Kubernetes1.6集群部署完全指南 ——基于CentOS7二进制方式部署并开启TLS安全认证

作者： Jimmy Song

本文档GitHub地址：https://github.com/rootsongjc/follow-me-install-kubernetes-cluster

Fork自：https://github.com/opsnull/follow-me-install-kubernetes-cluster

版本：V1.0

时间：2017-04-13

访问 kibana

前言

本系列文档介绍使用二进制部署 kubernetes 集群的所有步骤，而不是使用 kubeadm 等自动化方式来部署集群，同时开启了集群的TLS安全认证；

在部署的过程中，将详细列出各组件的启动参数，给出配置文件，详解它们的含义和可能遇到的问题。

部署完成后，你将理解系统各组件的交互原理，进而能快速解决实际问题。

所以本文档主要适合于那些有一定 kubernetes 基础，想通过一步步部署的方式来学习和了解系统配置、运行原理的人。

注：本文档中不包括docker和私有镜像仓库的安装。

集群详情

- CentOS 7.2.1511
- Docker 1.12.5
- 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)、EFK(elasticsearch、fluentd、kibana) 集群插件
- 私有docker镜像仓库harbor（请自行部署，harbor提供离线安装包，直接使用docker-compose启动即可）

准备

主机角色分配

IP	Hostname	Roles
172.20.0.112	sz-pg-oam-docker-hub-001.tendcloud.com	Harbor（私有镜像仓库）
172.20.0.113	sz-pg-oam-docker-test-001.tendcloud.com	master node kube-apiserver kube-controller-manager kube-scheduler kubelet kube-proxy etcd flannel
172.20.0.114	sz-pg-oam-docker-test-002.tendcloud.com	node kubectl kube-proxy flannel etcd
172.20.0.115	sz-pg-oam-docker-test-003.tendcloud.com	node kubectl kube-proxy flannel etcd

注：

- 172.20.0.112作为harbor私有镜像仓库，本文档不包括harbor的安装，请参考http://github.com/vmware/harbor上的文档自行安装。
- 172.20.0.113既作为master也作为node。

镜像准备

Google官方提供的kubernetes组件镜像在墙外，国内下载有困难，我将所有的镜像克隆的了一份放到了时速云上，以下公有镜像可以直接使用：

index.tenxcloud.com/jimmy/elasticsearch:v2.4.1-2

index.tenxcloud.com/jimmy/fluentd-elasticsearch:1.22

index.tenxcloud.com/jimmy/kibana:v4.6.1-1

index.tenxcloud.com/jimmy/kubernetes-dashboard-amd64:v1.6.0

index.tenxcloud.com/jimmy/heapster-grafana-amd64:v4.0.2

index.tenxcloud.com/jimmy/heapster-amd64:v1.3.0-beta.1

index.tenxcloud.com/jimmy/heapster-influxdb-amd64:v1.1.1

index.tenxcloud.com/jimmy/k8s-dns-kube-dns-amd64:1.14.1

index.tenxcloud.com/jimmy/k8s-dns-dnsmasq-nanny-amd64:1.14.1

index.tenxcloud.com/jimmy/k8s-dns-sidecar-amd64:1.14.1

注：

文档中使用的是我们的私有镜像仓库中的镜像，地址与上述不同。

1.创建 kubernetes 各组件 TLS 加密通信的证书和秘钥

`kubernetes` 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 工具集 cfssl 来生成 Certificate Authority (CA) 和其它证书；

生成的 CA 证书和秘钥文件如下：

- ca-key.pem
- ca.pem
- kubernetes-key.pem
- kubernetes.pem
- kube-proxy.pem
- kube-proxy-key.pem
- admin.pem
- admin-key.pem

使用证书的组件如下：

- etcd：使用 ca.pem、kubernetes-key.pem、kubernetes.pem；
- kube-apiserver：使用 ca.pem、kubernetes-key.pem、kubernetes.pem；
- kubelet：使用 ca.pem；
- kube-proxy：使用 ca.pem、kube-proxy-key.pem、kube-proxy.pem；
- kubectl：使用 ca.pem、admin-key.pem、admin.pem；

`kube-controller`、`kube-scheduler` 当前需要和 `kube-apiserver` 部署在同一台机器上且使用非安全端口通信，故不需要证书。

安装 `CFSSL`

方式一：直接使用二进制源码包安装

```
$ 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

我们的系统中安装了Go1.7.5，使用以下命令安装更快捷：

```
$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
```

在`$GOPATH/bin`目录下得到以cfssl开头的几个命令。

创建 CA (Certificate Authority)

创建 CA 配置文件

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

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

```
$ 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
```

2.创建 kubernetes 证书

创建 kubernetes 证书签名请求

```
$ cat kubernetes-csr.json
{
    "CN": "kubernetes",
    "hosts": [
      "127.0.0.1",
      "172.20.0.112",
      "172.20.0.113",
      "172.20.0.114",
      "172.20.0.115",
      "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"
        }
    ]
}
```

- 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 `kue-apiserver` 指定的 `service-cluster-ip-range` 网段的第一个IP，如 10.254.0.1。

生成 kubernetes 证书和私钥

```
$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -
profile=kubernetes kubernetes-csr.json | cfssljson -bare kubernetes
$ ls kuberntes*
kubernetes.csr  kubernetes-csr.json  kubernetes-key.pem  kubernetes.pem
```

或者直接在命令行上指定相关参数：

