二进制部署K8S集群从0到1

**[管理集群的TLS](#n1)**

**[集群部署](#n2)**

环境规划

环境准备

**[创建TLS证书和密钥](#n3)**

创建CA（certficate authority）

**[创建Kubeconfig文件](#n4)**

下载kubectl

创建kubectl kubeconfig文件

创建TLS Bootstrapping token

创建kubelet bootstrapping kubeconfig文件

创建kube-proxy kubeconfig文件

分发kubeconfig 文件

[**创建ETCD HA 集群**](#n5)

TLS 证书认证

下载二进制文件

创建etcd的数据目录

创建etcd的环境变量配置文件

部署node节点的etcd

启动&验证服务

[**部署master节点**](#n6)

TLS证书文件

下载二进制版本

配置和启动kube-apiserver

配置和启动kube-scheduler

配置和启动kube-controller-manager

验证master节点功能

[**安装flannel网络插件**](#n7)

[**部署node节点**](#n8)

配置Docker

安装配置kubectl

配置kube-proxy

脚本方式配置

验证测试

[**DNS服务搭建与配置**](#n9)

安装CoreDNS插件

[**安装dashboard插件**](#n10)

采用kubectl proxy服务dashboard

[**部署metrics-server插件**](#n11)

[**K8S集群实现自动扩容与缩容：autoscale**](#n12)

介绍：K8S集群系统的各组件需要使用TLS证书对通信进行加密，本文档使用CloudFlare的PKI工具cfssl来生成Certificate authority 和其他证书

**管理集群中的TLS**

**前言**：

每一个K8S集群都有一个集群证书颁发机构（CA）。集群中组件通常使用CA来验证API server 证书，有API server 验证kubectl客户端证书等。为了支持这一点，CA证书被分发到集群中的每一个节点，并作为一个secret附加分发到默认service account上。

**集群中的TLS信任**

让POD 中运行的应用程序信任集群根CA需要额外的应用程序配置，需要将CA证书包添加到TLS客户端或者服务器信任的CA列表中。

**集群部署**

**环境规划**

|  |  |
| --- | --- |
| **软件** | **版本** |
| **Linux操作系统** | **CentOS Linux release 7.7.1908 (Core)** |
| **Kubernetes** | **v1.14.3** |
| **Docker** | **18.06.3-ce** |
| **Etcd** | **3.3.11** |

|  |  |  |  |
| --- | --- | --- | --- |
| **角色** | **IP** | **组件** | **配置** |
| **K8S-master** | **135.251.206.39** | **kube-apiserver  kube-controller-manager  kube-scheduler etcd**  **flannel**  **docker** | **8core和16GB内存** |
| **K8s-node1** | **135.251.206.38** | **kubelet  kube-proxy docker  flannel  etcd** | **根据需要运行的容器数量进行配置** |
| **K8s-node2** | **135.251.206.37** | **kubelet  kube-proxy docker  flannel  etcd** | **根据需要运行的容器数量进行配置** |

|  |  |
| --- | --- |
| **组件** | **证书** |
| **Etcd** | **ca.pem, server.pem, server-key.pem** |
| **Kube-apiserver** | **ca.pem, server.pem, server-key.pem** |
| **Kubelet** | **ca.pem, ca-key.pem** |
| **Kube-proxy** | **ca.pem, kube-proxy.pem, kube-proxy-key.pem** |
| **Kubectl** | **ca.pem, admin.pem, admin-key.pem** |
| **Kube-controller-manager** | **ca.pem, ca-key.pem** |
| **Flannel** | **ca.pem, server.pem, server-key.pem** |

**环境准备**

**以下操作在master节点和node节点上执行**

**安装必要的软件包**

# 安装net-tools，可以使用ping,ifconfig等命令

yum install -y net-tools

# 安装curl，telnet命令

yum install -y curl telnet

# 安装vim编辑器

yum install -y vim

# 安装wget下载命令

yum install -y wget

# 安装lrzsz工具，可以直接拖拽文件到Xshell中上传文件到服务器或下载文件到本地。

yum -y install lrzsz

**关闭防火墙**

systemctl stop firewalld

systemctl disable firewalld

**关闭selinux**

sed -i 's/enforcing/disabled' /etc/selinux/config

setenforce 0

# 或者进入到/etc/selinux/config将以下字段设置并重启生效：

SELINUX=disabled

**关闭swap**

swapoff -a # 临时

vim /etc/fstab #永久

**确保net.bridge.bridge-nf-call-iptables在sysctl配置为1**

cat <<EOF > /etc/sysctl.d/k8s.conf

net.ipv4.ip\_forward =1

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

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

EOF

$ sysctl –system

**同步时间**

# yum install ntpdate -y

# ntpdate ntp.api.bz

**k8s需要容器运行时（Container Runtime Interface，CRI）的支持，目前官方支持的容器运行时包括：Docker、Containerd、CRI-O和frakti。此处以Docker作为容器运行环境，推荐版本为Docker CE 18.06 或 18.09**

**安装docker**

# 为Docker配置阿里云源,注意是在/etc/yum.repos.d目录执行下述命令。

wget <https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

yum update && yum repolist

# 列出可用的docker-ce版本，推荐使用18.06或18.09的稳定版。

yum list docker-ce.x86\_64 --showduplicates | sort -r

# 正式安装docker，此处以docker-ce-18.06.3.ce-3.el7为例。

yum -y install docker-ce-18.06.3.ce-3.el7

# 在此处可能会报错：Delta RPMs disabled because /usr/bin/applydeltarpm not installed.采用如下命令解决。

yum provides '\*/applydeltarpm'

yum install deltarpm -y

# 然后重新执行安装命令

yum -y install docker-ce-18.06.3.ce-3.el7

# 安装完成设置docker开机自启动。

systemctl enable docker

**以下操作仅需要在master节点上执行，证书只需创建一次即可，之后需在集群中加入新的node，只需要将/etc/kubernetes/目录下的证书copy到node上即可。**

**创建TLS证书和密钥**

**采用二进制源码包安装CFSSL**

# 首先创建存放证书的位置

$ mkdir ssl && cd ssl

# 下载用于生成证书的

wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64

# 用于将证书的json文本导入

wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64

# 查看证书信息

wget https://pkg.cfssl.org/R1.2/cfssl-certinfo\_linux-amd64

# 修改文件，使其具备执行权限

chmod +x cfssl\_linux-amd64 cfssljson\_linux-amd64 cfssl-certinfo\_linux-amd64

# 将文件移到/usr/local/bin/cfssl

mv cfssl\_linux-amd64 /usr/local/bin/cfssl

mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson

mv cfssl-certinfo\_linux-amd64 /usr/local/bin/cfssl-certinfo

# 如果是普通用户，可能需要将环境变量设置下

export PATH=/usr/local/bin:$PATH

**创建CA（Certificate Authority）**

**注意以下命令，仍旧在/root/ssl文件目录下执行。**

**创建CA配置文件**

****

字段说明

* 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；

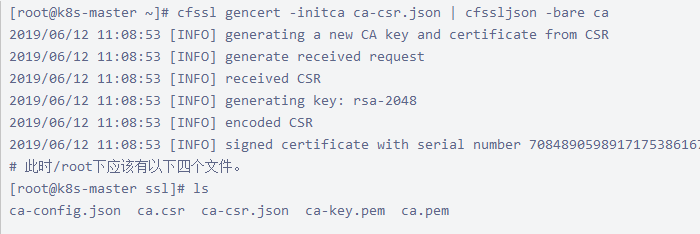
**创建CA证书签名请求**

****

 CN”：Common Name，kube-apiserver 从证书中提取该字段作为请求的用户名 (User Name)；浏览器使用该字段验证网站是否合法；

 “O”：Organization，kube-apiserver 从证书中提取该字段作为请求用户所属的组 (Group)；

**生成CA证书和私钥**

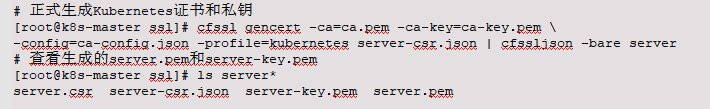
****

**创建Kubernetes证书**

创建Kubernetes证书签名请求文件server-csr.json（kubernetes-csr.json），并将受信任的IP修改添加到hosts，比如我的三个节点的IP为：135.251.206.39，135.251.206.38，135.251.206.37



**生成Kubernetes证书和私钥**

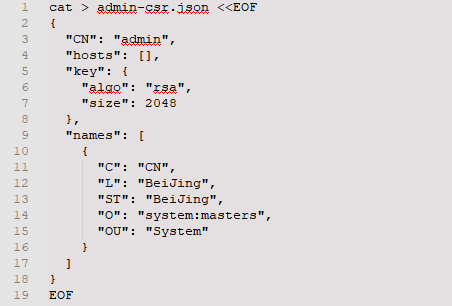


如果 hosts 字段不为空则需要指定授权使用该证书的 **IP 或域名列表**，由于该证书后续被 etcd 集群和 kubernetes master 集群使用，所以上面分别指定了 etcd集群、kubernetes master 集群的主机 IP 和 **kubernetes 服务的服务 IP**（一般是 kube-apiserver 指定的 service-cluster-ip-range 网段的第一个IP，如 10.10.10.1）。

