ，run命令用于新建一个部署。我们需要提供部署名称和应用镜像地址（DockerHub以外的镜像需要全路径）作为参数。通过--port参数，还可以指定app使用的端口。

kubectl run kubernetes-bootcamp --image=docker.io/jocatalin/kubernetes-bootcamp:v1 --port=8080

具体后台的操作包括：

查找适合这个应用运行的node

调度这个应用在选定的node上运行

配置集群，在需要的时候为这个应用调配新的node

查看应用部署情况kubectl get deployments。

$ kubectl get deployments

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

kubernetes-bootcamp 1 1 1 1 5m

查看应用

在Kubernetes中，pod运行于私有的、隔离的网络。默认情况下，Pod对集群内的其他pod和服务是可见的，但对网络外部是不可见的。我们在使用kubectl时，实际上是通过API端点(endpoint)（可以理解为URL）与应用进行交互。

通过kubectl proxy可以创建一个代理，让你能与集群内的私有网络进行通讯。代理运行过程中没有任何输出，按Ctrl+C可以关闭代理程序。

重新打开一个终端运行代理程序

# kubectl proxy

这样就建立了一条从本地主机到集群的连接，代理程序允许从终端直接访问API。通过代理端点，你可以查看所有的API，端点地址是：http://localhost:8001。你可以通过curl命令直接查看：

$ curl http://localhost:8001/version

{

"major": "1",

"minor": "5",

"gitVersion": "v1.5.2",

"gitCommit": "08e099554f3c31f6e6f07b448ab3ed78d0520507",

"gitTreeState": "clean",

"buildDate": "1970-01-01T00:00:00Z",

"goVersion": "go1.7.1",

"compiler": "gc",

"platform": "linux/amd64"

}$

API服务器自动为每个pod建立同名的API端点，而且这些端点可以通过代理访问。

首先获取Pod名称存储到POD\_NAME中：

$ export POD\_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}')

$ echo Name of the Pod: $POD\_NAME

Name of the Pod: kubernetes-bootcamp-390780338-x81xj

然后向pod中运行的应用发送一个http请求，这个url就指向了Pod的API。

$ curl http://localhost:8001/api/v1/proxy/namespaces/default/pods/kubernetes-bootcamp-5d7f968ccb-n5vqb/

Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-390780338-x81xj | v=1

注意：命令行中的参数“kubernetes-bootcamp-5d7f968ccb-dtdv5”，要从“echo POD\_NAME”中获得。

=================================================

网站服务应用：

在控制台中创建

应用一：

apiVersion: v1

kind: Service

metadata:

name: result

spec:

type: NodePort

ports:

- name: "result-service"

port: 5001

targetPort: 80

nodePort: 31001

selector:

app: result

应用二：

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: result

spec:

replicas: 1

template:

metadata:

labels:

app: result

spec:

containers:

- image: dockersamples/examplevotingapp\_result:before

name: result

确认image运行

[root@server manifests]# kubectl get pods

NAME READY STATUS RESTARTS AGE

kubernetes-bootcamp-5d7f968ccb-n5d4c 1/1 Running 0 9m

result-76784c98fb-nw6jp 1/1 Running 0 42s

确认端口

[root@server manifests]# kubectl get svc

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

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 33m

nginx-service NodePort 10.98.157.233 <none> 80:30001/TCP 13m

result NodePort 10.106.86.50 <none> 5001:31001/TCP 2m

访问网站：

[root@server manifests]# curl 192.168.224.10:31001

================================

Mysql

部署MYSQL持久化卷

kind: PersistentVolume

apiVersion: v1

metadata:

name: mysql-pv-volume

labels:

type: local

spec:

storageClassName: manual

capacity:

storage: 5Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/mnt/data"

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mysql-pv-claim

spec:

storageClassName: manual

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

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

部署MySQL服务：

apiVersion: v1

kind: Service

metadata:

name: mysql

spec:

ports:

