阿良技术博客：<https://blog.51cto.com/lizhenliang/2325770>

**官方提供的几种Kubernetes部署方式**

* **minikube**

Minikube是一个工具，可以在本地快速运行一个单点的Kubernetes，尝试Kubernetes或日常开发的用户使用。不能用于生产环境。

官方地址：<https://kubernetes.io/docs/setup/minikube/>

* **kubeadm**

Kubeadm也是一个工具，提供kubeadm init和kubeadm join，用于快速部署Kubernetes集群。

官方地址：<https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/>

* **二进制包**

从官方下载发行版的二进制包，手动部署每个组件，组成Kubernetes集群。

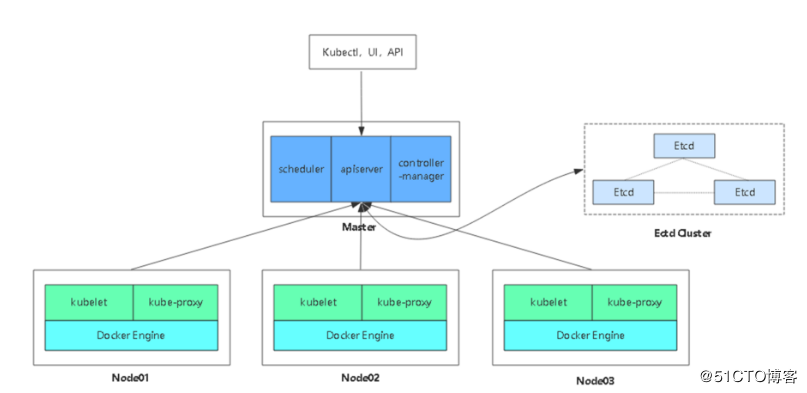
**小结：**  
生产环境中部署Kubernetes集群，只有Kubeadm和二进制包可选，Kubeadm降低部署门槛，但屏蔽了很多细节，遇到问题很难排查。我们这里使用二进制包部署Kubernetes集群，我也是推荐大家使用这种方式，虽然手动部署麻烦点，但学习很多工作原理，更有利于后期维护。

**软件环境**

| **软件** | **版本** |
| --- | --- |
| 操作系统 | CentOS7.5\_x64 |
| Docker | 18-ce |
| Kubernetes | 1.12 |

**服务器角色**

| **角色** | **IP** | **组件** |
| --- | --- | --- |
| k8s-master | 192.168.31.63 | kube-apiserver，kube-controller-manager，kube-scheduler，etcd |
| k8s-node1 | 192.168.31.65 | kubelet，kube-proxy，docker，flannel，etcd |
| k8s-node2 | 192.168.31.66 | kubelet，kube-proxy，docker，flannel，etcd |

  
​ 架构图

**1. 部署Etcd集群**

使用cfssl来生成自签证书，先下载cfssl工具：

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

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

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

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

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

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

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

**1.1 生成证书**

创建以下三个文件：

*# cat ca-config.json*

{

"signing": {

"default": {

"expiry": "87600h"

},

"profiles": {

"www": {

"expiry": "87600h",

"usages": [

"signing",

"key encipherment",

"server auth",

"client auth"

]

}

}

}

}

*# cat ca-csr.json*

{

"CN": "etcd CA",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "Beijing",

"ST": "Beijing"

}

]

}

*# cat server-csr.json*

{

"CN": "etcd",

"hosts": [

"192.168.31.63",

"192.168.31.65",

"192.168.31.66"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "BeiJing",

"ST": "BeiJing"

}

]