```
$ echo '{"CN":"kubernetes","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes -hostname="127.0.0.1,172.20.0.112,172.20.0.113,172.20.0.114,172.20.0.115,kubernetes,kubernetes.default" - | cfssljson -bare kubernetes
```

创建 admin 证书

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` 的**所有API**的权限;
- OU 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;

生成 admin 证书和私钥

```
$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes admin-csr.json | cfssljson -bare admin
$ ls admin*
admin.csr  admin-csr.json  admin-key.pem  admin.pem
```

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 `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；

生成 kube-proxy 客户端证书和私钥

```
$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=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
```

校验证书

以 kubernetes 证书为例

使用 `opsnssl` 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:172.20.0.112, IP Address:172.20.0.113, IP
Address:172.20.0.114, IP Address:172.20.0.115, 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",
    "10.64.3.7",
    "10.254.0.1"
  ],
  "not_before": "2017-04-05T05:36:00Z",
  "not_after": "2018-04-05T05:36:00Z",
  "sigalg": "SHA256WithRSA",
...
```


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；

```
$ sudo mkdir -p /etc/kubernetes/ssl
$ sudo cp *.pem /etc/kubernetes/ssl
```

参考

- Generate self-signed certificates
- Setting up a Certificate Authority and Creating TLS Certificates
- Client Certificates V/s Server Certificates
- 数字证书及 CA 的扫盲介绍

2.创建 kubeconfig 文件

`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；

kubernetes 1.4 开始支持由 `kube-apiserver` 为客户端生成 TLS 证书的 TLS Bootstrapping 功能，这样就不需要为每个客户端生成证书了；该功能**当前仅支持为** `kubelet` 生成证书；

创建 TLS Bootstrapping Token

Token auth file

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

```
export BOOTSTRAP_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')
cat > token.csv <<EOF
${BOOTSTRAP_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap"
EOF
```

> 后三行是一句，直接复制上面的脚本运行即可。

将token.csv发到所有机器（Master 和 Node）的 `/etc/kubernetes/` 目录。

```
$cp token.csv /etc/kubernetes/
```

创建 kubelet bootstrapping kubeconfig 文件

```
$ cd /etc/kubernetes
$ export KUBE_APISERVER="https://172.20.0.113:6443"
$ # 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
```

- `--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

```
$ export KUBE_APISERVER="https://172.20.0.113:6443"
$ # 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
```

- 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

```
$ cp bootstrap.kubeconfig kube-proxy.kubeconfig /etc/kubernetes/
```

3.创建高可用 etcd 集群

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`sz-pg-oam-docker-test-001.tendcloud.com`、`sz-pg-oam-docker-test-002.tendcloud.com`、`sz-pg-oam-docker-test-003.tendcloud.com`：

- sz-pg-oam-docker-test-001.tendcloud.com：172.20.0.113
- sz-pg-oam-docker-test-002.tendcloud.com：172.20.0.114
- sz-pg-oam-docker-test-003.tendcloud.com：172.20.0.115

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

```
$ cp ca.pem kubernetes-key.pem kubernetes.pem /etc/kubernetes/ssl
```

- 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

到 `https://github.com/coreos/etcd/releases` 页面下载最新版本的二进制文件

```
$ https://github.com/coreos/etcd/releases/download/v3.1.5/etcd-v3.1.5-linux-amd64.tar.gz
$ tar -xvf etcd-v3.1.4-linux-amd64.tar.gz
$ sudo mv etcd-v3.1.4-linux-amd64/etcd* /root/local/bin
```

创建 etcd 的 systemd unit 文件

注意替换 `ETCD_NAME` 和 `INTERNAL_IP` 变量的值；

```
$ export ETCD_NAME=sz-pg-oam-docker-test-001.tendcloud.com
$ export INTERNAL_IP=172.20.0.113
$ sudo mkdir -p /var/lib/etcd /var/lib/etcd
$ cat > etcd.service <<EOF
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target
Documentation=https://github.com/coreos

[Service]
Type=notify
WorkingDirectory=/var/lib/etcd/
EnvironmentFile=-/etc/etcd/etcd.conf
ExecStart=/root/local/bin/etcd \\
  --name ${ETCD_NAME} \\
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \\
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \\
  --peer-cert-file=/etc/kubernetes/ssl/kubernetes.pem \\
  --peer-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \\
  --trusted-ca-file=/etc/kubernetes/ssl/ca.pem \\
  --peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \\
  --initial-advertise-peer-urls https://${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-0 \\
  --initial-cluster sz-pg-oam-docker-test-
001.tendcloud.com=https://172.20.0.113:2380,sz-pg-oam-docker-test-
002.tendcloud.com=https://172.20.0.114:2380,sz-pg-oam-docker-test-
003.tendcloud.com=https://172.20.0.115:2380 \\
  --initial-cluster-state new \\
  --data-dir=/var/lib/etcd
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOF
```

- 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两个目录；
- 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 `kubernetes.pem` 证书时使用的 `kubernetes-csr.json` 文件的 `hosts` 字段**包含所有 etcd 节点的 INTERNAL_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；

完整 unit 文件见：etcd.service

启动 etcd 服务

```
$ sudo mv etcd.service /etc/systemd/system/
$ sudo systemctl daemon-reload
$ sudo systemctl enable etcd
$ sudo systemctl start etcd
$ systemctl status etcd
```