- port: 3306

selector:

app: mysql

clusterIP: None

---

apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2

kind: Deployment

metadata:

name: mysql

spec:

selector:

matchLabels:

app: mysql

strategy:

type: Recreate

template:

metadata:

labels:

app: mysql

spec:

containers:

- image: mysql:5.7

name: mysql

env:

# Use secret in real usage

- name: MYSQL\_ROOT\_PASSWORD

value: password

ports:

- containerPort: 3306

name: mysql

volumeMounts:

- name: mysql-persistent-storage

mountPath: /var/lib/mysql

volumes:

- name: mysql-persistent-storage

persistentVolumeClaim:

claimName: mysql-pv-claim

查看deployment mysql 资源详细信息

[root@server manifests]# kubectl describe deployment mysql

Name: mysql

Namespace: default

CreationTimestamp: Tue, 30 Oct 2018 08:51:52 -0700

Labels: app=mysql

Annotations: deployment.kubernetes.io/revision=1

Selector: app=mysql

Replicas: 1 desired | 1 updated | 1 total | 0 available | 1 unavailable

StrategyType: Recreate

MinReadySeconds: 0

Pod Template:

Labels: app=mysql

Containers:

mysql:

Image: mysql:5.6

Port: 3306/TCP

Environment:

MYSQL\_ROOT\_PASSWORD: password

Mounts:

/var/lib/mysql from mysql-persistent-storage (rw)

Volumes:

mysql-persistent-storage:

Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)

ClaimName: mysql-pv-claim

ReadOnly: false

Conditions:

Type Status Reason

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

Available False MinimumReplicasUnavailable

Progressing True ReplicaSetUpdated

OldReplicaSets: <none>

NewReplicaSet: mysql-544bbdcd6f (1/1 replicas created)

Events:

Type Reason Age From Message

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

Normal ScalingReplicaSet 4s deployment-controller Scaled up replica set mysql-544bbdcd6f to 1

确认mysql的服务都已经安装完毕

[root@server manifests]# kubectl get pods -l app=mysql

NAME READY STATUS RESTARTS AGE

mysql-544bbdcd6f-8jbvq 1/1 Running 0 19s

查看mysql-pv-claim 资源详细信息

[root@server manifests]# kubectl describe pvc mysql-pv-claim

Name: mysql-pv-claim

Namespace: default

StorageClass: manual

Status: Bound

Volume: mysql-pv-volume

Labels: <none>

Annotations: pv.kubernetes.io/bind-completed=yes

pv.kubernetes.io/bound-by-controller=yes

Finalizers: []

Capacity: 20Gi

Access Modes: RWO

Events:

Type Reason Age From Message

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

Warning ProvisioningFailed 32s persistentvolume-controller storageclass.storage.k8s.io "manual" not found

通过虚拟机宿主机登陆mysql镜像

[root@server manifests]# kubectl run -it --rm --image=mysql:5.6 --restart=Never mysql-client -- mysql -h mysql -ppassword

If you don't see a command prompt, try pressing enter.

mysql> create database dbname default charset utf8;

mysql> use dbname;

Database changed

mysql> show tables;

Empty set (0.00 sec)

1. 创建集群目录

在集群组件部署之前，先进行对应的目录创建

## 所有节点所需目录

[root@k8smaster ~]# ansible k8sall -m file -a 'path=/etc/kubernetes/ssl state=directory'

[root@k8smaster01 ~]# ansible k8sall -m file -a 'path=/etc/kubernetes/config state=directory'

## k8smaster节点所需目录

[root@k8smaster ~]# mkdir /opt/k8s/{certs,cfg,unit} -p

# [K8S从入门到放弃系列-(2)集群根证书准备](https://www.cnblogs.com/tchua/p/10750571.html)

k8s从1.8版本开始，集群中各个组件需要使用TLS证书对通信进行加密，每个k8s集群都需要有独立的CA证书体系，这里我们采用比较常用的CloudFlare 的 PKI 工具集 cfssl 来生成 Certificate Authority (CA) 证书和秘钥文件，CA 是自签名的证书，用来签名后续创建的其它 TLS 证书。

