# Kubernetes安装手册

本文档介绍Kubernetes1.6.x集群的安装

## 集群详情

* Kubernetes 1.6.0
* Docker 1.12.5（使用yum安装）
* Etcd 3.1.5
* Flanneld 0.7 vxlan 网络
* TLS 认证通信 (所有组件，如 etcd、kubernetes master 和 node)
* RBAC 授权
* kublet TLS BootStrapping
* kubedns、dashboard、
* heapster(influxdb、grafana)、监控模块
* EFK(elasticsearch、fluentd、kibana) 日志采集模块
* 私有docker镜像仓库harbor

## 参考博客

本安装手册主要参考了开源书[Kubernetes handbook](https://rootsongjc.gitbooks.io/kubernetes-handbook/content/)中的最佳实践部分，[青蛙小白](http://blog.frognew.com)的博客也给了我很多帮助，在此表示感谢。在实践中，结合自己在安装中遇到的一些坑，进行了适当的修改，得到这篇手册，以方便自己日后的使用。文档还没有完善好，近期可能会有一些变动。

# 安装Docker

使用[阿里云加速器](https://cr.console.aliyun.com)安装  
1. 确保本机没有旧版本的docker

|  |
| --- |
| sudo yum remove docker docker-engine docker.io 1. 安装docker-engine, docker-ce curl -sSL http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/docker-engine/internet | sh - |

3. 添加镜像

|  |
| --- |
| sudo mkdir -p /etc/docker sudo tee /etc/docker/daemon.json <<-'EOF' {  "registry-mirrors": ["https://8difvy5w.mirror.aliyuncs.com"] } EOF sudo systemctl daemon-reload sudo systemctl restart docker |

# 创建 CA 证书和秘钥

kubernetes 系统各组件需要使用 TLS 证书对通信进行加密，本文档使用 CloudFlare 的 PKI 工具集 [cfssl](https://github.com/cloudflare/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。

## 安装 CFSSL

|  |
| --- |
| $ wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64 $ chmod +x cfssl\_linux-amd64 $ sudo mv cfssl\_linux-amd64 /root/local/bin/cfssl  $ wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64 $ chmod +x cfssljson\_linux-amd64 $ sudo mv cfssljson\_linux-amd64 /root/local/bin/cfssljson  $ wget https://pkg.cfssl.org/R1.2/cfssl-certinfo\_linux-amd64 $ chmod +x cfssl-certinfo\_linux-amd64 $ sudo mv cfssl-certinfo\_linux-amd64 /root/local/bin/cfssl-certinfo  $ export PATH=/root/local/bin:$PATH $ mkdir ssl$ cd ssl $ cfssl print-defaults ca-config > ca-config.json $ cfssl print-defaults ca-csr > ca-csr.json $ |

## 创建 CA (Certificate Authority)

创建 CA 配置文件：

|  |
| --- |
| $ cat ca-config.json {  "signing": {  "default": {  "expiry": "8760h"  },  "profiles": {  "kubernetes": {  "usages": [  "signing",  "key encipherment",  "server auth",  "client auth"  ],  "expiry": "8760h"  }  }  } } |

* 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：

|  |
| --- |
| $ cat ca-csr.json {  "CN": "kubernetes",  "key": {  "algo": "rsa",  "size": 2048  },  "names": [  {  "C": "CN",  "ST": "BeiJing",  "L": "BeiJing",  "O": "k8s",  "OU": "System"  }  ] } |

* "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：

|  |
| --- |
| $ cfssl gencert -initca ca-csr.json | cfssljson -bare ca $ ls ca\* ca-config.json ca.csr ca-csr.json ca-key.pem ca.pem |

## 分发证书

将生成的 CA 证书、秘钥文件、配置文件拷贝到**所有机器**的 /etc/kubernetes/ssl 目录下

|  |
| --- |
| $ sudo mkdir -p /etc/kubernetes/ssl $ sudo cp ca\* /etc/kubernetes/ssl |

## 校验证书

以校验 kubernetes 证书(后续部署 master 节点时生成的)为例：

### 使用 openssl 命令

|  |
| --- |
| $ openssl x509 -noout -text -in kubernetes.pem ...  Signature Algorithm: sha256WithRSAEncryption  Issuer: C=CN, ST=BeiJing, L=BeiJing, O=k8s, OU=System, CN=Kubernetes  Validity  Not Before: Apr 5 05:36:00 2017 GMT  Not After : Apr 5 05:36:00 2018 GMT  Subject: C=CN, ST=BeiJing, L=BeiJing, O=k8s, OU=System, CN=kubernetes ...  X509v3 extensions:  X509v3 Key Usage: critical  Digital Signature, Key Encipherment  X509v3 Extended Key Usage:  TLS Web Server Authentication, TLS Web Client Authentication  X509v3 Basic Constraints: critical  CA:FALSE  X509v3 Subject Key Identifier:  DD:52:04:43:10:13:A9:29:24:17:3A:0E:D7:14:DB:36:F8:6C:E0:E0  X509v3 Authority Key Identifier:  keyid:44:04:3B:60:BD:69:78:14:68:AF:A0:41:13:F6:17:07:13:63:58:CD   X509v3 Subject Alternative Name:  DNS:kubernetes, DNS:kubernetes.default, DNS:kubernetes.default.svc, DNS:kubernetes.default.svc.cluster, DNS:kubernetes.default.svc.cluster.local, IP Address:127.0.0.1, IP Address:10.64.3.7, IP Address:10.254.0.1 ... |

* 确认 Issuer 字段的内容和 ca-csr.json 一致；
* 确认 Subject 字段的内容和 kubernetes-csr.json 一致；
* 确认 X509v3 Subject Alternative Name 字段的内容和 kubernetes-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；

### 使用 cfssl-certinfo 命令

|  |
| --- |
| $ cfssl-certinfo -cert kubernetes.pem ... {  "subject": {  "common\_name": "kubernetes",  "country": "CN",  "organization": "k8s",  "organizational\_unit": "System",  "locality": "BeiJing",  "province": "BeiJing",  "names": [  "CN",  "BeiJing",  "BeiJing",  "k8s",  "System",  "kubernetes"  ]  },  "issuer": {  "common\_name": "Kubernetes",  "country": "CN",  "organization": "k8s",  "organizational\_unit": "System",  "locality": "BeiJing",  "province": "BeiJing",  "names": [  "CN",  "BeiJing",  "BeiJing",  "k8s",  "System",  "Kubernetes"  ]  },  "serial\_number": "174360492872423263473151971632292895707129022309",  "sans": [  "kubernetes",  "kubernetes.default",  "kubernetes.default.svc",  "kubernetes.default.svc.cluster",  "kubernetes.default.svc.cluster.local",  "127.0.0.1",  "192.168.202.131",  "192.168.202.132",  "192.168.202.133",  "10.254.0.1"  ],  "not\_before": "2017-04-05T05:36:00Z",  "not\_after": "2018-04-05T05:36:00Z",  "sigalg": "SHA256WithRSA", ... |

# 2、Etcd集群安装

## 环境信息

|  |
| --- |
| Centos 7  192.168.202.131 node1 192.168.202.132 node2 192.168.202.133 node3 |

## TLS密钥和证书

部署的etcd集群使用TLS证书对集群中节点间通信进行加密，并开启基于CA根证书签名的双向数字证书认证。本文档使用[cfssl](https://github.com/cloudflare/cfssl)来生成CA证书以及其他需要的证书。生成的证书列表如下：

* ca.pem
* etcd.pem
* etcd-key.pem

下面介绍使用cfssl生成所需要的私钥和证书.

## 安装[cfssl](https://github.com/cloudflare/cfssl)

### 方式一：直接使用二进制包安装

|  |
| --- |
| $ wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64 $ chmod +x cfssl\_linux-amd64 $ sudo mv cfssl\_linux-amd64 /root/local/bin/cfssl  $ wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64 $ chmod +x cfssljson\_linux-amd64 $ sudo mv cfssljson\_linux-amd64 /root/local/bin/cfssljson  $ wget https://pkg.cfssl.org/R1.2/cfssl-certinfo\_linux-amd64 $ chmod +x cfssl-certinfo\_linux-amd64 $ sudo mv cfssl-certinfo\_linux-amd64 /root/local/bin/cfssl-certinfo  $ export PATH=/root/local/bin:$PATH |

### 方式二：使用go命令安装

如果系统中安装过Go的话，可以直接使用命令安装

|  |
| --- |
| $ go get -u github.com/cloudflare/cfssl/cmd/... $ echo $GOPATH /usr/local $ ls /usr/local/bin/cfssl\* cfssl cfssl-bundle cfssl-certinfo cfssljson cfssl-newkey cfssl-scan |

## 创建CA证书

### 创建CA的配置文件ca-config.json

|  |
| --- |
| $ mkdir /root/ssl $ cd /root/ssl $ cfssl print-defaults config > ca-config.json $ cfssl print-defaults csr > ca-csr.json $ cat ca-config.json {  "signing": {  "default": {  "expiry": "8760h"  },  "profiles": {  "frognew": {  "usages": [  "signing",  "key encipherment",  "server auth",  "client auth"  ],  "expiry": "87600h"  }  }  } } |

* ca-config.json中可以定义多个profile，分别设置不同的expiry和usages等参数。如上面的ca-config.json中定义了名称为frognew的profile，这个profile的expiry 87600h为10年，useages中：
* signing表示此CA证书可以用于签名其他证书，ca.pem中的CA=TRUE
* server auth表示TLS Server Authentication, 即client可以用该 CA 对server提供的证书进行验证
* client auth表示TLS Client Authentication，即server可以用该CA对client提供的证书进行验证

### 创建CA证书签名请求配置ca-csr.json：

|  |
| --- |
| {  "CN": "frognew",  "key": {  "algo": "rsa",  "size": 2048  },  "names": [  {  "C": "CN",  "ST": "BeiJing",  "L": "BeiJing",  "O": "frognew",  "OU": "cloudnative"  }  ] } |

下面使用cfssl生成CA证书和私钥:

|  |
| --- |
| $ cfssl gencert -initca ca-csr.json | cfssljson -bare ca $ ls ca\* ca-config.json ca.csr ca-csr.json ca-key.pem ca.pem |

