
Table of Contents

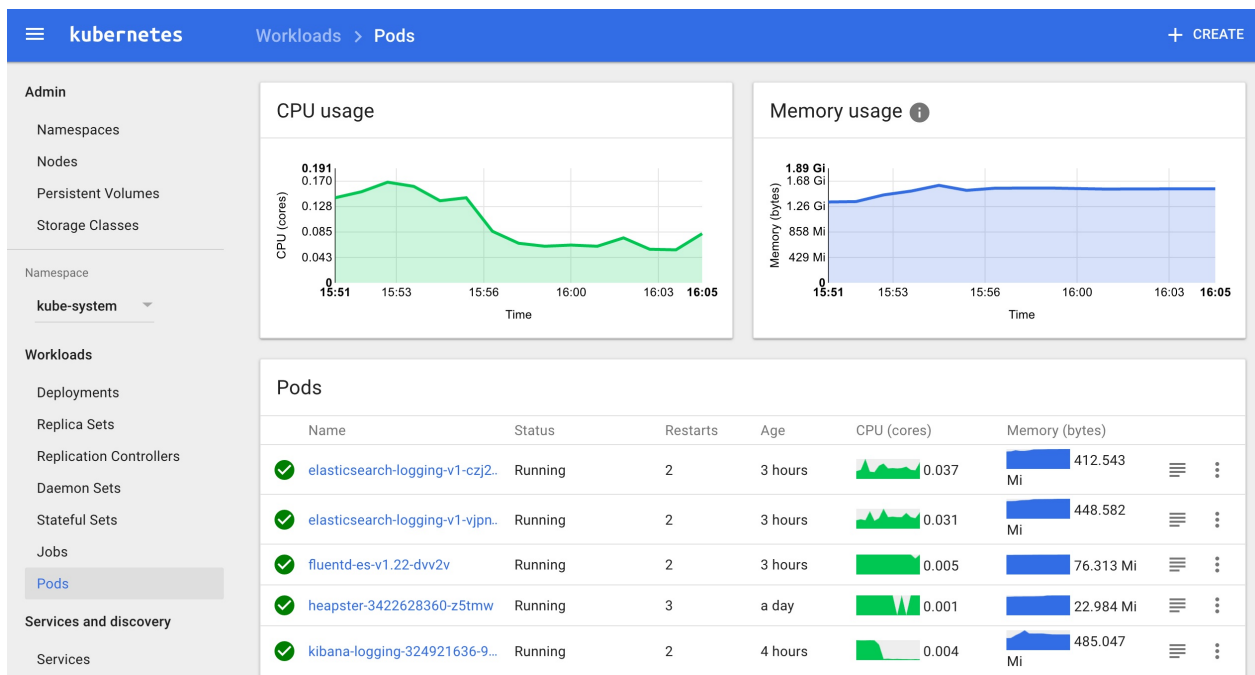
和我一步步部署 **kubernetes** 集群

Introduction	1.1
01. 组件版本和集群环境	1.2
02. 创建 CA 证书和秘钥	1.3
03. 部署高可用 Etcd 集群	1.4
04. 下载和配置 Kubectl 命令行工具	1.5
05. 配置 Flannel 网络	1.6
06. 部署 Master 节点	1.7
07. 部署 Node 节点	1.8
08. 部署 DNS 插件	1.9
09. 部署 Dashboard 插件	1.10
10. 部署 Heapster 插件	1.11
11. 部署 EFK 插件	1.12
12. 部署 Docker Registry	1.13
13. 部署 Harbor 私有仓库	1.14
14. 清理集群	1.15

标签集合

标签	2.1
----	-----

和我一步步部署 **kubernetes** 集群



图片 - *dashboard-home*

本系列文档介绍使用二进制部署最新 **kubernetes v1.6.2** 集群的所有步骤，而不是使用 **kubeadm** 等自动化方式来部署集群。

在部署的过程中，将详细列出各组件的启动参数，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。

本系列文档适用于 **CentOS 7**、**Ubuntu 16.04** 及以上版本系统，随着各组件的更新而更新，有任何问题欢迎提 **issue**！

由于启用了 **TLS** 双向认证、**RBAC** 授权等严格的安全机制，建议从头开始部署，否则可能会认证、授权等失败！

步骤列表

1. 组件版本和集群环境
2. 创建 CA 证书和密钥
3. 部署高可用 Etcd 集群
4. 下载和配置 Kubectl 命令行工具
5. 配置 Flannel 网络
6. 部署 Master 节点
7. 部署 Node 节点
8. 部署 DNS 插件
9. 部署 Dashboard 插件
10. 部署 Heapster 插件
11. 部署 EFK 插件
12. 部署 Docker Registry
13. 部署 Harbor 私有仓库
14. 清理集群

在线阅读

- 建议：[GitBook](#)
- [Github](#)

电子书

- pdf 格式 [下载](#)
- epub 格式 [下载](#)

版权

Copyright 2017 zhangjun (geekard@qq.com)

Apache License 2.0，详情见 [LICENSE](#) 文件。

如果你觉得这份文档对你有帮助，请微信扫描下方二维码进行捐赠，加油后的 opsnul1 将会和你分享更多的原创教程，谢谢！



zhangjun 最后更新： 2017-08-11 02:50:58

tags: kubernetes, environment

组件版本和集群环境

集群组件和版本

- Kubernetes 1.6.2
- Docker 17.04.0-ce
- Etcd 3.1.6
- Flannel 0.7.1 vxlan 网络
- TLS 认证通信 (所有组件，如 etcd、kubernetes master 和 node)
- RBAC 授权
- kubelet TLS BootStrapping
- kubedns、dashboard、heapster (influxdb、grafana)、EFK (elasticsearch、fluentd、kibana) 插件
- 私有 docker registry，使用 ceph rgw 后端存储，TLS + HTTP Basic 认证

集群机器

- 10.64.3.7
- 10.64.3.8
- 10.66.3.86

本着测试的目的，etcd 集群、kubernetes master 集群、kubernetes node 均使用这三台机器。

若有安装 Vagrant 与 Virtualbox，这三台机器可以用本着提供的 Vagrantfile 来建置：

```
$ cd vagrant
$ vagrant up
```

集群环境变量

后续的部署步骤将使用下面定义的全局环境变量，根据自己的机器、网络情况修改：

```
# TLS Bootstrapping 使用的 Token，可以使用命令 head -c 16 /dev/urandom | od -An -t x | tr -d ' ' 生成
BOOTSTRAP_TOKEN="41f7e4ba8b7be874fcff18bf5cf41a7c"

# 建议用 未用的网段 来定义服务网段和 Pod 网段

# 服务网段 (Service CIDR)，部署前路由不可达，部署后集群内使用 IP:Port 可达
SERVICE_CIDR="10.254.0.0/16"

# POD 网段 (Cluster CIDR)，部署前路由不可达，**部署后**路由可达 (flanneld 保证)
CLUSTER_CIDR="172.30.0.0/16"

# 服务端口范围 (NodePort Range)
NODE_PORT_RANGE="8400-9000"

# etcd 集群服务地址列表
ETCD_ENDPOINTS="https://10.64.3.7:2379,https://10.64.3.8:2379,https://10.66.3.86:2379"

# flanneld 网络配置前缀
FLANNEL_ETCD_PREFIX="/kubernetes/network"

# kubernetes 服务 IP (预分配，一般是 SERVICE_CIDR 中第一个IP)
CLUSTER_KUBERNETES_SVC_IP="10.254.0.1"

# 集群 DNS 服务 IP (从 SERVICE_CIDR 中预分配)
CLUSTER_DNS_SVC_IP="10.254.0.2"

# 集群 DNS 域名
CLUSTER_DNS_DOMAIN="cluster.local."
```

- 打包后的变量定义见 [environment.sh](#)，后续部署时会提示导入该脚本；

分发集群环境变量定义脚本

把全局变量定义脚本拷贝到所有机器的 `/root/local/bin` 目录：

```
$ cp environment.sh /root/local/bin
$
```

zhangjun 最后更新：2017-08-11 02:50:58

tags: TLS, CA

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](#) 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 config > config.json
$ cfssl print-defaults csr > 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、私钥文件、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 Au
thentication
            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:kube
rnetes.default.svc, DNS:kubernetes.default.svc.cluster, DNS:kube
rnetes.default.svc.cluster.local, IP Address:127.0.0.1, IP Addre
ss: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": "17436049287242326347315197163229289570712902
2309",
  "sans": [
    "kubernetes",
```

```
"kubernetes.default",
"kubernetes.default.svc",
"kubernetes.default.svc.cluster",
"kubernetes.default.svc.cluster.local",
"127.0.0.1",
"10.64.3.7",
"10.64.3.8",
"10.66.3.86",
"10.254.0.1"
],
"not_before": "2017-04-05T05:36:00Z",
"not_after": "2018-04-05T05:36:00Z",
"sigalg": "SHA256WithRSA",
...
```

参考

- [Generate self-signed certificates](#)
- [Setting up a Certificate Authority and Creating TLS Certificates](#)
- [Client Certificates V/s Server Certificates](#)

zhangjun 最后更新： 2017-08-11 02:50:58

tags: etcd

部署高可用 etcd 集群

kubernetes 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 `etcd-host0` 、 `etcd-host1` 、 `etcd-host2` ：

- `etcd-host0` : 10.64.3.7
- `etcd-host1` : 10.64.3.8
- `etcd-host2` : 10.66.3.86

使用的变量

本文档用到的变量定义如下：

```
$ export NODE_NAME=etcd-host0 # 当前部署的机器名称(随便定义，只要能区分不同机器即可)
$ export NODE_IP=10.64.3.7 # 当前部署的机器 IP
$ export NODE_IPS="10.64.3.7 10.64.3.8 10.66.3.86" # etcd 集群所有机器 IP
$ # etcd 集群间通信的IP和端口
$ export ETCD_NODES=etcd-host0=https://10.64.3.7:2380,etcd-host1=https://10.64.3.8:2380,etcd-host2=https://10.66.3.86:2380
$ # 导入用到的其它全局变量：ETCD_ENDPOINTS、FLANNEL_ETCD_PREFIX、CLUSTER_CIDR
$ source /root/local/bin/environment.sh
$
```

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：

