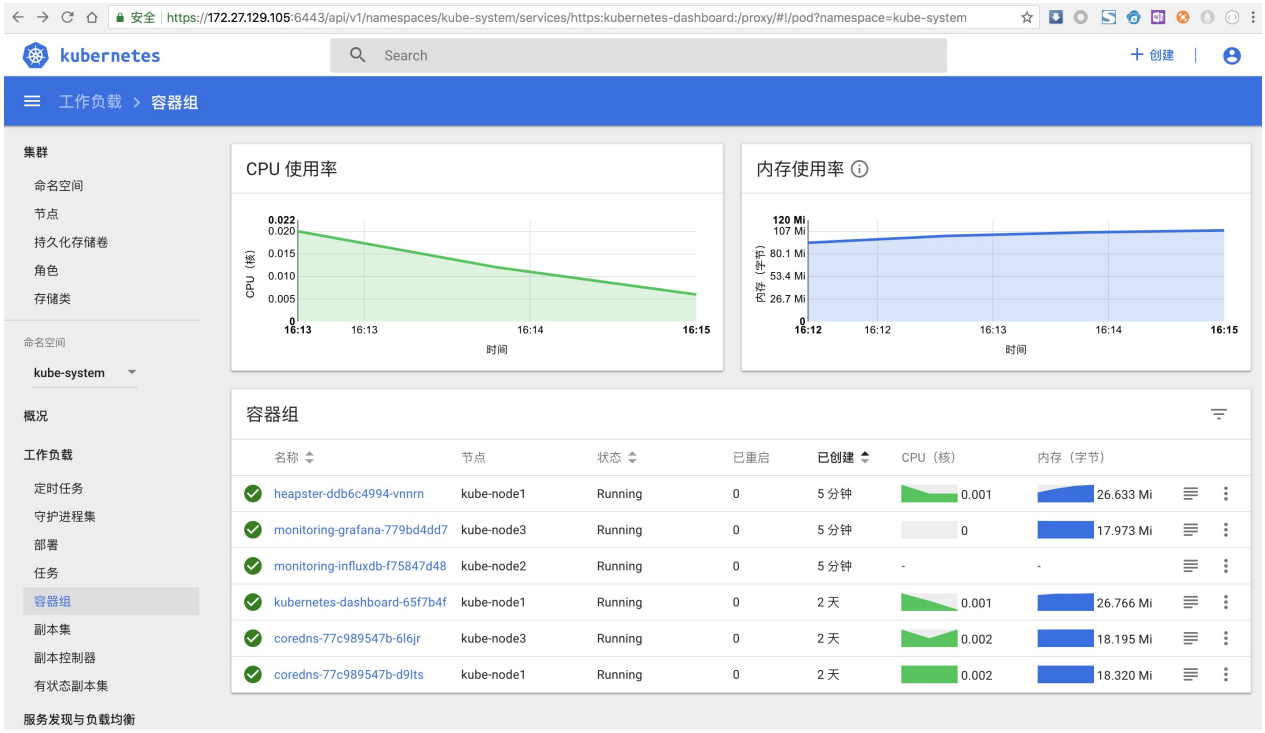
目錄

和我一步步部署 kubernetes 集群

标签集合

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 v1.10.4` 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。

本系列系文档适用于 `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 `x509` 证书双向认证、 `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！

历史版本

- v1.6.2：已停止更新；
- v1.8.x：继续更新；

步骤列表

在线阅读

- 建议：GitBook
- Github

电子书

- pdf 格式 下载
- epub 格式 下载

打赏

如果你觉得这份文档对你有帮助，请微信扫描下方的二维码进行捐赠，加油后的 opsnull 将会和你分享更多的原创教程，谢谢！

版权

zhangjun 最后更新： 2018-10-18 04:04:02

tags: version

00.组件版本和配置策略

组件版本

- Kubernetes 1.10.4
- Docker 18.03.1-ce
- Etcd 3.3.7
- Flanneld 0.10.0
- 插件：
  - Coredns
  - Dashboard
  - Heapster (influxdb、grafana)
  - Metrics-Server
  - EFK (elasticsearch、fluentd、kibana)
- 镜像仓库：
  - docker registry
  - harbor