 这是最小化安装的kubernetes集群，包括一个私有镜像仓库，三个节点的kubernetes集群，以上物理节点的IP也可以更换为主机名

**创建admin证书**

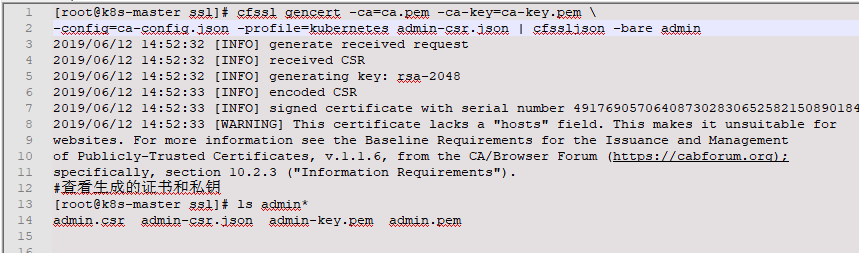
**创建admin证书签名请求文件，admin-csr.json:**



* 后续 kube-apiserver 使用 RBAC 对客户端(如 kubelet、kube-proxy、Pod)请求进行授权；
* kube-apiserver 预定义了一些 RBAC 使用的 RoleBindings，如 cluster-admin 将 Group system:masters 与 Role cluster-admin 绑定，该 Role 授予了调用kube-apiserver 的**所有 API**的权限；
* O 指定该证书的 Group 为 system:masters，kubelet 使用该证书访问 kube-apiserver 时 ，由于证书被 CA 签名，所以认证通过，同时由于证书用户组为经过预授权的 system:masters，所以被授予访问所有 API 的权限；

**注意**：这个admin 证书，是将来生成管理员用的kube config 配置文件用的，现在我们一般建议使用RBAC 来对kubernetes 进行角色权限控制， kubernetes 将证书中的CN 字段 作为User， O 字段作为 Group（具体参考 [Kubernetes中的用户与身份认证授权](https://jimmysong.io/kubernetes-handbook/guide/authentication.html)中 X509 Client Certs 一段）

**生成admin证书和私钥**



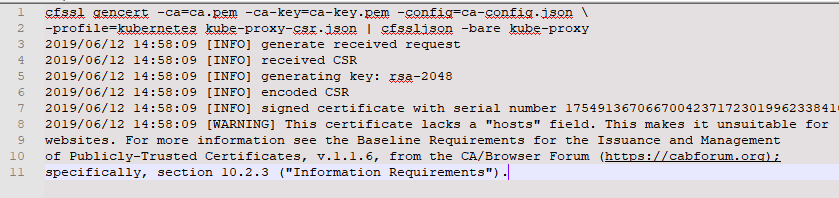
**创建kube-proxy证书**

创建 kube-proxy 证书签名请求文件 kube-proxy-csr.json，让它携带证书访问集群：



* CN 指定该证书的 User 为 system:kube-proxy；
* kube-apiserver 预定义的 RoleBinding system:node-proxier 将User system:kube-proxy 与 Role system:node-proxier 绑定，该 Role 授予了调用 kube-apiserver Proxy 相关 API 的权限；

**生成 kube-proxy 客户端证书和私钥**



**校验证书**

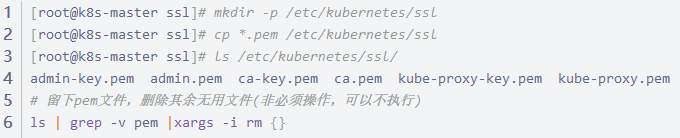
openssl x509 -noout -text -in server.pem

* 确认 Issuer 字段的内容和 ca-csr.json 一致；
* 确认 Subject 字段的内容和 server-csr.json 一致；
* 确认 X509v3 Subject Alternative Name 字段的内容和 server-csr.json 一致；
* 确认 X509v3 Key Usage、Extended Key Usage 字段的内容和 ca-config.json 中 ``kubernetes profile` 一致；

cfssl-certinfo -cert server.pem

**分发证书**

将生成的证书和秘钥文件（后缀名为.pem）拷贝到所有机器的 /etc/kubernetes/ssl目录下备用；

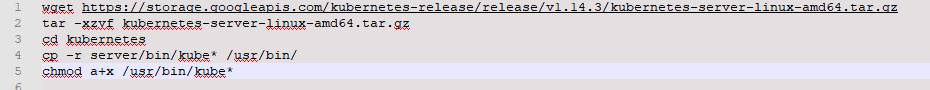


**创建kubeconfig文件**

以下命令在master节点运行，没有指定运行目录，则默认是用户家目录，root用户则在/root下执行

**下载kubectl**

注意请下载对应的Kubernetes版本的安装包。



**创建kubectl kubeconfig文件**



* admin.pem 证书 OU 字段值为 system:masters，kube-apiserver 预定义的 RoleBinding cluster-admin 将 Group system:masters 与 Role cluster-admin绑定，该 Role 授予了调用kube-apiserver 相关 API 的权限；
* 生成的 kubeconfig 被保存到 ~/.kube/config 文件；

**注意：**~/.kube/config文件拥有对该集群的最高权限，请妥善保管。

kubelet、kube-proxy 等 Node 机器上的进程与 Master 机器的 kube-apiserver 进程通信时需要认证和授权；

以下操作只需要在master节点上执行，生成的\*.kubeconfig文件可以直接拷贝到node节点的/etc/kubernetes目录下。

**创建TLS Bootstrapping token**

**Token auth file**

Token可以是任意的包含128 bit的字符串，可以使用安全的随机数发生器生成



**注意：在进行后续操作前请检查 token.csv 文件，确认其中的 ${BOOTSTRAP\_TOKEN}环境变量已经被真实的值替换。**

**BOOTSTRAP\_TOKEN** 将被写入到 kube-apiserver 使用的 token.csv 文件和 kubelet 使用的 bootstrap.kubeconfig 文件，**如果后续重新生成了 BOOTSTRAP\_TOKEN，则需要**：

1. 更新 token.csv 文件，分发到所有机器 (master 和 node）的 /etc/kubernetes/ 目录下，分发到node节点上非必需；
2. 重新生成 bootstrap.kubeconfig 文件，分发到所有 node 机器的 /etc/kubernetes/ 目录下；
3. 重启 kube-apiserver 和 kubelet 进程；
4. 重新 approve kubelet 的 csr 请求；

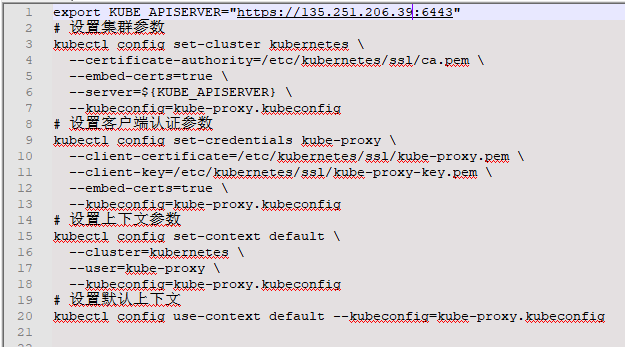


**创建kubelet bootstrapping kubeconfig文件**



* --embed-certs 为 true 时表示将 certificate-authority 证书写入到生成的 bootstrap.kubeconfig 文件中；
* 设置客户端认证参数时**没有**指定秘钥和证书，后续由 kube-apiserver 自动生成；

**创建kube-proxy kubeconfig文件**

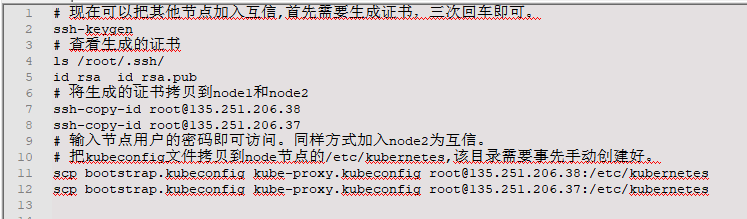


 设置集群参数和客户端认证参数时 --embed-certs 都为 true，这会将 certificate-authority、client-certificate 和 client-key 指向的证书文件内容写入到生成的 kube-proxy.kubeconfig 文件中；

 kube-proxy.pem 证书中 CN 为 system:kube-proxy，kube-apiserver 预定义的 RoleBinding cluster-admin 将User system:kube-proxy 与 Role system:node-proxier 绑定，该 Role 授予了调用 kube-apiserver Proxy 相关 API 的权限