}

生成证书：

cfssl gencert -initca ca-csr.json | cfssljson -bare ca -

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

*# ls \*pem*

ca-key.pem ca.pem server-key.pem server.pem

证书这块知道怎么生成、怎么用即可，建议暂时不必过多研究。

**1.2 部署Etcd**

二进制包下载地址：<https://github.com/coreos/etcd/releases/tag/v3.2.12>

**以下部署步骤在规划的三个etcd节点操作一样，唯一不同的是etcd配置文件中的服务器IP要写当前的：**

解压二进制包：

**#** mkdir /opt/etcd/{bin,cfg,ssl} -p

**#** tar zxvf etcd-v3.2.12-linux-amd64.tar.gz

**#** mv etcd-v3.2.12-linux-amd64/{etcd,etcdctl} /opt/etcd/bin/

创建etcd配置文件：

*# cat /opt/etcd/cfg/etcd*

*#[Member]*

ETCD\_NAME="etcd01"

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

ETCD\_LISTEN\_PEER\_URLS="https://192.168.31.63:2380"

ETCD\_LISTEN\_CLIENT\_URLS="https://192.168.31.63:2379"

*#[Clustering]*

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="https://192.168.31.63:2380"

ETCD\_ADVERTISE\_CLIENT\_URLS="https://192.168.31.63:2379"

ETCD\_INITIAL\_CLUSTER="etcd01=https://192.168.31.63:2380,etcd02=https://192.168.31.65:2380,etcd03=https://192.168.31.66:2380"

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

ETCD\_INITIAL\_CLUSTER\_STATE="new"

* ETCD\_NAME 节点名称
* ETCD\_DATA\_DIR 数据目录
* ETCD\_LISTEN\_PEER\_URLS 集群通信监听地址
* ETCD\_LISTEN\_CLIENT\_URLS 客户端访问监听地址
* ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS 集群通告地址
* ETCD\_ADVERTISE\_CLIENT\_URLS 客户端通告地址
* ETCD\_INITIAL\_CLUSTER 集群节点地址
* ETCD\_INITIAL\_CLUSTER\_TOKEN 集群Token
* ETCD\_INITIAL\_CLUSTER\_STATE 加入集群的当前状态，new是新集群，existing表示加入已有集群

systemd管理etcd：

*# cat /usr/lib/systemd/system/etcd.service*

**[Unit]**

Description=Etcd Server

After=network.target

After=network-online.target

Wants=network-online.target

**[Service]**

Type=notify

EnvironmentFile=/opt/etcd/cfg/etcd

ExecStart=/opt/etcd/bin/etcd \

--name=${ETCD\_NAME} \

--data-dir=${ETCD\_DATA\_DIR} \

--listen-peer-urls=${ETCD\_LISTEN\_PEER\_URLS} \

--listen-client-urls=${ETCD\_LISTEN\_CLIENT\_URLS},http://127.0.0.1:2379 \

--advertise-client-urls=${ETCD\_ADVERTISE\_CLIENT\_URLS} \

--initial-advertise-peer-urls=${ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS} \

--initial-cluster=${ETCD\_INITIAL\_CLUSTER} \

--initial-cluster-token=${ETCD\_INITIAL\_CLUSTER\_TOKEN} \

--initial-cluster-state=new \

--cert-file=/opt/etcd/ssl/server.pem \

--key-file=/opt/etcd/ssl/server-key.pem \

--peer-cert-file=/opt/etcd/ssl/server.pem \

--peer-key-file=/opt/etcd/ssl/server-key.pem \

--trusted-ca-file=/opt/etcd/ssl/ca.pem \

--peer-trusted-ca-file=/opt/etcd/ssl/ca.pem

Restart=on-failure

LimitNOFILE=65536

**[Install]**

WantedBy=multi-user.target

把刚才生成的证书拷贝到配置文件中的位置：

**#** cp ca\*pem server\*pem /opt/etcd/ssl

启动并设置开启启动：

**#** systemctl start etcd

**#** systemctl enable etcd

都部署完成后，检查etcd集群状态：

**# /opt/etcd/bin/etcdctl \**

--ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem \

--endpoints="https://192.168.31.63:2379,https://192.168.31.65:2379,https://192.168.31.66:2379" \