```
$ wget https://github.com/coreos/etcd/releases/download/v3.1.6/etcd-v3.1.6-linux-amd64.tar.gz
$ tar -xvf etcd-v3.1.6-linux-amd64.tar.gz
$ sudo mv etcd-v3.1.6-linux-amd64/etcd* /root/local/bin
$
```

创建 TLS 秘钥和证书

为了保证通信安全，客户端(如 etcdctl)与 etcd 集群、etcd 集群之间的通信需要使用 TLS 加密，本节创建 etcd TLS 加密所需的证书和私钥。

创建 etcd 证书签名请求：

```
$ cat > etcd-csr.json <<EOF
{
  "CN": "etcd",
  "hosts": [
    "127.0.0.1",
    "${NODE_IP}"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "System"
    }
  ]
}
EOF
```

- hosts 字段指定授权使用该证书的 etcd 节点 IP；

生成 etcd 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 etcd-csr.json | cfssljson -bare etcd  
$ ls etcd*  
etcd.csr  etcd-csr.json  etcd-key.pem  etcd.pem  
$ sudo mkdir -p /etc/etcd/ssl  
$ sudo mv etcd*.pem /etc/etcd/ssl  
$ rm etcd.csr  etcd-csr.json
```

创建 **etcd** 的 **systemd unit** 文件

```
$ sudo mkdir -p /var/lib/etcd # 必须先创建工作目录
$ 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/
ExecStart=/root/local/bin/etcd \\\
    --name=${NODE_NAME} \\\
    --cert-file=/etc/etcd/ssl/etcd.pem \\\
    --key-file=/etc/etcd/ssl/etcd-key.pem \\\
    --peer-cert-file=/etc/etcd/ssl/etcd.pem \\\
    --peer-key-file=/etc/etcd/ssl/etcd-key.pem \\\
    --trusted-ca-file=/etc/kubernetes/ssl/ca.pem \\\
    --peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \\\
    --initial-advertise-peer-urls=https://${NODE_IP}:2380 \\\
    --listen-peer-urls=https://${NODE_IP}:2380 \\\
    --listen-client-urls=https://${NODE_IP}:2379,http://127.0.0.1:2379 \\\
    --advertise-client-urls=https://${NODE_IP}:2379 \\\
    --initial-cluster-token=etcd-cluster-0 \\\
    --initial-cluster=${ETCD_NODES} \\\
    --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` 的工作目录和数据目录为 `/var/lib/etcd`，需在启动服务前创建这个目录；
- 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）；

- `--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  
$
```

最先启动的 etcd 进程会卡住一段时间，等待其它节点上的 etcd 进程加入集群，为正常现象。

在所有的 etcd 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

部署完 etcd 集群后，在任一 etcd 集群节点上执行如下命令：

```
$ for ip in ${NODE_IPS}; do  
    ETCDCTL_API=3 /root/local/bin/etcdctl \  
    --endpoints=https://${ip}:2379 \  
    --cacert=/etc/kubernetes/ssl/ca.pem \  
    --cert=/etc/etcd/ssl/etcd.pem \  
    --key=/etc/etcd/ssl/etcd-key.pem \  
    endpoint health; done
```

预期结果：

```
2017-04-10 14:50:50.011317 I | warning: ignoring ServerName for
user-provided CA for backwards compatibility is deprecated
https://10.64.3.7:2379 is healthy: successfully committed propos
al: took = 1.687897ms
2017-04-10 14:50:50.061577 I | warning: ignoring ServerName for
user-provided CA for backwards compatibility is deprecated
https://10.64.3.8:2379 is healthy: successfully committed propos
al: took = 1.246915ms
2017-04-10 14:50:50.104718 I | warning: ignoring ServerName for
user-provided CA for backwards compatibility is deprecated
https://10.66.3.86:2379 is healthy: successfully committed propo
sal: took = 1.509229ms
```

三台 etcd 的输出均为 healthy 时表示集群服务正常（忽略 warning 信息）。

zhangjun 最后更新：2017-08-11 02:50:58

tags: kubectl

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=10.64.3.7 # 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。

zhangjun 最后更新：2017-08-11 02:50:58

tags: flanneld

部署 Flannel 网络

kubernetes 要求集群内各节点能通过 Pod 网段互联互通，本文档介绍使用 Flannel 在所有节点 (Master、Node) 上创建互联互通的 Pod 网段的步骤。

使用的变量

本文档用到的变量定义如下：

```
$ export NODE_IP=10.64.3.7 # 当前部署节点的 IP
$ # 导入用到的其它全局变量：ETCD_ENDPOINTS、FLANNEL_ETCD_PREFIX、CLUSTER_CIDR
$ source /root/local/bin/environment.sh
$
```

创建 TLS 秘钥和证书

etcd 集群启用了双向 TLS 认证，所以需要为 flanneld 指定与 etcd 集群通信的 CA 和秘钥。

创建 flanneld 证书签名请求：

```
$ cat > flanneld-csr.json <<EOF
{
  "CN": "flanneld",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "System"
    }
  ]
}
EOF
```

- hosts 字段为空；

生成 flanneld 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 flanneld-csr.json | cfssljson -bare flanneld
$ ls flanneld*
flanneld.csr  flanneld-csr.json  flanneld-key.pem  flanneld.pem
$ sudo mkdir -p /etc/flanneld/ssl
$ sudo mv flanneld*.pem /etc/flanneld/ssl
$ rm flanneld.csr  flanneld-csr.json
```

向 etcd 写入集群 Pod 网段信息

注意：本步骤只需在第一次部署 Flannel 网络时执行，后续在其它节点上部署 Flannel 时无需再写入该信息！

```
$ /root/local/bin/etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/flanneld/ssl/flanneld.pem \
  --key-file=/etc/flanneld/ssl/flanneld-key.pem \
  set ${FLANNEL_ETCD_PREFIX}/config '{"Network":"'${CLUSTER_CIDR}'
  ', "SubnetLen": 24, "Backend": {"Type": "vxlan"}}'
```

- flanneld 目前版本 (**v0.7.1**) 不支持 **etcd v3**，故使用 etcd v2 API 写入配置 key 和网段数据；
- 写入的 Pod 网段(`${CLUSTER_CIDR}`，172.30.0.0/16) 必须与 kube-controller-manager 的 `--cluster-cidr` 选项值一致；

安装和配置 flanneld

下载 flanneld

```
$ mkdir flannel
$ wget https://github.com/coreos/flannel/releases/download/v0.7.1/flannel-v0.7.1-linux-amd64.tar.gz
$ tar -xzvf flannel-v0.7.1-linux-amd64.tar.gz -C flannel
$ sudo cp flannel/{flanneld,mk-docker-opts.sh} /root/local/bin
$
```

创建 flanneld 的 systemd unit 文件

```
$ cat > flanneld.service << EOF
[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
ExecStart=/root/local/bin/flanneld \\\
    -etcd-cafile=/etc/kubernetes/ssl/ca.pem \\\
    -etcd-certfile=/etc/flanneld/ssl/flanneld.pem \\\
    -etcd-keyfile=/etc/flanneld/ssl/flanneld-key.pem \\\
    -etcd-endpoints=${ETCD_ENDPOINTS} \\\
    -etcd-prefix=${FLANNEL_ETCD_PREFIX}
ExecStartPost=/root/local/bin/mk-docker-opts.sh -k DOCKER_NETWORK_OPTIONS -d /run/flannel/docker
Restart=on-failure

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

- `mk-docker-opts.sh` 脚本将分配给 `flanneld` 的 Pod 子网网段信息写入到 `/run/flannel/docker` 文件中，后续 `docker` 启动时使用这个文件中参数值设置 `docker0` 网桥；
- `flanneld` 使用系统缺省路由所在的接口和其它节点通信，对于有多个网络接口的机器（如，内网和公网），可以用 `--iface` 选项值指定通信接口(上面的 `systemd unit` 文件没指定这个选项)，如本着 `Vagrant + Virtualbox`，就要指定 `--iface=enp0s8`；

完整 unit 见 [flanneld.service](#)

启动 flanneld

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

检查 flanneld 服务

```
$ journalctl -u flanneld |grep 'Lease acquired'  
$ ifconfig flannel.1  
$
```

检查分配给各 flanneld 的 Pod 网段信息

```
$ # 查看集群 Pod 网段(/16)
$ /root/local/bin/etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/flanneld/ssl/flanneld.pem \
  --key-file=/etc/flanneld/ssl/flanneld-key.pem \
  get ${FLANNEL_ETCD_PREFIX}/config
{ "Network": "172.30.0.0/16", "SubnetLen": 24, "Backend": { "Type": "vxlan" } }
$ # 查看已分配的 Pod 子网段列表(/24)
$ /root/local/bin/etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/flanneld/ssl/flanneld.pem \
  --key-file=/etc/flanneld/ssl/flanneld-key.pem \
  ls ${FLANNEL_ETCD_PREFIX}/subnets
/kubernetes/network/subnets/172.30.19.0-24
$ # 查看某一 Pod 网段对应的 flanneld 进程监听的 IP 和网络参数
$ /root/local/bin/etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/flanneld/ssl/flanneld.pem \
  --key-file=/etc/flanneld/ssl/flanneld-key.pem \
  get ${FLANNEL_ETCD_PREFIX}/subnets/172.30.19.0-24
{"PublicIP":"10.64.3.7", "BackendType":"vxlan", "BackendData":{"VtepMAC":"d6:51:2e:80:5c:69"}}
```