**分发Kubeconfig文件**

将两个 kubeconfig 文件分发到所有 Node 机器的 /etc/kubernetes/ 目录下



**创建 ETCD HA集群**

etcd服务作为k8s集群的主数据库，在安装k8s各服务之前需要首先安装和启动。kuberntes 系统使用 etcd 存储所有数据，本文档介绍部署一个三节点高可用 etcd 集群的步骤，这三个节点复用 kubernetes master 机器，分别命名为k8s-master、k8s-node1、k8s-node2：

|  |  |
| --- | --- |
| 角色 | IP |
| K8S-master | 135.251.206.39 |
| KS8-node1 | 135.251.206.38 |
| K8S-node2 | 135.251.206.37 |

**TLS认证文件**

需要为 etcd 集群创建加密通信的 TLS 证书，这里复用以前创建的 kubernetes 证书，之前已经将相关证书copy到/etc/kubernetes/ssl目录下了。

kubernetes 证书的 hosts 字段列表中包含上面三台机器的 IP，否则后续证书校验会失败

**下载二进制文件**



或者直接使用yum命令安装：

Yum install -y etcd

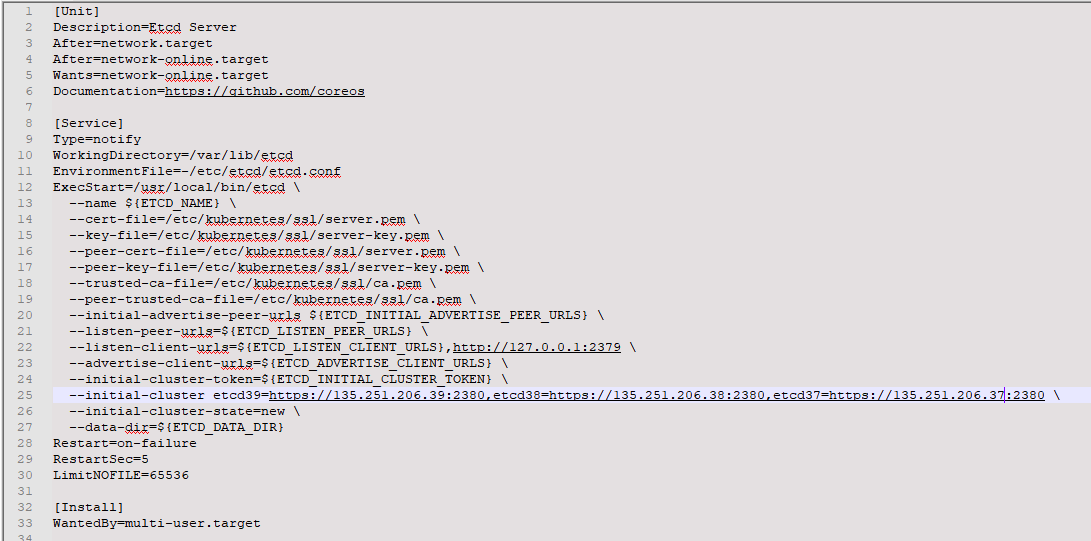
**注意**：若使用yum安装，默认etcd命令将在/usr/bin目录下，注意修改下面的etcd.service文件中的启动命令地址为/usr/bin/etcd

**创建etcd的数据目录**

Mkdir -p /var/lib/etcd/default.etcd

**创建etcd的systemd unit文件**

在/usr/lib/systemd/system/目录下创建文件etcd.service，内容如下。注意替换IP地址为你自己的etcd集群的主机IP。



 指定 etcd 的工作目录为 /var/lib/etcd，数据目录为 /var/lib/etcd，**需在启动服务前创建这个目录**，否则启动服务的时候会报错“Failed at step CHDIR spawning /usr/bin/etcd: No such file or directory”；

 为了保证通信安全，需要指定 etcd 的公私钥(cert-file和key-file)、Peers 通信的公私钥和 CA 证书(peer-cert-file、peer-key-file、peer-trusted-ca-file)、客户端的CA证书（trusted-ca-file）；

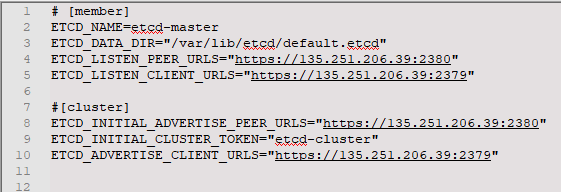
 创建 server.pem 证书时使用的 server-csr.json 文件的 hosts 字段**包含所有 etcd 节点的IP**，否则证书校验会出错；

 --initial-cluster-state 值为 new 时，--name 的参数值必须位于 --initial-cluster 列表中

**创建etcd的环境变量配置文件/etc/etcd/etcd.conf**

Mkdir -p /etcd/etcd

Touch etcd.conf

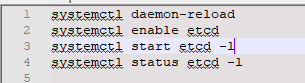


其他两个etcd节点只要将上面的IP地址改成相应节点的IP地址即可。ETCD\_NAME换成对应节点的

**部署node节点的etcd**

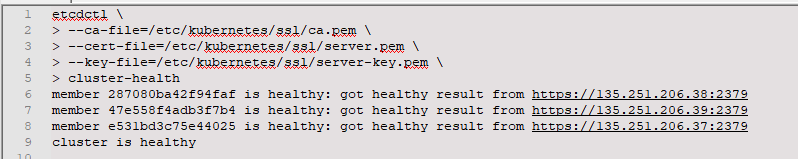


**启动服务**



**验证服务**

在master节点上执行



此时，表面集群服务正常工作

**部署master节点**

kubernetes master 节点包含的组件：

* kube-apiserver
* kube-scheduler
* kube-controller-manager

目前这三个组件需要部署在同一台机器上。

* kube-scheduler、kube-controller-manager 和 kube-apiserver 三者的功能紧密相关；
* 同时只能有一个 kube-scheduler、kube-controller-manager 进程处于工作状态，如果运行多个，则需要通过选举产生一个 leader；

**TLS证书文件**

以下pem证书文件我们在”创建TLS证书和秘钥“这一步中已经创建过了，token.csv文件在“创建kubeconfig文件”的时候创建。

将二进制文件拷贝到指定路径

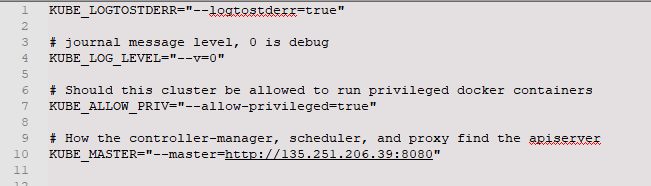
cp -r server/bin/{kube-apiserver,kube-controller-manager,kube-scheduler,kubectl,kube-proxy,kubelet} /usr/local/bin/

### 配置和启动kube-apiserver

创建kube-apiserver的service配置文件，/usr/lib/systemd/system/kube-apiserver.service内容：

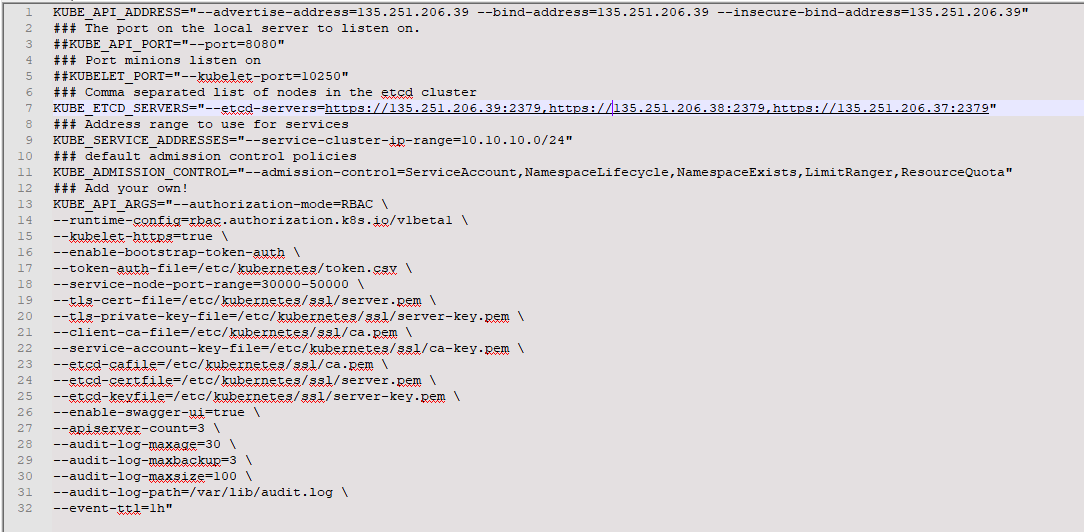


创建/etc/kubernetes/config文件内容为：



该配置文件同时被kube-apiserver、kube-controller-manager、kube-scheduler、kubelet、kube-proxy使用。