使用证书的组件如下：

* etcd：使用 ca.pem、etcd-key.pem、etcd.pem；(etcd对外提供服务、节点间通信（etcd peer）使用同一套证书)
* kube-apiserver：使用 ca.pem、ca-key.pem、kube-apiserver-key.pem、kube-apiserver.pem；
* kubelet：使用 ca.pem ca-key.pem；
* kube-proxy：使用 ca.pem、kube-proxy-key.pem、kube-proxy.pem；
* kubectl：使用 ca.pem、admin-key.pem、admin.pem；
* kube-controller-manager：使用 ca-key.pem、ca.pem、kube-controller-manager.pem、kube-controller-manager-key.pem；
* kube-scheduler：使用ca-key.pem、ca.pem、kube-scheduler-key.pem、kube-scheduler.pem；

1）安装cfssl

生成证书时可在任一节点完成，这里在k8smaster主机执行，证书只需要创建一次即可，以后在向集群中添加新节点时只要将 /etc/kubernetes/ssl 目录下的证书拷贝到新节点上即可。

[root@k8s\_masker ~]# mkdir k8s/cfssl -p

[root@k8s\_masker ~]# cd k8s/cfssl/

[root@k8s\_masker cfssl]# wget <https://pkg.cfssl.org/R1.2/cfssl_linux-amd64>

[root@k8s\_masker cfssl]# chmod +x cfssl\_linux-amd64

[root@k8s\_masker cfssl]# cp cfssl\_linux-amd64 /usr/local/bin/cfssl

[root@k8s\_masker cfssl]# wget <https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64>

[root@k8s\_masker cfssl]# chmod +x cfssljson\_linux-amd64

[root@k8s\_masker cfssl]# cp cfssljson\_linux-amd64 /usr/local/bin/cfssljson

[root@k8s\_masker cfssl]# wget https://pkg.cfssl.org/R1.2/cfssl-certinfo\_linux-amd64

[root@k8s\_masker cfssl]# chmod +x cfssl-certinfo\_linux-amd64

[root@k8s\_masker cfssl]# cp cfssl-certinfo\_linux-amd64 /usr/local/bin/cfssl-certinfo

2）创建CA根证书

由于维护多套CA实在过于繁杂，这里CA证书用来签署集群其它组件的证书

这个文件中包含后面签署etcd、kubernetes等其它证书的时候用到的配置

[root@k8s\_masker ~]# vim /opt/k8s/certs/ca-config.json

{

"signing": {

"default": {

"expiry": "87600h"

},

"profiles": {

"kubernetes": {

"usages": [

"signing",

"key encipherment",

"server auth",

"client auth"

],

"expiry": "87600h"

}

}

}

}

* ca-config.json：可以定义多个 profiles，分别指定不同的过期时间、使用场景等参数；后续在签名证书时使用某个 profile;
* signing：表示该证书可用于签名其它证书；生成的 ca.pem 证书中 CA=TRUE;
* server auth：表示client可以用该 CA 对server提供的证书进行验证;
* client auth：表示server可以用该CA对client提供的证书进行验证;
* expiry: 表示证书过期时间，我们设置10年，当然你如果比较在意安全性，可以适当减少

1. 创建 CA 证书签名请求模板

[root@k8smaster ~]# vim /opt/k8s/certs/ca-csr.json

{

"CN": "kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "ShangHai",

"L": "ShangHai",

"O": "k8s",

"OU": "System"

}

]