cluster-health

member 18218cfabd4e0dea **is** healthy: got healthy result **from** https:*//192.168.31.63:2379*

member 541c1c40994c939b **is** healthy: got healthy result **from** https:*//192.168.31.65:2379*

member a342ea2798d20705 **is** healthy: got healthy result **from** https:*//192.168.31.66:2379*

cluster **is** healthy

如果输出上面信息，就说明集群部署成功。如果有问题第一步先看日志：/var/log/message 或 journalctl -u etcd

**2. 在Node安装Docker**

**#** yum install -y yum-utils device-mapper-persistent-data lvm2

**#** yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

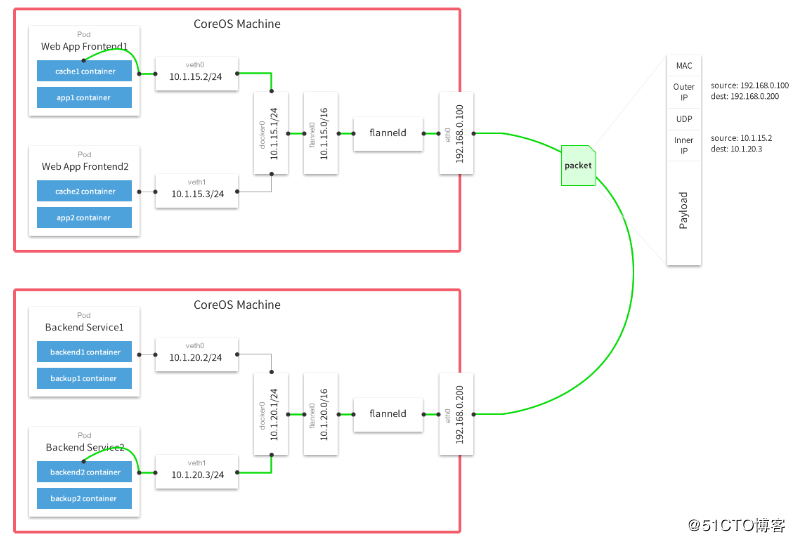
**#** yum install docker-ce -y

**#** curl -sSL https://get.daocloud.io/daotools/set\_mirror.sh | sh -s http://bc437cce.m.daocloud.io

**#** systemctl start docker

**#** systemctl enable docker

**3. 部署Flannel网络**

工作原理：  


Falnnel要用etcd存储自身一个子网信息，所以要保证能成功连接Etcd，写入预定义子网段：

*# /opt/etcd/bin/etcdctl \*

--ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem \

**--endpoints="https://192.168.31.63:2379,https://192.168.31.65:2379,https://192.168.31.66:2379" \**

set /coreos.com/network/config '{ "Network": "172.17.0.0/16", "Backend": {"Type": "vxlan"}}'

**以下部署步骤在规划的每个node节点都操作。**

下载二进制包：

**#** wget https://github.com/coreos/flannel/releases/download/v0.10.0/flannel-v0.10.0-linux-amd64.tar.gz

**#** tar zxvf flannel-v0.9.1-linux-amd64.tar.gz

**#** mv flanneld mk-docker-opts.sh /opt/kubernetes/bin

配置Flannel：

*# cat /opt/kubernetes/cfg/flanneld*

FLANNEL\_OPTIONS="--etcd-endpoints=https://192.168.31.63:2379,https://192.168.31.65:2379,https://192.168.31.66:2379 -etcd-cafile=/opt/etcd/ssl/ca.pem -etcd-certfile=/opt/etcd/ssl/server.pem -etcd-keyfile=/opt/etcd/ssl/server-key.pem"

systemd管理Flannel：

*# cat /usr/lib/systemd/system/flanneld.service*

**[Unit]**

Description=Flanneld overlay address etcd agent

After=network-online.target network.target

Before=docker.service

**[Service]**

Type=notify

EnvironmentFile=/opt/kubernetes/cfg/flanneld

ExecStart=/opt/kubernetes/bin/flanneld --ip-masq $FLANNEL\_OPTIONS

ExecStartPost=/opt/kubernetes/bin/mk-docker-opts.sh -k DOCKER\_NETWORK\_OPTIONS -d /run/flannel/subnet.env