apiserver配置文件/etc/kubernetes/apiserver内容为：



* 如果中途修改过--service-cluster-ip-range地址，则必须将default命名空间的kubernetes的service给删除，使用命令：kubectl delete service kubernetes，然后系统会自动用新的ip重建这个service，不然apiserver的log有报错the cluster IP x.x.x.x for service kubernetes/default is not within the service CIDR x.x.x.x/24; please recreate
* --authorization-mode=RBAC 指定在安全端口使用 RBAC 授权模式，拒绝未通过授权的请求；
* kube-scheduler、kube-controller-manager 一般和 kube-apiserver 部署在同一台机器上，它们使用**非安全端口**和 kube-apiserver通信;
* kubelet、kube-proxy、kubectl 部署在其它 Node 节点上，如果通过**安全端口**访问 kube-apiserver，则必须先通过 TLS 证书认证，再通过 RBAC 授权；
* kube-proxy、kubectl 通过在使用的证书里指定相关的 User、Group 来达到通过 RBAC 授权的目的；
* 如果使用了 kubelet TLS Boostrap 机制，则不能再指定 --kubelet-certificate-authority、--kubelet-client-certificate 和 --kubelet-client-key 选项，否则后续 kube-apiserver 校验 kubelet 证书时出现 ”x509: certificate signed by unknown authority“ 错误；
* --admission-control 值必须包含 ServiceAccount；
* --bind-address 不能为 127.0.0.1；
* runtime-config配置为rbac.authorization.k8s.io/v1beta1，表示运行时的apiVersion；
* --service-cluster-ip-range 指定 Service Cluster IP 地址段，该地址段不能路由可达；
* 缺省情况下 kubernetes 对象保存在 etcd/registry 路径下，可以通过 --etcd-prefix 参数进行调整；
* 如果需要开通http的无认证的接口，则可以增加以下两个参数：--insecure-port=8080 --insecure-bind-address=127.0.0.1。注意，生产上不要绑定到非127.0.0.1的地址上。

启动服务

systemctl daemon-reload

systemctl enable kube-apiserver

systemctl start kube-apiserver -l

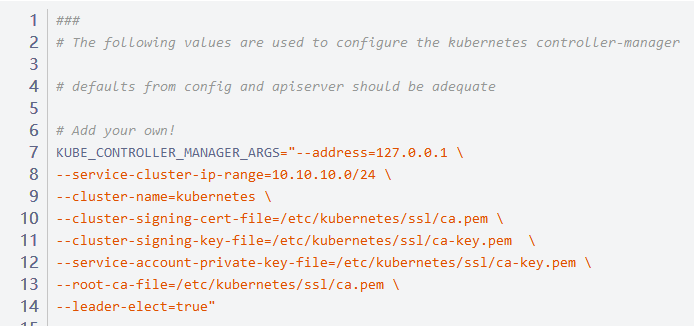
systemctl status kube-apiserver -l

### 配置和启动kube-controller-manager

创建kube-controller-manager的service配置文件,usr/lib/systemd/system/kube-controller-manager.service



配置文件/etc/kubernetes/controller-manager



* --service-cluster-ip-range 参数指定 Cluster 中 Service 的CIDR范围，该网络在各 Node 间必须路由不可达，必须和 kube-apiserver 中的参数一致；
* --cluster-signing-\* 指定的证书和私钥文件用来签名为 TLS BootStrap 创建的证书和私钥；
* --root-ca-file 用来对 kube-apiserver 证书进行校验，**指定该参数后，才会在Pod 容器的 ServiceAccount 中放置该 CA 证书文件**；
* --address 值必须为 127.0.0.1，kube-apiserver 期望 scheduler 和 controller-manager 在同一台机器；

**启动kube-controller-manager**

systemctl daemon-reload

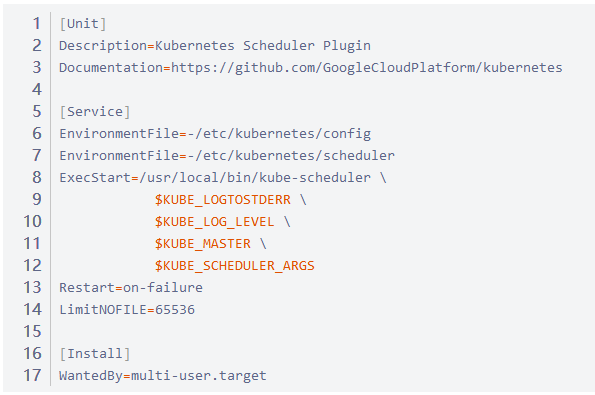
systemctl enable kube-controller-manager

systemctl start kube-controller-manager

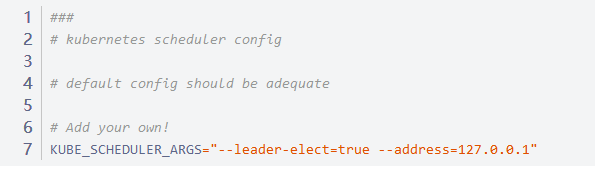
systemctl status kube-controller-manager

### 配置和启动kube-scheduler

创建kube-scheduler的service的配置文件,/usr/lib/systemd/system/kube-scheduler.service



配置文件/etc/kubernetes/scheduler



--address 值必须为 127.0.0.1，因为当前 kube-apiserver 期望 scheduler 和 controller-manager 在同一台机器；

**启动kube-scheduler**

systemctl daemon-reload

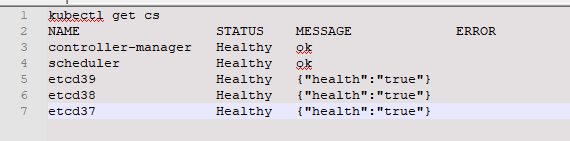
systemctl enable kube-scheduler

systemctl start kube-scheduler -l

systemctl status kube-scheduler -l

**验证master节点功能**

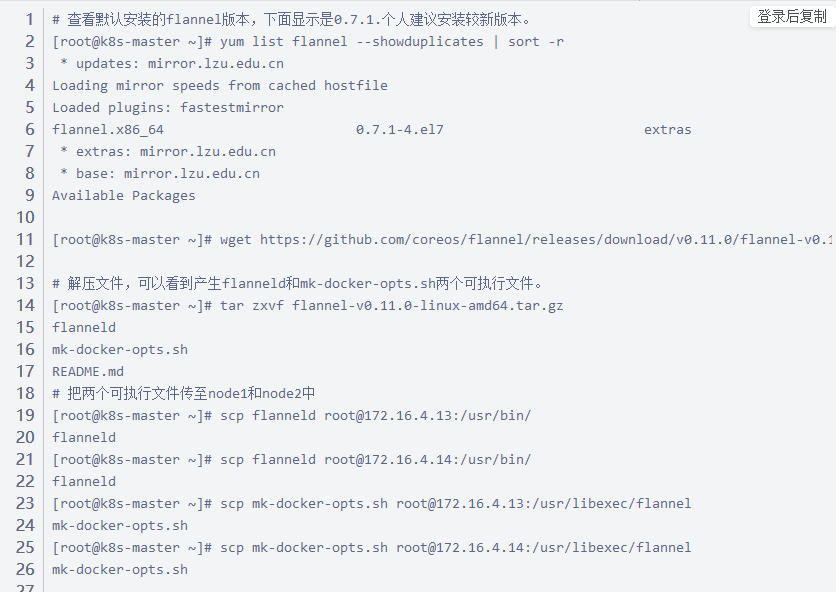
启动每个组件后可以通过执行命令kubectl get cs，来查看各个组件的状态;



**安装flannel网络插件**

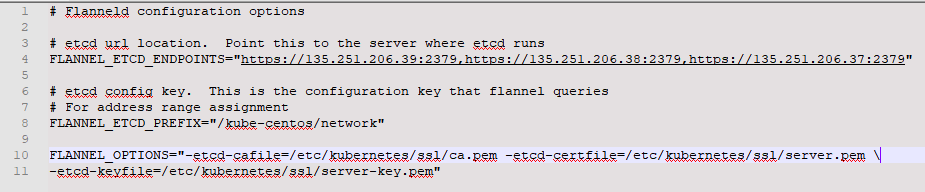
所有的node节点都需要安装网络插件才能让所有的Pod加入到同一个局域网中，本文是安装flannel网络插件的参考文档。