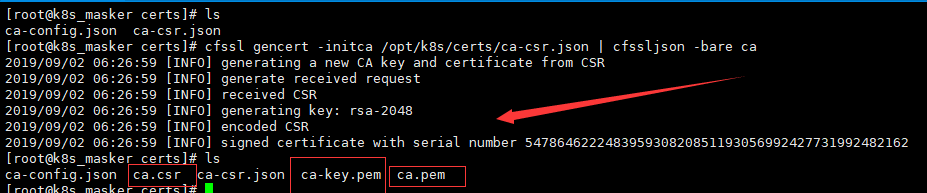
}

4生成CA证书、私钥和csr证书签名请求

该命令会生成运行CA所必需的文件ca-key.pem（私钥）和ca.pem（证书），还会生成ca.csr（证书签名请求），用于交叉签名或重新签名

[root@k8s\_masker ~]# cd /opt/k8s/certs/

[root@k8s\_masker certs]# cfssl gencert -initca /opt/k8s/certs/ca-csr.json | cfssljson -bare ca

  
5）分发证书

 可采用scp直接传输到所有节点，我这里还是采用ansible

[root@k8s\_masker certs]# ansible k8sall -m copy -a 'src=/opt/k8s/certs/ca.csr dest=/etc/kubernetes/ssl/'

[root@k8s\_masker certs]# ansible k8sall -m copy -a 'src=/opt/k8s/certs/ca-key.pem dest=/etc/kubernetes/ssl/'

[root@k8s\_masker certs]# ansible k8sall -m copy -a 'src=/opt/k8s/certs/ca.pem dest=/etc/kubernetes/ssl/'

# [K8S从入门到放弃系列-(3)部署etcd集群](https://www.cnblogs.com/tchua/p/10755871.html)

摘要：etcd 是k8s集群最重要的组件，用来存储k8s的所有服务信息， etcd 挂了，集群就挂了，我们这里把etcd部署在master三台节点上做高可用，etcd集群采用raft算法选举Leader， 由于Raft算法在做决策时需要多数节点的投票，所以etcd一般部署集群推荐奇数个节点，推荐的数量为3、5或者7个节点构成一个集群。

官方地址 <https://github.com/coreos/etcd/releases>

1）下载etcd二进制文件

etcd命令为下载的二进制文件，解压后复制到指定目录即可

[root@k8s\_masker ~]# cd k8s/

[root@k8s\_masker k8s]# wget <https://github.com/etcd-io/etcd/releases/download/v3.3.12/etcd-v3.3.12-linux-amd64.tar.gz>

[root@k8s\_masker k8s]# tar -xf etcd-v3.3.12-linux-amd64.tar.gz

[root@k8s\_masker k8s]# cd etcd-v3.3.12-linux-amd64 ##有2个文件，etcdctl是操作etcd的命令



##把etcd二进制文件传输到三个master节点 我这里这有一台

[root@k8s\_masker etcd-v3.3.12-linux-amd64]# ansible k8smaster -m copy -a 'src=/root/k8s/etcd-v3.3.12-linux-amd64/etcd dest=/usr/local/bin/ mode=0755'

[root@k8s\_masker etcd-v3.3.12-linux-amd64]# ansible k8smaster -m copy -a 'src=/root/k8s/etcd-v3.3.12-linux-amd64/etcdctl dest=/usr/local/bin/ mode=0755'

1. 创建etcd证书请求模板文件

[root@k8s\_masker]# vim /opt/k8s/certs/etcd-csr.json ##证书请求文件

{

"CN": "etcd",

"hosts": [

"127.0.0.1",

"192.168.224.11",

"192.168.224.12"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "ShangHai",

"L": "ShangHai",

"O": "k8s",

"OU": "System"

}

]

}

说明：hosts中的IP为各etcd节点IP及本地127地址，在生产环境中hosts列表最好多预留几个IP，这样后续扩展节点或者因故障需要迁移时不需要再重新生成证书。(我生产环境使用阿里云VPC网络，所以会预留指定段的IP)