Restart=on-failure

**[Install]**

WantedBy=multi-user.target

配置Docker启动指定子网段：

*# cat /usr/lib/systemd/system/docker.service*

**[Unit]**

Description=Docker Application Container Engine

Documentation=https://docs.docker.com

After=network-online.target firewalld.service

Wants=network-online.target

**[Service]**

Type=notify

EnvironmentFile=/run/flannel/subnet.env

ExecStart=/usr/bin/dockerd $DOCKER\_NETWORK\_OPTIONS

ExecReload=/bin/kill -s HUP $MAINPID

LimitNOFILE=infinity

LimitNPROC=infinity

LimitCORE=infinity

TimeoutStartSec=0

Delegate=yes

KillMode=process

Restart=on-failure

StartLimitBurst=3

StartLimitInterval=60s

**[Install]**

WantedBy=multi-user.target

重启flannel和docker：

**#** systemctl daemon-reload

**#** systemctl start flanneld

**#** systemctl enable flanneld

**#** systemctl restart docker

检查是否生效：

*# ps -ef |grep docker*

root 20941 1 1 Jun28 ? 09:15:34 /usr/bin/dockerd --bip=172.17.34.1/24 --ip-masq=false --mtu=1450

*# ip addr*

3607: flannel.1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue state UNKNOWN

link/ether 8a:2e:3d:09:dd:82 brd ff:ff:ff:ff:ff:ff

inet 172.17.34.0/32 scope global flannel.1

valid\_lft forever preferred\_lft forever

3608: docker0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue state UP

link/ether 02:42:31:8f:d3:02 brd ff:ff:ff:ff:ff:ff

inet 172.17.34.1/24 brd 172.17.34.255 scope global docker0

valid\_lft forever preferred\_lft forever

inet6 fe80::42:31ff:fe8f:d302/64 scope link

valid\_lft forever preferred\_lft forever

确保docker0与flannel.1在同一网段。  
测试不同节点互通，在当前节点访问另一个Node节点docker0 IP：

*# ping 172.17.58.1*

PING 172.17.58.1 (172.17.58.1) 56(84) bytes of data.

64 bytes from 172.17.58.1: icmp\_seq=1 ttl=64 time=0.263 ms

64 bytes from 172.17.58.1: icmp\_seq=2 ttl=64 time=0.204 ms

如果能通说明Flannel部署成功。如果不通检查下日志：journalctl -u flannel

**4. 在Master节点部署组件**

在部署Kubernetes之前一定要确保etcd、flannel、docker是正常工作的，否则先解决问题再继续。

**4.1 生成证书**

创建CA证书：

*# cat ca-config.json*

{

"signing": {

"default": {

"expiry": "87600h"

},

"profiles": {

"kubernetes": {

"expiry": "87600h",

"usages": [

"signing",

"key encipherment",

"server auth",

"client auth"

]

}

}

}

}

*# cat ca-csr.json*

{

"CN": "kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "Beijing",

"ST": "Beijing",

"O": "k8s",

"OU": "System"

}

]

}

*# cfssl gencert -initca ca-csr.json | cfssljson -bare ca -*

生成apiserver证书：

*# cat server-csr.json*

{

"CN": "kubernetes",

"hosts": [

"10.0.0.1",

"127.0.0.1",

"192.168.31.63",

"kubernetes",

"kubernetes.default",

"kubernetes.default.svc",

"kubernetes.default.svc.cluster",

"kubernetes.default.svc.cluster.local"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "BeiJing",