确保各节点间 **Pod** 网段能互联互通

在各节点上部署完 Flannel 后，查看已分配的 Pod 子网段列表(/24)

```
$ /root/local/bin/etcdctl \  
  --endpoints=${ETCD_ENDPOINTS} \  
  --ca-file=/etc/kubernetes/ssl/ca.pem \  
  --cert-file=/etc/flannel/ssl/flannel.pem \  
  --key-file=/etc/flannel/ssl/flannel-key.pem \  
  ls ${FLANNEL_ETCD_PREFIX}/subnets  
/kubernetes/network/subnets/172.30.19.0-24  
/kubernetes/network/subnets/172.30.20.0-24  
/kubernetes/network/subnets/172.30.21.0-24
```

当前三个节点分配的 Pod 网段分别是：172.30.19.0-24、172.30.20.0-24、172.30.21.0-24。

在各节点上分配 ping 这三个网段的网关地址，确保能通：

```
$ ping 172.30.19.1  
$ ping 172.30.20.2  
$ ping 172.30.21.3  
$
```

zhangjun 最后更新：2017-08-11 02:50:58

tags: master, kube-apiserver, kube-scheduler, kube-controller-manager

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=10.64.3.7 # 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](#) 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
...
```

2. 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/v1.6.2/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 [05-部署Flannel网络.md](#)

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",
    "${MASTER_IP}",
    "${CLUSTER_KUBERNETES_SVC_IP}",
    "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 kube
rnetes
$ ls kubernetes*
kubernetes.csr  kubernetes-csr.json  kubernetes-key.pem  kuberne
tes.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-boots
trap"
EOF
$ mv token.csv /etc/kubernetes/
$
```

创建 kube-apiserver 的 systemd unit 文件

```
$ cat > kube-apiserver.service <<EOF
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target

[Service]
ExecStart=/root/local/bin/kube-apiserver \\\
    --admission-control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota \\\
    --advertise-address=${MASTER_IP} \\\
    --bind-address=${MASTER_IP} \\\
    --insecure-bind-address=${MASTER_IP} \\\
    --authorization-mode=RBAC \\\
    --runtime-config=rbac.authorization.k8s.io/v1alpha1 \\\
    --kubelet-https=true \\\
    --experimental-bootstrap-token-auth \\\
    --token-auth-file=/etc/kubernetes/token.csv \\\
    --service-cluster-ip-range=${SERVICE_CIDR} \\\
    --service-node-port-range=${NODE_PORT_RANGE} \\\
    --tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem \\\
    --tls-private-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \\\
    --client-ca-file=/etc/kubernetes/ssl/ca.pem \\\
    --service-account-key-file=/etc/kubernetes/ssl/ca-key.pem \\\
    --etcd-cafile=/etc/kubernetes/ssl/ca.pem \\\
    --etcd-certfile=/etc/kubernetes/ssl/kubernetes.pem \\\
    --etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem \\\
    --etcd-servers=${ETCD_ENDPOINTS} \\\
    --enable-swagger-ui=true \\\
    --allow-privileged=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 \\\
    --v=2
Restart=on-failure
RestartSec=5
Type=notify
```

```
LimitNOFILE=65536
```

```
[Install]
```

```
WantedBy=multi-user.target
```

```
EOF
```

- 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 Bootstrap 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=${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](#)

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 <<EOF  
[Unit]  
Description=Kubernetes Controller Manager  
Documentation=https://github.com/GoogleCloudPlatform/kubernetes  
  
[Service]  
ExecStart=/root/local/bin/kube-controller-manager \  
    --address=127.0.0.1 \  
    --master=http://${MASTER_IP}:8080 \  
    --allocate-node-cidrs=true \  
    --service-cluster-ip-range=${SERVICE_CIDR} \  
    --cluster-cidr=${CLUSTER_CIDR} \  
    --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 \  
    --v=2  
Restart=on-failure  
RestartSec=5  
  
[Install]  
WantedBy=multi-user.target  
EOF
```

- `--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://127.0.0.1:1025
2/healthz: dial tcp 127.0.0.1:10252: getsockopt: connection
refused
scheduler           Unhealthy  Get http://127.0.0.1:1025
1/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(flannel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](#)

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 <<EOF  
[Unit]  
Description=Kubernetes Scheduler  
Documentation=https://github.com/GoogleCloudPlatform/kubernetes  
  
[Service]  
ExecStart=/root/local/bin/kube-scheduler \  
    --address=127.0.0.1 \  
    --master=http://{MASTER_IP}:8080 \  
    --leader-elect=true \  
    --v=2  
Restart=on-failure  
RestartSec=5  
  
[Install]  
WantedBy=multi-user.target  
EOF
```

- `--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](#)。

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"}	

zhangjun 最后更新：2017-08-11 02:50:58

tags: node, flanneld, docker, kubeconfig, kubelet, kube-proxy

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

本文档用到的变量定义如下：

```
$ # 替换为 kubernetes master 集群任一机器 IP
$ export MASTER_IP=10.64.3.7
$ export KUBE_APISERVER="https://${MASTER_IP}:6443"
$ # 当前部署的节点 IP
$ export NODE_IP=10.64.3.7
$ # 导入用到的其它全局变量：ETCD_ENDPOINTS、FLANNEL_ETCD_PREFIX、CLUSTER_CIDR、CLUSTER_DNS_SVC_IP、CLUSTER_DNS_DOMAIN、SERVICE_CIDR
$ source /root/local/bin/environment.sh
$
```

安装和配置 flanneld

参考 [05-部署Flannel网络.md](#)

安装和配置 docker

下载最新的 docker 二进制文件

```
$ wget https://get.docker.com/builds/Linux/x86_64/docker-17.04.0-ce.tgz
$ tar -xvf docker-17.04.0-ce.tgz
$ cp docker/docker* /root/local/bin
$ cp docker/completion/bash/docker /etc/bash_completion.d/
$
```

创建 **docker** 的 **systemd unit** 文件

```
$ cat docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=http://docs.docker.io

[Service]
Environment="PATH=/root/local/bin:/bin:/sbin:/usr/bin:/usr/sbin"
EnvironmentFile=-/run/flannel/docker
ExecStart=/root/local/bin/dockerd --log-level=error $DOCKER_NETWORK_OPTIONS
ExecReload=/bin/kill -s HUP $MAINPID
Restart=on-failure
RestartSec=5
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
Delegate=yes
KillMode=process

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

- dockerd 运行时会调用其它 docker 命令，如 docker-proxy，所以需要将 docker 命令所在的目录加到 PATH 环境变量中；
- flanneld 启动时将网络配置写入到 `/run/flannel/docker` 文件中的变量 `DOCKER_NETWORK_OPTIONS`，dockerd 命令行上指定该变量值来设置 docker0 网桥参数；
- 如果指定了多个 `EnvironmentFile` 选项，则必须将 `/run/flannel/docker` 放在最后(确保 docker0 使用 flanneld 生成的 `bip` 参

数)；

- 不能关闭默认开启的 `--iptables` 和 `--ip-masq` 选项；
- 如果内核版本比较新，建议使用 `overlay` 存储驱动；
- docker 从 1.13 版本开始，可能将 **iptables FORWARD chain** 的默认策略设置为 **DROP**，从而导致 ping 其它 Node 上的 Pod IP 失败，遇到这种情况时，需要手动设置策略为 `ACCEPT`：

```
$ sudo iptables -P FORWARD ACCEPT
$
```

并且把以下命令写入 `/etc/rc.local` 文件中，防止节点重启 **iptables FORWARD chain** 的默认策略又还原为 **DROP**

```
sleep 60 && /sbin/iptables -P FORWARD ACCEPT
```

- 为了加快 pull image 的速度，可以使用国内的仓库镜像服务器，同时增加下载的并发数。(如果 `dockerd` 已经运行，则需要重启 `dockerd` 生效。)

```
$ cat /etc/docker/daemon.json
{
  "registry-mirrors": ["https://docker.mirrors.ustc.edu.cn",
    "hub-mirror.c.163.com"],
  "max-concurrent-downloads": 10
}
```

完整 unit 见 [docker.service](#)

启动 dockerd

```
$ sudo cp docker.service /etc/systemd/system/docker.service
$ sudo systemctl daemon-reload
$ sudo systemctl stop firewalld
$ sudo systemctl disable firewalld
$ sudo iptables -F && sudo iptables -X && sudo iptables -F -t nat && sudo iptables -X -t nat
$ sudo systemctl enable docker
$ sudo systemctl start docker
$
```

- 需要关闭 firewalld(centos7)/ufw(ubuntu16.04)，否则可能会重复创建的 iptables 规则；
- 最好清理旧的 iptables rules 和 chains 规则；

检查 docker 服务

```
$ docker version
$
```

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/
```

- `--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密钥和证书，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 <<EOF
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=docker.service
Requires=docker.service

[Service]
WorkingDirectory=/var/lib/kubelet
ExecStart=/root/local/bin/kubelet \\\
    --address=${NODE_IP} \\\
    --hostname-override=${NODE_IP} \\\
    --pod-infra-container-image=registry.access.redhat.com/rhel7/pod-infrastructure:latest \\\
    --experimental-bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig \\\
    --kubeconfig=/etc/kubernetes/kubelet.kubeconfig \\\
    --require-kubeconfig \\\
    --cert-dir=/etc/kubernetes/ssl \\\
    --cluster-dns=${CLUSTER_DNS_SVC_IP} \\\
    --cluster-domain=${CLUSTER_DNS_DOMAIN} \\\
    --hairpin-mode promiscuous-bridge \\\
    --allow-privileged=true \\\
    --serialize-image-pulls=false \\\
    --logtostderr=true \\\
    --v=2
ExecStartPost=/sbin/iptables -A INPUT -s 10.0.0.0/8 -p tcp --dport 4194 -j ACCEPT
ExecStartPost=/sbin/iptables -A INPUT -s 172.16.0.0/12 -p tcp --dport 4194 -j ACCEPT
ExecStartPost=/sbin/iptables -A INPUT -s 192.168.0.0/16 -p tcp --dport 4194 -j ACCEPT
ExecStartPost=/sbin/iptables -A INPUT -p tcp --dport 4194 -j DROP
Restart=on-failure
RestartSec=5