3）生成证书及私钥

注意命令中使用的证书的具体位置

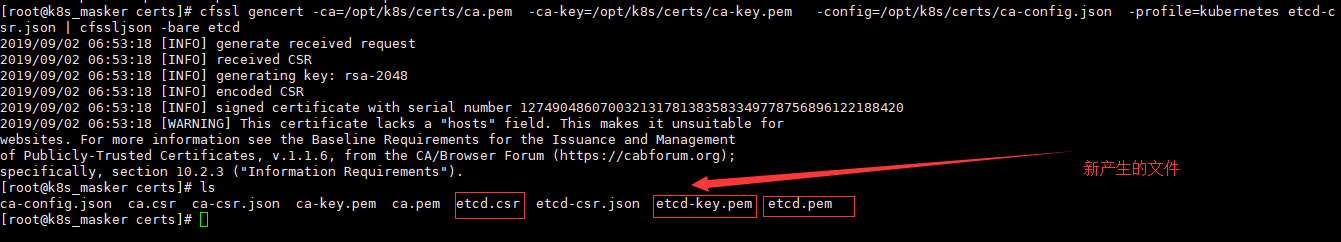
[root@k8s\_masker certs]# cd /opt/k8s/certs/

[root@k8smaster certs]# cfssl gencert -ca=/opt/k8s/certs/ca.pem \ -ca-key=/opt/k8s/certs/ca-key.pem \ -config=/opt/k8s/certs/ca-config.json \ -profile=kubernetes etcd-csr.json | cfssljson -bare etcd

这条命令会报错，用下条。

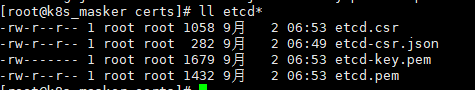


[root@k8s\_masker certs]# cfssl gencert -ca=/opt/k8s/certs/ca.pem -ca-key=/opt/k8s/certs/ca-key.pem -config=/opt/k8s/certs/ca-config.json -profile=kubernetes etcd-csr.json | cfssljson -bare etcd



1. 查看证书

etcd.csr是签署时用到的中间文件，如果你不打算自己签署证书，而是让第三方的CA机构签署，只需要把etcd.csr文件提交给CA机构。



5）证书分发 我只有一台master 所以这步可以不用

把生成的etcd证书复制到创建的证书目录并放至另2台etcd节点

正常情况下只需要copy这三个文件即可，ca.pem(已经存在)、etcd-key.pem、etcd.pem

[root@k8s\_masker certs]# ansible k8smaster -m copy -a 'src=/opt/k8s/certs/etcd.pem dest=/etc/kubernetes/ssl/'

[root@k8s\_masker certs]# ansible k8smaster -m copy -a 'src=/opt/k8s/certs/etcd-key.pem dest=/etc/kubernetes/ssl/'

6）修改etcd配置参数

为了安全性起我这里使用单独的用户启动 Etcd

##创建etcd用户和组

[root@k8smaster ~]# ansible k8smaster -m group -a 'name=etcd'

[root@k8smaster ~]# ansible k8smaster -m user -a 'name=etcd group=etcd comment="etcd user" shell=/sbin/nologin home=/var/lib/etcd createhome=no'

##创建etcd数据存放目录并授权

[root@k8smaster ~]# ansible k8smaster -m file -a 'path=/var/lib/etcd state=directory owner=etcd group=etcd'

说明：

以上步骤若是感觉比较麻烦，可以直接在对应三台master主机执行以下命令即可

mkdir /etc/kubernetes/config

groupadd -r etcd

useradd -r -g etcd -d /var/lib/etcd -s /sbin/nologin -c "etcd user" etcd

mkdir /var/lib/etcd/

chown -R etcd:etcd /var/lib/etcd/

7）配置etcd配置文件

etcd.conf配置文件信息，配置文件中涉及证书，etcd用户需要对其有可读权限，否则会提示无法获取证书，644权限即可。

[root@k8s\_masker certs]# vim /opt/k8s/cfg/etcd.con

#[member]

ETCD\_NAME="etcd01"

ETCD\_DATA\_DIR="/var/lib/etcd"

#ETCD\_SNAPSHOT\_COUNTER="10000"