"ST": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

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

生成kube-proxy证书：

*# cat kube-proxy-csr.json*

{

"CN": "system:kube-proxy",

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "BeiJing",

"ST": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

*# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy*

最终生成以下证书文件：

# **ls** \***pem**

**ca-key**.pem **ca**.pem **kube-proxy-key**.pem **kube-proxy**.pem **server-key**.pem **server**.pem

**4.2 部署apiserver组件**

下载二进制包：<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.12.md>  
下载这个包（kubernetes-server-linux-amd64.tar.gz）就够了，包含了所需的所有组件。

**#** mkdir /opt/kubernetes/{bin,cfg,ssl} -p

**#** tar zxvf kubernetes-server-linux-amd64.tar.gz

**#** cd kubernetes/server/bin

**#** cp kube-apiserver kube-scheduler kube-controller-manager kubectl /opt/kubernetes/bin

创建token文件，用途后面会讲到：

*# cat /opt/kubernetes/cfg/token.csv*

674c457d4dcf2eefe4920d7dbb6b0ddc,kubelet-bootstrap,10001,"system:kubelet-bootstrap"

第一列：随机字符串，自己可生成  
第二列：用户名  
第三列：UID  
第四列：用户组

创建apiserver配置文件：

# cat /opt/kubernetes/cfg/kube-apiserver

KUBE\_APISERVER\_OPTS="--logtostderr=true \

--v=4 \

--etcd-servers=https://192.168.31.63:2379,https://192.168.31.65:2379,https://192.168.31.66:2379 \

--bind-address=192.168.31.63 \

--secure-port=6443 \

--advertise-address=192.168.31.63 \

--allow-privileged=true \

--service-cluster-ip-range=10.0.0.0/24 \

--enable-admission-plugins=NamespaceLifecycle,LimitRanger,SecurityContextDeny,ServiceAccount,ResourceQuota,NodeRestriction \

--authorization-mode=RBAC,Node \

--enable-bootstrap-token-auth \

--token-auth-file=/opt/kubernetes/cfg/token.csv \

--service-node-port-range=30000-50000 \

--tls-cert-file=/opt/kubernetes/ssl/server.pem \

--tls-private-key-file=/opt/kubernetes/ssl/server-key.pem \

--client-ca-file=/opt/kubernetes/ssl/ca.pem \

--service-account-key-file=/opt/kubernetes/ssl/ca-key.pem \

--etcd-cafile=/opt/etcd/ssl/ca.pem \

--etcd-certfile=/opt/etcd/ssl/server.pem \

--etcd-keyfile=/opt/etcd/ssl/server-key.pem"

配置好前面生成的证书，确保能连接etcd。

参数说明：

* --logtostderr 启用日志
* ---v 日志等级
* --etcd-servers etcd集群地址
* --bind-address 监听地址
* --secure-port https安全端口
* --advertise-address 集群通告地址
* --allow-privileged 启用授权
* --service-cluster-ip-range Service虚拟IP地址段
* --enable-admission-plugins 准入控制模块
* --authorization-mode 认证授权，启用RBAC授权和节点自管理
* --enable-bootstrap-token-auth 启用TLS bootstrap功能，后面会讲到
* --token-auth-file token文件
* --service-node-port-range Service Node类型默认分配端口范围

systemd管理apiserver：

*# cat /usr/lib/systemd/system/kube-apiserver.service*

**[Unit]**

Description=Kubernetes API Server

Documentation=https://github.com/kubernetes/kubernetes

**[Service]**

EnvironmentFile=-/opt/kubernetes/cfg/kube-apiserver

ExecStart=/opt/kubernetes/bin/kube-apiserver $KUBE\_APISERVER\_OPTS

Restart=on-failure

**[Install]**

WantedBy=multi-user.target

启动：

**#** systemctl daemon-reload

**#** systemctl enable kube-apiserver

**#** systemctl restart kube-apiserver

**4.3 部署scheduler组件**

创建schduler配置文件：

*# cat /opt/kubernetes/cfg/kube-scheduler*

KUBE\_SCHEDULER\_OPTS="--logtostderr=true \

--v=4 \

--master=127.0.0.1:8080 \

--leader-elect"