建议直接使用yum安装flanneld，除非对版本有特殊需求，默认安装的是0.7.1版本的flannel。



注意在node节点上一定要实现创建好盛放flanneld和mk-docker-opts.sh的目录

**/etc/sysconfig/flanneld配置文件**



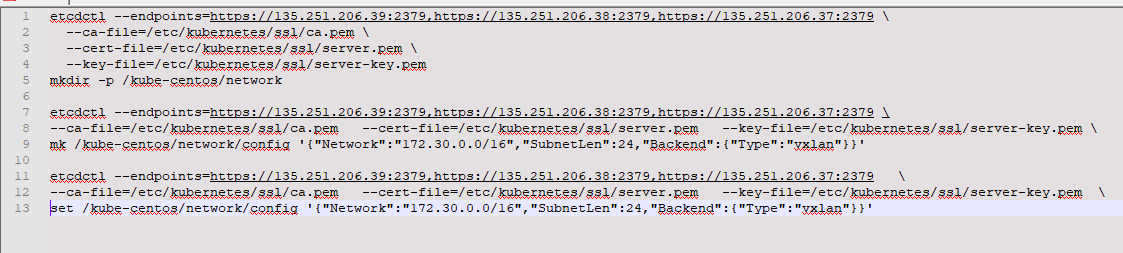
创建service配置文件/usr/lib/systemd/system/flanneld.service



注意如果是多网卡（例如vagrant环境），则需要在FLANNEL\_OPTIONS中增加指定的外网出口的网卡，例如-iface=eth1

在etcd中创建网络配置

执行下面的命令为docker分配IP地址段



**启动flannel**

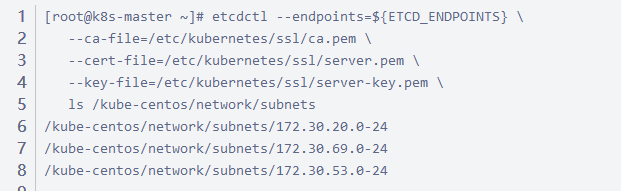
systemctl daemon-reload

systemctl enable flanneld

systemctl start flanneld

systemctl status flanneld

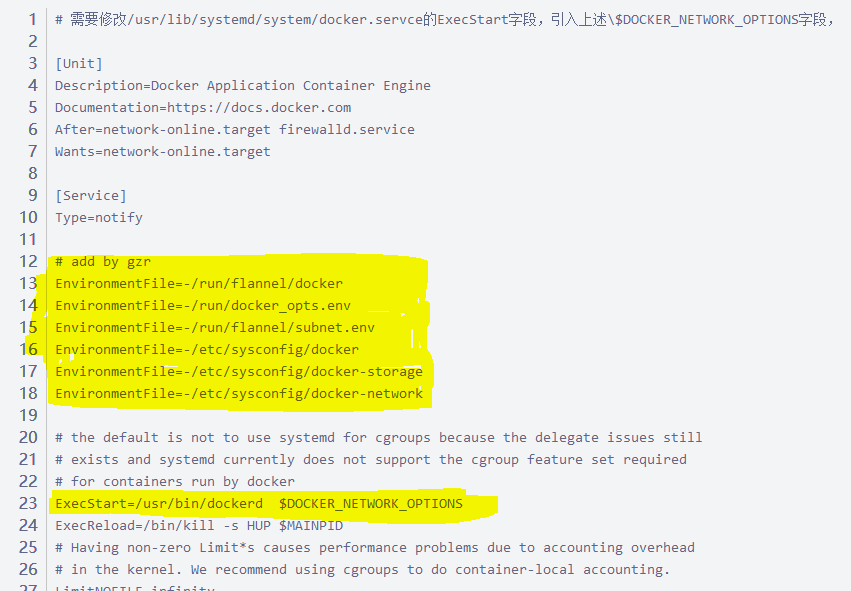
查询etcd中的内容

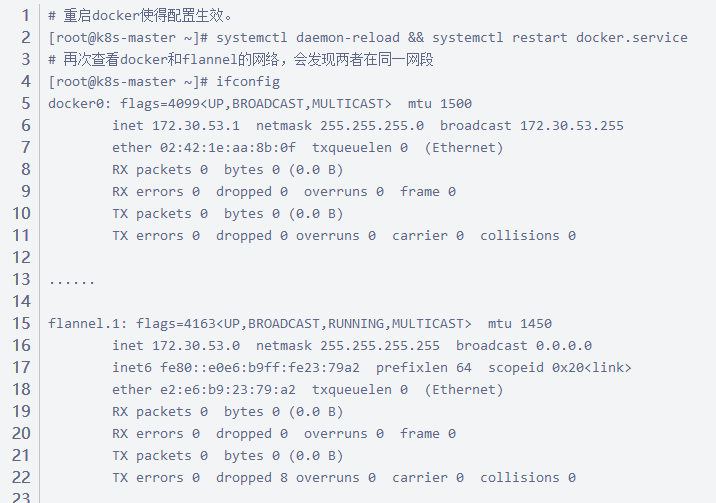


查看到flannel网络的信息



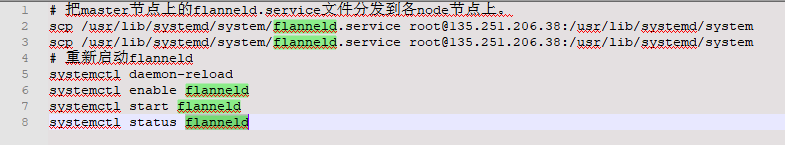
**将docker应用于flannel**

****

****

同理，可以参考以上配置应用到其他各节点上

**部署node节点**

****

**重新启动docker**

systemctl daemon-reload && systemctl restart docker

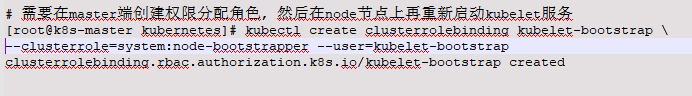
### 安装配置kubelet

kubelet 启动时向 kube-apiserver 发送 TLS bootstrapping 请求，需要先将 bootstrap token 文件中的 kubelet-bootstrap 用户赋予 system:node-bootstrapper cluster 角色(role)， 然后 kubelet 才能有权限创建认证请求(certificate signing requests)：

从master节点的/usr/local/bin将kubelet和kube-proxy文件传至各节点

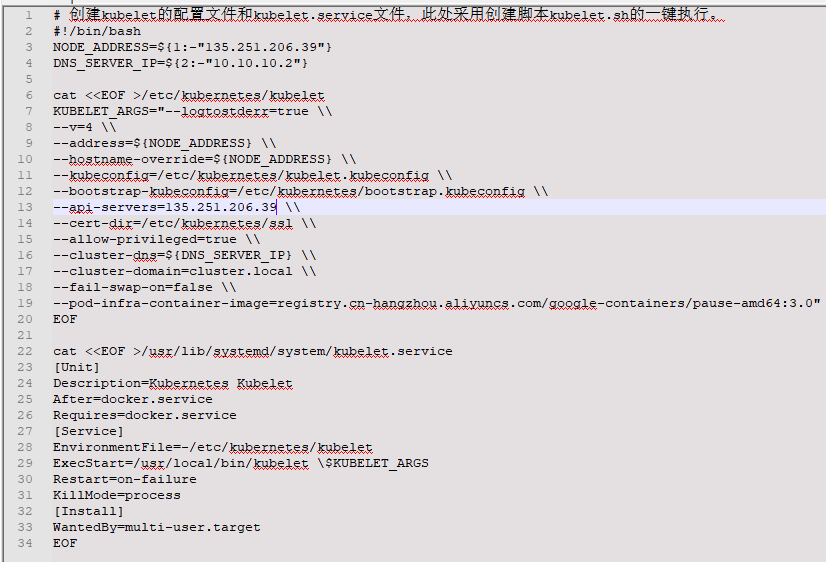
****

在master节点上创建角色

****

**创建kubelet服务**

**第一种方式：在所有节点上创建执行脚本**

****

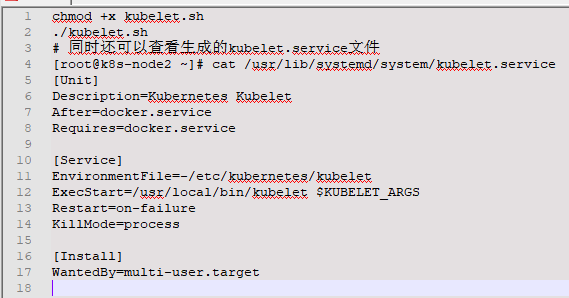
启动服务

systemctl daemon-reload

systemctl enable kubelet

systemctl restart kubelet && systemctl status kubelet

**执行脚本**

****

注意：在node1，node2上执行kubelet.sh脚本，传入135.251.206.38（node1 IP），135.251.206.37（node2 IP）和 10.10.10.2（DNS服务器IP）。在其他节点执行脚本时，记得替换相应的参数

**参数解析**

如果使用systemd方式启动，则需要额外增加两个参数--runtime-cgroups=/systemd/system.slice --kubelet-cgroups=/systemd/system.slice

 --address 不能设置为 127.0.0.1，否则后续 Pods 访问 kubelet 的 API 接口时会失败，因为 Pods 访问的 127.0.0.1 指向自己而不是 kubelet；