#ETCD\_HEARTBEAT\_INTERVAL="100"

#ETCD\_ELECTION\_TIMEOUT="1000"

ETCD\_LISTEN\_PEER\_URLS="https://192.168.224.10:2380"

ETCD\_LISTEN\_CLIENT\_URLS="https://192.168.224.10:2379,https://127.0.0.1:2379"

#ETCD\_MAX\_SNAPSHOTS="5"

#ETCD\_MAX\_WALS="5"

#ETCD\_CORS=""

ETCD\_AUTO\_COMPACTION\_RETENTION="1"

ETCD\_QUOTA\_BACKEND\_BYTES="8589934592"

ETCD\_MAX\_REQUEST\_BYTES="5242880"

#[cluster]

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="https://192.168.224.10:2380"

# if you use different ETCD\_NAME (e.g. test),

# set ETCD\_INITIAL\_CLUSTER value for this name, i.e. "test=http://..."

ETCD\_INITIAL\_CLUSTER="etcd01=https://192.168.224.10:2380,etcd02=https://10.10.0.19:2380,etcd03=https://10.10.0.20:2380"

ETCD\_INITIAL\_CLUSTER\_STATE="new"

ETCD\_INITIAL\_CLUSTER\_TOKEN="k8s-etcd-cluster"

ETCD\_ADVERTISE\_CLIENT\_URLS="https://192.168.224.10:2379"

#[security]

CLIENT\_CERT\_AUTH="true"

ETCD\_CA\_FILE="/etc/kubernetes/ssl/ca.pem"

ETCD\_CERT\_FILE="/etc/kubernetes/ssl/etcd.pem"

ETCD\_KEY\_FILE="/etc/kubernetes/ssl/etcd-key.pem"

PEER\_CLIENT\_CERT\_AUTH="true"

ETCD\_PEER\_CA\_FILE="/etc/kubernetes/ssl/ca.pem"

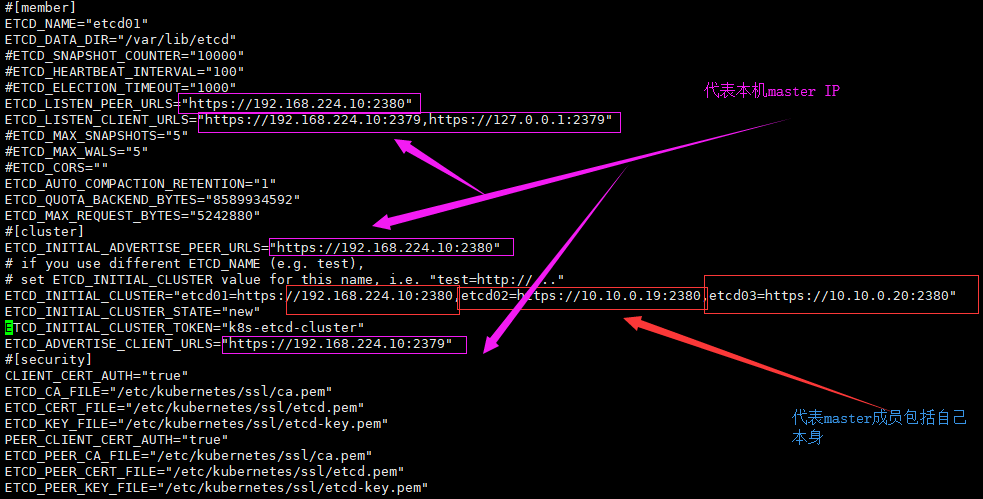
ETCD\_PEER\_CERT\_FILE="/etc/kubernetes/ssl/etcd.pem"

ETCD\_PEER\_KEY\_FILE="/etc/kubernetes/ssl/etcd-key.pem"