## Etcd证书和私钥

创建etcd证书签名请求配置etcd-csr.json：

|  |
| --- |
| {  "CN": "frognew",  "hosts": [  "127.0.0.1",  "192.168.202.131",  "192.168.202.132",  "192.168.202.133",  "node1",  "node2",  "node3"  ],  "key": {  "algo": "rsa",  "size": 2048  },  "names": [  {  "C": "CN",  "ST": "BeiJing",  "L": "BeiJing",  "O": "frognew",  "OU": "cloudnative"  }  ] } |

**注意：**上面配置hosts字段中制定授权使用该证书的IP和域名列表，因为现在要生成的证书需要被etcd集群各个节点使用，所以这里指定了各个节点的IP和hostname。

下面生成etcd的证书和私钥：

|  |
| --- |
| $ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=frognew etcd-csr.json | cfssljson -bare etcd  $ ls etcd\* etcd.csr etcd-csr.json etcd-key.pem etcd.pem |

对生成的证书可以使用cfssl或者openssl查看：

|  |
| --- |
| $ cfssl-certinfo -cert etcd.pem  $ openssl x509 -noout -text -in etcd.pem |

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

## 安装etcd Etcd

可以使用二进制安装和yum源安装两种方式

### 二进制安装

|  |
| --- |
| wget <https://github.com/coreos/etcd/releases/download/v3.1.6/etcd-v3.1.6-linux-amd64.tar.gz> |

* 解压缩etcd-v3.1.6-linux-amd64.tar.gz，将其中的etcd和etcdctl两个可执行文件复制到各节点的/usr/bin目录。

### yum源安装

|  |
| --- |
| $ yum list etcd $ yum install -y etcd |

安装完成之后，在各节点创建etcd的数据目录

|  |
| --- |
| mkdir -p /var/lib/etcd |

使用systemctl启动和管理etcd服务，在每个节点上创建etcd的systemd unit文件/usr/lib/systemd/system/etcd.service，注意替换ETCD\_NAME和INTERNAL\_IP变量的值：

|  |
| --- |
| $ export ETCD\_NAME=node1 $ export INTERNAL\_IP=192.168.202.131 $ 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 WorkingDirectory=/var/lib/etcd/ EnvironmentFile=-/etc/etcd/etcd.conf ExecStart=/usr/bin/etcd \  --name ${ETCD\_NAME} \  --cert-file=/etc/etcd/ssl/etcd.pem \  --key-file=/etc/etcd/ssl/etcd-key.pem \  --peer-cert-file=/etc/etcd/ssl/etcd.pem \  --peer-key-file=/etc/etcd/ssl/etcd-key.pem \  --trusted-ca-file=/etc/etcd/ssl/ca.pem \  --peer-trusted-ca-file=/etc/etcd/ssl/ca.pem \  --initial-advertise-peer-urls https://${INTERNAL\_IP}:2380 \  --listen-peer-urls https://${INTERNAL\_IP}:2380 \  --listen-client-urls https://${INTERNAL\_IP}:2379,https://127.0.0.1:2379 \  --advertise-client-urls https://${INTERNAL\_IP}:2379 \  --initial-cluster-token etcd-cluster-1 \  --initial-cluster node1=https://192.168.202.131:2380,node2=https://192.168.202.132:2380,node3=https://192.168.202.133:2380 \  --initial-cluster-state new \  --data-dir=/var/lib/etcd Restart=on-failure RestartSec=5 LimitNOFILE=65536  [Install] WantedBy=multi-user.target |

* --data-dir指定了etcd的工作目录和数据目录是/var/lib/etcd
* --cert-file和--key-file分别指定etcd的公钥证书和私钥
* --peer-cert-file和--peer-key-file分别指定了etcd的Peers通信的公钥证书和私钥。
* --trusted-ca-file指定了客户端的CA证书
* --peer-trusted-ca-file指定了Peers的CA证书
* --initial-cluster-state new表示这是新初始化集群，--name指定的参数值必须在--initial-cluster中

**注意**：在etcd.pem生成时hosts配置了Ip地址列表和hostname列表，在etcd的service(/usr/lib/systemd/system/etcd.service)文件中，所有ip不能代替为未包含的hostname，如master ## 启动Etcd 在各节点上启动etcd：

|  |
| --- |
| $ systemctl daemon-reload $ systemctl enable etcd $ systemctl start etcd $ systemctl status etcd |

在启动etcd的时候，可以开启另一个命令窗口，查看启动日志，确保没有报错

|  |
| --- |
| journalctl –f |

* 如果出现了形如 unkown flag的字段，表示启动参数错误，不识别,说明该参数拼写错误(*如--keyfile应当为--key-file*),可以到官方配置文档[Configuration flags](https://github.com/coreos/etcd/blob/master/Documentation/op-guide/configuration.md)查看该参数的写法，确保正确。
* 如果出现Failed to find member fXXXXXX的错误，这说明之前启动的etcd时，标识号出现错误，此时删除/var/lib/etcd/member目录，让etcd重新为每个节点分配标识号, /var/lib/etcd为etcd启动配置工作目录

如果日志一切正常，可以使用etcdctl检查集群是否健康，在任一节点执行：

|  |
| --- |
| $ etcdctl \  --ca-file=/etc/etcd/ssl/ca.pem \  --cert-file=/etc/etcd/ssl/etcd.pem \  --key-file=/etc/etcd/ssl/etcd-key.pem \  --endpoints=https://node1:2379,https://node2:2379,https://node3:2379 \  cluster-health   2017-04-24 19:53:40.545148 I | warning: ignoring ServerName for user-provided CA for backwards compatibility is deprecated 2017-04-24 19:53:40.546127 I | warning: ignoring ServerName for user-provided CA for backwards compatibility is deprecated member 4f2f99d70000fc19 is healthy: got healthy result from https://192.168.202.132:2379 member 99a756f799eb4163 is healthy: got healthy result from https://192.168.202.131:2379 member a9aff19397de2e4e is healthy: got healthy result from https://192.168.202.133:2379 cluster is healthy |

确保输出cluster is healthy的信息。 上面的命令使用证书访问，返回正常信息，若未添加证书，使用etcdctl member list访问，应当报错，否则，TLS(安全认证)未生效，即使用http访问etcd集群。

## etcdctl配置

由于使用了TLS安全认证，etcdctl 查询时需要在命令行中指定证书和endpoints，会使得一条命令变得很长，可以预先创建一个etcdctl配置文件，进行相应的配置.

1. 创建etcdctl配置文件

|  |
| --- |
| * $ vi /etc/etcd/etcdctl $ cat /etc/etcd/etcdctl ETCDCTL\_ENDPOINT="https://node1:2379,https://node2:2379,https://node3:2379" ETCDCTL\_CACERT=/etc/etcd/ssl/ca.pem ETCDCTL\_CERT=/etc/etcd/ssl/etcd.pem ETCDCTL\_KEY=/etc/etcd/ssl/etcd-key.pem |

1. 使配置文件生效

|  |
| --- |
| $ source /etc/etcd/etcdctl |

1. 查看集群状态

|  |
| --- |
| $ etcdctl cluster-health 2017-04-24 19:53:40.545148 I | warning: ignoring ServerName for user-provided CA for backwards compatibility is deprecated 2017-04-24 19:53:40.546127 I | warning: ignoring ServerName for user-provided CA for backwards compatibility is deprecated member 4f2f99d70000fc19 is healthy: got healthy result from https://192.168.202.132:2379 member 99a756f799eb4163 is healthy: got healthy result from https://192.168.202.131:2379 member a9aff19397de2e4e is healthy: got healthy result from https://192.168.202.133:2379 cluster is healthy |

etcdctl配置的本质是定义ETCDCTL\_ENDPOINT常量，etcdctl运行时读取该常量值，进行连接，具体的常量名称可以参考官方的配置说明[etcdctl config](https://github.com/coreos/etcd/tree/master/etcdctl)

## 参考

* [Clustering Guide](https://github.com/coreos/etcd/blob/master/Documentation/op-guide/clustering.md)
* [etcdctl config](https://github.com/coreos/etcd/tree/master/etcdctl)
* [cloudflare/cfssl](https://github.com/cloudflare/cfssl)

# 部署 kubectl 命令行工具

kubectl 默认从 ~/.kube/config 配置文件获取访问 kube-apiserver 地址、证书、用户名等信息，如果没有配置该文件，执行命令时出错：

|  |
| --- |
| $ kubectl get pods The connection to the server localhost:8080 was refused - did you specify the right host or port? |

本文档介绍下载和配置 kubernetes 集群命令行工具 kubectl 的步骤。

需要将下载的 kubectl 二进制程序和生成的 ~/.kube/config 配置文件拷贝到**所有使用 kubectl 命令的机器**。

## 使用的变量

本文档用到的变量定义如下：

|  |
| --- |
| $ export MASTER\_IP=192.168.202.131 # 替换为 kubernetes master 集群任一机器 IP $ export KUBE\_APISERVER="https://${MASTER\_IP}:6443" $ |

* 变量 KUBE\_APISERVER 指定 kubelet 访问的 kube-apiserver 的地址，后续被写入~/.kube/config 配置文件；

## 下载 kubectl

|  |
| --- |
| $ wget https://dl.k8s.io/v1.6.2/kubernetes-client-linux-amd64.tar.gz $ tar -xzvf kubernetes-client-linux-amd64.tar.gz $ sudo cp kubernetes/client/bin/kube\* /root/local/bin/ $ chmod a+x /root/local/bin/kube\* $ export PATH=/root/local/bin:$PATH |

## 创建 admin 证书

kubectl 与 kube-apiserver 的安全端口通信，需要为安全通信提供 TLS 证书和秘钥。

创建 admin 证书签名请求

|  |
| --- |
| $ cat admin-csr.json {  "CN": "admin",  "hosts": [],  "key": {  "algo": "rsa",  "size": 2048  },  "names": [  {  "C": "CN",  "ST": "BeiJing",  "L": "BeiJing",  "O": "system:masters",  "OU": "System"  }  ] } |

* 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 **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；
* hosts 属性值为空列表；

生成 admin 证书和私钥：

|  |
| --- |
| $ cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem \  -ca-key=/etc/kubernetes/ssl/ca-key.pem \  -config=/etc/kubernetes/ssl/ca-config.json \  -profile=kubernetes admin-csr.json | cfssljson -bare admin $ ls admin\* admin.csr admin-csr.json admin-key.pem admin.pem $ sudo mv admin\*.pem /etc/kubernetes/ssl/ $ rm admin.csr admin-csr.json |