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

- `--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(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 kubernetes 的 Service IP(可以先分配，后续创建 kubernetes 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](#)

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 <<EOF
[Unit]
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target

[Service]
WorkingDirectory=/var/lib/kube-proxy
ExecStart=/root/local/bin/kube-proxy \\\
  --bind-address=${NODE_IP} \\\
  --hostname-override=${NODE_IP} \\\
  --cluster-cidr=${SERVICE_CIDR} \\\
  --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig \\\
  --logtostderr=true \\\
  --v=2
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

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

- `--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](#)

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.yml
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
10.64.3.7      Ready     8d            v1.6.2
10.64.3.8      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=10.64.3.7 # 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。

zhangjun 最后更新：2017-08-11 02:50:58

tags: kubedns

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](#)。

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/clusterrolebindings/system%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](#) 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: xuejipeng/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: xuejipeng/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: xuejipeng/k8s-dns-sidecar-amd64:v1.14.1
161,162c161,162
<         - --probe=kubedns,127.0.0.1:10053,kubernetes.default.s
vc.__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.s
vc.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 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  kubed
ns-svc.yaml
$ kubectl create -f .
$
```

检查 kubernetes 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 dat
a bytes
^C--- kube-dns.kube-system.svc.cluster.local ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```


tags: dashboard

部署 dashboard 插件

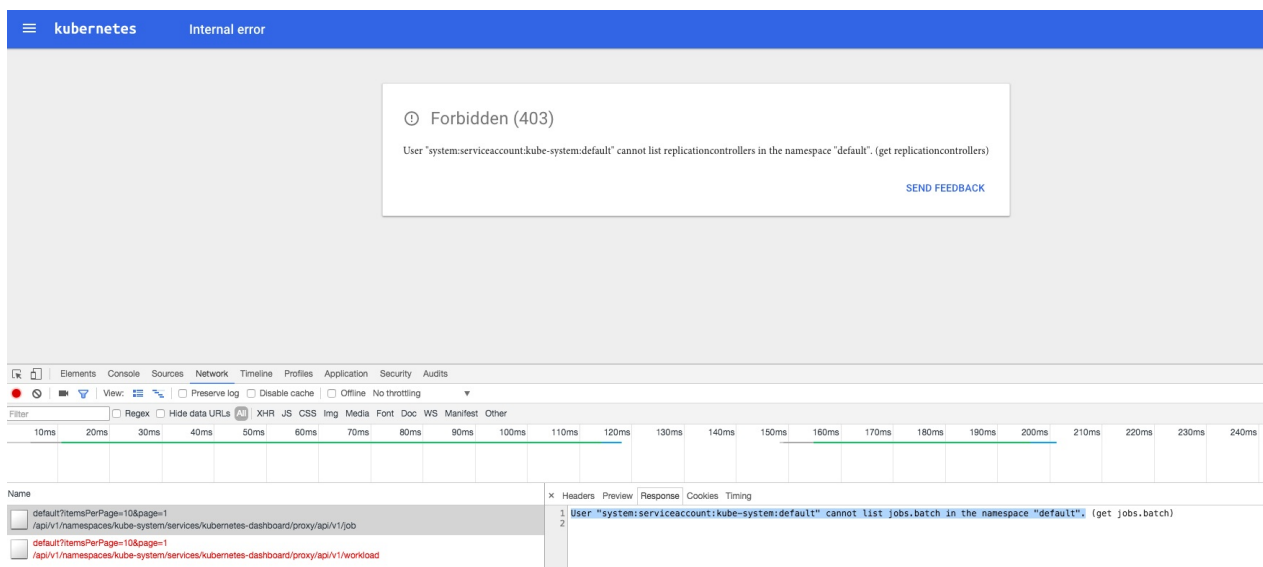
官方文件目录：`kubernetes/cluster/addons/dashboard`

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，前端界面提示：



图片 - `dashboard-403.png`

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。

已经修改好的 yaml 文件见：[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-a
md64:v1.6.0
---
>     image: cokabug/kubernetes-dashboard-amd64:v1.6.0
```

- 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-servic
e.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)
kubernetes-dashboard	10.254.224.130	<nodes>	80:30312/TCP

CP 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
kubernetes-dashboard	1	1	1	1

E AGE
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='10.64.3.7' --port=8086 --accept-hosts='^*$'
```

Starting to serve on 10.64.3.7:8086

- 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://10.64.3.7:8086/ui` 自动跳转到 : `http://10.64.3.7:8086/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard/#/workload?namespace=default`

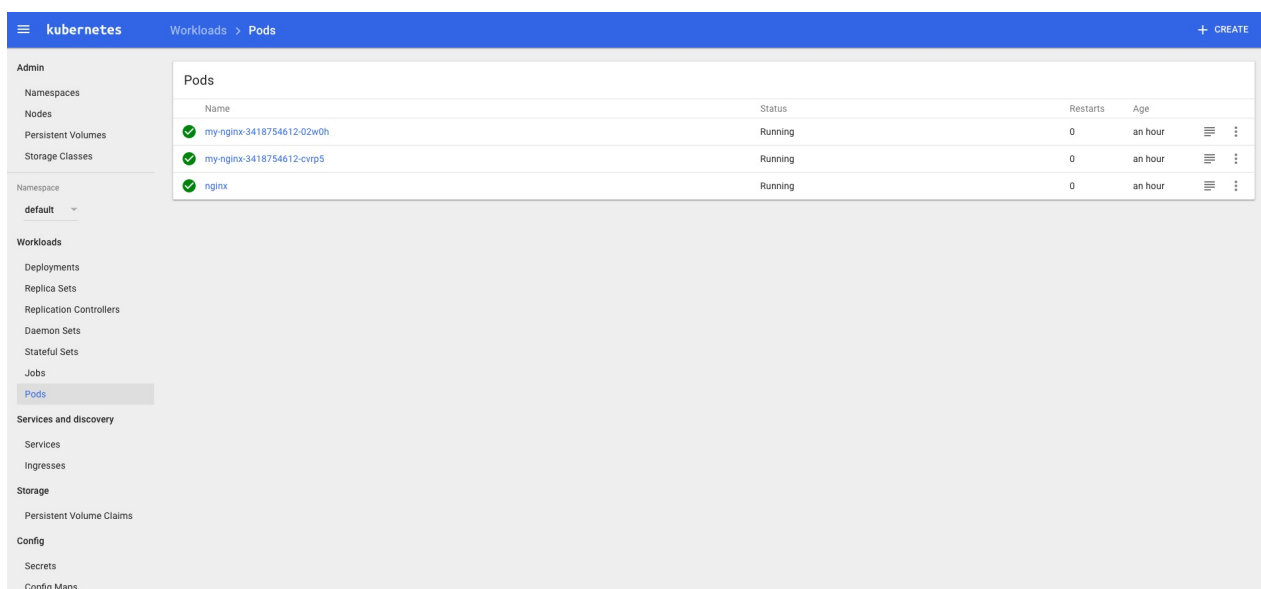
通过 kube-apiserver 访问 dashboard

获取集群服务地址列表

```
$ kubectl cluster-info
Kubernetes master is running at https://10.64.3.7:6443
KubeDNS is running at https://10.64.3.7:6443/api/v1/proxy/namespaces/kube-system/services/kube-dns
kubernetes-dashboard is running at https://10.64.3.7:6443/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://10.64.3.7:8080/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard`



图片 - *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；

zhangjun 最后更新：2017-08-11 02:50:58

tags: heapster

部署 heapster 插件

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
$
```

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

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

- 新加了 `heapster-rbac.yaml` 和 `influxdb-cm.yaml` 文件，分别定义 RoleBinding 和 influxdb 的配置；

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: lvanneo/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
13a14
>         serviceAccountName: heapster
16c17
<         image: gcr.io/google_containers/heapster-amd64:v1.3.0-beta.1
---
>         image: lvanneo/heapster-amd64:v1.3.0-beta.1
```

- 使用的是自定义的、名为 heapster 的 ServiceAccount；

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。相关步骤如下：

注意：无需自己导出、修改和创建 ConfigMap，可以直接使用放在 manifests 目录下的 [ConfigMap 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: gcr.io/google_containers/heapster-influxdb-amd64:v1.1.1
---
>         image: lvanneo/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-deployment.yaml  heapster-deployment.yaml  heapster-service.yaml  influxdb-deployment.yaml
grafana-service.yaml     heapster-rbac.yaml        influxdb-cm.yaml       influxdb-service.yaml
$ kubectl create -f .
$
```

检查执行结果

检查 Deployment