在所有的 kubernetes master 节点重复上面的步骤，直到所有机器的 etcd 服务都已启动。

验证服务

在任一 kubernetes master 机器上执行如下命令：

```
$ etcdctl \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  cluster-health
2017-04-11 15:17:09.082250 I | warning: ignoring ServerName for user-provided CA
for backwards compatibility is deprecated
2017-04-11 15:17:09.083681 I | warning: ignoring ServerName for user-provided CA
for backwards compatibility is deprecated
member 9a2ec640d25672e5 is healthy: got healthy result from
https://172.20.0.115:2379
member bc6f27ae3be34308 is healthy: got healthy result from
https://172.20.0.114:2379
member e5c92ea26c4edba0 is healthy: got healthy result from
https://172.20.0.113:2379
cluster is healthy
```

结果最后一行为 `cluster is healthy` 时表示集群服务正常。

4.下载和配置 kubectl 命令行工具

下载 kubectl

```
$ wget https://dl.k8s.io/v1.6.0/kubernetes-client-linux-amd64.tar.gz
$ tar -xzvf kubernetes-client-linux-amd64.tar.gz
$ cp kubernetes/client/bin/kube* /usr/bin/
$ chmod a+x /usr/bin/kube*
```

创建 kubectl kubeconfig 文件

```
$ export KUBE_APISERVER="https://172.20.0.113:6443"
$ # 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 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；

5.部署高可用 kubernetes master 集群

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；

本文档记录部署一个三个节点的高可用 kubernetes master 集群步骤。（后续创建一个 load balancer 来代理访问 kube-apiserver 的请求）

TLS 证书文件

pem和token.csv证书文件我们在TLS证书和秘钥这一步中已经创建过了。我们再检查一下。

```
$ ls /etc/kubernetes/ssl
admin-key.pem  admin.pem  ca-key.pem  ca.pem  kube-proxy-key.pem  kube-proxy.pem
kubernetes-key.pem  kubernetes.pem
```

下载最新版本的二进制文件

有两种下载方式

方式一

从 github release 页面 下载发布版 tarball，解压后再执行下载脚本

```
$ wget
https://github.com/kubernetes/kubernetes/releases/download/v1.6.0/kubernetes.tar.gz
$ tar -xzvf kubernetes.tar.gz
...
$ cd kubernetes
$ ./cluster/get-kube-binaries.sh
...
```

方式二

从 `CHANGELOG` 页面 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/v1.6.0/kubernetes-client-linux-amd64.tar.gz
$ wget https://dl.k8s.io/v1.6.0/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

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

配置和启动 kube-apiserver

创建 kube-apiserver的service配置文件

serivce配置文件`/usr/lib/systemd/system/kube-apiserver.service`内容：

```
[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://sz-pg-oam-docker-test-001.tendcloud.com:8080"
KUBE_MASTER="--master=http://172.20.0.113:8080"
```

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=172.20.0.113 --bind-address=172.20.0.113 --
insecure-bind-address=172.20.0.113"
#
## 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://172.20.0.113:2379,172.20.0.114:2379,172.20.0.115:2379"
#
## 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"
```

- `--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；

完整 unit 见 kube-apiserver.service

启动kube-apiserver

```
$ systemctl daemon-reload
$ systemctl enable kube-apiserver
$ systemctl start kube-apiserver
$ systemctl status kube-apiserver
```

配置和启动 kube-controller-manager

创建 kube-controller-manager的serivce配置文件

文件路径`/usr/lib/systemd/system/kube-controller-manager.service`

```
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[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"
```

- `--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`，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
scheduler            Unhealthy   Get http://127.0.0.1:10251/healthz: dial tcp
127.0.0.1:10251: getsockopt: connection refused
controller-manager   Healthy     ok

etcd-2               Unhealthy   Get http://172.20.0.113:2379/health:
malformed HTTP response "\x15\x03\x01\x00\x02\x02"
etcd-0               Healthy     {"health": "true"}

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

参考：https://github.com/kubernetes-incubator/bootkube/issues/64

完整 unit 见 kube-controller-manager.service

启动 kube-controller-manager

```
$ systemctl daemon-reload
$ systemctl enable kube-controller-manager
$ systemctl start kube-controller-manager
```

配置和启动 kube-scheduler

创建 kube-scheduler的serivce配置文件

文件路径`/usr/lib/systemd/system/kube-scheduler.serivce`。

```
[Unit]
Description=Kubernetes Scheduler Plugin
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[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；

完整 unit 见 kube-scheduler.service

启动 kube-scheduler

```
$ systemctl daemon-reload
$ systemctl enable kube-scheduler
$ systemctl start kube-scheduler
```

验证 master 节点功能

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

6.部署kubernetes node节点

kubernetes node 节点包含如下组件：

- Flanneld：参考我之前写的文章Kubernetes基于Flannel的网络配置，之前没有配置TLS，现在需要在serivce配置文件中增加TLS配置。
- Docker1.12.5：docker的安装很简单，这里也不说了。
- kubelet
- kube-proxy

下面着重讲`kubelet`和`kube-proxy`的安装，同时还要将之前安装的flannel集成TLS验证。

目录和文件

我们再检查一下三个节点上，经过前几步操作生成的配置文件。

```
$ ls /etc/kubernetes/ssl
admin-key.pem  admin.pem  ca-key.pem  ca.pem  kube-proxy-key.pem  kube-proxy.pem
kubernetes-key.pem  kubernetes.pem
$ ls /etc/kubernetes/
apiserver  bootstrap.kubeconfig  config  controller-manager  kubelet  kube-
proxy.kubeconfig  proxy  scheduler  ssl  token.csv
```

配置Flanneld

参考我之前写的文章Kubernetes基于Flannel的网络配置，之前没有配置TLS，现在需要在serivce配置文件中增加TLS配置。

service配置文件`/usr/lib/systemd/system/flanneld.service`。