主要配置策略

kube-apiserver：

- 使用 keepalived 和 haproxy 实现 3 节点高可用；
- 关闭非安全端口 8080 和匿名访问；
- 在安全端口 6443 接收 https 请求；
- 严格的认证和授权策略 (x509、token、RBAC)；
- 开启 bootstrap token 认证，支持 kubelet TLS bootstrapping；
- 使用 https 访问 kubelet、etcd，加密通信；

kube-controller-manager：

- 3 节点高可用；
- 关闭非安全端口，在安全端口 10252 接收 https 请求；
- 使用 kubeconfig 访问 apiserver 的安全端口；
- 自动 approve kubelet 证书签名请求 (CSR)，证书过期后自动轮转；

- 各 controller 使用自己的 ServiceAccount 访问 apiserver；

kube-scheduler：

- 3 节点高可用；
- 使用 kubeconfig 访问 apiserver 的安全端口；

kubelet：

- 使用 kubeadm 动态创建 bootstrap token，而不是在 apiserver 中静态配置；
- 使用 TLS bootstrap 机制自动生成 client 和 server 证书，过期后自动轮转；
- 在 KubeletConfiguration 类型的 JSON 文件配置主要参数；
- 关闭只读端口，在安全端口 10250 接收 https 请求，对请求进行认证和授权，拒绝匿名访问和非授权访问；
- 使用 kubeconfig 访问 apiserver 的安全端口；

kube-proxy：

- 使用 kubeconfig 访问 apiserver 的安全端口；
- 在 KubeProxyConfiguration 类型的 JSON 文件配置主要参数；
- 使用 ipvs 代理模式；

集群插件：

- DNS：使用功能、性能更好的 coredns；
- Dashboard：支持登录认证；
- Metric：heapster、metrics-server，使用 https 访问 kubelet 安全端口；
- Log：Elasticsearch、Fluend、Kibana；
- Registry 镜像库：docker-registry、harbor；

zhangjun 最后更新： 2018-10-18 04:04:02

tags: environment

01.系统初始化和全局变量

集群机器

- kube-node1：172.27.129.105
- kube-node2：172.27.129.111
- kube-node3：172.27.129.112

可以使用 vagrant 和 Vagrantfile 创建三台虚机：

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

本文档中的 etcd 集群、master 节点、worker 节点均使用这三台机器。

主机名

设置永久主机名称，然后重新登录:

```
$ sudo hostnamectl set-hostname kube-node1 # 将 kube-node1 替换为当前主机名
```

- 设置的主机名保存在 `/etc/hostname` 文件中；

修改每台机器的 `/etc/hosts` 文件，添加主机名和 IP 的对应关系：

```
$ grep kube-node /etc/hosts
172.27.129.105 kube-node1    kube-node1
172.27.129.111 kube-node2    kube-node2
172.27.129.112 kube-node3    kube-node3
```

添加 **k8s** 和 **docker** 账户

在每台机器上添加 k8s 账户，可以无密码 sudo：

```
$ sudo useradd -m k8s
$ sudo sh -c 'echo 123456 | passwd k8s --stdin' # 为 k8s 账户设置密码
$ sudo visudo
$ sudo grep '%wheel.*NOPASSWD: ALL' /etc/sudoers
%wheel    ALL=(ALL)    NOPASSWD: ALL
$ sudo gpasswd -a k8s wheel
```

在每台机器上添加 docker 账户，将 k8s 账户添加到 docker 组中，同时配置 dockerd 参数：

```
$ sudo useradd -m docker
$ sudo gpasswd -a k8s docker
$ sudo mkdir -p  /etc/docker/
$ cat /etc/docker/daemon.json
{
    "registry-mirrors": ["https://hub-mirror.c.163.com", "https://doc
ker.mirrors.ustc.edu.cn"],
    "max-concurrent-downloads": 20
}
```

无密码 ssh 登录其它节点

如果没有特殊指明，本文档的所有操作均在 **kube-node1** 节点上执行，然后远程分发文件和执行命令。

设置 kube-node1 可以无密码登录所有节点的 k8s 和 root 账户：

```
[k8s@kube-node1 k8s]$ ssh-keygen -t rsa
[k8s@kube-node1 k8s]$ ssh-copy-id root@kube-node1
[k8s@kube-node1 k8s]$ ssh-copy-id root@kube-node2
[k8s@kube-node1 k8s]$ ssh-copy-id root@kube-node3

[k8s@kube-node1 k8s]$ ssh-copy-id k8s@kube-node1
[k8s@kube-node1 k8s]$ ssh-copy-id k8s@kube-node2
[k8s@kube-node1 k8s]$ ssh-copy-id k8s@kube-node3
```