```
$ kubectl get deployments -n kube-system | grep -E 'heapster|monitoring'
```

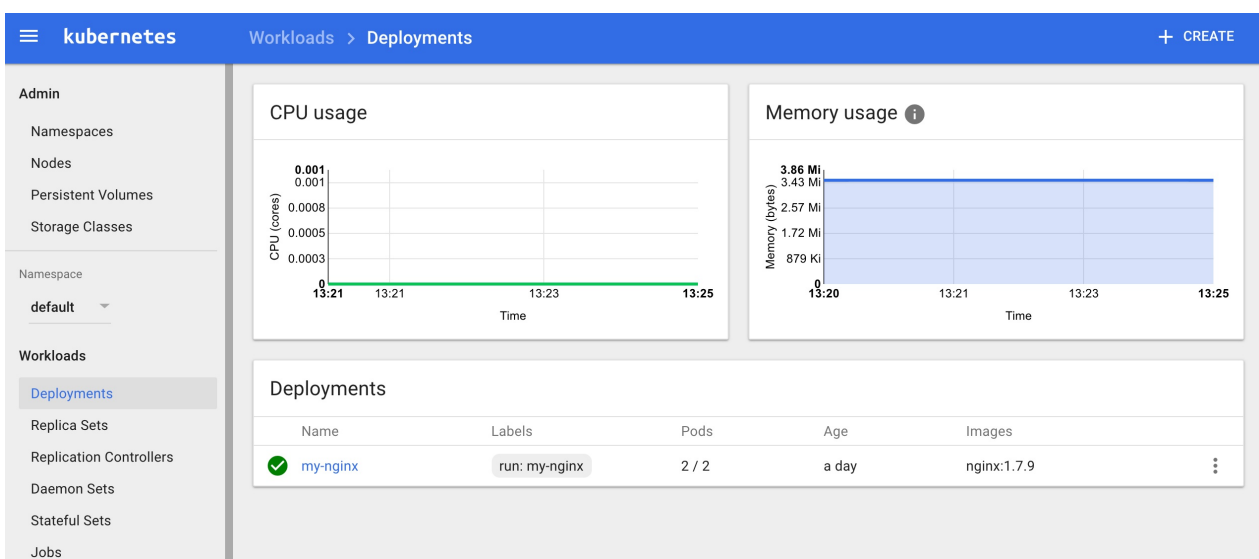
Deployment Name	Replicas	Updated Replicas	Available Replicas	Unavailable Replicas
heapster	1	1	1	0
monitoring-grafana	1	1	1	0
monitoring-influxdb	1	1	1	0

检查 Pods

```
$ kubectl get pods -n kube-system | grep -E 'heapster|monitoring'
```

Pod Name	Ready	Running	Terminated
heapster-3273315324-tmxbg	1/1	Running	0
monitoring-grafana-2255110352-94lpn	1/1	Running	0
monitoring-influxdb-884893134-3vb6n	1/1	Running	0

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



图片 - *dashboard-heapster*

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://10.64.3.7:6443
Heapster is running at https://10.64.3.7:6443/api/v1/proxy/n
namespaces/kube-system/services/heapster
KubeDNS is running at https://10.64.3.7:6443/api/v1/proxy/n
amespaces/kube-system/services/kube-dns
kubernetes-dashboard is running at https://10.64.3.7:6443/a
pi/v1/proxy/namespaces/kube-system/services/kubernetes-dashb
oard
monitoring-grafana is running at https://10.64.3.7:6443/api
/v1/proxy/namespaces/kube-system/services/monitoring-grafana
monitoring-influxdb is running at https://10.64.3.7:6443/ap
i/v1/proxy/namespaces/kube-system/services/monitoring-influx
db
$
```

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://10.64.3.7:8080/api/v1/proxy/namespaces/kube-
system/services/monitoring-grafana
```

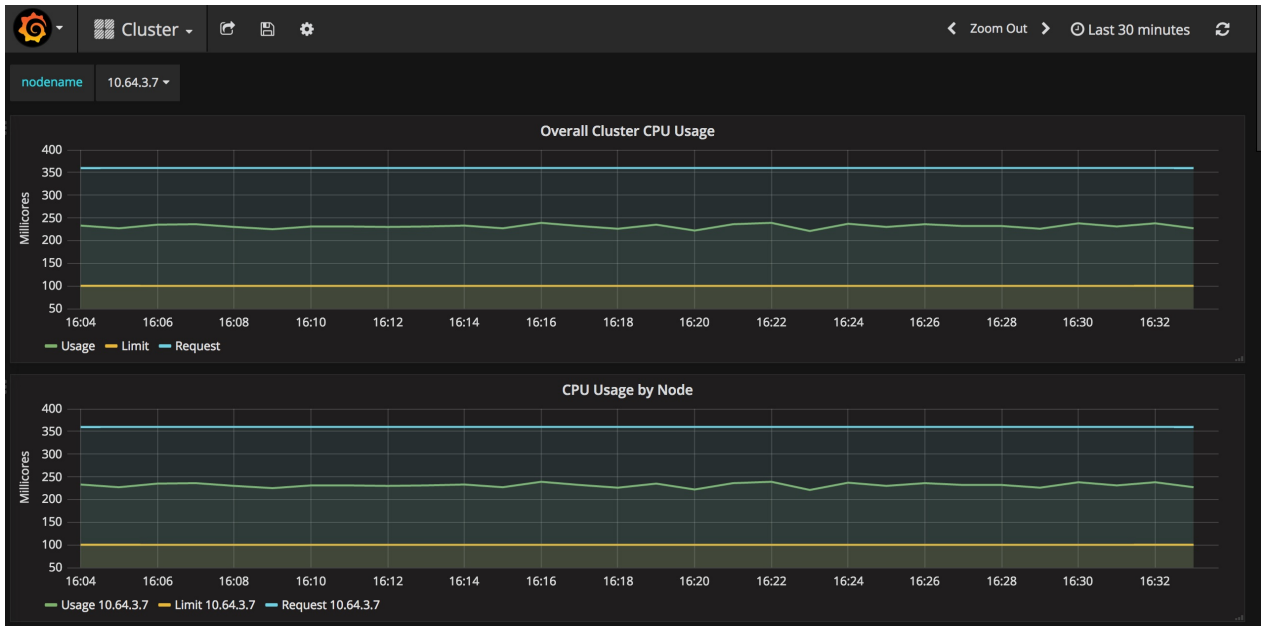
2. 通过 kubectl proxy 访问：

创建代理

```
$ kubectl proxy --address='10.64.3.7' --port=8086 --accept-
hosts='^*$'
Starting to serve on 10.64.3.7:8086
```

浏览器访问

URL : `http://10.64.3.7:8086/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana`



图片 - grafana

访问 influxdb admin UI

获取 influxdb http 8086 映射的 NodePort

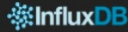
```
$ kubectl get svc -n kube-system | grep influxdb
monitoring-influxdb    10.254.255.183    <nodes>          8086:8670/
TCP,8083:8595/TCP     21m
```

通过 kube-apiserver 的非安全端口访问 influxdb 的 admin UI 界面：

`http://10.64.3.7:8080/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 8670，点击“Save”即可：

10. 部署 Heapster 插件

 Write Data Documentation Database: _internal ⌵ ⚙

Connection Settings

Host10.64.3.7

Port8670

Username

Password

☐ SSL

Save

Query:

Generate Query URL

Query Templates ⌵

zhangjun 最后更新：2017-08-11 02:50:58

tags: EFK, fluentd, elasticsearch, kibana

部署 EFK 插件

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

```
$ ls *.yaml
es-controller.yaml es-rbac.yaml es-service.yaml fluentd-es-ds.y
aml kibana-controller.yaml kibana-service.yaml fluentd-es-rbac
.yaml
```

- 新加了 `es-rbac.yaml` 和 `fluentd-es-rbac.yaml` 文件，定义了 elasticsearch 和 fluentd 使用的 Role 和 RoleBinding；

已经修改好的 yaml 文件见：[EFK](#)。

配置 es-controller.yaml

```
$ diff es-controller.yaml.orig es-controller.yaml
22a23
>     serviceAccountName: elasticsearch
24c25
<     - image: gcr.io/google_containers/elasticsearch:v2.4.1-2
---
>     - image: onlyyerich/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
23a24
>     serviceAccountName: fluentd
26c27
<     image: gcr.io/google_containers/fluentd-elasticsearch:
1.22
---
>     image: onlyyerich/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: onlyyerich/kibana:v4.6.1-1
```

给 Node 设置标签

DaemonSet `fluentd-es-v1.22` 只会调度到设置了标签

`beta.kubernetes.io/fluentd-ds-ready=true` 的 Node，需要在期望运行 `fluentd` 的 Node 上设置该标签：

```
$ kubectl get nodes
NAME           STATUS    AGE           VERSION
10.64.3.7      Ready     1d            v1.6.2

$ kubectl label nodes 10.64.3.7 beta.kubernetes.io/fluentd-ds-re
ady=true
node "10.64.3.7" labeled
```

执行定义文件

```
$ pwd
/root/kubernetes/cluster/addons/fluentd-elasticsearch
$ ls *.yaml
es-controller.yaml es-rbac.yaml es-service.yaml fluentd-es-ds.yaml
kibana-controller.yaml kibana-service.yaml fluentd-es-rbac.yaml
$ kubectl create -f .
$
```

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-kwc9w          1/1          Running    0
4m
elasticsearch-logging-v1-ws9mk          1/1          Running    0
4m
fluentd-es-v1.22-g76x0                  1/1          Running    0
4m
kibana-logging-324921636-ph7sn          1/1          Running    0
4m

$ kubectl get service -n kube-system | grep -E 'elasticsearch|kibana'
elasticsearch-logging    10.254.128.156    <none>          9200/TCP
3m
kibana-logging           10.254.88.109     <none>          5601/TCP
3m
```

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-324921636-ph7sn -n kube-system -f
ELASTICSEARCH_URL=http://elasticsearch-logging:9200
server.basePath: /api/v1/proxy/namespaces/kube-system/services/kibana-logging
{"type":"log","@timestamp":"2017-04-08T09:30:30Z","tags":["info","optimize"],"pid":7,"message":"Optimizing and caching bundles for kibana and statusPage. This may take a few minutes"}
{"type":"log","@timestamp":"2017-04-08T09:44:01Z","tags":["info","optimize"],"pid":7,"message":"Optimization of bundles for kibana and statusPage complete in 811.00 seconds"}
{"type":"log","@timestamp":"2017-04-08T09:44:02Z","tags":["status","plugin:kibana@1.0.0","info"],"pid":7,"state":"green","message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}
```

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://10.64.3.7:6443
Elasticsearch is running at https://10.64.3.7:6443/api/v1/proxy/namespaces/kube-system/services/elasticsearch-logging
Heapster is running at https://10.64.3.7:6443/api/v1/proxy/namespaces/kube-system/services/heapster
Kibana is running at https://10.64.3.7:6443/api/v1/proxy/namespaces/kube-system/services/kibana-logging
KubeDNS is running at https://10.64.3.7:6443/api/v1/proxy/namespaces/kube-system/services/kube-dns
kubernetes-dashboard is running at https://10.64.3.7:6443/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard
monitoring-grafana is running at https://10.64.3.7:6443/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana
monitoring-influxdb is running at https://10.64.3.7:6443/api/v1/proxy/namespaces/kube-system/services/monitoring-influxdb
```