## 创建 kubectl kubeconfig 文件

|  |
| --- |
| $ # 设置集群参数 $ kubectl config set-cluster kubernetes \  --certificate-authority=/etc/kubernetes/ssl/ca.pem \  --embed-certs=true \  --server=${KUBE\_APISERVER} $ # 设置客户端认证参数 $ kubectl config set-credentials admin \  --client-certificate=/etc/kubernetes/ssl/admin.pem \  --embed-certs=true \  --client-key=/etc/kubernetes/ssl/admin-key.pem $ # 设置上下文参数 $ kubectl config set-context kubernetes \  --cluster=kubernetes \  --user=admin $ # 设置默认上下文 $ kubectl config use-context kubernetes |

* admin.pem 证书 O 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；

## 分发 kubeconfig 文件

将 ~/.kube/config 文件拷贝到运行 kubelet 命令的机器的 ~/.kube/ 目录下。

# Flannel安装和配置

## 安装Flannel

直接使用yum进行安装

|  |
| --- |
| $ yum list flanneld $ yum install -y flannel |

## 配置Flannel

使用yum安装后，会生成/usr/lib/systemd/system/flanneld.service配置文件

|  |
| --- |
| [Unit] Description=Flanneld overlay address etcd agent After=network.target After=network-online.target Wants=network-online.target After=etcd.service Before=docker.service  [Service] Type=notify EnvironmentFile=/etc/sysconfig/flanneld EnvironmentFile=-/etc/sysconfig/docker-network ExecStart=/usr/bin/flanneld-start $FLANNEL\_OPTIONS ExecStartPost=/usr/libexec/flannel/mk-docker-opts.sh -k DOCKER\_NETWORK\_OPTIONS -d /run/flannel/docker Restart=on-failure  [Install] WantedBy=multi-user.target RequiredBy=docker.service |

可以看到flannel环境变量配置文件在/etc/sysconfig/flannel

|  |
| --- |
| # Flanneld configuration options   # etcd url location. Point this to the server where etcd runs FLANNEL\_ETCD\_ENDPOINTS="https://172.20.0.113:2379,https://172.20.0.114:2379,https://172.20.0.115:2379"  # etcd config key. This is the configuration key that flannel queries # For address range assignment FLANNEL\_ETCD\_PREFIX="/kube-centos/network"  # Any additional options that you want to pass FLANNEL\_OPTIONS="-etcd-cafile=/etc/etcd/ssl/ca.pem -etcd-certfile=/etc/etcd/ssl/etcd.pem -etcd-keyfile=/etc/etcd/ssl/etcd-key.pem" |

* etcd的地址FLANNEL\_ETCD\_ENDPOINT
* etcd查询的目录，包含docker的IP地址段配置。FLANNEL\_ETCD\_PREFIX, 需要在etcd集群中有对应的路径
* FLANNEL\_OPTIONS配置了TLS证书

## 在etcd中创建网络配置

执行下面的命令为docker分配IP地址段，用于启动容器时分配ip

|  |
| --- |
| $ etcdctl mkdir /kube-centos/network $ etcdctl mk /kube-centos/network/config "{ \"Network\": \"10.254.0.0/16\", \"SubnetLen\": 24, \"Backend\": { \"Type\": \"vxlan\" } }" |

ip地址需要为10开头的，kubernetes集群安装kube-dns时，配置为别的曾经出错

## 启动flannel

|  |
| --- |
| $ systemctl daemon-reload $ systemctl enable flanneld $ systemctl start flanneld $ systemctl status flannel |

## flannel 子网段

Flannel启动后，应当有/run/flannel/subnet.env文件, source使之生效

|  |
| --- |
| $ cat /run/flannel/subnet.env FLANNEL\_NETWORK=10.254.0.0/16 FLANNEL\_SUBNET=10.254.46.1/24 FLANNEL\_MTU=1450 FLANNEL\_IPMASQ=false $ source /run/flannel/subnet.env |

## 查询Etcd中的内容

|  |
| --- |
| $ etcdctl ls /kube-centos/network/subnets /kube-centos/network/subnets/10.254.14.0-24 /kube-centos/network/subnets/10.254.38.0-24 /kube-centos/network/subnets/10.254.46.0-24 $ etcdctl get /kube-centos/network/config { "Network": "10.254.0.0/16", "SubnetLen": 24, "Backend": { "Type": "vxlan" } } $ etcdctl get /kube-centos/network/subnets/10.254.14.0-24 {"PublicIP":"10.254.0.114","BackendType":"vxlan","BackendData":{"VtepMAC":"56:27:7d:1c:08:22"}} $ etcdctl get /kube-centos/network/subnets/10.254.38.0-24 {"PublicIP":"10.254.0.115","BackendType":"vxlan","BackendData":{"VtepMAC":"12:82:83:59:cf:b8"}} $ etcdctl get /kube-centos/network/subnets/10.254.46.0-24 {"PublicIP":"10.254.0.113","BackendType":"vxlan","BackendData":{"VtepMAC":"e6:b2:fd:f6:66:96"}} |

## 查询ip

使用ip addr,此时docker和flannel应当在同一网段中

|  |
| --- |
| 6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN   link/ether 02:42:da:bf:83:a2 brd ff:ff:ff:ff:ff:ff  inet 10.254.38.1/24 brd 172.30.38.255 scope global docker0  valid\_lft forever preferred\_lft forever 7: flannel.1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue state UNKNOWN   link/ether 9a:29:46:61:03:44 brd ff:ff:ff:ff:ff:ff  inet 10.254.38.0/32 scope global flannel.1  valid\_lft forever preferred\_lft forever |

# 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；

本文档介绍部署单机 kubernetes master 节点的步骤，**没有实现高可用 master 集群**。

计划后续再介绍部署 LB 的步骤，客户端 (kubectl、kubelet、kube-proxy) 使用 LB 的 VIP 来访问 kube-apiserver，从而实现高可用 master 集群。

master 节点与 node 节点上的 Pods 通过 Pod 网络通信，所以需要在 master 节点上部署 Flannel 网络。

## 使用的变量

本文档用到的变量定义如下：

|  |
| --- |
| $ export MASTER\_IP=192.168.202.131 # 替换为当前部署的 master 机器 IP $ # 导入用到的其它全局变量：SERVICE\_CIDR、CLUSTER\_CIDR、NODE\_PORT\_RANGE、ETCD\_ENDPOINTS、BOOTSTRAP\_TOKEN $ source /root/local/bin/environment.sh |

## 下载最新版本的二进制文件

有两种下载方式：

1. 从 [github release 页面](https://github.com/kubernetes/kubernetes/releases) 下载发布版 tarball，解压后再执行下载脚本

|  |
| --- |
| $ wget https://github.com/kubernetes/kubernetes/releases/download/v1.6.2/kubernetes.tar.gz $ tar -xzvf kubernetes.tar.gz ... $ cd kubernetes $ ./cluster/get-kube-binaries.sh ... |

1. 从 [CHANGELOG页面](https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG.md) 下载 client 或 server tarball 文件

* server 的 tarball kubernetes-server-linux-amd64.tar.gz 已经包含了 client(kubectl) 二进制文件，所以不用单独下载kubernetes-client-linux-amd64.tar.gz文件；

|  |
| --- |
| $ # wget https://dl.k8s.io/v1.6.2/kubernetes-client-linux-amd64.tar.gz $ wget https://dl.k8s.io/v1.6.2/kubernetes-server-linux-amd64.tar.gz $ tar -xzvf kubernetes-server-linux-amd64.tar.gz ... $ cd kubernetes $ tar -xzvf kubernetes-src.tar.gz |

将二进制文件拷贝到指定路径：

|  |
| --- |
| $ sudo cp -r server/bin/{kube-apiserver,kube-controller-manager,kube-scheduler,kubectl,kube-proxy,kubelet} /root/local/bin/ |

## 安装和配置 flanneld

参考 [Flannel安装与配置](#flannel安装和配置)

## 创建 kubernetes 证书

创建 kubernetes 证书签名请求

|  |
| --- |
| $ cat > kubernetes-csr.json <<EOF {  "CN": "kubernetes",  "hosts": [  "127.0.0.1",  "${MASTER\_IP}",  "10.254.0.1",  "kubernetes",  "kubernetes.default",  "kubernetes.default.svc",  "kubernetes.default.svc.cluster",  "kubernetes.default.svc.cluster.local"  ],  "key": {  "algo": "rsa",  "size": 2048  },  "names": [  {  "C": "CN",  "ST": "BeiJing",  "L": "BeiJing",  "O": "k8s",  "OU": "System"  }  ] } EOF |

* 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**，所以上面分别指定了当前部署的 master 节点主机 IP；
* 还需要添加 kube-apiserver 注册的名为 kubernetes 的服务 IP (Service Cluster IP)，一般是 kube-apiserver --service-cluster-ip-range 选项值指定的网段的**第一个IP**，如 "10.254.0.1"；

|  |
| --- |
| $ kubectl get svc kubernetes  NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE  kubernetes 10.254.0.1 <none> 443/TCP 1d |

生成 kubernetes 证书和私钥

|  |
| --- |
| $ cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem \  -ca-key=/etc/kubernetes/ssl/ca-key.pem \  -config=/etc/kubernetes/ssl/ca-config.json \  -profile=kubernetes kubernetes-csr.json | cfssljson -bare kubernetes $ ls kubernetes\* kubernetes.csr kubernetes-csr.json kubernetes-key.pem kubernetes.pem $ sudo mkdir -p /etc/kubernetes/ssl/ $ sudo mv kubernetes\*.pem /etc/kubernetes/ssl/ $ rm kubernetes.csr kubernetes-csr.json |

## 配置和启动 kube-apiserver

### 创建 kube-apiserver 使用的客户端 token 文件

kubelet **首次启动**时向 kube-apiserver 发送 TLS Bootstrapping 请求，kube-apiserver 验证 kubelet 请求中的 token 是否与它配置的 token.csv 一致，如果一致则自动为 kubelet生成证书和秘钥。

|  |
| --- |
| $ # 导入的 environment.sh 文件定义了 BOOTSTRAP\_TOKEN 变量 $ cat > token.csv <<EOF ${BOOTSTRAP\_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap" EOF $ mv token.csv /etc/kubernetes/ |