参数说明：

* --master 连接本地apiserver
* --leader-elect 当该组件启动多个时，自动选举（HA）

systemd管理schduler组件：

*# cat /usr/lib/systemd/system/kube-scheduler.service*

**[Unit]**

Description=Kubernetes Scheduler

Documentation=https://github.com/kubernetes/kubernetes

**[Service]**

EnvironmentFile=-/opt/kubernetes/cfg/kube-scheduler

ExecStart=/opt/kubernetes/bin/kube-scheduler $KUBE\_SCHEDULER\_OPTS

Restart=on-failure

**[Install]**

WantedBy=multi-user.target

启动：

**#** systemctl daemon-reload

**#** systemctl enable kube-scheduler

**#** systemctl restart kube-scheduler

**4.4 部署controller-manager组件**

创建controller-manager配置文件：

# cat /opt/kubernetes/cfg/kube-controller-manager

KUBE\_CONTROLLER\_MANAGER\_OPTS="--logtostderr=true \

--v=4 \

--master=127.0.0.1:8080 \

--leader-elect=true \

--address=127.0.0.1 \

--service-cluster-ip-range=10.0.0.0/24 \

--cluster-name=kubernetes \

--cluster-signing-cert-file=/opt/kubernetes/ssl/ca.pem \

--cluster-signing-key-file=/opt/kubernetes/ssl/ca-key.pem \

--root-ca-file=/opt/kubernetes/ssl/ca.pem \

--service-account-private-key-file=/opt/kubernetes/ssl/ca-key.pem"

systemd管理controller-manager组件：

*# cat /usr/lib/systemd/system/kube-controller-manager.service*

**[Unit]**

Description=Kubernetes Controller Manager

Documentation=https://github.com/kubernetes/kubernetes

**[Service]**

EnvironmentFile=-/opt/kubernetes/cfg/kube-controller-manager

ExecStart=/opt/kubernetes/bin/kube-controller-manager $KUBE\_CONTROLLER\_MANAGER\_OPTS