将可执行文件路径 /opt/k8s/bin 添加到 PATH 变量中

在每台机器上添加环境变量：

```
$ sudo sh -c "echo 'PATH=/opt/k8s/bin:$PATH:$HOME/bin:$JAVA_HOME/bin'
 >>/root/.bashrc"
$ echo 'PATH=/opt/k8s/bin:$PATH:$HOME/bin:$JAVA_HOME/bin' >>~/.bashrc
```

安装依赖包

在每台机器上安装依赖包：

CentOS:

```
$ sudo yum install -y epel-release
$ sudo yum install -y conntrack ipvsadm ipset jq sysstat curl iptable
s libseccomp
```

Ubuntu:

```
$ sudo apt-get install -y conntrack ipvsadm ipset jq sysstat curl ipt
ables libseccomp
```

- ipvs 依赖 ipset；

关闭防火墙

在每台机器上关闭防火墙：

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

关闭 **swap** 分区

如果开启了 swap 分区，kubelet 会启动失败(可以通过将参数 --fail-swap-on 设置为 false 来忽略 swap on)，故需要在每台机器上关闭 swap 分区：

```
$ sudo swapoff -a
```

为了防止开机自动挂载 swap 分区，可以注释 `/etc/fstab` 中相应的条目：

```
$ sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

关闭 SELinux

关闭 SELinux，否则后续 K8S 挂载目录时可能报错 `Permission denied` ：

```
$ sudo setenforce 0
$ grep SELINUX /etc/selinux/config
SELINUX=disabled
```

- 修改配置文件，永久生效；

关闭 dnsmasq (可选)

linux 系统开启了 dnsmasq 后(如 GUI 环境)，将系统 DNS Server 设置为 127.0.0.1，这会导致 docker 容器无法解析域名，需要关闭它：

```
$ sudo service dnsmasq stop
$ sudo systemctl disable dnsmasq
```

加载内核模块

```
$ sudo modprobe br_netfilter
$ sudo modprobe ip_vs
```

设置系统参数

```
$ cat > kubernetes.conf <<EOF
net.bridge.bridge-nf-call-iptables=1
net.bridge.bridge-nf-call-ip6tables=1
net.ipv4.ip_forward=1
net.ipv4.tcp_tw_recycle=0
vm.swappiness=0
vm.overcommit_memory=1
vm.panic_on_oom=0
fs.inotify.max_user_watches=89100
fs.file-max=52706963
fs.nr_open=52706963
net.ipv6.conf.all.disable_ipv6=1
net.netfilter.nf_conntrack_max=2310720
EOF
$ sudo cp kubernetes.conf  /etc/sysctl.d/kubernetes.conf
$ sudo sysctl -p /etc/sysctl.d/kubernetes.conf
$ sudo mount -t cgroup -o cpu,cpuacct none /sys/fs/cgroup/cpu,cpuacct
```

- tcp_tw_recycle 和 Kubernetes 的 NAT 冲突，必须关闭 ，否则会导致服务不通；
- 关闭不使用的 IPV6 协议栈，防止触发 docker BUG；

设置系统时区

```
$ # 调整系统 TimeZone
$ sudo timedatectl set-timezone Asia/Shanghai

$ # 将当前的 UTC 时间写入硬件时钟
$ sudo timedatectl set-local-rtc 0

$ # 重启依赖于系统时间的服务
$ sudo systemctl restart rsyslog
$ sudo systemctl restart crond
```

更新系统时间

```
$ sudo ntpdate cn.pool.ntp.org
```

创建目录

在每台机器上创建目录：

```
$ sudo mkdir -p /opt/k8s/bin
$ sudo chown -R k8s /opt/k8s

$ sudo sudo mkdir -p /etc/kubernetes/cert
$ sudo chown -R k8s /etc/kubernetes

$ sudo mkdir -p /etc/etcd/cert
$ sudo chown -R k8s /etc/etcd/cert

$ sudo mkdir -p /var/lib/etcd && chown -R k8s /etc/etcd/cert
```

检查系统内核和模块是否适合运行 docker (仅适用于 linux 系统)