### 创建 kube-apiserver 的 systemd unit 文件

|  |
| --- |
| $ cat kube-apiserver.service [Unit] Description=Kubernetes API Service Documentation=https://github.com/GoogleCloudPlatform/kubernetes After=network.target After=etcd.service  [Service] EnvironmentFile=-/etc/kubernetes/config EnvironmentFile=-/etc/kubernetes/apiserver ExecStart=/usr/bin/kube-apiserver \  $KUBE\_LOGTOSTDERR \  $KUBE\_LOG\_LEVEL \  $KUBE\_ETCD\_SERVERS \  $KUBE\_API\_ADDRESS \  $KUBE\_API\_PORT \  $KUBELET\_PORT \  $KUBE\_ALLOW\_PRIV \  $KUBE\_SERVICE\_ADDRESSES \  $KUBE\_ADMISSION\_CONTROL \  $KUBE\_API\_ARGS Restart=on-failure Type=notify LimitNOFILE=65536  [Install] WantedBy=multi-user.target |

/etc/kubernetes/config文件的内容为：

|  |
| --- |
| ### # kubernetes system config # # The following values are used to configure various aspects of all # kubernetes services, including # # kube-apiserver.service # kube-controller-manager.service # kube-scheduler.service # kubelet.service # kube-proxy.service # logging to stderr means we get it in the systemd journal KUBE\_LOGTOSTDERR="--logtostderr=true"  # journal message level, 0 is debug KUBE\_LOG\_LEVEL="--v=0"  # Should this cluster be allowed to run privileged docker containers KUBE\_ALLOW\_PRIV="--allow-privileged=true"  # How the controller-manager, scheduler, and proxy find the apiserver KUBE\_MASTER="--master=http://192.168.202.131:8080" |

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：

|  |
| --- |
| ### ## kubernetes system config ## ## The following values are used to configure the kube-apiserver ## # ## The address on the local server to listen to. #KUBE\_API\_ADDRESS="--insecure-bind-address=sz-pg-oam-docker-test-001.tendcloud.com" KUBE\_API\_ADDRESS="--advertise-address=192.168.202.131 --bind-address=192.168.202.131 --insecure-bind-address=192.168.202.131" # ## The port on the local server to listen on. #KUBE\_API\_PORT="--port=8080" # ## Port minions listen on #KUBELET\_PORT="--kubelet-port=10250" # ## Comma separated list of nodes in the etcd cluster KUBE\_ETCD\_SERVERS="--etcd-servers=https://192.168.202.131:2379,192.168.202.132:2379,192.168.202.133:2379" # ## Address range to use for services KUBE\_SERVICE\_ADDRESSES="--service-cluster-ip-range=10.254.0.0/16" # ## default admission control policies KUBE\_ADMISSION\_CONTROL="--admission-control=ServiceAccount,NamespaceLifecycle,NamespaceExists,LimitRanger,ResourceQuota" # ## Add your own! KUBE\_API\_ARGS="--authorization-mode=RBAC --runtime-config=rbac.authorization.k8s.io/v1beta1 --kubelet-https=true --experimental-bootstrap-token-auth --token-auth-file=/etc/kubernetes/token.csv --service-node-port-range=30000-32767 --tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem --tls-private-key-file=/etc/kubernetes/ssl/kubernetes-key.pem --client-ca-file=/etc/kubernetes/ssl/ca.pem --service-account-key-file=/etc/kubernetes/ssl/ca-key.pem --etcd-cafile=/etc/kubernetes/ssl/ca.pem --etcd-certfile=/etc/kubernetes/ssl/kubernetes.pem --etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem --enable-swagger-ui=true --apiserver-count=3 --audit-log-maxage=30 --audit-log-maxbackup=3 --audit-log-maxsize=100 --audit-log-path=/var/lib/audit.log --event-ttl=1h" |

* kube-apiserver 1.6 版本开始使用 etcd v3 API 和存储格式；
* --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，否则部署集群插件时会失败；
* --bind-address 不能为 127.0.0.1；
* --service-cluster-ip-range 指定 Service Cluster IP 地址段，该地址段不能路由可达；
* --service-node-port-range 指定 NodePort 的端口范围；
* 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；

完整 unit 见 [kube-apiserver.service](https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master/systemd/kube-apiserver.service)

### 启动 kube-apiserver

|  |
| --- |
| $ sudo cp kube-apiserver.service /etc/systemd/system/ $ sudo systemctl daemon-reload $ sudo systemctl enable kube-apiserver $ sudo systemctl start kube-apiserver $ sudo systemctl status kube-apiserver |

## 配置和启动 kube-controller-manager

### 创建 kube-controller-manager 的 systemd unit 文件

|  |
| --- |
| $ cat kube-controller-manager.service [Unit] Description=Kubernetes Controller Manager Documentation=https://github.com/GoogleCloudPlatform/kubernetes After=kube-apiserver.service  [Service] EnvironmentFile=-/etc/kubernetes/config EnvironmentFile=-/etc/kubernetes/controller-manager ExecStart=/usr/bin/kube-controller-manager \  $KUBE\_LOGTOSTDERR \  $KUBE\_LOG\_LEVEL \  $KUBE\_MASTER \  $KUBE\_CONTROLLER\_MANAGER\_ARGS Restart=on-failure LimitNOFILE=65536  [Install] WantedBy=multi-user.target |

配置文件/etc/kubernetes/controller-manager

|  |
| --- |
| ### # The following values are used to configure the kubernetes controller-manager  # defaults from config and apiserver should be adequate  # Add your own! KUBE\_CONTROLLER\_MANAGER\_ARGS="--address=127.0.0.1 --service-cluster-ip-range=10.254.0.0/16 --cluster-name=kubernetes --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem --service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem --root-ca-file=/etc/kubernetes/ssl/ca.pem --leader-elect=true" |

* --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，否则：

|  |
| --- |
| $ kubectl get componentstatuses NAME STATUS MESSAGE ERROR controller-manager Unhealthy Get http://127.0.0.1:10252/healthz: dial tcp 127.0.0.1:10252: getsockopt: connection refused scheduler Unhealthy Get http://127.0.0.1:10251/healthz: dial tcp 127.0.0.1:10251: getsockopt: connection refused |

* 参考：https://github.com/kubernetes-incubator/bootkube/issues/64
* --master=http://{MASTER\_IP}:8080：使用非安全 8080 端口与 kube-apiserver 通信；
* --cluster-cidr 指定 Cluster 中 Pod 的 CIDR 范围，该网段在各 Node 间必须路由可达(flanneld保证)；
* --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**；
* --leader-elect=true 部署多台机器组成的 master 集群时选举产生一处于工作状态的 kube-controller-manager 进程；

完整 unit 见 [kube-controller-manager.service](https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master/systemd/kube-controller-manager.service)

### 启动 kube-controller-manager

|  |
| --- |
| $ sudo cp kube-controller-manager.service /etc/systemd/system/ $ sudo systemctl daemon-reload $ sudo systemctl enable kube-controller-manager $ sudo systemctl start kube-controller-manager |

## 配置和启动 kube-scheduler

### 创建 kube-scheduler 的 systemd unit 文件

|  |
| --- |
| $ cat kube-scheduler.service  [Unit] Description=Kubernetes Scheduler Documentation=https://github.com/GoogleCloudPlatform/kubernetes After=kube-apiserver.service  [Service] EnvironmentFile=-/etc/kubernetes/config EnvironmentFile=-/etc/kubernetes/scheduler ExecStart=/usr/bin/kube-scheduler \  $KUBE\_LOGTOSTDERR \  $KUBE\_LOG\_LEVEL \  $KUBE\_MASTER \  $KUBE\_SCHEDULER\_ARGS Restart=on-failure LimitNOFILE=65536  [Install] WantedBy=multi-user.target |

配置文件/etc/kubernetes/scheduler。

|  |
| --- |
| ### # kubernetes scheduler config  # default config should be adequate  # Add your own! KUBE\_SCHEDULER\_ARGS="--leader-elect=true --address=127.0.0.1" |

* --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；
* --master=http://{MASTER\_IP}:8080：使用非安全 8080 端口与 kube-apiserver 通信；
* --leader-elect=true 部署多台机器组成的 master 集群时选举产生一处于工作状态的 kube-controller-manager 进程；

完整 unit 见 [kube-scheduler.service](https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master/systemd/kube-scheduler.service)。

### 启动 kube-scheduler

|  |
| --- |
| $ sudo cp kube-scheduler.service /etc/systemd/system/ $ sudo systemctl daemon-reload $ sudo systemctl enable kube-scheduler $ sudo systemctl start kube-scheduler |

## 验证 master 节点功能

|  |
| --- |
| $ kubectl get componentstatuses NAME STATUS MESSAGE ERROR controller-manager Healthy ok scheduler Healthy ok etcd-0 Healthy {"health": "true"} etcd-1 Healthy {"health": "true"} etcd-2 Healthy {"health": "true"} |

# 部署 Node 节点

kubernetes Node 节点包含如下组件：

* flanneld
* docker
* kubelet
* kube-proxy

## 使用的变量

## 安装和配置 flanneld

参考 [部署Flannel网络](#_Flannel安装和配置)

## 安装和配置 docker

参考 [Docker安装](#_安装Docker)

### 检查 docker 服务

|  |
| --- |
| $ service status docker |

## 安装和配置 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 角色，然后 kubelet 才有权限创建认证请求(certificatesigningrequests)：

|  |
| --- |
| $ kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-bootstrapper --user=kubelet-bootstrap |

* --user=kubelet-bootstrap 是文件 /etc/kubernetes/token.csv 中指定的用户名，同时也写入了文件 /etc/kubernetes/bootstrap.kubeconfig；

### 下载最新的 kubelet 和 kube-proxy 二进制文件

|  |
| --- |
| $ wget https://dl.k8s.io/v1.6.2/kubernetes-server-linux-amd64.tar.gz $ tar -xzvf kubernetes-server-linux-amd64.tar.gz $ cd kubernetes $ tar -xzvf kubernetes-src.tar.gz $ sudo cp -r ./server/bin/{kube-proxy,kubelet} /root/local/bin/ |

## 创建 kubelet bootstrapping kubeconfig 文件

|  |
| --- |
| $ # 设置集群参数 $ kubectl config set-cluster kubernetes \  --certificate-authority=/etc/kubernetes/ssl/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 $ mv bootstrap.kubeconfig /etc/kubernetes/ |