参数解释：

1. ETCD\_NAME：etcd节点成员名称，在一个etcd集群中必须唯一性，可使用Hostname或者machine-id
2. ETCD\_LISTEN\_PEER\_URLS：和其它成员节点间通信地址，每个节点不同，必须使用IP，使用域名无效
3. ETCD\_LISTEN\_CLIENT\_URLS：对外提供服务的地址，通常为本机节点。使用域名无效
4. ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS：节点监听地址，并会通告集群其它节点
5. ETCD\_INITIAL\_CLUSTER：集群中所有节点信息，格式为：节点名称+监听的本地端口，及：ETCD\_NAME：[https://ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS](https://etcd_initial_advertise_peer_urls/)
6. ETCD\_ADVERTISE\_CLIENT\_URLS：节点成员客户端url列表，对外公告此节点客户端监听地址，可以使用域名
7. ETCD\_AUTO\_COMPACTION\_RETENTION:  在一个小时内为mvcc键值存储的自动压实保留。0表示禁用自动压缩
8. ETCD\_QUOTA\_BACKEND\_BYTES: 事务中允许的最大操作数,默认1.5M,官方推荐10M，我这里设置5M，大家根据自己实际业务设置
9. ETCD\_MAX\_REQUEST\_BYTES: ETCDdb存储数据大小，默认2G，推荐8G

       由于我们是三个节点etcd集群，所以需要把etcd.conf配置文件复制到另外2个节点，并把上面参数解释中红色参数修改为对应主机IP。



 分发etcd.conf配置文件，当然你不用ansible，可以直接用scp命令把配置文件传输到三台机器对应位置，然后三台机器分别修改IP、ETCD\_NAME等参数。

[root@k8smaster config]# ansible k8smaster -m copy -a 'src=/opt/k8s/cfg/etcd.conf dest=/etc/kubernetes/config/etcd.conf'

##登陆对应主机修改配置文件，把对应IP修改为本地IP

[root@k8smaster ~]# vim /opt/k8s/unit/etcd.service

[Unit]

Description=Etcd Server

After=network.target

After=network-online.target

Wants=network-online.target

[Service]

Type=notify

WorkingDirectory=/var/lib/etcd/

EnvironmentFile=-/etc/kubernetes/config/etcd.conf

User=etcd

# set GOMAXPROCS to number of processors

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/local/bin/etcd --name=\"${ETCD\_NAME}\" --data-dir=\"${ETCD\_DATA\_DIR}\" --listen-client-urls=\"${ETCD\_LISTEN\_CLIENT\_URLS}\""

Restart=on-failure

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

[root@k8smaster ~]# ansible k8smaster -m copy -a 'src=/opt/k8s/unit/etcd.service dest=/usr/lib/systemd/system/etcd.service'

[root@k8smaster ~]# ansible k8smaster -m shell -a 'systemctl daemon-reload'

[root@k8smaster ~]# ansible k8smaster -m shell -a 'systemctl enable etcd'

[root@k8smaster ~]# ansible k8smaster -m shell -a 'systemctl start etcd'

注：

这里需要三台etcd服务同时启动，在三台机器上同时执行启动命令，启动其中一台后，服务会卡在那里，直到集群中所有etcd节点都已启动。我这里因为是ansible远程执行，所以没有出现这个问题。

 8）验证集群

etcd3版本，查看集群状态时，需要指定对应的证书位置 只有一台master,写一个ip 就可以了

[root@k8smaster ~]# etcdctl --endpoints=https://192.168.224.10:2379,https://10.10.0.19:2379,https://10.10.0.20:2379 --cert-file=/etc/kubernetes/ssl/etcd.pem --ca-file=/etc/kubernetes/ssl/ca.pem --key-file=/etc/kubernetes/ssl/etcd-key.pem cluster-health

etcdctl --endpoints=https://192.168.224.10:2379,https://10.10.0.19:2379,https://10.10.0.20:2379 --cert-file=/etc/kubernetes/ssl/etcd.pem --ca-file=/etc/kubernetes/ssl/ca.pem --key-file=/etc/kubernetes/ssl/etcd-key.pem member list