```
$ curl https://raw.githubusercontent.com/docker/docker/master/contrib
/check-config.sh > check-config.sh
$ bash ./check-config.sh
```

修改和分发集群环境变量定义脚本

后续的部署步骤将使用 environment.sh 中定义的全局环境变量，请根据自己的机器、网络情况修改。

然后，把全局变量定义脚本拷贝到所有节点的 `/opt/k8s/bin` 目录：

```
source environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp environment.sh k8s@${node_ip}:/opt/k8s/bin/
    ssh k8s@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

参考

1. 系统内核相关参数参

考：https://docs.openshift.com/enterprise/3.2/admin_guide/overcommit.html

zhangjun 最后更新： 2018-10-18 04:04:02

tags: TLS, CA, x509

02.创建 CA 证书和秘钥

为确保安全，`kubernetes` 系统各组件需要使用 `x509` 证书对通信进行加密和认证。

CA (Certificate Authority) 是自签名的根证书，用来签名后续创建的其它证书。

本文档使用 `CloudFlare` 的 PKI 工具集 cfssl 创建所有证书。

安装 cfssl 工具集

```
sudo mkdir -p /opt/k8s/cert && sudo chown -R k8s /opt/k8s && cd /opt/
k8s
wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
mv cfssl_linux-amd64 /opt/k8s/bin/cfssl

wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
mv cfssljson_linux-amd64 /opt/k8s/bin/cfssljson

wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64
mv cfssl-certinfo_linux-amd64 /opt/k8s/bin/cfssl-certinfo

chmod +x /opt/k8s/bin/*
export PATH=/opt/k8s/bin:$PATH
```

创建根证书 (CA)

CA 证书是集群所有节点共享的，只需要创建一个 **CA** 证书，后续创建的所有证书都由它签名。

创建配置文件

CA 配置文件用于配置根证书的使用场景 (profile) 和具体参数 (usage，过期时间、服务端认证、客户端认证、加密等)，后续在签名其它证书时需要指定特定场景。

```
cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "87600h"
    },
    "profiles": {
      "kubernetes": {
        "usages": [
            "signing",
            "key encipherment",
            "server auth",
            "client auth"
        ],
        "expiry": "87600h"
      }
    }
  }
}
EOF
```

- `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 可以用该该证书对 server 提供的证书进行验证；
- `client auth`：表示 server 可以用该该证书对 client 提供的证书进行验证；

创建证书签名请求文件

```
cat > ca-csr.json <<EOF
{
  "CN": "kubernetes",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "4Paradigm"
    }
  ]
}
EOF
```

- 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)**；
- kube-apiserver 将提取的 User、Group 作为 `RBAC` 授权的用户标识；

生成 CA 证书和私钥

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

分发证书文件

将生成的 CA 证书、秘钥文件、配置文件拷贝到所有节点的 `/etc/kubernetes/cert` 目录下：

```
source /opt/k8s/bin/environment.sh # 导入 NODE_IPS 环境变量
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /etc/kubernetes/cert && chown -R k8
s /etc/kubernetes"
    scp ca*.pem ca-config.json k8s@${node_ip}:/etc/kubernetes/cert
  done
```

- k8s 账户需要有读写 /etc/kubernetes 目录及其子目录文件的权限；

参考

1. 各种 CA 证书类型：https://github.com/kubernetes-incubator/apiserver-builder/blob/master/docs/concepts/auth.md

zhangjun 最后更新： 2018-10-18 04:04:02

tags: kubectl

03.部署 kubectl 命令行工具

kubectl 是 kubernetes 集群的命令行管理工具，本文档介绍安装和配置它的步骤。

kubectl 默认从 `~/.kube/config` 文件读取 kube-apiserver 地址、证书、用户名等信息，如果没有配置，执行 kubectl 命令时可能会出错：

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

本文档只需要部署一次，生成的 kubeconfig 文件与机器无关。

下载和分发 kubectl 二进制文件

下载和解压：

```
wget https://dl.k8s.io/v1.10.4/kubernetes-client-linux-amd64.tar.gz
tar -xzvf kubernetes-client-linux-amd64.tar.gz
```