* BOOTSTRAP\_TOKEN必须和master节点上生成的一致
* --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 kubelet 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 kubelet 的 systemd unit 文件

|  |
| --- |
| $ sudo mkdir /var/lib/kubelet # 必须先创建工作目录 $ cat kubelet.service  [Unit] Description=Kubernetes Kubelet Documentation=https://github.com/GoogleCloudPlatform/kubernetes After=docker.service Requires=docker.service  [Service] WorkingDirectory=/var/lib/kubelet EnvironmentFile=-/etc/kubernetes/config EnvironmentFile=-/etc/kubernetes/kubelet ExecStart=/usr/bin/kubelet \  $KUBE\_LOGTOSTDERR \  $KUBE\_LOG\_LEVEL \  $KUBELET\_API\_SERVER \  $KUBELET\_ADDRESS \  $KUBELET\_PORT \  $KUBELET\_HOSTNAME \  $KUBE\_ALLOW\_PRIV \  $KUBELET\_POD\_INFRA\_CONTAINER \  $KUBELET\_ARGS Restart=on-failure  [Install] WantedBy=multi-user.target |

kubelet的配置文件/etc/kubernetes/kubelet。其中的IP地址更改为你的每台node节点的IP地址

|  |
| --- |
| ### ## kubernetes kubelet (minion) config # ## The address for the info server to serve on (set to 0.0.0.0 or "" for all interfaces) KUBELET\_ADDRESS="--address=192.168.202.131" # ## The port for the info server to serve on #KUBELET\_PORT="--port=10250" # ## You may leave this blank to use the actual hostname KUBELET\_HOSTNAME="--hostname-override=192.168.202.131" # ## location of the api-server KUBELET\_API\_SERVER="--api-servers=http://192.168.202.131:8080" # ## pod infrastructure container KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=sz-pg-oam-docker-hub-001.tendcloud.com/library/pod-infrastructure:rhel7" # ## Add your own! KUBELET\_ARGS="--cgroup-driver=systemd --cluster-dns=10.254.0.2 --experimental-bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig --kubeconfig=/etc/kubernetes/kubelet.kubeconfig --require-kubeconfig --cert-dir=/etc/kubernetes/ssl --cluster-domain=cluster.local. --hairpin-mode promiscuous-bridge --serialize-image-pulls=false" |

* KUBELET\_ADDRESS 填写本机ip，批量建议0.0.0.0
* --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；
* 如果设置了 --hostname-override 选项，则 kube-proxy 也需要设置该选项，否则会出现找不到 Node 的情况；
* KUBELET\_POD\_INFRA\_CONTAINER：Pod 启动时的一个基础容器，你可以通过Dokcer ps -a命令看到这个容器，类似windows系统服务,供kubenetes内部使用。
* --experimental-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(自动创建 --kubeconfig 指定的文件)；
* 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 kue-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;
* --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；
* kubelet cAdvisor 默认在**所有接口**监听 4194 端口的请求，对于有外网的机器来说不安全，ExecStartPost 选项指定的 iptables 规则只允许内网机器访问 4194 端口；

完整 unit 见 [kubelet.service](https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master/systemd/kubelet.service)

### 启动 kubelet

|  |
| --- |
| $ sudo cp kubelet.service /etc/systemd/system/kubelet.service $ sudo systemctl daemon-reload $ sudo systemctl enable kubelet $ sudo systemctl start kubelet $ systemctl status kubelet |

### 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。

查看未授权的 CSR 请求：

|  |
| --- |
| $ kubectl get csr NAME AGE REQUESTOR CONDITION csr-2b308 4m kubelet-bootstrap Pending $ kubectl get nodes No resources found. |

通过 CSR 请求：

|  |
| --- |
| $ kubectl certificate approve csr-2b308 certificatesigningrequest "csr-2b308" approved $ kubectl get nodes NAME STATUS AGE VERSION 10.64.3.7 Ready 49m v1.6.2 |

自动生成了 kubelet kubeconfig 文件和公私钥：

|  |
| --- |
| $ ls -l /etc/kubernetes/kubelet.kubeconfig -rw------- 1 root root 2284 Apr 7 02:07 /etc/kubernetes/kubelet.kubeconfig $ ls -l /etc/kubernetes/ssl/kubelet\* -rw-r--r-- 1 root root 1046 Apr 7 02:07 /etc/kubernetes/ssl/kubelet-client.crt -rw------- 1 root root 227 Apr 7 02:04 /etc/kubernetes/ssl/kubelet-client.key -rw-r--r-- 1 root root 1103 Apr 7 02:07 /etc/kubernetes/ssl/kubelet.crt -rw------- 1 root root 1675 Apr 7 02:07 /etc/kubernetes/ssl/kubelet.key |

## 配置 kube-proxy

### 创建 kube-proxy 证书

创建 kube-proxy 证书签名请求：

|  |
| --- |
| $ cat kube-proxy-csr.json {  "CN": "system:kube-proxy",  "hosts": [],  "key": {  "algo": "rsa",  "size": 2048  },  "names": [  {  "C": "CN",  "ST": "BeiJing",  "L": "BeiJing",  "O": "k8s",  "OU": "System"  }  ] } |

* 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；
* hosts 属性值为空列表；

生成 kube-proxy 客户端证书和私钥：

|  |
| --- |
| $ cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem \  -ca-key=/etc/kubernetes/ssl/ca-key.pem \  -config=/etc/kubernetes/ssl/ca-config.json \  -profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy $ ls kube-proxy\* kube-proxy.csr kube-proxy-csr.json kube-proxy-key.pem kube-proxy.pem $ sudo mv kube-proxy\*.pem /etc/kubernetes/ssl/ $ rm kube-proxy.csr kube-proxy-csr.json |

### 创建 kube-proxy kubeconfig 文件

|  |
| --- |
| $ # 设置集群参数 $ kubectl config set-cluster kubernetes \  --certificate-authority=/etc/kubernetes/ssl/ca.pem \  --embed-certs=true \  --server=${KUBE\_APISERVER} \  --kubeconfig=kube-proxy.kubeconfig $ # 设置客户端认证参数 $ kubectl config set-credentials kube-proxy \  --client-certificate=/etc/kubernetes/ssl/kube-proxy.pem \  --client-key=/etc/kubernetes/ssl/kube-proxy-key.pem \  --embed-certs=true \  --kubeconfig=kube-proxy.kubeconfig $ # 设置上下文参数 $ kubectl config set-context default \  --cluster=kubernetes \  --user=kube-proxy \  --kubeconfig=kube-proxy.kubeconfig $ # 设置默认上下文 $ kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig $ mv kube-proxy.kubeconfig /etc/kubernetes/ |

* 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；

### 创建 kube-proxy 的 systemd unit 文件

|  |
| --- |
| $ sudo mkdir -p /var/lib/kube-proxy #必须先创建工作目录 $ cat kube-proxy.service  [Unit] Description=Kubernetes Kube-Proxy Server Documentation=https://github.com/GoogleCloudPlatform/kubernetes After=network.target  [Service] EnvironmentFile=-/etc/kubernetes/config EnvironmentFile=-/etc/kubernetes/proxy ExecStart=/usr/bin/kube-proxy \  $KUBE\_LOGTOSTDERR \  $KUBE\_LOG\_LEVEL \  $KUBE\_MASTER \  $KUBE\_PROXY\_ARGS Restart=on-failure LimitNOFILE=65536  [Install] WantedBy=multi-user.target |

kube-proxy配置文件/etc/kubernetes/proxy

|  |
| --- |
| ### # kubernetes proxy config  # default config should be adequate  # Add your own! KUBE\_PROXY\_ARGS="--bind-address=172.20.0.113 --hostname-override=172.20.0.113 --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig --cluster-cidr=10.254.0.0/16" |

* --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；
* --cluster-cidr 必须与 kube-apiserver 的 --service-cluster-ip-range 选项值一致；
* 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；

完整 unit 见 [kube-proxy.service](https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master/systemd/kube-proxy.service)

### 启动 kube-proxy

|  |
| --- |
| $ sudo cp kube-proxy.service /etc/systemd/system/ $ sudo systemctl daemon-reload $ sudo systemctl enable kube-proxy $ sudo systemctl start kube-proxy $ systemctl status kube-proxy |

## 验证集群功能

定义文件：

|  |
| --- |
| $ cat nginx-ds.yaml apiVersion: v1 kind: Service metadata:  name: nginx-ds  labels:  app: nginx-ds spec:  type: NodePort  selector:  app: nginx-ds  ports:  - name: http  port: 80  targetPort: 80  ---  apiVersion: extensions/v1beta1 kind: DaemonSet metadata:  name: nginx-ds  labels:  addonmanager.kubernetes.io/mode: Reconcile spec:  template:  metadata:  labels:  app: nginx-ds  spec:  containers:  - name: my-nginx  image: nginx:1.7.9  ports:  - containerPort: 80 |

创建 Pod 和服务：

|  |
| --- |
| $ kubectl create -f nginx-ds.yml service "nginx-ds" created daemonset "nginx-ds" created |

### 检查节点状态

|  |
| --- |
| $ kubectl get nodes NAME STATUS AGE VERSION 192.168.202.131 Ready 8d v1.6.2 192.168.202.132 Ready 8d v1.6.2 |

都为 Ready 时正常。

### 检查各 Node 上的 Pod IP 连通性

|  |
| --- |
| $ kubectl get pods -o wide|grep nginx-ds nginx-ds-6ktz8 1/1 Running 0 5m 172.30.25.19 10.64.3.7 nginx-ds-6ktz9 1/1 Running 0 5m 172.30.20.20 10.64.3.8 |

可见，nginx-ds 的 Pod IP 分别是 172.30.25.19、172.30.20.20，在所有 Node 上分别 ping 这两个 IP，看是否连通。

### 检查服务 IP 和端口可达性

|  |
| --- |
| $ kubectl get svc |grep nginx-ds nginx-ds 10.254.136.178 <nodes> 80:8744/TCP 11m |

可见：

* 服务IP：10.254.136.178
* 服务端口：80
* NodePort端口：8744

在所有 Node 上执行：

|  |
| --- |
| $ curl 10.254.136.178 # `kubectl get svc |grep nginx-ds` 输出中的服务 IP |

预期输出 nginx 欢迎页面内容。

### 检查服务的 NodePort 可达性

在所有 Node 上执行：

$ export NODE\_IP=192.168.202.131 # 当前 Node 的 IP  
$ export NODE\_PORT=8744 # `kubectl get svc |grep nginx-ds` 输出中 80 端口映射的 NodePort  
$ curl ${NODE\_IP}:${NODE\_PORT}  
$