```
[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
```

`/etc/sysconfig/flanneld`配置文件。

```
# 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/kubernetes/ssl/ca.pem -etcd-
certfile=/etc/kubernetes/ssl/kubernetes.pem -etcd-
keyfile=/etc/kubernetes/ssl/kubernetes-key.pem"
```

在FLANNEL_OPTIONS中增加TLS的配置。

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 cluster 角色(role),
然后 kubelet 才能有权限创建认证请求(certificate signing requests):

```
$ cd /etc/kubernetes
$ kubectl create clusterrolebinding kubelet-bootstrap \
  --clusterrole=system:node-bootstrapper \
  --user=kubelet-bootstrap
```

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

下载最新的 kubelet 和 kube-proxy 二进制文件

```
$ wget https://dl.k8s.io/v1.6.0/kubernetes-server-linux-amd64.tar.gz
$ tar -xzvf kubernetes-server-linux-amd64.tar.gz
$ cd kubernetes
$ tar -xzvf  kubernetes-src.tar.gz
$ cp -r ./server/bin/{kube-proxy,kubelet} /usr/bin/
```

创建 kubelet 的service配置文件

文件位置`/usr/lib/systemd/system/kubelet.serivce`。

```
[Unit]
Description=Kubernetes Kubelet Server
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=172.20.0.113"
#
## 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=172.20.0.113"
#
## location of the api-server
KUBELET_API_SERVER="--api-servers=http://172.20.0.113:8080"
#
## 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"
```

- `--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；
- `--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；
- 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;
- `--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；

完整 unit 见 kubelet.service

启动kublet

```
$ systemctl daemon-reload
$ systemctl enable kubelet
$ systemctl start kubelet
$ systemctl status kubelet
```

通过 kublet 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       v1.6.1
```

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 的service配置文件

文件路径`/usr/lib/systemd/system/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；
- 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

启动 kube-proxy

```
$ systemctl daemon-reload
$ systemctl enable kube-proxy
$ systemctl start kube-proxy
$ systemctl status kube-proxy
```

验证测试

我们创建一个niginx的service试一下集群是否可用。

```
$ kubectl run nginx --replicas=2 --labels="run=load-balancer-example" --image=sz-pg-oam-docker-hub-001.tendcloud.com/library/nginx:1.9  --port=80
deployment "nginx" created
$ kubectl expose deployment nginx --type=NodePort --name=example-service
service "example-service" exposed
$ kubectl describe svc example-service
Name:            example-service
Namespace:       default
Labels:          run=load-balancer-example
Annotations:         <none>
Selector:        run=load-balancer-example
Type:            NodePort
IP:          10.254.62.207
Port:            <unset> 80/TCP
NodePort:        <unset> 32724/TCP
Endpoints:       172.30.60.2:80,172.30.94.2:80
Session Affinity:    None
Events:          <none>
$ curl "10.254.62.207:80"
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

访问`172.20.0.113:32724`或`172.20.0.114:32724`或者`172.20.0.115:32724`都可以得到nginx的页面。

7.安装和配置 kubedns 插件

官方的yaml文件目录：`kubernetes/cluster/addons/dns`。

该插件直接使用kubernetes部署，官方的配置文件中包含以下镜像：

```
gcr.io/google_containers/k8s-dns-dnsmasq-nanny-amd64:1.14.1
gcr.io/google_containers/k8s-dns-kube-dns-amd64:1.14.1
gcr.io/google_containers/k8s-dns-sidecar-amd64:1.14.1
```

我clone了上述镜像，上传到我的私有镜像仓库：

```
sz-pg-oam-docker-hub-001.tendcloud.com/library/k8s-dns-dnsmasq-nanny-amd64:1.14.1
sz-pg-oam-docker-hub-001.tendcloud.com/library/k8s-dns-kube-dns-amd64:1.14.1
sz-pg-oam-docker-hub-001.tendcloud.com/library/k8s-dns-sidecar-amd64:1.14.1
```

同时上传了一份到时速云备份：

```
index.tenxcloud.com/jimmy/k8s-dns-dnsmasq-nanny-amd64:1.14.1
index.tenxcloud.com/jimmy/k8s-dns-kube-dns-amd64:1.14.1
index.tenxcloud.com/jimmy/k8s-dns-sidecar-amd64:1.14.1
```

以下yaml配置文件中使用的是私有镜像仓库中的镜像。

```
kubedns-cm.yaml
kubedns-sa.yaml
kubedns-controller.yaml
kubedns-svc.yaml
```

已经修改好的 yaml 文件见：dns

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: 2017-04-11T11:20:42Z
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:kube-dns
  resourceVersion: "58"
  selfLink:
/apis/rbac.authorization.k8s.io/v1beta1/clusterrolebindingssystem%3Akube-dns
  uid: e61f4d92-1ea8-11e7-8cd7-f4e9d49f8ed0
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
```

- spec.clusterIP = 10.254.0.2，即明确指定了 kube-dns Service IP，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: sz-pg-oam-docker-hub-001.tendcloud.com/library/k8s-dns-kube-dns-
amd64:v1.14.1
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: sz-pg-oam-docker-hub-001.tendcloud.com/library/k8s-dns-dnsmasq-
nanny-amd64:v1.14.1
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: sz-pg-oam-docker-hub-001.tendcloud.com/library/k8s-dns-sidecar-
amd64:v1.14.1
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
```

- 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/kubedns
$ 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: sz-pg-oam-docker-hub-001.tendcloud.com/library/nginx:1.9
        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     10.254.179.239   <none>        80/TCP          42m
```