Restart=on-failure

**[Install]**

WantedBy=multi-user.target

启动：

**#** systemctl daemon-reload

**#** systemctl enable kube-controller-manager

**#** systemctl restart kube-controller-manager

所有组件都已经启动成功，通过kubectl工具查看当前集群组件状态：

*# /opt/kubernetes/bin/kubectl get cs*

NAME STATUS MESSAGE ERROR

scheduler Healthy ok

etcd-0 Healthy {"health":"true"}

etcd-2 Healthy {"health":"true"}

etcd-1 Healthy {"health":"true"}

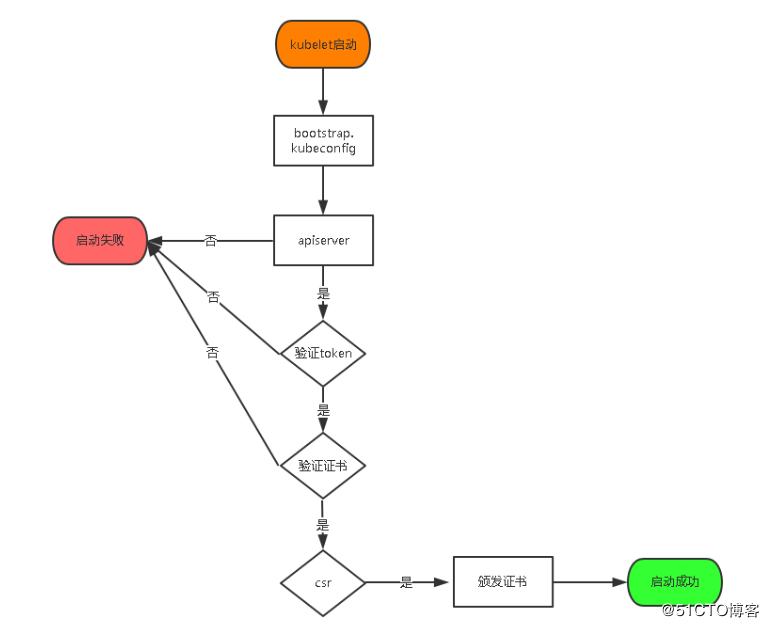
controller-manager Healthy ok

如上输出说明组件都正常。

**5. 在Node节点部署组件**

Master apiserver启用TLS认证后，Node节点kubelet组件想要加入集群，必须使用CA签发的有效证书才能与apiserver通信，当Node节点很多时，签署证书是一件很繁琐的事情，因此有了TLS Bootstrapping机制，kubelet会以一个低权限用户自动向apiserver申请证书，kubelet的证书由apiserver动态签署。

认证大致工作流程如图所示：



**5.1 将kubelet-bootstrap用户绑定到系统集群角色**

kubectl **create** clusterrolebinding kubelet-bootstrap \

*--clusterrole=system:node-bootstrapper \*

*--user=kubelet-bootstrap*

**5.2 创建kubeconfig文件**

在生成kubernetes证书的目录下执行以下命令生成kubeconfig文件：

**# 创建kubelet bootstrapping kubeconfig**

BOOTSTRAP\_TOKEN=674c457d4dcf2eefe4920d7dbb6b0ddc

KUBE\_APISERVER="https://192.168.31.63:6443"

**# 设置集群参数**

kubectl config **set**-cluster kubernetes \

--certificate-authority=./ca.pem \

--embed-certs=true \

--server=${KUBE\_APISERVER} \

--kubeconfig=bootstrap.kubeconfig

**# 设置客户端认证参数**

kubectl config **set**-credentials kubelet-bootstrap \

--token=${BOOTSTRAP\_TOKEN} \

--kubeconfig=bootstrap.kubeconfig

**# 设置上下文参数**

kubectl config **set**-context **default** \

--cluster=kubernetes \

--user=kubelet-bootstrap \

--kubeconfig=bootstrap.kubeconfig

**# 设置默认上下文**

kubectl config use-context **default** --kubeconfig=bootstrap.kubeconfig

**#----------------------**

**# 创建kube-proxy kubeconfig文件**

kubectl config **set**-cluster kubernetes \

--certificate-authority=./ca.pem \

--embed-certs=true \

--server=${KUBE\_APISERVER} \

--kubeconfig=kube-proxy.kubeconfig

kubectl config **set**-credentials kube-proxy \

--client-certificate=./kube-proxy.pem \

--client-key=./kube-proxy-key.pem \

--embed-certs=true \

--kubeconfig=kube-proxy.kubeconfig

kubectl config **set**-context **default** \

--cluster=kubernetes \

--user=kube-proxy \

--kubeconfig=kube-proxy.kubeconfig

kubectl config use-context **default** --kubeconfig=kube-proxy.kubeconfig

# **ls**

**bootstrap**.kubeconfig **kube-proxy**.kubeconfig

将这两个文件拷贝到Node节点/opt/kubernetes/cfg目录下。

**5.2 部署kubelet组件**

将前面下载的二进制包中的kubelet和kube-proxy拷贝到/opt/kubernetes/bin目录下。

**创建kubelet配置文件：**

# cat /opt/kubernetes/cfg/kubelet

KUBELET\_OPTS="--logtostderr=true \

--v=4 \

--hostname-override=192.168.31.65 \

--kubeconfig=/opt/kubernetes/cfg/kubelet.kubeconfig \

--bootstrap-kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig \

--config=/opt/kubernetes/cfg/kubelet.config \

--cert-dir=/opt/kubernetes/ssl \

--pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google-containers/pause-amd64:3.0"