 "--cgroup-driver 配置成 systemd，不要使用cgroup，否则在 CentOS 系统中 kubelet 将启动失败（保持docker和kubelet中的cgroup driver配置一致即可，不一定非使用systemd）。

 --bootstrap-kubeconfig 指向 bootstrap kubeconfig 文件，kubelet 使用该文件中的用户名和 token 向 kube-apiserver 发送 TLS Bootstrapping 请求；

 管理员通过了 CSR 请求后，kubelet 自动在 --cert-dir 目录创建证书和私钥文件(kubelet-client.crt 和 kubelet-client.key)，然后写入 --kubeconfig 文件；

 建议在 --kubeconfig 配置文件中指定 kube-apiserver 地址，如果未指定 --api-servers 选项，则必须指定 --require-kubeconfig 选项后才从配置文件中读取 kube-apiserver 的地址，否则 kubelet 启动后将找不到 kube-apiserver (日志中提示未找到 API Server），kubectl get nodes 不会返回对应的 Node 信息; --require-kubeconfig 在1.10版本被移除，参看[PR](https://github.com/kubernetes/kops/pull/4357/commits/30b10cb1c8c9d8d67fdf6371f1fda952a2b02004)；

 --cluster-dns 指定 kubedns 的 Service IP(可以先分配，后续创建 kubedns 服务时指定该 IP)，--cluster-domain 指定域名后缀，这两个参数同时指定后才会生效；

 --cluster-domain 指定 pod 启动时 /etc/resolve.conf 文件中的 search domain ，起初我们将其配置成了 cluster.local.，这样在解析 service 的 DNS 名称时是正常的，可是在解析 headless service 中的 FQDN pod name 的时候却错误，因此我们将其修改为 cluster.local，去掉最后面的 ”点号“ 就可以解决该问题，关于 kubernetes 中的域名/服务名称解析请参见我的另一篇文章。

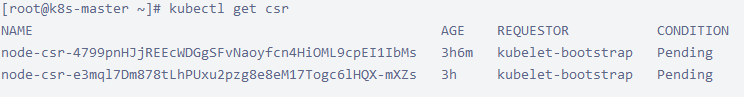
 --kubeconfig=/etc/kubernetes/kubelet.kubeconfig中指定的kubelet.kubeconfig文件在第一次启动kubelet之前并不存在，请看下文，当通过CSR请求后会自动生成kubelet.kubeconfig文件，如果你的节点上已经生成了~/.kube/config文件，你可以将该文件拷贝到该路径下，并重命名为kubelet.kubeconfig，所有node节点可以共用同一个kubelet.kubeconfig文件，这样新添加的节点就不需要再创建CSR请求就能自动添加到kubernetes集群中。同样，在任意能够访问到kubernetes集群的主机上使用kubectl --kubeconfig命令操作集群时，只要使用~/.kube/config文件就可以通过权限认证，因为这里面已经有认证信息并认为你是admin用户，对集群拥有所有权限。

 KUBELET\_POD\_INFRA\_CONTAINER 是基础镜像容器，这里我用的是私有镜像仓库地址，**大家部署的时候需要修改为自己的镜像**。可以使用Google的pause镜像gcr.io/google\_containers/pause-amd64:3.0，这个镜像只有300多K

**通过kubelet的TLS证书请求**

kubelet 首次启动时向 kube-apiserver 发送证书签名请求，必须通过后 kubernetes 系统才会将该 Node 加入到集群。

在master节点上查看未授权的CSR请求

****

通过CSR请求

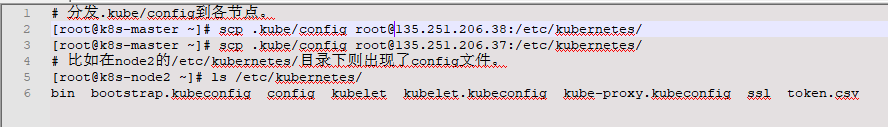
****

自动生成了 kubelet kubeconfig 文件和公私钥

假如你更新kubernetes的证书，只要没有更新token.csv，当重启kubelet后，该node就会自动加入到kuberentes集群中，而不会重新发送certificaterequest，也不需要在master节点上执行kubectl certificate approve操作。前提是不要删除node节点上的/etc/kubernetes/ssl/kubelet\*和/etc/kubernetes/kubelet.kubeconfig文件。否则kubelet启动时会提示找不到证书而失败。

****

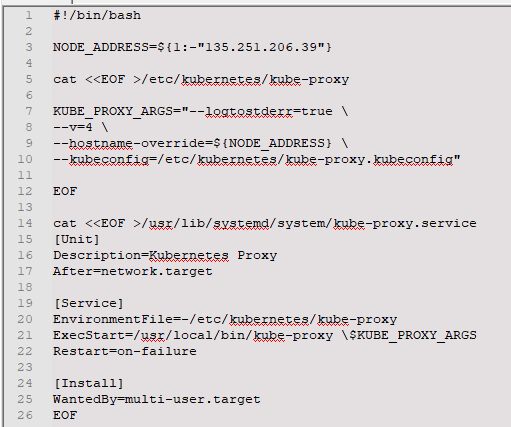
\*\*注意：\*\*如果启动kubelet的时候见到证书相关的报错，有个trick可以解决这个问题，可以将master节点上的~/.kube/config文件（该文件在[安装kubectl命令行工具]:这一步中将会自动生成）拷贝到node节点的/etc/kubernetes/kubelet.kubeconfig位置，这样就不需要通过CSR，当kubelet启动后就会自动加入的集群中。注意同时记得也把.kube/config中的内容复制粘贴到/etc/kubernetes/kubelet.kubeconfig中，替换原先内容。

****

### 配置kube-proxy

### 脚本方式配置

编写kube-proxy.sh脚本内容如下（在各node上编写该脚本）：

****

 --hostname-override 参数值必须与 kubelet 的值一致，否则 kube-proxy 启动后会找不到该 Node，从而不会创建任何 iptables 规则；

 kube-proxy 根据 --cluster-cidr 判断集群内部和外部流量，指定 --cluster-cidr 或 --masquerade-all 选项后 kube-proxy 才会对访问 Service IP 的请求做 SNAT；

 --kubeconfig 指定的配置文件嵌入了 kube-apiserver 的地址、用户名、证书、秘钥等请求和认证信息；

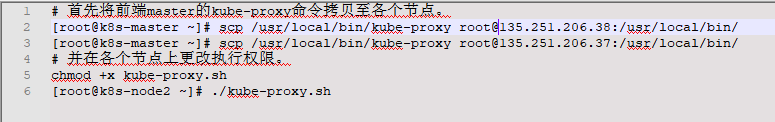
 预定义的 RoleBinding cluster-admin 将User system:kube-proxy 与 Role system:node-proxier 绑定，该 Role 授予了调用 kube-apiserver Proxy 相关 API 的权限

**启动服务**

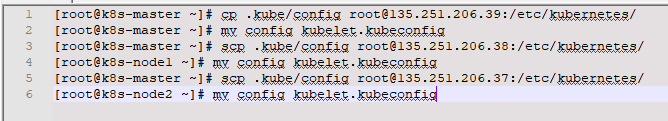
systemctl daemon-reload && systemctl enable kube-proxy

systemctl restart kube-proxy && systemctl status kube-proxy

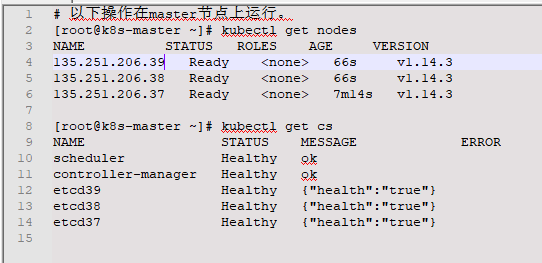
**执行脚本**

****

--kubeconfig=/etc/kubernetes/kubelet.kubeconfig中指定的kubelet.kubeconfig文件在第一次启动kubelet之前并不存在，请看下文，当通过CSR请求后会自动生成kubelet.kubeconfig文件，如果你的节点上已经生成了~/.kube/config文件，你可以将该文件拷贝到该路径下，并重命名为kubelet.kubeconfig，所有node节点可以共用同一个kubelet.kubeconfig文件，这样新添加的节点就不需要再创建CSR请求就能自动添加到kubernetes集群中。同样，在任意能够访问到kubernetes集群的主机上使用kubectl --kubeconfig命令操作集群时，只要使用~/.kube/config`文件就可以通过权限认证，因为这里面已经有认证信息并认为你是admin用户，对集群拥有所有权限。

****

**验证测试**

****

**DNS服务搭建与配置**

从k8s v1.11版本开始，Kubernetes集群的DNS服务由CoreDNS提供。它是CNCF基金会的一个项目，使用Go语言实现的高性能、插件式、易扩展的DNS服务端。