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 Cluster IP `10.254.179.239`。

```
$ kubectl create -f nginx-pod.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. tendcloud.com
options ndots:5

root@nginx:/# ping my-nginx
PING my-nginx.default.svc.cluster.local (10.254.179.239): 56 data bytes
76 bytes from 119.147.223.109: Destination Net Unreachable
^C--- my-nginx.default.svc.cluster.local ping statistics ---

root@nginx:/# ping kubernetes
PING kubernetes.default.svc.cluster.local (10.254.0.1): 56 data bytes
^C--- kubernetes.default.svc.cluster.local ping statistics ---
11 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): 56 data bytes
^C--- kube-dns.kube-system.svc.cluster.local ping statistics ---
6 packets transmitted, 0 packets received, 100% packet loss
```

从结果来看，service名称可以正常解析。

8.配置和安装 dashboard

官方文件目录：`kubernetes/cluster/addons/dashboard`

我们使用的文件

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

已经修改好的 yaml 文件见：dashboard

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，web中提示：

```
Forbidden (403)

User "system:serviceaccount:kube-system:default" cannot list jobs.batch in the
namespace "default". (get jobs.batch)
```

增加了一个`dashboard-rbac.yaml`文件，定义一个名为 dashboard 的 ServiceAccount，然后将它和 Cluster Role view 绑定。

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

```
$ diff dashboard-controller.yaml.orig dashboard-controller.yaml
23c23
<         image: gcr.io/google_containers/kubernetes-dashboard-amd64:v1.6.0
---
>         image: sz-pg-oam-docker-hub-001.tendcloud.com/library/kubernetes-
dashboard-amd64:v1.6.0
```

执行所有定义文件

```
$ pwd
/root/kubernetes/cluster/addons/dashboard
$ ls *.yaml
dashboard-controller.yaml  dashboard-service.yaml
$ kubectl create -f  .
service "kubernetes-dashboard" created
deployment "kubernetes-dashboard" created
```

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

有以下三种方式：

- kubernetes-dashboard 服务暴露了 NodePort，可以使用 `http://NodeIP:nodePort` 地址访问 dashboard；
- 通过 kube-apiserver 访问 dashboard（https 6443端口和http 8080端口方式）；
- 通过 kubectl proxy 访问 dashboard：

通过 kubectl proxy 访问 dashboard

启动代理

```
$ kubectl proxy --address='172.20.0.113' --port=8086 --accept-hosts='^*$'
Starting to serve on 172.20.0.113:8086
```

- 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://172.20.0.113:8086/ui`
自动跳转到：`http://172.20.0.113:8086/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://172.20.0.113:6443
KubeDNS is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-system/services/kube-dns
kubernetes-dashboard is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard
```

浏览器访问 URL：`https://172.20.0.113:6443/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，已经解决。

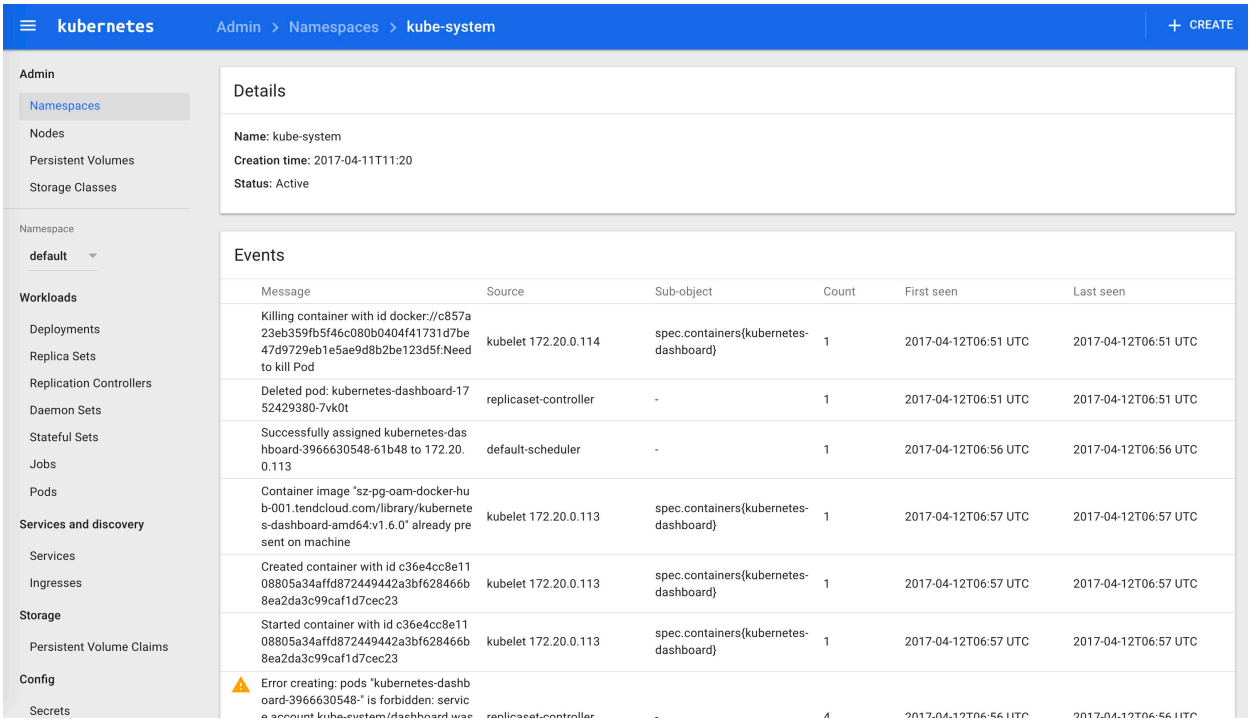
导入证书

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。

如果你不想使用**https**的话，可以直接访问insecure port 8080端口:`http://172.20.0.113:8080/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard`



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

9.配置和安装 Heapster

到 heapster release 页面 下载最新版本的 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
```