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://10.64.3.7:8080/api/v1/proxy/namespaces/kube-system/services/kibana-logging
```

2. 通过 kubectl proxy 访问：

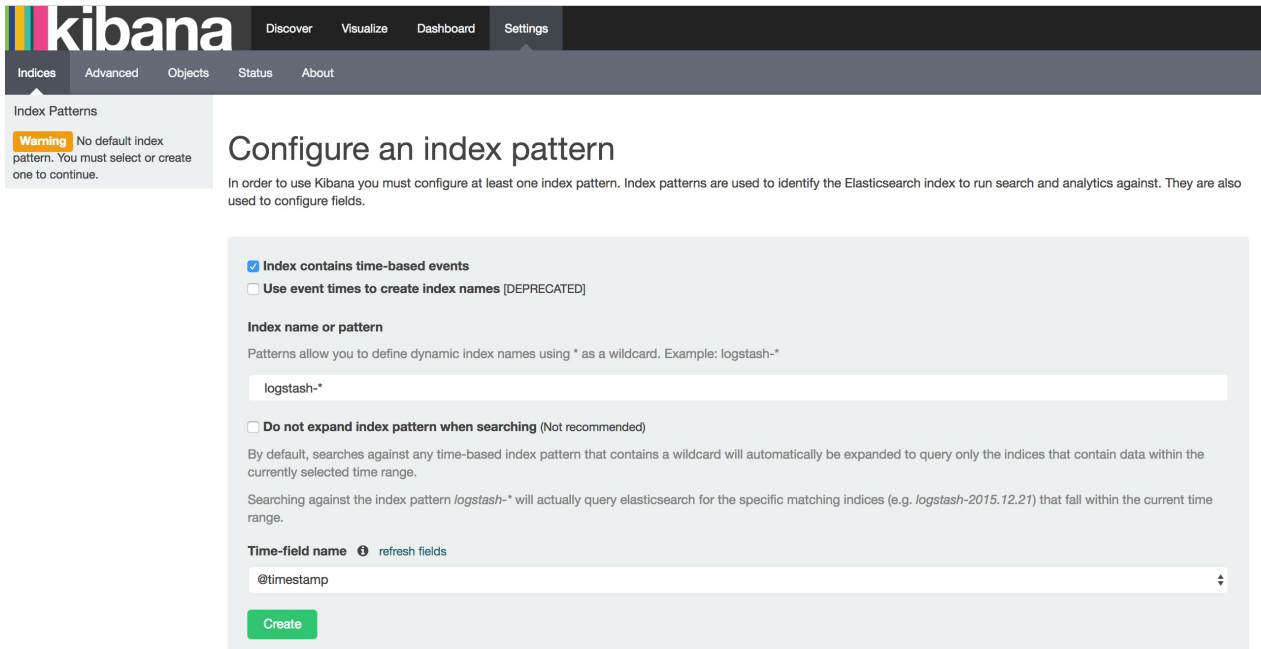
创建代理

```
$ kubectl proxy --address='10.64.3.7' --port=8086 --accept-hosts='^*$'
Starting to serve on 10.64.3.7:8086
```

浏览器访问

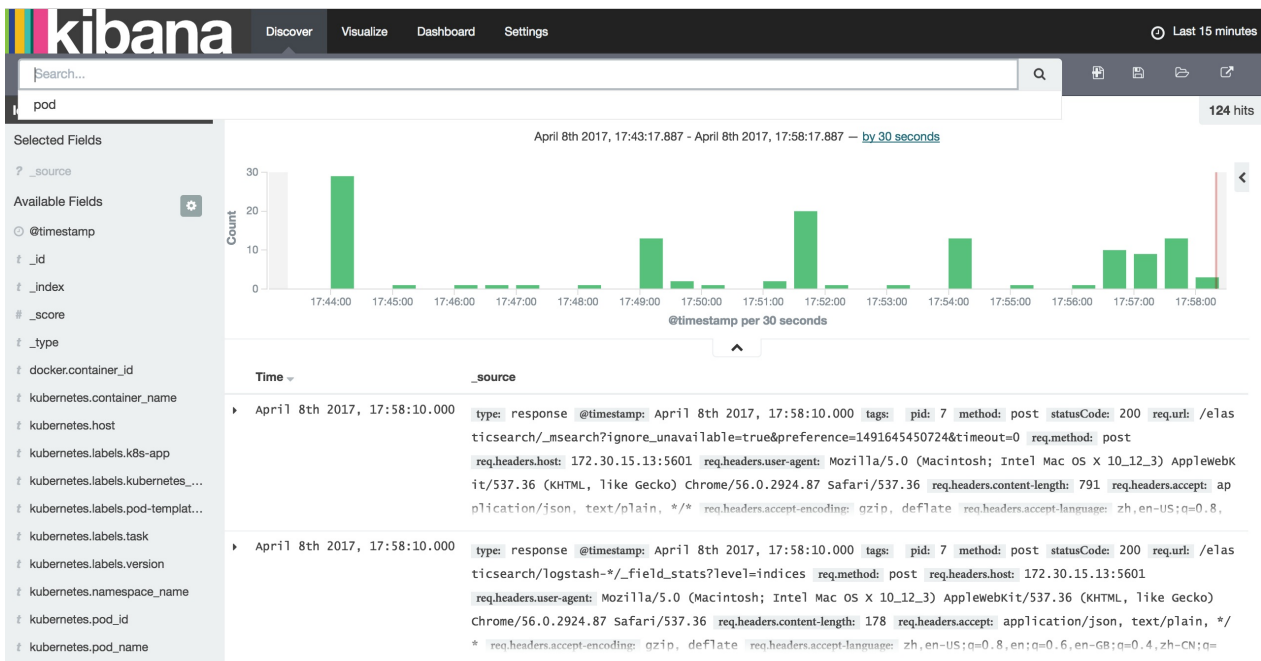
URL： `http://10.64.3.7:8086/api/v1/proxy/namespaces/kube-system/services/kibana-logging`

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`；



图片 - es-setting

创建Index后，稍等几分钟就可以在 `Discover` 菜单下看到 Elasticsearch logging 中汇聚的日志；



zhangjun

最后更新：2017-08-11 02:50:58

tags: registry, ceph

部署私有 **docker registry**

注意：本文档介绍使用 docker 官方的 registry v2 镜像部署私有仓库的步骤，你也可以部署 Harbor 私有仓库（[部署 Harbor 私有仓库](#)）。

本文档讲解部署一个 TLS 加密、HTTP Basic 认证、用 ceph rgw 做后端存储的私有 docker registry 步骤，如果使用其它类型的后端存储，则可以从“创建 docker registry”节开始；

示例两台机器 IP 如下：

- ceph rgw: 10.64.3.9
- docker registry: 10.64.3.7

部署 **ceph RGW** 节点

```
$ ceph-deploy rgw create 10.64.3.9 # rgw 默认监听7480端口
$
```

创建测试账号 **demo**

```
$ radosgw-admin user create --uid=demo --display-name="ceph rgw
demo user"
$
```

创建 **demo** 账号的子账号 **swift**

当前 registry 只支持使用 swift 协议访问 ceph rgw 存储，暂时不支持 s3 协议；

```
$ radosgw-admin subuser create --uid demo --subuser=demo:swift -
-access=full --secret=secretkey --key-type=swift
$
```

创建 **demo:swift** 子账号的 **secret key**

```
$ radosgw-admin key create --subuser=demo:swift --key-type=swift
--gen-secret
{
  "user_id": "demo",
  "display_name": "ceph rgw demo user",
  "email": "",
  "suspended": 0,
  "max_buckets": 1000,
  "auid": 0,
  "subusers": [
    {
      "id": "demo:swift",
      "permissions": "full-control"
    }
  ],
  "keys": [
    {
      "user": "demo",
      "access_key": "5Y1B1SIJ2YHKEH05U36B",
      "secret_key": "nrIvtPqUj7pUlcclYPuR3ntVzIa50DToIpe7x
FjT"
    }
  ],
  "swift_keys": [
    {
      "user": "demo:swift",
      "secret_key": "aCgVTx3Gfz1dBiFS4NfjIRmvT0sgpHDP6aa0Y
frh"
    }
  ],
  "caps": [],
  "op_mask": "read, write, delete",
  "default_placement": "",
  "placement_tags": [],
  "bucket_quota": {
    "enabled": false,
    "max_size_kb": -1,
    "max_objects": -1
  },
}
```

```
    "user_quota": {  
      "enabled": false,  
      "max_size_kb": -1,  
      "max_objects": -1  
    },  
    "temp_url_keys": []  
  }  
}
```

- `aCgVTx3Gfz1dBiFS4NfjIRmvT0sgpHDP6aa0Yfrh` 为子账号 `demo:swift` 的 secret key ;

创建 **docker registry**

创建 registry 使用的 TLS 证书

```
$ mkdir -p registry/{auth,certs}
$ cat registry-csr.json
{
  "CN": "registry",
  "hosts": [
    "127.0.0.1",
    "10.64.3.7"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "System"
    }
  ]
}
$ 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 registry-csr.json | cfssljson -bare registry
$ cp registry.pem registry-key.pem registry/certs
$
```

- 这里复用以前创建的 CA 证书和秘钥文件；
- hosts 字段指定 registry 的 NodeIP；

创建 HTTP Basic 认证文件