参数说明：

* --hostname-override 在集群中显示的主机名
* --kubeconfig 指定kubeconfig文件位置，会自动生成
* --bootstrap-kubeconfig 指定刚才生成的bootstrap.kubeconfig文件
* --cert-dir 颁发证书存放位置
* --pod-infra-container-image 管理Pod网络的镜像

**其中/opt/kubernetes/cfg/kubelet.config配置文件如下：**

**kind: KubeletConfiguration**

**apiVersion: kubelet.config.k8s.io/v1beta1**

**address: 192.168.31.65**

**port: 10250**

**readOnlyPort: 10255**

**cgroupDriver: cgroupfs**

**clusterDNS: ["10.0.0.2"]**

**clusterDomain: cluster.local.**

**failSwapOn: false**

**authentication:**

anonymous:

enabled: true

**systemd管理kubelet组件：**

*# cat /usr/lib/systemd/system/kubelet.service*

**[Unit]**

Description=Kubernetes Kubelet

After=docker.service

Requires=docker.service

**[Service]**

EnvironmentFile=/opt/kubernetes/cfg/kubelet

ExecStart=/opt/kubernetes/bin/kubelet $KUBELET\_OPTS

Restart=on-failure

KillMode=process

**[Install]**

WantedBy=multi-user.target

启动：

**#** systemctl daemon-reload

**#** systemctl enable kubelet

**#** systemctl restart kubelet

**在Master审批Node加入集群：**

启动后还没加入到集群中，需要手动允许该节点才可以。  
在Master节点查看请求签名的Node：

**#** kubectl get csr

**#** kubectl certificate approve XXXXID

**#** kubectl get node

**5.3 部署kube-proxy组件**

创建kube-proxy配置文件：

# cat /opt/kubernetes/cfg/kube-proxy

KUBE\_PROXY\_OPTS="--logtostderr=true \

--v=4 \

--hostname-override=192.168.31.65 \

--cluster-cidr=10.0.0.0/24 \

--kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig"

systemd管理kube-proxy组件：

*# cat /usr/lib/systemd/system/kube-proxy.service*

**[Unit]**

Description=Kubernetes Proxy

After=network.target

**[Service]**

EnvironmentFile=-/opt/kubernetes/cfg/kube-proxy

ExecStart=/opt/kubernetes/bin/kube-proxy $KUBE\_PROXY\_OPTS

Restart=on-failure

**[Install]**

WantedBy=multi-user.target

启动：

**#** systemctl daemon-reload

**#** systemctl enable kube-proxy

**#** systemctl restart kube-proxy

Node2部署方式一样。

**6. 查看集群状态**

*# kubectl get node*

NAME STATUS ROLES AGE VERSION

**192.168.31.65 Ready <none>** 1d v1.12.0

**192.168.31.66 Ready <none>** 1d v1.12.0

*# kubectl get cs*

NAME STATUS MESSAGE ERROR

controller-manager Healthy ok

scheduler Healthy ok

etcd-2 Healthy {"health":"true"}

etcd-1 Healthy {"health":"true"}

etcd-0 Healthy {"health":"true"}

**7. 运行一个测试示例**

创建一个Nginx Web，测试集群是否正常工作：

**#** kubectl run nginx --image=nginx --replicas=3

**#** kubectl expose deployment nginx --port=88 --target-port=80 --type=NodePort

查看Pod，Service：

*# kubectl get pods*

NAME READY STATUS RESTARTS AGE

nginx-64f497f8fd-fjgt2 1/1 Running 3 1d

nginx-64f497f8fd-gmstq 1/1 Running 3 1d

nginx-64f497f8fd-q6wk9 1/1 Running 3 1d

*# kubectl get svc*

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

kubernetes ClusterIP 10.0.0.1 <none> 443/TCP 28d

nginx NodePort 10.0.0.175 <none> 88:38696/TCP 28d

访问集群中部署的Nginx，打开浏览器输入：[http://192.168.31.66:38696](http://192.168.31.66:38696/)