我们自己创建了heapster的rbac配置`heapster-rbac.yaml`。

已经修改好的 yaml 文件见：heapster

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: sz-pg-oam-docker-hub-001.tendcloud.com/library/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: /
```

- 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-deployment

```
$ diff heapster-deployment.yaml.orig heapster-deployment.yaml
16c16
<         image: gcr.io/google_containers/heapster-amd64:v1.3.0-beta.1
---
>         image: sz-pg-oam-docker-hub-001.tendcloud.com/library/heapster-
amd64:v1.3.0-beta.1
```

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

```
$ # 导出镜像中的 influxdb 配置文件
$ docker run --rm --entrypoint 'cat'  -ti lvanneo/heapster-influxdb-amd64:v1.1.1
/etc/config.toml >config.toml.orig
$ cp config.toml.orig config.toml
$ # 修改：启用 admin 接口
$ vim config.toml
$ diff config.toml.orig config.toml
35c35
<   enabled = false
---
>   enabled = true
$ # 将修改后的配置写入到 ConfigMap 对象中
$ kubectl create configmap influxdb-config --from-file=config.toml  -n kube-system
configmap "influxdb-config" created
$ # 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

```
$ pwd
/root/heapster-1.3.0/deploy/kube-config/influxdb
$ 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
```

检查执行结果

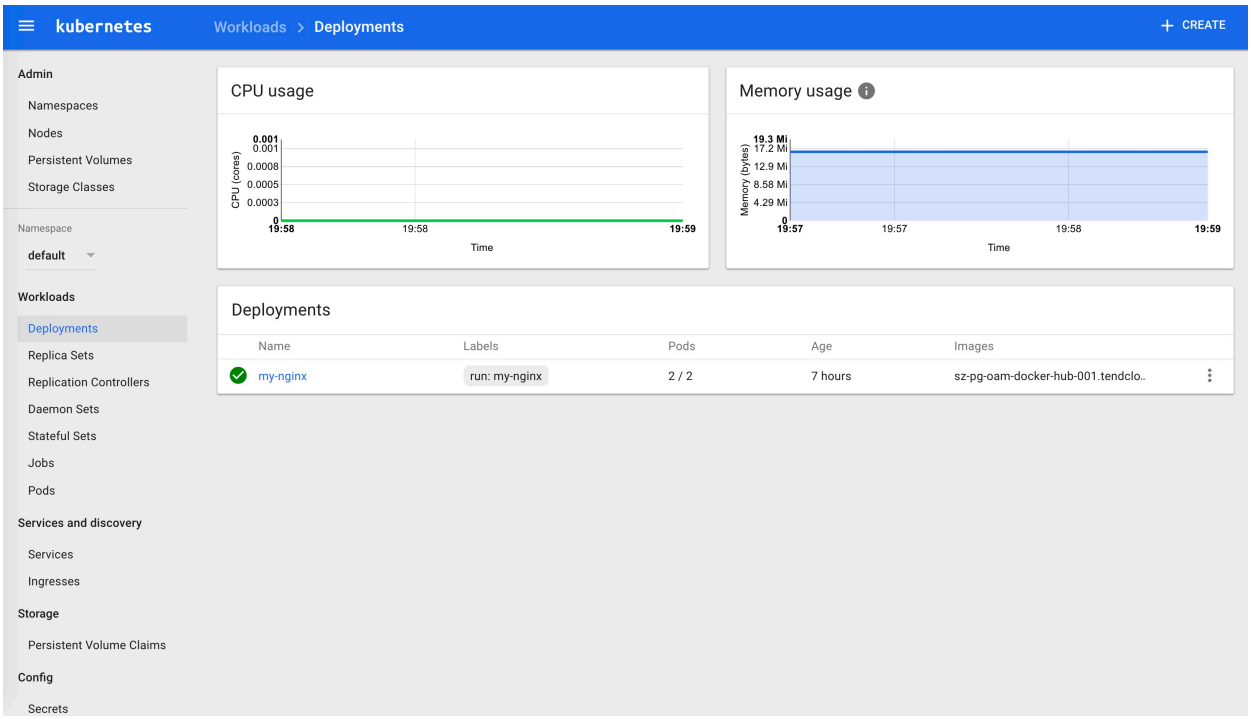
检查 Deployment

```
$ 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://172.20.0.113:6443
   Heapster is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-system/services/heapster
   KubeDNS is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-system/services/kube-dns
   kubernetes-dashboard is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard
   monitoring-grafana is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana
   monitoring-influxdb is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-system/services/monitoring-influxdb

   To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
   ```