```
$ docker run --entrypoint htpasswd registry:2 -Bbn foo foo123 >
  auth/htpasswd
$ cat auth/htpasswd
foo:$2y$05$I60z69MdluAQ8i1Ka3x3Neb332yz1ioow2C4oroZS0E0fqPogAmZm
```


配置 registry 参数

```
$ export RGW_AUTH_URL="http://10.64.3.9:7480/auth/v1"
$ export RGW_USER="demo:swift"
$ export RGW_SECRET_KEY="aCgVTx3Gfz1dBiFS4NfjIRmvT0sgpHDP6aa0Yfrh"
$ cat > config.yml << EOF
# https://docs.docker.com/registry/configuration/#list-of-configuration-options
version: 0.1
log:
  level: info
  formatter: text
  fields:
    service: registry

storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
  swift:
    authurl: ${RGW_AUTH_URL}
    username: ${RGW_USER}
    password: ${RGW_SECRET_KEY}
    container: registry

auth:
  htpasswd:
    realm: basic-realm
    path: /auth/htpasswd

http:
  addr: 0.0.0.0:8000
  headers:
    X-Content-Type-Options: [nosniff]
  tls:
    certificate: /certs/registry.pem
    key: /certs/registry-key.pem

health:
```

```
storagedriver:  
  enabled: true  
  interval: 10s  
  threshold: 3  
EOF
```

- storage.swift 指定后端使用 swift 接口协议的存储，这里配置的是 ceph rgw 存储参数；
- auth.htpasswd 指定了 HTTP Basic 认证的 token 文件路径；
- http.tls 指定了 registry http 服务器的证书和秘钥文件路径；

创建 docker registry

```
$ docker run -d -p 8000:8000 \  
  -v $(pwd)/registry/auth:/auth \  
  -v $(pwd)/registry/certs:/certs \  
  -v $(pwd)/config.yml:/etc/docker/registry/config.yml \  
  --name registry registry:2
```

- 执行该 docker run 命令的机器 IP 为 10.64.3.7；

向 registry push image

将签署 registry 证书的 CA 证书拷贝到 `/etc/docker/certs.d/10.64.3.7:8000` 目录下

```
$ sudo mkdir -p /etc/docker/certs.d/10.64.3.7:8000  
$ sudo cp /etc/kubernetes/ssl/ca.pem /etc/docker/certs.d/10.64.3  
.7:8000/ca.crt  
$
```

登陆私有 registry

```
$ docker login 10.64.3.7:8000  
Username: foo  
Password:  
Login Succeeded
```

登陆信息被写入 `~/.docker/config.json` 文件

```
$ cat ~/.docker/config.json
{
  "auths": {
    "10.64.3.7:8000": {
      "auth": "Zm9vOmZvbzEyMw=="
    }
  }
}
```

将本地的 image 打上私有 registry 的 tag

```
$ docker tag docker.io/kubernetes/pause 10.64.3.7:8000/zhangjun3/pause
$ docker images |grep pause
docker.io/kubernetes/pause          latest
f9d5de079539      2 years ago      239.8 kB
10.64.3.7:8000/zhangjun3/pause      latest
f9d5de079539      2 years ago      239.8 kB
```

将 image push 到私有 registry

```
$ docker push 10.64.3.7:8000/zhangjun3/pause
The push refers to a repository [10.64.3.7:8000/zhangjun3/pause]
5f70bf18a086: Pushed
e16a89738269: Pushed
latest: digest: sha256:9a6b437e896acad3f5a2a8084625fdd4177b2e7124ee943af642259f2f283359 size: 916
```

查看 ceph 上是否已经有 push 的 pause 容器文件

```
$ rados lspools
rbd
.rgw.root
default.rgw.control
default.rgw.data.root
default.rgw.gc
default.rgw.log
default.rgw.users.uid
default.rgw.users.keys
default.rgw.users.swift
default.rgw.buckets.index
default.rgw.buckets.data

$ rados --pool default.rgw.buckets.data ls|grep pause
9c2d5a9d-19e6-4003-90b5-b1cbf15e890d.4310.1_files/docker/registry/v2/repositories/zhangjun3/pause/_layers/sha256/f9d5de0795395db6c50cb1ac82ebd1bd8eb3eefcebb1aa724e01239594e937b/link
9c2d5a9d-19e6-4003-90b5-b1cbf15e890d.4310.1_files/docker/registry/v2/repositories/zhangjun3/pause/_layers/sha256/f72a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845/link
9c2d5a9d-19e6-4003-90b5-b1cbf15e890d.4310.1_files/docker/registry/v2/repositories/zhangjun3/pause/_layers/sha256/a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4/link
9c2d5a9d-19e6-4003-90b5-b1cbf15e890d.4310.1_files/docker/registry/v2/repositories/zhangjun3/pause/_manifests/tags/latest/current/link
9c2d5a9d-19e6-4003-90b5-b1cbf15e890d.4310.1_files/docker/registry/v2/repositories/zhangjun3/pause/_manifests/tags/latest/index/sha256/9a6b437e896acad3f5a2a8084625fdd4177b2e7124ee943af642259f2f283359/link
9c2d5a9d-19e6-4003-90b5-b1cbf15e890d.4310.1_files/docker/registry/v2/repositories/zhangjun3/pause/_manifests/revisions/sha256/9a6b437e896acad3f5a2a8084625fdd4177b2e7124ee943af642259f2f283359/link
```

私有 **registry** 的运维操作

查询私有镜像中的 **images**

```
$ curl --user zhangjun3:xxx --cacert /etc/docker/certs.d/10.64.3.7\8000/ca.crt https://10.64.3.7:8000/v2/_catalog
{"repositories":["library/redis","zhangjun3/busybox","zhangjun3/pause","zhangjun3/pause2"]}
```

查询某个镜像的 **tags** 列表

```
$ curl --user zhangjun3:xxx --cacert /etc/docker/certs.d/10.64.3.7\8000/ca.crt https://10.64.3.7:8000/v2/zhangjun3/busybox/tags/list
{"name":"zhangjun3/busybox","tags":["latest"]}
```

获取 **image** 或 **layer** 的 **digest**

向 `v2/<repoName>/manifests/<tagName>` 发 GET 请求，从响应的头部

`Docker-Content-Digest` 获取 image digest，从响应的 body 的

`fsLayers.blobSum` 中获取 layDigests;

注意，必须包含请求头：`Accept:`

`application/vnd.docker.distribution.manifest.v2+json` :

```
$ curl -v -H "Accept: application/vnd.docker.distribution.manifest.v2+json" --user zhangjun3:xxx --cacert /etc/docker/certs.d/10.64.3.7\8000/ca.crt https://10.64.3.7:8000/v2/zhangjun3/busybox/manifests/latest
```

```
> GET /v2/zhangjun3/busybox/manifests/latest HTTP/1.1
> User-Agent: curl/7.29.0
> Host: 10.64.3.7:8000
> Accept: application/vnd.docker.distribution.manifest.v2+json
>
< HTTP/1.1 200 OK
< Content-Length: 527
< Content-Type: application/vnd.docker.distribution.manifest.v2+json
< Docker-Content-Digest: sha256:68effe31a4ae8312e47f54bec52d1fc925908009ce7e6f734e1b54a4169081c5
```

```
< Docker-Distribution-API-Version: registry/2.0
< Etag: "sha256:68effe31a4ae8312e47f54bec52d1fc925908009ce7e6f73
4e1b54a4169081c5"
< X-Content-Type-Options: nosniff
< Date: Tue, 21 Mar 2017 15:19:42 GMT
<
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.v2
+json",
  "config": {
    "mediaType": "application/vnd.docker.container.image.v1+js
on",
    "size": 1465,
    "digest": "sha256:00f017a8c2a6e1fe2ffd05c281f27d069d2a9932
3a8cd514dd35f228ba26d2ff"
  },
  "layers": [
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.
tar.gz",
      "size": 701102,
      "digest": "sha256:04176c8b224aa0eb9942af765f66dae866f43
6e75acef028fe44b8a98e045515"
    }
  ]
}
```

删除 image

向 `/v2/<name>/manifests/<reference>` 发送 DELETE 请求，reference 为上一步返回的 Docker-Content-Digest 字段内容：

```
$ curl -X DELETE --user zhangjun3:xxx --cacert /etc/docker/cert
s.d/10.64.3.7\8000/ca.crt https://10.64.3.7:8000/v2/zhangjun3/b
usybox/manifests/sha256:68effe31a4ae8312e47f54bec52d1fc925908009
ce7e6f734e1b54a4169081c5
$
```

删除 layer

向 `/v2/<name>/blobs/<digest>` 发送 DELETE 请求，其中 `digest` 是上一步返回的 `fsLayers.blobSum` 字段内容：

```
$ curl -X DELETE --user zhangjun3:xxx --cacert /etc/docker/certs.d/10.64.3.7\8000/ca.crt https://10.64.3.7:8000/v2/zhangjun3/busybox/blobs/sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
$ curl -X DELETE --cacert /etc/docker/certs.d/10.64.3.7\8000/ca.crt https://10.64.3.7:8000/v2/zhangjun3/busybox/blobs/sha256:04176c8b224aa0eb9942af765f66dae866f436e75acef028fe44b8a98e045515
$
```

zhangjun 最后更新：2017-08-11 02:50:58

部署 harbor 私有仓库

本文档介绍使用 docker-compose 部署 harbor 私有仓库的步骤，你也可以使用 docker 官方的 registry 镜像部署私有仓库([部署 Docker Registry](#))。

使用的变量

本文档用到的变量定义如下：

```
$ export NODE_IP=10.64.3.7 # 当前部署 harbor 的节点 IP
$
```

下载文件

从 docker compose [发布页面](#) 下载最新的 `docker-compose` 二进制文件