预期输出 nginx 欢迎页面内容。

# 部署 kubedns 插件

官方文件目录：kubernetes/cluster/addons/dns

使用的文件：

$ ls \*.yaml \*.base  
kubedns-cm.yaml kubedns-sa.yaml kubedns-controller.yaml.base kubedns-svc.yaml.base

已经修改好的 yaml 文件见：[dns](https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master/manifests/kubedns)。

## 系统预定义的 RoleBinding

预定义的 RoleBinding system:kube-dns 将 kube-system 命名空间的 kube-dns ServiceAccount 与 system:kube-dns Role 绑定， 该 Role 具有访问 kube-apiserver DNS 相关 API 的权限；

$ kubectl get clusterrolebindings system:kube-dns -o yaml  
apiVersion: rbac.authorization.k8s.io/v1beta1  
kind: ClusterRoleBinding  
metadata:  
 annotations:  
 rbac.authorization.kubernetes.io/autoupdate: "true"  
 creationTimestamp: 2017-04-06T17:40:47Z  
 labels:  
 kubernetes.io/bootstrapping: rbac-defaults  
 name: system:kube-dns  
 resourceVersion: "56"  
 selfLink: /apis/rbac.authorization.k8s.io/v1beta1/clusterrolebindingssystem%3Akube-dns  
 uid: 2b55cdbe-1af0-11e7-af35-8cdcd4b3be48  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: system:kube-dns  
subjects:  
- kind: ServiceAccount  
 name: kube-dns  
 namespace: kube-system

kubedns-controller.yaml 中定义的 Pods 时使用了 kubedns-sa.yaml 文件定义的 kube-dns ServiceAccount，所以具有访问 kube-apiserver DNS 相关 API 的权限；

## 配置 kube-dns ServiceAccount

无需修改；

## 配置 kube-dns 服务

$ diff kubedns-svc.yaml.base kubedns-svc.yaml  
30c30  
< clusterIP: \_\_PILLAR\_\_DNS\_\_SERVER\_\_  
---  
> clusterIP: 10.254.0.2

* 需要将 spec.clusterIP 设置为[集群环境变量](https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master/manifests/environment.sh)中变量 CLUSTER\_DNS\_SVC\_IP 值，这个 IP 需要和 kubelet 的 —cluster-dns 参数值一致；

## 配置 kube-dns Deployment

$ diff kubedns-controller.yaml.base kubedns-controller.yaml  
58c58  
< image: gcr.io/google\_containers/k8s-dns-kube-dns-amd64:1.14.1  
---  
> image: registry.cn-hangzhou.aliyuncs.com/kube\_containers/k8s-dns-kube-dns-amd64:1.14.1  
88c88  
< - --domain=\_\_PILLAR\_\_DNS\_\_DOMAIN\_\_.  
---  
> - --domain=cluster.local.  
92c92  
< \_\_PILLAR\_\_FEDERATIONS\_\_DOMAIN\_\_MAP\_\_  
---  
> #\_\_PILLAR\_\_FEDERATIONS\_\_DOMAIN\_\_MAP\_\_  
110c110  
< image: gcr.io/google\_containers/k8s-dns-dnsmasq-nanny-amd64:1.14.1  
---  
> image: registry.cn-hangzhou.aliyuncs.com/google-containers/k8s-dns-dnsmasq-nanny-amd64:1.14.1  
129c129  
< - --server=/\_\_PILLAR\_\_DNS\_\_DOMAIN\_\_/127.0.0.1#10053  
---  
> - --server=/cluster.local./127.0.0.1#10053  
148c148  
< image: gcr.io/google\_containers/k8s-dns-sidecar-amd64:1.14.1  
---  
> image: registry.cn-hangzhou.aliyuncs.com/google\_containers\_mirror/k8s-dns-sidecar-amd64:1.14.1  
161,162c161,162  
< - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.\_\_PILLAR\_\_DNS\_\_DOMAIN\_\_,5,A  
< - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.\_\_PILLAR\_\_DNS\_\_DOMAIN\_\_,5,A  
---  
> - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local.,5,A  
> - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local.,5,A

* --domain 为[集群环境文档](file:///C:\Program%20Files%20(x86)\Pandoc\01-environment.md) 变量 CLUSTER\_DNS\_DOMAIN 的值；
* 使用系统已经做了 RoleBinding 的 kube-dns ServiceAccount，该账户具有访问 kube-apiserver DNS 相关 API 的权限；

## 执行所有定义文件

$ pwd  
/root/kubernetes-git/cluster/addons/dns  
$ ls \*.yaml  
kubedns-cm.yaml kubedns-controller.yaml kubedns-sa.yaml kubedns-svc.yaml  
$ kubectl create -f .  
$

## 检查 kubedns 功能

新建一个 Deployment

$ cat my-nginx.yaml  
apiVersion: extensions/v1beta1  
kind: Deployment  
metadata:  
 name: my-nginx  
spec:  
 replicas: 2  
 template:  
 metadata:  
 labels:  
 run: my-nginx  
 spec:  
 containers:  
 - name: my-nginx  
 image: nginx:1.7.9  
 ports:  
 - containerPort: 80  
$ kubectl create -f my-nginx.yaml  
$

Export 该 Deployment, 生成 my-nginx 服务

$ kubectl expose deploy my-nginx  
$ kubectl get services --all-namespaces |grep my-nginx  
default my-nginx 10.254.86.48 <none> 80/TCP 1d

创建另一个 Pod，查看 /etc/resolv.conf 是否包含 kubelet 配置的 --cluster-dns 和 --cluster-domain，是否能够将服务 my-nginx 解析到上面显示的 Cluster IP 10.254.86.48

$ cat pod-nginx.yaml  
apiVersion: v1  
kind: Pod  
metadata:  
 name: nginx  
spec:  
 containers:  
 - name: nginx  
 image: nginx:1.7.9  
 ports:  
 - containerPort: 80  
$ kubectl create -f pod-nginx.yaml  
$ kubectl exec nginx -i -t -- /bin/bash  
root@nginx:/# cat /etc/resolv.conf  
nameserver 10.254.0.2  
search default.svc.cluster.local svc.cluster.local cluster.local tjwq01.ksyun.com  
options ndots:5  
  
root@nginx:/# ping my-nginx  
PING my-nginx.default.svc.cluster.local (10.254.86.48): 48 data bytes  
^C--- my-nginx.default.svc.cluster.local ping statistics ---  
2 packets transmitted, 0 packets received, 100% packet loss  
  
root@nginx:/# ping kubernetes  
PING kubernetes.default.svc.cluster.local (10.254.0.1): 48 data bytes  
^C--- kubernetes.default.svc.cluster.local ping statistics ---  
1 packets transmitted, 0 packets received, 100% packet loss  
  
root@nginx:/# ping kube-dns.kube-system.svc.cluster.local  
PING kube-dns.kube-system.svc.cluster.local (10.254.0.2): 48 data bytes  
^C--- kube-dns.kube-system.svc.cluster.local ping statistics ---  
1 packets transmitted, 0 packets received, 100% packet loss

## 遇到的错误

### dns调试

kubectl get pods --namespace=kube-system查看dns的pods是否正常  
kubectl describe pods --namespace=kube-system查看详细情况  
kubectl logs pod\_name -n kube-system 查看kube-dns的日志

### dns出错,DNS Server not ready

原因: Pod内部网络为10开头 clusterIP:只能配置为10开头的内网地址

# 部署 dashboard 插件

官方文件目录：[kubernetes/cluster/addons/dashboard](https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/)

使用的文件：

$ ls \*.yaml  
dashboard-controller.yaml dashboard-rbac.yaml dashboard-service.yaml

* 新加了 dashboard-rbac.yaml 文件，定义 dashboard 使用的 RoleBinding。

由于 kube-apiserver 启用了 RBAC 授权，而官方源码目录的 dashboard-controller.yaml 没有定义授权的 ServiceAccount，所以后续访问 kube-apiserver 的 API 时会被拒绝，

解决办法是：定义一个名为 dashboard 的 ServiceAccount，然后将它和 Cluster Role view 绑定，具体参考 [dashboard-rbac.yaml文件](https://github.com/zaixiandemiao/kubernetes-install/blob/master/manifest/dashboard/dashboard-rbac.yaml)。

已经修改好的 yaml 文件见：[dashboard](https://github.com/zaixiandemiao/kubernetes-install/tree/master/manifest/dashboard)。

## 配置dashboard-service

$ diff dashboard-service.yaml.orig dashboard-service.yaml  
10a11  
> type: NodePort

* 指定端口类型为 NodePort，这样外界可以通过地址 nodeIP:nodePort 访问 dashboard；

## 配置dashboard-controller

20a21  
> serviceAccountName: dashboard  
23c24  
< image: gcr.io/google\_containers/kubernetes-dashboard-amd64:v1.6.0  
---  
> image: registry.cn-hangzhou.aliyuncs.com/google-containers/kubernetes-dashboard-amd64:v1.6.3

* 使用名为 dashboard 的自定义 ServiceAccount；

## 执行所有定义文件

$ pwd  
/root/kubernetes/cluster/addons/dashboard  
$ ls \*.yaml  
dashboard-controller.yaml dashboard-rbac.yaml dashboard-service.yaml  
$ kubectl create -f .  
$

## 检查执行结果

查看分配的 NodePort

$ kubectl get services kubernetes-dashboard -n kube-system  
NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
kubernetes-dashboard 10.254.224.130 <nodes> 80:30312/TCP 25s

* NodePort 30312映射到 dashboard pod 80端口；

检查 controller

$ kubectl get deployment kubernetes-dashboard -n kube-system  
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE  
kubernetes-dashboard 1 1 1 1 3m  
$ kubectl get pods -n kube-system | grep dashboard  
kubernetes-dashboard-1339745653-pmn6z 1/1 Running 0 4m

## 访问dashboard

1. kubernetes-dashboard 服务暴露了 NodePort，可以使用 http://NodeIP:nodePort 地址访问 dashboard；
2. 通过 kube-apiserver 访问 dashboard；
3. 通过 kubectl proxy 访问 dashboard：

### 通过 kubectl proxy 访问 dashboard

启动代理

$ kubectl proxy --address='192.168.202.131' --port=8086 --accept-hosts='^\*$'  
Starting to serve on 192.168.202.131:8086

* 需要指定 --accept-hosts 选项，否则浏览器访问 dashboard 页面时提示 “Unauthorized”；

浏览器访问 URL：http://192.168.202.131:8086/ui 自动跳转到：http://192.168.202.131:8086/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard/#/workload?namespace=default