分发到所有使用 kubectl 的节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kubernetes/client/bin/kubectl k8s@${node_ip}:/opt/k8s/bin/
    ssh k8s@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

创建 admin 证书和私钥

kubectl 与 apiserver https 安全端口通信，apiserver 对提供的证书进行认证和授权。

kubectl 作为集群的管理工具，需要被授予最高权限。这里创建具有最高权限的 admin 证书。

创建证书签名请求：

```
cat > admin-csr.json <<EOF
{
  "CN": "admin",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "system:masters",
      "OU": "4Paradigm"
    }
  ]
}
EOF
```

- O 为 `system:masters`，kube-apiserver 收到该证书后将请求的 Group 设置为 system:masters；
- 预定义的 ClusterRoleBinding `cluster-admin` 将 Group `system:masters` 与 Role `cluster-admin` 绑定，该 Role 授予所有 **API**的权限；
- 该证书只会被 kubectl 当做 client 证书使用，所以 hosts 字段为空；

生成证书和私钥：

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

创建 **kubeconfig** 文件

kubeconfig 为 kubectl 的配置文件，包含访问 apiserver 的所有信息，如 apiserver 地址、CA 证书和自身使用的证书；

```
source /opt/k8s/bin/environment.sh
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/cert/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kubectl.kubeconfig

# 设置客户端认证参数
kubectl config set-credentials admin \
  --client-certificate=admin.pem \
  --client-key=admin-key.pem \
  --embed-certs=true \
  --kubeconfig=kubectl.kubeconfig

# 设置上下文参数
kubectl config set-context kubernetes \
  --cluster=kubernetes \
  --user=admin \
  --kubeconfig=kubectl.kubeconfig

# 设置默认上下文
kubectl config use-context kubernetes --kubeconfig=kubectl.kubeconfig
```

- `--certificate-authority`：验证 kube-apiserver 证书的根证书；
- `--client-certificate`、`--client-key`：刚生成的 `admin` 证书和私钥，连接 kube-apiserver 时使用；
- `--embed-certs=true`：将 ca.pem 和 admin.pem 证书内容嵌入到生成的 kubectl.kubeconfig 文件中(不加时，写入的是证书文件路径)；

分发 kubeconfig 文件

分发到所有使用 `kubectl` 命令的节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh k8s@${node_ip} "mkdir -p ~/.kube"
    scp kubectl.kubeconfig k8s@${node_ip}:~/.kube/config
    ssh root@${node_ip} "mkdir -p ~/.kube"
    scp kubectl.kubeconfig root@${node_ip}:~/.kube/config
  done
```

- 保存到用户的 `~/.kube/config` 文件；

zhangjun 最后更新： 2018-10-18 04:04:02

tags: etcd

04.部署 etcd 集群

etcd 是基于 Raft 的分布式 key-value 存储系统，由 CoreOS 开发，常用于服务发现、共享配置以及并发控制（如 leader 选举、分布式锁等）。kubernetes 使用 etcd 存储所有运行数据。

本文档介绍部署一个三节点高可用 etcd 集群的步骤：

- 下载和分发 etcd 二进制文件；
- 创建 etcd 集群各节点的 x509 证书，用于加密客户端(如 etcdctl) 与 etcd 集群、etcd 集群之间的数据流；
- 创建 etcd 的 systemd unit 文件，配置服务参数；
- 检查集群工作状态；

etcd 集群各节点的名称和 IP 如下：

- kube-node1：172.27.129.105
- kube-node2：172.27.129.111
- kube-node3：172.27.129.112

下载和分发 etcd 二进制文件

到 https://github.com/coreos/etcd/releases 页面下载最新版本的发布包：

```
wget https://github.com/coreos/etcd/releases/download/v3.3.7/etcd-v3.3.7-linux-amd64.tar.gz
tar -xvf etcd-v3.3.7-linux-amd64.tar.gz
```

分发二进制文件到集群所有节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp etcd-v3.3.7-linux-amd64/etcd* k8s@${node_ip}:/opt/k8s/bin
    ssh k8s@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

创建 etcd 证书和私钥

创建证书签名请求：