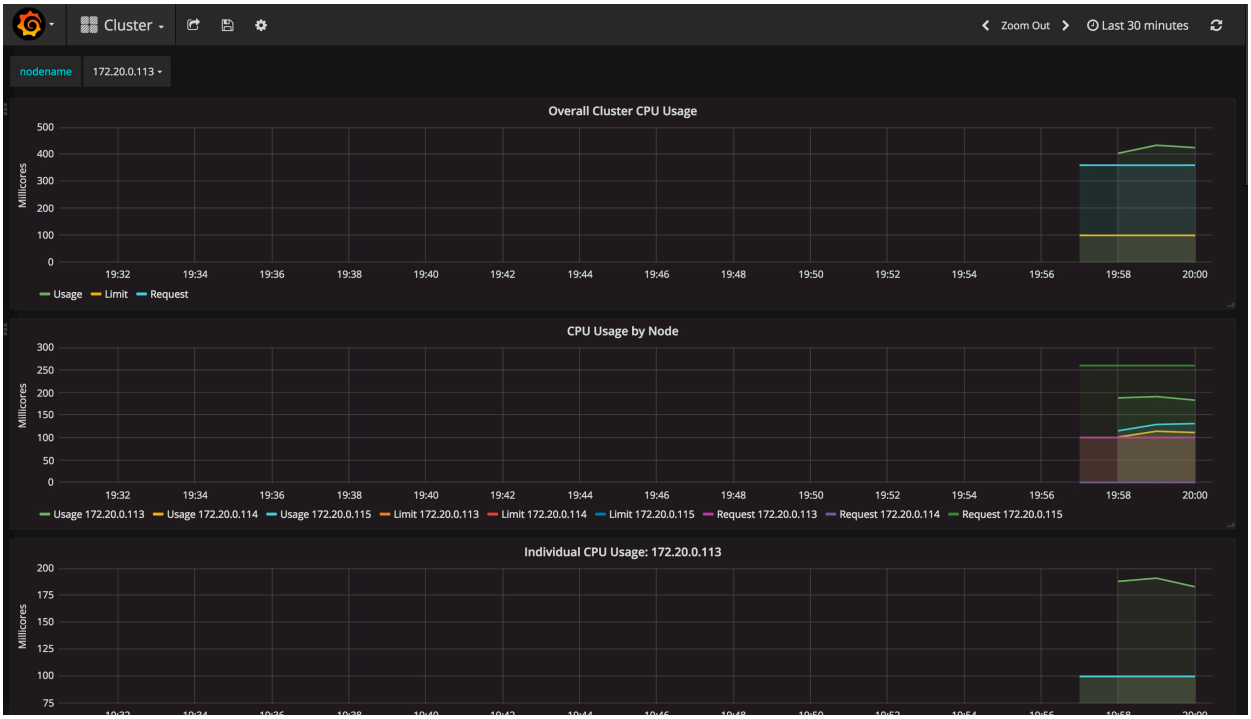
   浏览器访问 URL： `http://172.20.0.113:8080/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana`

2. 通过 kubectl proxy 访问：

   创建代理

```
$ kubectl proxy --address='172.20.0.113' --port=8086 --accept-hosts='^*$'
Starting to serve on 172.20.0.113:8086
```

浏览器访问 URL：`http://172.20.0.113:8086/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana`