### 通过 kube-apiserver 访问dashboard

获取集群服务地址列表

$ kubectl cluster-info  
Kubernetes master is running at https://192.168.202.131:6443  
KubeDNS is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/kube-dns  
kubernetes-dashboard is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard

由于 kube-apiserver 开启了 RBAC 授权，而浏览器访问 kube-apiserver 的时候使用的是匿名证书，所以访问安全端口会导致授权失败。这里需要使用**非安全**端口访问 kube-apiserver：

浏览器访问 URL：http://192.168.202.131:8080/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard

kubernetes-dashboard

kubernetes-dashboard

浏览器访问https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard浏览器会提示证书验证，因为通过加密通道，以改方式访问的话，需要提前导入证书到你的计算机中）。这是我当时在这遇到的坑：[通过 kube-apiserver 访问dashboard，提示User “system:anonymous” cannot proxy services in the namespace “kube-system”. #5](https://github.com/opsnull/follow-me-install-kubernetes-cluster/issues/5)，已经解决。

**导入证书**  
将生成的admin.pem证书转换格式

openssl pkcs12 -export -in admin.pem -out admin.p12 -inkey admin-key.pem

将生成的admin.p12证书导入的你的电脑，导出的时候记住你设置的密码，导入的时候还要用到。

由于缺少 Heapster 插件，当前 dashboard 不能展示 Pod、Nodes 的 CPU、内存等 metric 图形；

# Heapster+InfluxDB+Grafana安装

## 下载压缩包

到 [heapster release 页面](https://github.com/kubernetes/heapster/releases) 下载 heapster。

$ wget https://github.com/kubernetes/heapster/archive/v1.3.0.zip  
$ unzip v1.3.0.zip  
$ mv v1.3.0.zip heapster-1.3.0

文件目录：heapster-1.3.0/deploy/kube-config/influxdb

$ cd heapster-1.3.0/deploy/kube-config/influxdb  
$ ls \*.yaml  
grafana-deployment.yaml grafana-service.yaml heapster-deployment.yaml heapster-service.yaml influxdb-deployment.yaml influxdb-service.yaml heapster-rbac.yaml

由于kubernetes集群中使用了RBAC认证，因此需要创建heapster的RBAC配置heapster-rbac.yaml  
已经修改好的yaml文件: [heapster](https://github.com/zaixiandemiao/kubernetes-install/tree/master/manifest/heapster),也可以结合kubernetes源码中的yaml配置文件：[kubernetes heapster](https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/cluster-monitoring/influxdb)

## 配置 grafana-deployment

$ diff grafana-deployment.yaml.orig grafana-deployment.yaml  
16c16  
< image: gcr.io/google\_containers/heapster-grafana-amd64:v4.0.2  
---  
> image: registry.cn-hangzhou.aliyuncs.com/lczean/heapster-grafana-amd64:v4.0.2  
40,41c40,41  
< # value: /api/v1/proxy/namespaces/kube-system/services/monitoring-grafana/  
< value: /  
---  
> value: /api/v1/proxy/namespaces/kube-system/services/monitoring-grafana/  
> #value: /

* image可以暂时使用[阿里云](https://cr.console.aliyun.com/#/imageSearch),确认可以使用后，再上传到私有仓库中,为保证不是镜像下载时出现问题，可以先docker pull到本地，确保镜像完整下载
* 如果后续使用 kube-apiserver 或者 kubectl proxy 访问 grafana dashboard，则必须将 GF\_SERVER\_ROOT\_URL 设置为/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana/，否则后续访问grafana时访问时提示找不到http://10.64.3.7:8086/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana/api/dashboards/home 页面

## 配置heapster-deployments.yaml

$ diff heapster-deployment.yaml.orig heapster-deployment.yaml  
16c16  
> image: sz-pg-oam-docker-hub-001.tendcloud.com/library/heapster-amd64:v1.3.0-beta.1  
> - --source=kubernetes:http://192.168.202.131:8080?inClusterConfig=false

* --source指定的是kube-apiserver的地址，使用8080 insecure端口进行访问

## 配置 influxdb-deployment

influxdb 官方建议使用命令行或 HTTP API 接口来查询数据库，从 v1.1.0 版本开始默认关闭 admin UI，将在后续版本中移除 admin UI 插件。

开启镜像中 admin UI的办法如下：先导出镜像中的 influxdb 配置文件，开启 admin 插件后，再将配置文件内容写入 ConfigMap，最后挂载到镜像中，达到覆盖原始配置的目的：

注意：manifests 目录已经提供了 [修改后的 ConfigMap 定义文件](https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master/manifests/heapster/influxdb-cm.yaml), 可以直接将其放在heapster目录下，一起创建, 确保[admin]中的enable = true

$ # 将 ConfigMap 中的配置文件挂载到 Pod 中，达到覆盖原始配置的目的  
$ diff influxdb-deployment.yaml.orig influxdb-deployment.yaml  
16c16  
< image: grc.io/google\_containers/heapster-influxdb-amd64:v1.1.1  
---  
> image: sz-pg-oam-docker-hub-001.tendcloud.com/library/heapster-influxdb-amd64:v1.1.1  
19a20,21  
> - mountPath: /etc/  
> name: influxdb-config  
22a25,27  
> - name: influxdb-config  
> configMap:  
> name: influxdb-config

## 配置 monitoring-influxdb Service

diff influxdb-service.yaml.orig influxdb-service.yaml  
12a13  
> type: NodePort  
15a17,20  
> name: http  
> - port: 8083  
> targetPort: 8083  
> name: admin

* 定义端口类型为 NodePort，额外增加了 admin 端口映射，用于后续浏览器访问 influxdb 的 admin UI 界面

## 执行所有定义文件

$ ls \*.yaml  
grafana-service.yaml heapster-rbac.yaml influxdb-cm.yaml influxdb-service.yaml  
grafana-deployment.yaml heapster-deployment.yaml heapster-service.yaml influxdb-deployment.yaml  
$ kubectl create -f .  
deployment "monitoring-grafana" created  
service "monitoring-grafana" created  
deployment "heapster" created  
serviceaccount "heapster" created  
clusterrolebinding "heapster" created  
service "heapster" created  
configmap "influxdb-config" created  
deployment "monitoring-influxdb" created  
service "monitoring-influxdb" created

## 检查结果

检查 Deployment

$ kubectl get deployments -n kube-system | grep -E 'heapster|monitoring'  
heapster 1 1 1 1 2m  
monitoring-grafana 1 1 1 1 2m  
monitoring-influxdb 1 1 1 1 2m

检查 Pods

$ kubectl get pods -n kube-system | grep -E 'heapster|monitoring'  
heapster-110704576-gpg8v 1/1 Running 0 2m  
monitoring-grafana-2861879979-9z89f 1/1 Running 0 2m  
monitoring-influxdb-1411048194-lzrpc 1/1 Running 0 2m

检查 kubernets dashboard 界面，看是显示各 Nodes、Pods 的 CPU、内存、负载等利用率曲线图

## 访问 grafana

1. 通过 kube-apiserver 访问：

* 获取 monitoring-grafana 服务 URL
* $ kubectl cluster-info  
  Kubernetes master is running at https://192.168.202.131:6443  
  Heapster is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/heapster  
  KubeDNS is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/kube-dns  
  kubernetes-dashboard is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard  
  monitoring-grafana is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana  
  monitoring-influxdb is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/monitoring-influxdb  
  To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
* 浏览器访问 URL： http://192.168.202.131:8080/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana

1. 通过 kubectl proxy 访问：

* 创建代理
* $ kubectl proxy --address='192.168.202.131' --port=8086 --accept-hosts='^\*$'  
  Starting to serve on 192.168.202.131:8086
* 浏览器访问 URL：http://192.168.202.131:8086/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana

### grafana显示问题

grafana的图像是根据influxdb中的数据绘制出的，刚启动的数据库中不包含数据，所以说要稍微等待一下。

grafana中cluster只能获取到一个节点以及该节点的pod信息。这就要回到教程最初了，最开始希望的是可以通过配置iptables规则来实现访问，但是配置失败了，导致grafana只能获取到一个node的数据，这时只需要关闭防火墙就好了。

## 访问 influxdb admin UI

获取 influxdb http 8086 映射的 NodePort

$ kubectl get svc -n kube-system|grep influxdb  
monitoring-influxdb 10.254.22.46 <nodes> 8086:32201/TCP,8083:30269/TCP 9m

通过 kube-apiserver 的**非安全端口**访问 influxdb 的 admin UI 界面： http://192.168.202.131:8080/api/v1/proxy/namespaces/kube-system/services/monitoring-influxdb:8083/

在页面的 “Connection Settings” 的 Host 中输入 node IP， Port 中输入 8086 映射的 nodePort 如上面的 32201，点击 “Save” 即可（我的集群中的地址是192.168.202.131:32201）：

kubernetes-influxdb-heapster

kubernetes-influxdb-heapster

# 配置和安装 EFK

官方文件目录：[cluster/addons/fluentd-elasticsearch](https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/)

$ ls \*.yaml  
es-controller.yaml es-service.yaml fluentd-es-ds.yaml kibana-controller.yaml kibana-service.yaml efk-rbac.yaml

同样EFK服务也需要一个efk-rbac.yaml文件，配置serviceaccount为efk。

已经修改好的 yaml 文件见：[EFK](https://github.com/zaixiandemiao/kubernetes-install/tree/master/manifest/kibana)

## 配置 es-controller.yaml

$ diff es-controller.yaml.orig es-controller.yaml  
24c24  
< - image: gcr.io/google\_containers/elasticsearch:v2.4.1-2  
---  
> - image: registry.cn-hangzhou.aliyuncs.com/google\_images/elasticsearch:v2.4.1-2

## 配置 es-service.yaml

无需配置；

## 配置 fluentd-es-ds.yaml

$ diff fluentd-es-ds.yaml.orig fluentd-es-ds.yaml  
26c26  
< image: gcr.io/google\_containers/fluentd-elasticsearch:1.22  
---  
> image: registry.cn-hangzhou.aliyuncs.com/google\_images/fluentd-elasticsearch:1.22

## 配置 kibana-controller.yaml

$ diff kibana-controller.yaml.orig kibana-controller.yaml  
22c22  
< image: gcr.io/google\_containers/kibana:v4.6.1-1  
---  
> image: registry.cn-hangzhou.aliyuncs.com/google\_images/kibana:v4.6.1-1