### 安装CoreDNS插件

在部署CoreDNS应用前，至少需要创建一个ConfigMap，一个Deployment和一个Service共3个资源对象。在启用了RBAC的集群中，还可以设置ServiceAccount、ClusterRole、ClusterRoleBinding对CoreDNS容器进行权限限制

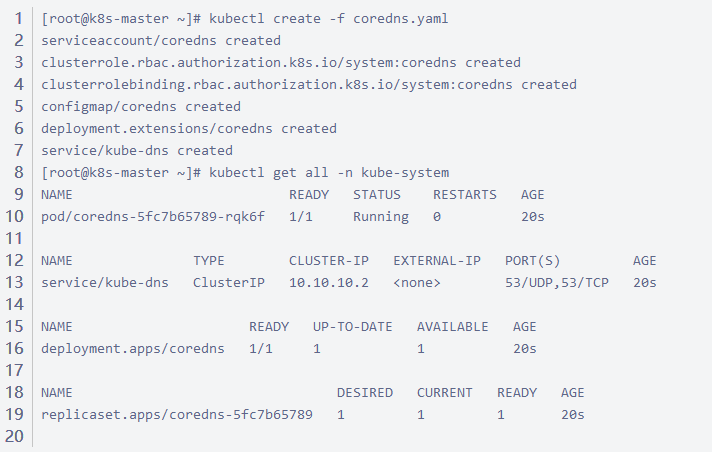
此处将svc,configmap,ServiceAccount等写在一个yaml文件里，coredns.yaml

可以从网址直接获取 <http://135.251.206.39/coredns.yaml>

clusterIP: 10.10.10.2是我集群各节点的DNS服务器IP，注意修改。并且在各node节点的kubelet的启动参数中加入以下两个参数：

* –cluster-dns=10.10.10.2：为DNS服务的ClusterIP地址。
* –cluster-domain=cluster.local：为在DNS服务中设置的域名

通过kubectl create -f coredns.yaml来创建服务

****

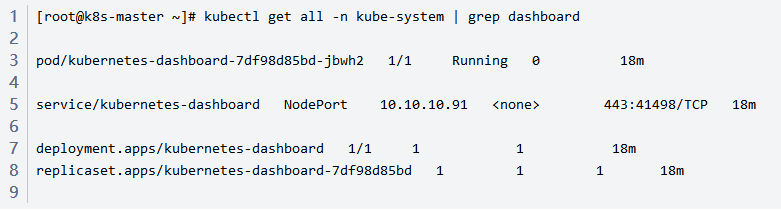
**安装dashboard插件**

Kubernetes的Web UI网页管理工具kubernetes-dashboard可提供部署应用、资源对象管理、容器日志查询、系统监控等常用的集群管理功能

创建部署文件kubernetes-dashboard.yaml，完整的文件参考：

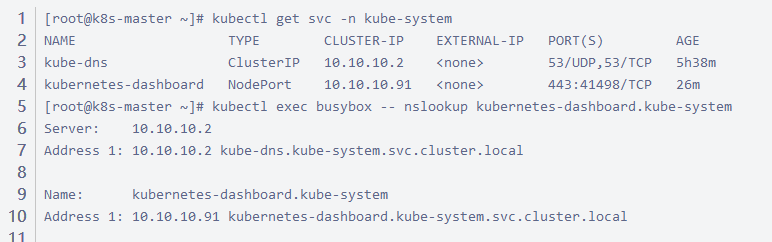
<http://135.251.206.39/dashboard.yaml>

**查看创建状态**

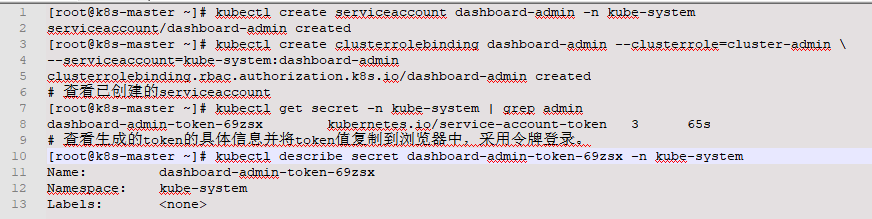
****

此时可以通过节点的41498端口进行访问。

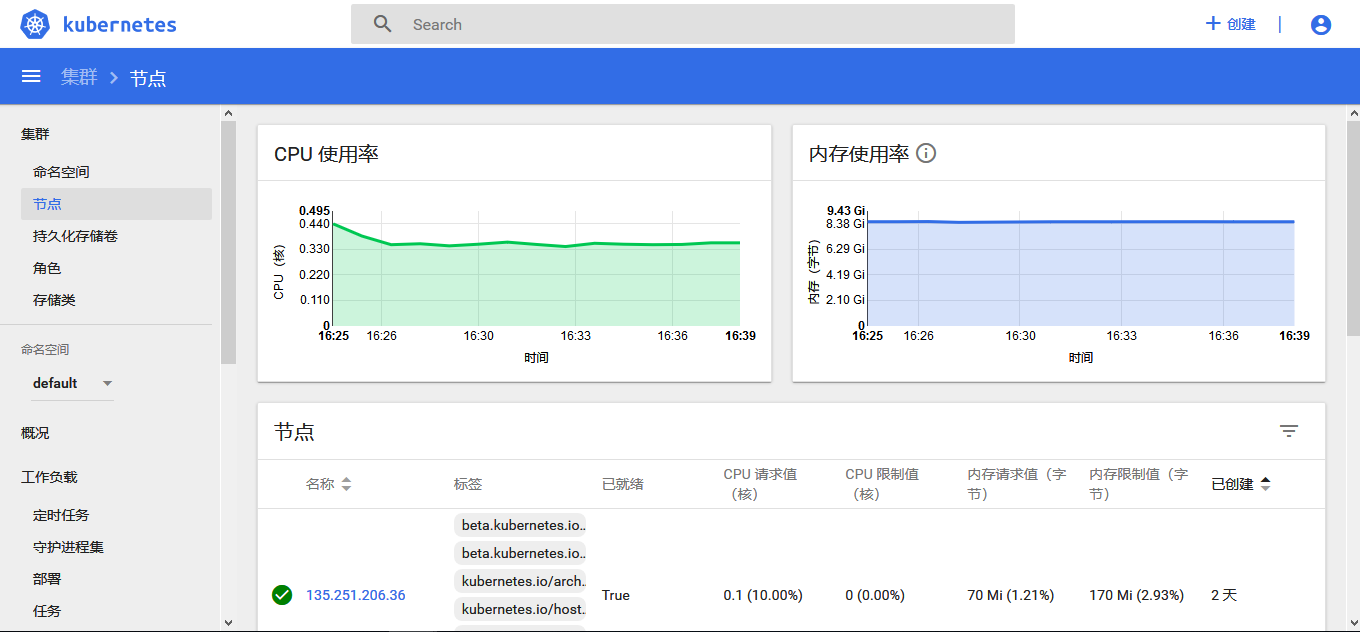
并且之前部署的CoreDNSf服务是能够解析到其服务的IP地址



创建SA并绑定cluster-admin管理员角色



在浏览器中选择token方式登录，即可查看到集群的状态：



访问dashboard实际上有三种方式，这里是第一种

* Kubernetes-dashboard服务暴露nodeport，即可使用<http://nodeip:nodeport> 进行访问

Kubectln kube-system describe secret $( kubectl -n kube-system get secret | grep dashboard | grep service-account | awk '{print $1}')

获取token，访问UI

* 通过api server 访问dashboard (https 6443端口 ，http 8080端口)
* 通过kubelet proxy 方式

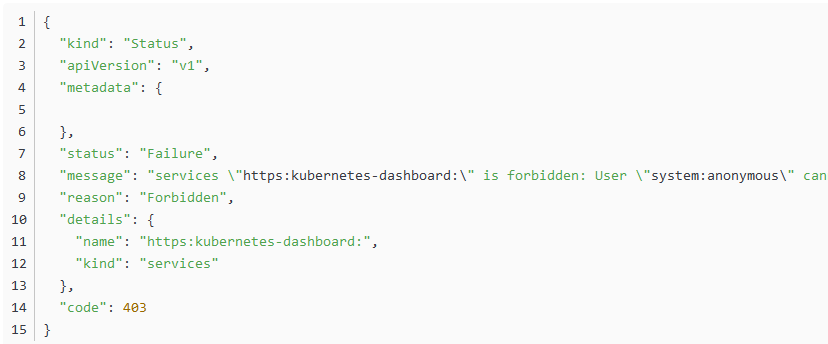
**采用kubelet proxy 方式访问dashboard**

1. **启用代理**

****

1. **访问dashboard**

**部署过程遇到的问题：**

****

**解决方案：**

**Kube-apiserver种新增了—anonymous-auth选项，允许匿名访问secure port，没有被authentication方法拒绝的请求就是anonymous requests，这样匿名请求的username是system:anonymous，归属的组为system:unauthenticated,并且这选项是默认的。所以只需要在kube-apiserver文件种设置—anonymous-auth=false即可**