```
$ wget https://github.com/docker/compose/releases/download/1.12.0/docker-compose-Linux-x86_64
$ mv ~/docker-compose-Linux-x86_64 /root/local/bin/docker-compose
$ chmod a+x /root/local/bin/docker-compose
$ export PATH=/root/local/bin:$PATH
$
```

从 harbor [发布页面](#) 下载最新的 harbor 离线安装包

```
$ wget --continue https://github.com/vmware/harbor/releases/download/v1.1.0/harbor-offline-installer-v1.1.0.tgz
$ tar -xvzf harbor-offline-installer-v1.1.0.tgz
$ cd harbor
$
```

导入 docker images

导入离线安装包中 harbor 相关的 docker images :

```
$ docker load -i harbor.v1.1.0.tar.gz
$
```

创建 harbor nginx 服务器使用的 TLS 证书

创建 harbor 证书签名请求：

```
$ cat > harbor-csr.json <<EOF
{
  "CN": "harbor",
  "hosts": [
    "127.0.0.1",
    "$NODE_IP"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "System"
    }
  ]
}
EOF
```

- **hosts** 字段指定授权使用该证书的当前部署节点 IP，如果后续使用域名访问 harbor 则还需要添加域名；

生成 harbor 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 harbor-csr.json | cfssljson -bare harbor  
$ ls harbor*  
harbor.csr  harbor-csr.json  harbor-key.pem harbor.pem  
$ sudo mkdir -p /etc/harbor/ssl  
$ sudo mv harbor*.pem /etc/harbor/ssl  
$ rm harbor.csr  harbor-csr.json
```

修改 harbor.cfg 文件

```
$ diff harbor.cfg.orig harbor.cfg  
5c5  
< hostname = reg.mydomain.com  
---  
> hostname = 10.64.3.7  
9c9  
< ui_url_protocol = http  
---  
> ui_url_protocol = https  
24,25c24,25  
< ssl_cert = /data/cert/server.crt  
< ssl_cert_key = /data/cert/server.key  
---  
> ssl_cert = /etc/harbor/ssl/harbor.pem  
> ssl_cert_key = /etc/harbor/ssl/harbor-key.pem
```

加载和启动 harbor 镜像

```
$ ./install.sh  
[Step 0]: checking installation environment ...  
  
Note: docker version: 17.04.0  
  
Note: docker-compose version: 1.12.0
```

[Step 1]: loading Harbor images ...

Loaded image: vmware/harbor-adminserver:v1.1.0

Loaded image: vmware/harbor-ui:v1.1.0

Loaded image: vmware/harbor-log:v1.1.0

Loaded image: vmware/harbor-jobservice:v1.1.0

Loaded image: vmware/registry:photon-2.6.0

Loaded image: vmware/harbor-notary-db:mariadb-10.1.10

Loaded image: vmware/harbor-db:v1.1.0

Loaded image: vmware/nginx:1.11.5-patched

Loaded image: photon:1.0

Loaded image: vmware/notary-photon:server-0.5.0

Loaded image: vmware/notary-photon:signer-0.5.0

[Step 2]: preparing environment ...

Generated and saved secret to file: /data/secretkey

Generated configuration file: ./common/config/nginx/nginx.conf

Generated configuration file: ./common/config/adminserver/env

Generated configuration file: ./common/config/ui/env

Generated configuration file: ./common/config/registry/config.yml

Generated configuration file: ./common/config/db/env

Generated configuration file: ./common/config/jobservice/env

Generated configuration file: ./common/config/jobservice/app.conf

Generated configuration file: ./common/config/ui/app.conf

Generated certificate, key file: ./common/config/ui/private_key.pem, cert file: ./common/config/registry/root.crt

The configuration files are ready, please use docker-compose to start the service.

[Step 3]: checking existing instance of Harbor ...

[Step 4]: starting Harbor ...

Creating network "harbor_harbor" with the default driver

Creating harbor-log

Creating registry

Creating harbor-adminserver

Creating harbor-db

Creating harbor-ui

```
Creating harbor-jobservice
```

```
Creating nginx
```

```
✓ ----Harbor has been installed and started successfully.----
```

```
Now you should be able to visit the admin portal at https://10.64.3.7.
```

```
For more details, please visit https://github.com/vmware/harbor .
```

访问管理界面

浏览器访问 `https://${NODE_IP}`，示例的是 `https://10.64.3.7`

用账号 `admin` 和 `harbor.cfg` 配置文件中的默认密码 `Harbor12345` 登陆系统：



图片 - harbor

harbor 运行时产生的文件、目录

```
$ # 日志目录
$ ls /var/log/harbor/2017-04-19/
adminserver.log  jobservice.log  mysql.log  proxy.log  registry.
log  ui.log
$ # 数据目录，包括数据库、镜像仓库
$ ls /data/
ca_download  config  database  job_logs  registry  secretkey
```

docker 客户端登陆

将签署 harbor 证书的 CA 证书拷贝到 `/etc/docker/certs.d/10.64.3.7` 目录下

```
$ sudo mkdir -p /etc/docker/certs.d/10.64.3.7
$ sudo cp /etc/kubernetes/ssl/ca.pem /etc/docker/certs.d/10.64.3.7/ca.crt
$
```

登陆 harbor

```
$ docker login 10.64.3.7
Username: admin
Password:
```

认证信息自动保存到 `~/.docker/config.json` 文件。

其它操作

下列操作的工作目录均为 解压离线安装文件后 生成的 harbor 目录。

```
$ # 停止 harbor
$ docker-compose down -v
$ # 修改配置
$ vim harbor.cfg
$ # 更修改的配置更新到 docker-compose.yml 文件
[root@tjwq01-sys-bs003007 harbor]# ./prepare
```

```
Clearing the configuration file: ./common/config/ui/app.conf
Clearing the configuration file: ./common/config/ui/env
Clearing the configuration file: ./common/config/ui/private_key.
pem
Clearing the configuration file: ./common/config/db/env
Clearing the configuration file: ./common/config/registry/root.c
rt
Clearing the configuration file: ./common/config/registry/config
.yml
Clearing the configuration file: ./common/config/jobservice/app.
conf
Clearing the configuration file: ./common/config/jobservice/env
Clearing the configuration file: ./common/config/nginx/cert/admi
n.pem
Clearing the configuration file: ./common/config/nginx/cert/admi
n-key.pem
Clearing the configuration file: ./common/config/nginx/nginx.con
f
Clearing the configuration file: ./common/config/adminserver/env
loaded secret from file: /data/secretkey
Generated configuration file: ./common/config/nginx/nginx.conf
Generated configuration file: ./common/config/adminserver/env
Generated configuration file: ./common/config/ui/env
Generated configuration file: ./common/config/registry/config.ym
l
Generated configuration file: ./common/config/db/env
Generated configuration file: ./common/config/jobservice/env
Generated configuration file: ./common/config/jobservice/app.con
f
Generated configuration file: ./common/config/ui/app.conf
Generated certificate, key file: ./common/config/ui/private_key.
pem, cert file: ./common/config/registry/root.crt
The configuration files are ready, please use docker-compose to
start the service.
$ # 启动 harbor
[root@tjqw01-sys-bs003007 harbor]# docker-compose up -d
```

zhangjun

最后更新：2017-08-11 02:50:58

tags: clean

清理集群

清理 **Node** 节点

停相关进程：

```
$ sudo systemctl stop kubelet kube-proxy flanneld docker
$
```

清理文件：

```
$ # umount kubelet 挂载的目录
$ mount | grep '/var/lib/kubelet' | awk '{print $3}' | xargs sudo u
mount
$ # 删除 kubelet 工作目录
$ sudo rm -rf /var/lib/kubelet
$ # 删除 docker 工作目录
$ sudo rm -rf /var/lib/docker
$ # 删除 flanneld 写入的网络配置文件
$ sudo rm -rf /var/run/flannel/
$ # 删除 docker 的一些运行文件
$ sudo rm -rf /var/run/docker/
$ # 删除 systemd unit 文件
$ sudo rm -rf /etc/systemd/system/{kubelet,docker,flanneld}.serv
ice
$ # 删除程序文件
$ sudo rm -rf /root/local/bin/{kubelet,docker,flanneld}
$ # 删除证书文件
$ sudo rm -rf /etc/flanneld/ssl /etc/kubernetes/ssl
$
```

清理 kube-proxy 和 docker 创建的 iptables：

```
$ sudo iptables -F && sudo iptables -X && sudo iptables -F -t nat && sudo iptables -X -t nat
$
```

删除 flanneld 和 docker 创建的网桥：

```
$ ip link del flannel.1
$ ip link del docker0
$
```

清理 Master 节点

停相关进程：

```
$ sudo systemctl stop kube-apiserver kube-controller-manager kube-scheduler
$
```

清理文件：

```
$ # 删除 kube-apiserver 工作目录
$ sudo rm -rf /var/run/kubernetes
$ # 删除 systemd unit 文件
$ sudo rm -rf /etc/systemd/system/{kube-apiserver,kube-controller-manager,kube-scheduler}.service
$ # 删除程序文件
$ sudo rm -rf /root/local/bin/{kube-apiserver,kube-controller-manager,kube-scheduler}
$ # 删除证书文件
$ sudo rm -rf /etc/flanneld/ssl /etc/kubernetes/ssl
$
```

清理 etcd 集群

停相关进程：


```
$ sudo systemctl stop etcd
$
```

清理文件：

```
$ # 删除 etcd 的工作目录和数据目录
$ sudo rm -rf /var/lib/etcd
$ # 删除 systemd unit 文件
$ sudo rm -rf /etc/systemd/system/etcd.service
$ # 删除程序文件
$ sudo rm -rf /root/local/bin/etcd
$ # 删除 TLS 证书文件
$ sudo rm -rf /etc/etcd/ssl/*
$
```

zhangjun 最后更新：2017-08-11 02:50:58

Tags

zhangjun

最后更新： 2017-08-11 02:50:58