## 给 Node 设置标签

定义 DaemonSet fluentd-es-v1.22 时设置了 nodeSelector beta.kubernetes.io/fluentd-ds-ready=true ，所以需要在期望运行 fluentd 的 Node 上设置该标签；

$ kubectl get nodes  
NAME STATUS AGE VERSION  
192.168.202.131 Ready 1d v1.6.0  
  
$ kubectl label nodes 192.168.202.131 beta.kubernetes.io/fluentd-ds-ready=true  
node "192.168.202.131" labeled

给其他两台node打上同样的标签。

## 执行定义文件

$ kubectl create -f .  
serviceaccount "efk" created  
clusterrolebinding "efk" created  
replicationcontroller "elasticsearch-logging-v1" created  
service "elasticsearch-logging" created  
daemonset "fluentd-es-v1.22" created  
deployment "kibana-logging" created  
service "kibana-logging" created

## 检查执行结果

$ kubectl get deployment -n kube-system|grep kibana  
kibana-logging 1 1 1 1 2m  
  
$ kubectl get pods -n kube-system|grep -E 'elasticsearch|fluentd|kibana'  
elasticsearch-logging-v1-mlstp 1/1 Running 0 1m  
elasticsearch-logging-v1-nfbbf 1/1 Running 0 1m  
fluentd-es-v1.22-31sm0 1/1 Running 0 1m  
fluentd-es-v1.22-bpgqs 1/1 Running 0 1m  
fluentd-es-v1.22-qmn7h 1/1 Running 0 1m  
kibana-logging-1432287342-0gdng 1/1 Running 0 1m  
  
$ kubectl get service -n kube-system|grep -E 'elasticsearch|kibana'  
elasticsearch-logging 10.254.77.62 <none> 9200/TCP 2m  
kibana-logging 10.254.8.113 <none> 5601/TCP 2m

kibana Pod 第一次启动时会用**较长时间(10-20分钟)**来优化和 Cache 状态页面，可以 tailf 该 Pod 的日志观察进度：

$ kubectl logs kibana-logging-1432287342-0gdng -n kube-system -f  
ELASTICSEARCH\_URL=http://elasticsearch-logging:9200  
server.basePath: /api/v1/proxy/namespaces/kube-system/services/kibana-logging  
{"type":"log","@timestamp":"2017-04-12T13:08:06Z","tags":["info","optimize"],"pid":7,"message":"Optimizing and caching bundles for kibana and statusPage. This may take a few minutes"}  
{"type":"log","@timestamp":"2017-04-12T13:18:17Z","tags":["info","optimize"],"pid":7,"message":"Optimization of bundles for kibana and statusPage complete in 610.40 seconds"}  
{"type":"log","@timestamp":"2017-04-12T13:18:17Z","tags":["status","plugin:kibana@1.0.0","info"],"pid":7,"state":"green","message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}  
{"type":"log","@timestamp":"2017-04-12T13:18:18Z","tags":["status","plugin:elasticsearch@1.0.0","info"],"pid":7,"state":"yellow","message":"Status changed from uninitialized to yellow - Waiting for Elasticsearch","prevState":"uninitialized","prevMsg":"uninitialized"}  
{"type":"log","@timestamp":"2017-04-12T13:18:19Z","tags":["status","plugin:kbn\_vislib\_vis\_types@1.0.0","info"],"pid":7,"state":"green","message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}  
{"type":"log","@timestamp":"2017-04-12T13:18:19Z","tags":["status","plugin:markdown\_vis@1.0.0","info"],"pid":7,"state":"green","message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}  
{"type":"log","@timestamp":"2017-04-12T13:18:19Z","tags":["status","plugin:metric\_vis@1.0.0","info"],"pid":7,"state":"green","message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}  
{"type":"log","@timestamp":"2017-04-12T13:18:19Z","tags":["status","plugin:spyModes@1.0.0","info"],"pid":7,"state":"green","message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}  
{"type":"log","@timestamp":"2017-04-12T13:18:19Z","tags":["status","plugin:statusPage@1.0.0","info"],"pid":7,"state":"green","message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}  
{"type":"log","@timestamp":"2017-04-12T13:18:19Z","tags":["status","plugin:table\_vis@1.0.0","info"],"pid":7,"state":"green","message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}  
{"type":"log","@timestamp":"2017-04-12T13:18:19Z","tags":["listening","info"],"pid":7,"message":"Server running at http://0.0.0.0:5601"}  
{"type":"log","@timestamp":"2017-04-12T13:18:24Z","tags":["status","plugin:elasticsearch@1.0.0","info"],"pid":7,"state":"yellow","message":"Status changed from yellow to yellow - No existing Kibana index found","prevState":"yellow","prevMsg":"Waiting for Elasticsearch"}  
{"type":"log","@timestamp":"2017-04-12T13:18:29Z","tags":["status","plugin:elasticsearch@1.0.0","info"],"pid":7,"state":"green","message":"Status changed from yellow to green - Kibana index ready","prevState":"yellow","prevMsg":"No existing Kibana index found"}

## 访问 kibana

1. 通过 kube-apiserver 访问：

* 获取 monitoring-grafana 服务 URL
* $ kubectl cluster-info  
  Kubernetes master is running at https://192.168.202.131:6443  
  Elasticsearch is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/elasticsearch-logging  
  Heapster is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/heapster  
  Kibana is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/kibana-logging  
  KubeDNS is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/kube-dns  
  kubernetes-dashboard is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard  
  monitoring-grafana is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana  
  monitoring-influxdb is running at https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/monitoring-influxdb
* 浏览器访问 URL： https://192.168.202.131:6443/api/v1/proxy/namespaces/kube-system/services/kibana-logging/app/kibana

1. 通过 kubectl proxy 访问：

* 创建代理
* $ kubectl proxy --address='192.168.202.131' --port=8086 --accept-hosts='^\*$'  
  Starting to serve on 192.168.202.131:8086
* 浏览器访问 URL：http://192.168.202.131:8086/api/v1/proxy/namespaces/kube-system/services/kibana-logging

在 Settings -> Indices 页面创建一个 index（相当于 mysql 中的一个 database），选中 Index contains time-based events，使用默认的 logstash-\* pattern，点击 Create ;

**可能遇到的问题**

1、fluent无法启动，kubectl logs pods\_name -n kube-system查看日志发现报了一堆警告，fluent不停的重启，找到最开始的地方，发现有一个Permission denied错误，无法打开一个文件。 这是因为没有关闭SELinux， 使用setenforce 0关闭就能正常启动了

2、如果你在这里发现Create按钮是灰色的无法点击，且Time-filed name中没有选项，fluentd要读取/var/log/containers/目录下的log日志，这些日志是从/var/lib/docker/containers/${CONTAINER\_ID}/${CONTAINER\_ID}-json.log链接过来的，查看你的docker配置，—log-dirver需要设置为**json-file**格式，默认的可能是**journald**，参考[docker logging](https://docs.docker.com/engine/admin/logging/overview/#examples)。

创建Index后，可以在 Discover 下看到 ElasticSearch logging 中汇聚的日志；

# Harbor（镜像私有仓库）安装

## 前言

Harbor的安装，[官方](https://github.com/vmware/harbor/blob/master/docs/installation_guide.md)提供了比较详尽的手册，基本可以参考完成整个安装过程。本文档采用离线安装方法，使用http访问私有仓库，后期探索证书认证方法。安装Harbor前需要的环境： \* Python >= 2.7 \* Docker >= 1.10，推荐使用[阿里云加速器](https://cr.console.aliyun.com)安装 \* Docker Compose >= 1.60(*Harbor使用docker-compose以容器的形式运行*)，直接下载[二进制文件](https://github.com/docker/compose/releases/download/1.16.0-rc1/docker-compose-Linux-x86_64)，加入path即可

## 1. 下载离线安装包

到Harbor的[发布页面](https://github.com/vmware/harbor/releases)下载，尽量下载没有rc的后缀的版本，相对比较稳定一些，我下载的是[1.1.2](https://github.com/vmware/harbor/releases/download/v1.1.2/harbor-offline-installer-v1.1.2.tgz)版本的离线包,下载完成后将压缩包解压

$ tar xvf harbor-offline-installer-<version>.tgz

## 2. 配置harbor.cfg

Harbor的安装配置都在harbor.cfg文件中，其中的属性分为required, optional, 详细信息可以参考[官方配置说明](https://github.com/vmware/harbor/blob/master/docs/installation_guide.md)，因为只使用Http访问私有仓库，这里需要改动的参数很少，修改情况如下： \* **hostname** 访问Harbor的地址（UI和registry service），可以被配置为ip地址或者域名，不可以配置成localhost或者127.0.0.1  
\* **ui\_url\_protocol:** (http 或https，默认为http)，此处使用http，https的配置参考[https配置](https://github.com/vmware/harbor/blob/master/docs/configure_https.md)  
\* **harbor\_admin\_password:** 第一次登陆时，admin的密码，登陆后必须修改，默认为Harbor12345,Harbor的密码必须包含大写，小写，数字三种字符 \* **self\_registration:** (on 或 off.默认为on), 是否支持自主注册，如果设为Off，则用户只能被admin用户创建  
\* **token\_expiration:** token的时效，单位为分钟，默认为30分钟

## 3. installation & starting

执行脚本文件进行安装

./install.sh

如果执行顺利的话，则可以访问http://hostname 进行访问

## 上传镜像到私有仓库

若要上传镜像的电脑可以访问外网，则可以使用docker先将镜像pull到本地，然后push到私有仓库上。

$ docker pull nginx  
$ docker login hostname  
$ docker tag nginx hostname/library/nginx  
$ docker push hostname/library/nginx

上面的四条命令将DockerHub上的nginx先Pull到本地，然后打标签后，上传到私有仓库中， hostname可以是Harbor所在的ip地址，library是一个所有用户都可以访问到的project名称。admin用户可以push，普通用户只可以pull  
如果docker login出错，则原因是，docker配置中，默认需要加密访问，可以到docker的配置文件中添加--insecure-registry hostname, 配置文件路径在/etc/sysconfig/docker,然后执行下面的命令

$ docker -D login hostname  
$ docker tag  
$ docker push

## 管理Harbor

停止Harbor

docker-compose stop

运行Harbor

docker-compose start

修改harbor.cfg, 重启Harbor

$ sudo docker-compose down -v  
$ vim harbor.cfg  
$ sudo prepare  
$ sudo docker-compose up -d