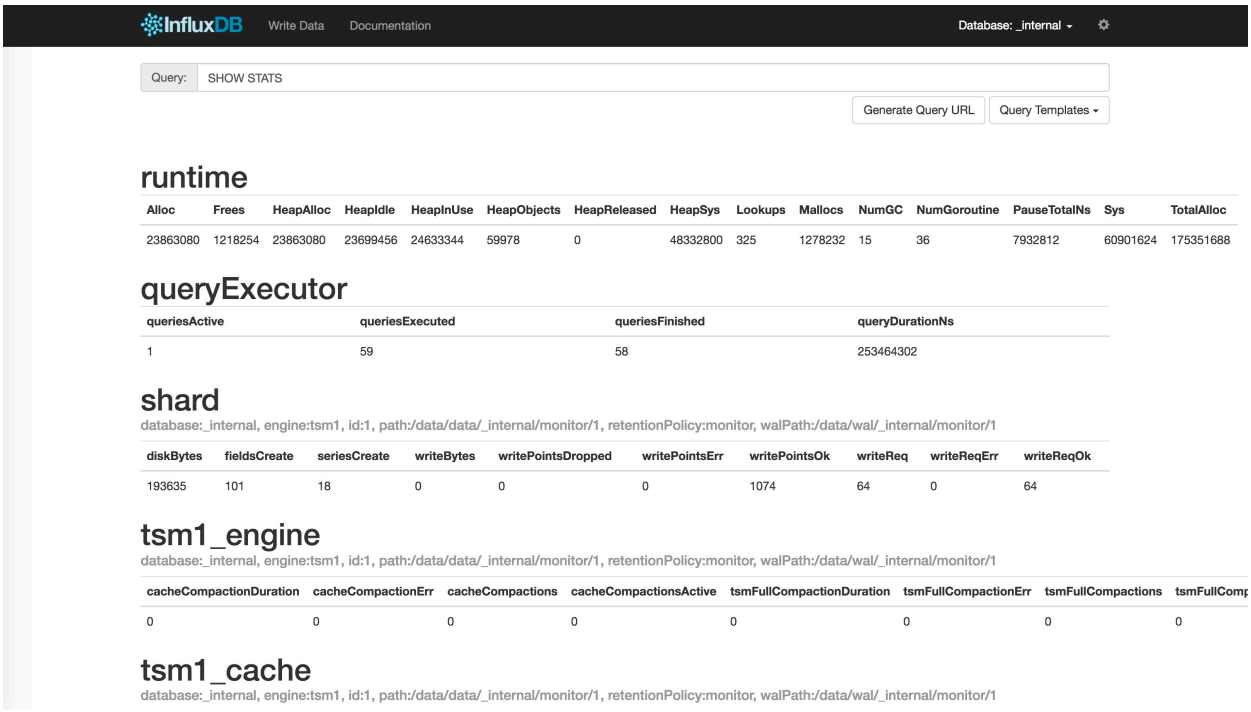
访问 influxdb admin UI

获取 influxdb http 8086 映射的 NodePort

```
$ kubectl get svc -n kube-system|grep influxdb
monitoring-influxdb    10.254.22.46    <nodes>       8086:32299/TCP,8083:30269/TCP
9m
```

通过 kube-apiserver 的**非安全端口**访问 influxdb 的 admin UI 界面：
`http://172.20.0.113:8080/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 32299，点击 “Save” 即可（我的集群中的地址是172.20.0.113:32299）：



10.配置和安装 EFK

官方文件目录：`cluster/addons/fluentd-elasticsearch`

```
$ 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

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: sz-pg-oam-docker-hub-
001.tendcloud.com/library/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: sz-pg-oam-docker-hub-001.tendcloud.com/library/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: sz-pg-oam-docker-hub-001.tendcloud.com/library/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
172.20.0.113   Ready     1d        v1.6.0

$ kubectl label nodes 172.20.0.113 beta.kubernetes.io/fluentd-ds-ready=true
node "172.20.0.113" 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":"S
tatus 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","messa
ge":"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":"Sta
tus 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":"Statu
s 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":"Statu
s 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":"S
tatus 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":"St
atus 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://172.20.0.113:6443
Elasticsearch is running at
https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-
system/services/elasticsearch-logging
Heapster is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-
system/services/heapster
Kibana is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-
system/services/kibana-logging
KubeDNS is running at https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-
system/services/kube-dns
kubernetes-dashboard is running at
https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-
system/services/kubernetes-dashboard
monitoring-grafana is running at
https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-
system/services/monitoring-grafana
monitoring-influxdb is running at
https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-
system/services/monitoring-influxdb
```

浏览器访问 URL：`https://172.20.0.113:6443/api/v1/proxy/namespaces/kube-system/services/kibana-logging/app/kibana`

2. 通过 kubectl proxy 访问：

   创建代理

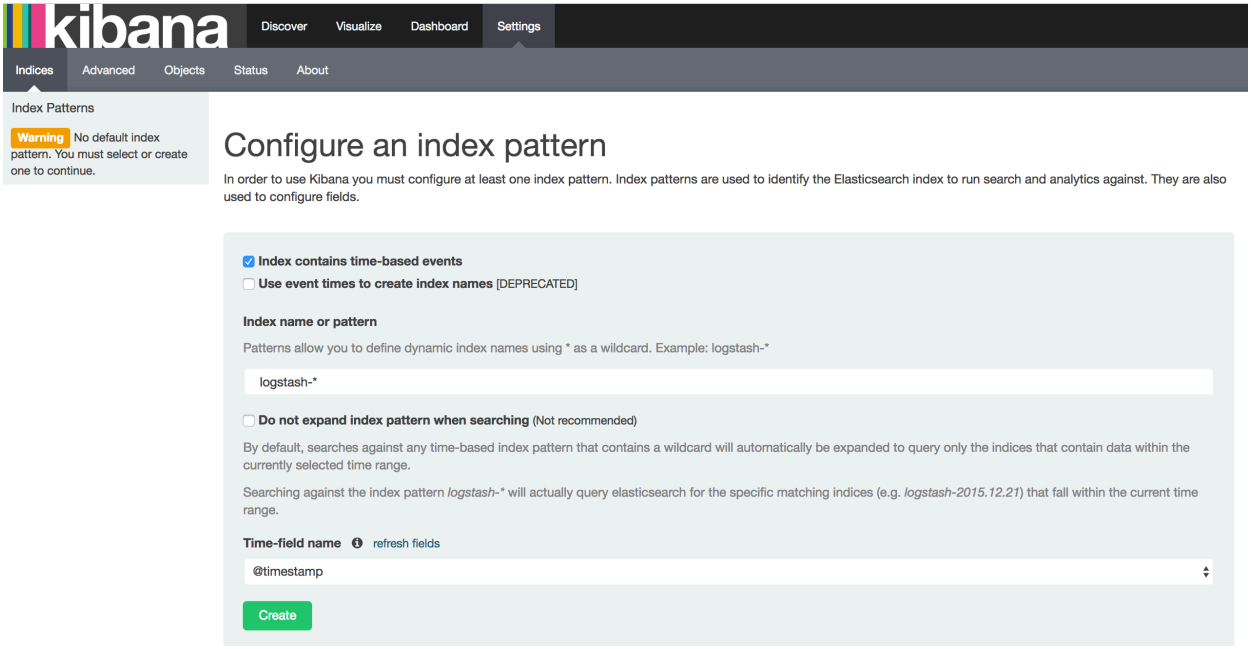
```
$ kubectl proxy --address='172.20.0.113' --port=8086 --accept-hosts='^*$'
Starting to serve on 172.20.0.113:8086
```

浏览器访问 URL：`http://172.20.0.113:8086/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

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。



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

到此整个kubernetes集群和插件已经安装完毕。