****

**解决方案：**

**新建/etc/kubernetes/basic\_auth\_file文件，**

**内容格式：passwd,username,uid**

**Admin123,admin,1002**

**在kube-apiserver文件种设置**

**--basic-auth-file=/etc/kubernetes/basic\_auth\_file**

**重启服务**

**最后执行角色绑定命令**

**kubectl create clusterrolebinding login-dashboard-admin --clusterrole=cluster-admin --user=admin**

**再次刷新访问即可**

**部署metrics-server插件**

**源码获取：**

**git clone** [**https://github.com/kubernetes-incubator/metrics-server**](https://github.com/kubernetes-incubator/metrics-server)

**目前metrics-server的最新版本是v0.3.1,如果部署v0.2版本以上的进1.8+这个目录，如果是v0.2版本以下进入1.7这个目录，此次 我部署的是v0.3.1，所以进入1.8+目录**

1. 使用cfssl工具生成证书



1. 生成证书

**cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes metrics-server-csr.json | cfssljson -bare metrics-server**

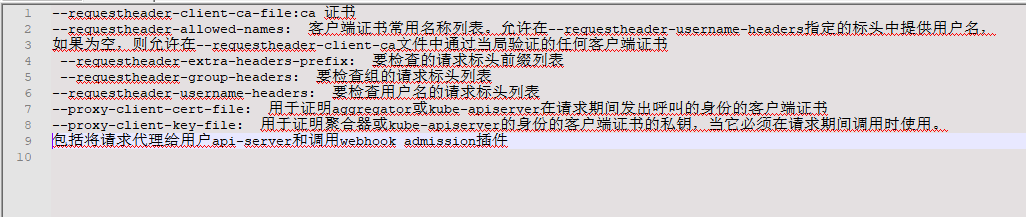
其中ca.pem ,ca-key.pem,ca-config.jsons是部署集群是生成的ca证书

将生成的证书copy到各个节点的证书目录下

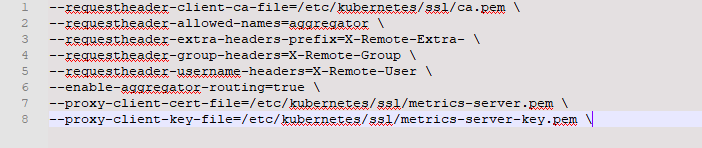
Cp metrics\*.prm /etc/kubernetes/ssl

1. 开启聚合层

设置kube-apiserver启动参数



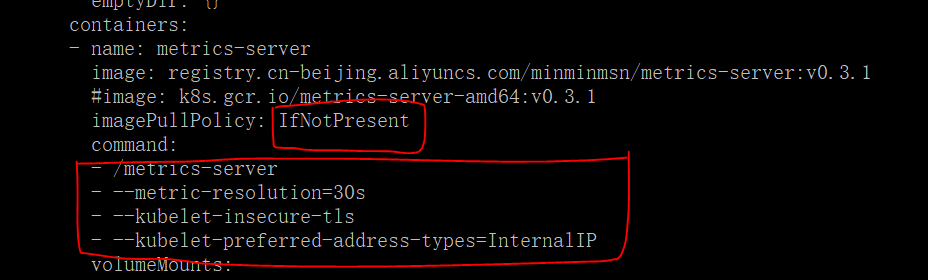
我的配置如下



重启kube-apiserver服务

**部署文件，最好提前load下所需要的镜像**

修改metrics-server-deployment.yaml文件



--kubelet-insecure-tls: 设置为不验证Kubelet 证书

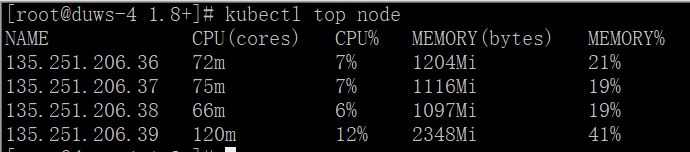
--kubelet-preferred-address-types=InternalIP:设置为能够解析节点的主机名

metrics-server连节点时默认是连接节点的主机名，需要加个参数，让它连接节点的IP，同时因为10250是https端口，连接它时需要提供证书，所以加上–kubelet-insecure-tls，表示不验证客户端证书，此前的版本中使用–source=这个参数来指定不验证客户端证书

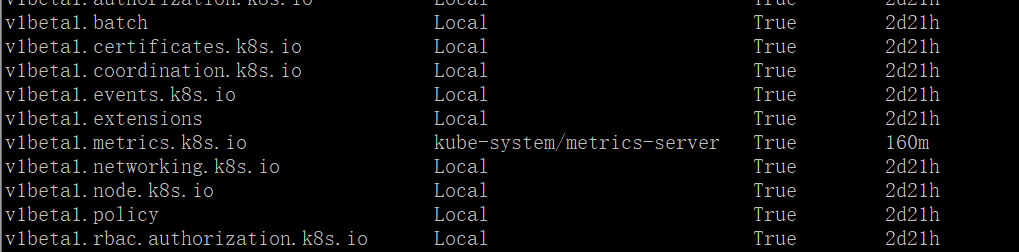
**部署：**

Kubelet create -f \*

**验证：**

****

**Kubectl get apiservice -n kube-system**



**此时表面metrics-server插件部署成功**

**通过kubelet命令行工具测试获取metrics-server数据**

**Yum install -y epel-release && Yum install -y jq**

**kubectl get --raw "/apis/metrics.k8s.io/v1beta1/nodes" | jq**



**部署k8s集群自动扩容与缩容**

实现调度平台能够根据服务的负载来智能的调控规模，进行弹性缩扩容。

K8S目前提供三个维度的autoscale

|  |  |  |
| --- | --- | --- |
| K8S | 节点 | Pods |
| 水平扩展 | Cluster autoscaler | HPA |
| 垂直扩展 | None | VPA |

K8S弹性伸缩分类：

1. 资源维度：保障集群资源池大小满足整体规划，当集群内部资源不足以支撑产出新的Pod,会触发边界进行扩容
2. 应用维度：保障应用的负载处在预期的容量规划内

对应的伸缩策略

1. 水平伸缩：

* 集群维度：自动调整资源池的规模(新增/减少worker节点)
* Pod维度：自动调整pod副本的数量

1. 垂直伸缩

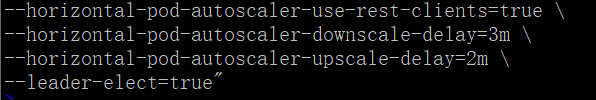
* 集群维度：不支持
* Pod维度：自动调整应用的资源分配(增加/减少cpu的数目，内存的占用)

目前最为成熟的策略即：HPA—水平Pod扩容。通过监控pod的CPU利用率和其他指标，自动调整Pod数量，但是有些类型是不支持的，比如daemonset，控制器会定期调整pod数量，判断的依据就是指定的指标，到达边界就会自动就行扩容/缩容pod数量。指标的收集就需要通过metrics来进行处理。

HPA（Horizontal Pod Autoscaler），基于观测CPU使用率(V1版本，v2beta版本支持memory或者其他自定义属性)。业务负载上升或超过HPA设定值，创建新的POD。

**周期**：

HPA控制器观测资源使用率并作出决策是有周期的，执行是需要时间的，在执行自动伸缩过程中metrics不是静止不变的，可能降低或者升高，如果执行太频繁可能导致资源的使用快速抖动，因此控制器每次决策后的一段时间内不再进行新的决策。对于扩容这个时间是3分钟，缩容则是5分钟，对应调整kube-controller-manager中如下参数



并且自动伸缩并不是一次性到位，而是逐渐逼近计算值，每次调整不超过当前副本数量的2倍或者1/2

**实例：**

1. **部署启动deployment文件**

**Wget** [**http://135.251.206.39/** **hpa-deployment.yaml**](http://135.251.206.39/%20hpa-deployment.yaml)

1. **部署service文件**

**Wget** [**http://135.251.206.39/** **hpa-deployment.yaml**](http://135.251.206.39/%20hpa-deployment.yaml)

1. **部署hpa文件**

**Wget** [**http://135.251.206.39/** **hpa-deployment.yaml**](http://135.251.206.39/%20hpa-deployment.yaml)

**压力测试，观察HPA效果**

* **生成一个压测客户端，持续压力测试**

kubectl run --generator=run-pod/v1 -i --tty load-generator --image=busybox /bin/sh

随着压力测试进行，deployment下pod的CPU使用率增加，超过HPA设定的百分比50%,之后逐次翻倍扩容replicaset。达到上限停止扩容。根据replicaset设置的request QoS逐渐稳定资源的使用率。

**停止压测**：

**while true; do wget -q -O- http://php-apache-svc.default.svc.cluster.local; done**