```
cat > etcd-csr.json <<EOF
{
  "CN": "etcd",
  "hosts": [
    "127.0.0.1",
    "172.27.129.105",
    "172.27.129.111",
    "172.27.129.112"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "4Paradigm"
    }
  ]
}
EOF
```

- hosts 字段指定授权使用该证书的 etcd 节点 IP 或域名列表，这里将 etcd 集群的三个节点 IP 都列在其中；

生成证书和私钥：

```
cfssl gencert -ca=/etc/kubernetes/cert/ca.pem \
    -ca-key=/etc/kubernetes/cert/ca-key.pem \
    -config=/etc/kubernetes/cert/ca-config.json \
    -profile=kubernetes etcd-csr.json | cfssljson -bare etcd
ls etcd*
```

分发生成的证书和私钥到各 etcd 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /etc/etcd/cert && chown -R k8s /etc
/etcd/cert"
    scp etcd*.pem k8s@${node_ip}:/etc/etcd/cert/
  done
```

创建 etcd 的 systemd unit 模板文件

```
source /opt/k8s/bin/environment.sh
cat > etcd.service.template <<EOF
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target
Documentation=https://github.com/coreos

[Service]
User=k8s
Type=notify
WorkingDirectory=/var/lib/etcd/
ExecStart=/opt/k8s/bin/etcd \\
  --data-dir=/var/lib/etcd \\
  --name=##NODE_NAME## \\
  --cert-file=/etc/etcd/cert/etcd.pem \\
  --key-file=/etc/etcd/cert/etcd-key.pem \\
  --trusted-ca-file=/etc/kubernetes/cert/ca.pem \\
  --peer-cert-file=/etc/etcd/cert/etcd.pem \\
  --peer-key-file=/etc/etcd/cert/etcd-key.pem \\
  --peer-trusted-ca-file=/etc/kubernetes/cert/ca.pem \\
  --peer-client-cert-auth \\
  --client-cert-auth \\
  --listen-peer-urls=https://##NODE_IP##:2380 \\
  --initial-advertise-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
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

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

- `User`：指定以 k8s 账户运行；
- `WorkingDirectory`、`--data-dir`：指定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；
- `--name`：指定节点名称，当 `--initial-cluster-state` 值为 `new` 时，`--`

`name` 的参数值必须位于 `--initial-cluster` 列表中；

- `--cert-file`、`--key-file`：etcd server 与 client 通信时使用的证书和私钥；
- `--trusted-ca-file`：签名 client 证书的 CA 证书，用于验证 client 证书；
- `--peer-cert-file`、`--peer-key-file`：etcd 与 peer 通信使用的证书和私钥；
- `--peer-trusted-ca-file`：签名 peer 证书的 CA 证书，用于验证 peer 证书；

为各节点创建和分发 etcd systemd unit 文件

替换模板文件中的变量，为各节点创建 systemd unit 文件：

```
source /opt/k8s/bin/environment.sh
for (( i=0; i < 3; i++ ))
  do
    sed -e "s/##NODE_NAME##/${NODE_NAMES[i]}/" -e "s/##NODE_IP##/${NO
DE_IPS[i]}/" etcd.service.template > etcd-${NODE_IPS[i]}.service
  done
ls *.service
```

- NODE_NAMES 和 NODE_IPS 为相同长度的 bash 数组，分别为节点名称和对应的 IP；

分发生成的 systemd unit 文件：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /var/lib/etcd && chown -R k8s /var/
lib/etcd"
    scp etcd-${node_ip}.service root@${node_ip}:/etc/systemd/system/e
tcd.service
  done
```

- 必须先创建 etcd 数据目录和工作目录;
- 文件重命名为 etcd.service;

完整 unit 文件见：etcd.service

启动 etcd 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable
etcd && systemctl restart etcd &"
  done
```

- etcd 进程首次启动时会等待其它节点的 etcd 加入集群，命令 `systemctl start etcd` 会卡住一段时间，为正常现象。

检查启动结果

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh k8s@${node_ip} "systemctl status etcd|grep Active"
  done
```

确保状态为 `active (running)`，否则查看日志，确认原因：

```
$ journalctl -u etcd
```

验证服务状态

部署完 etcd 集群后，在任一 etc 节点上执行如下命令：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ETCDCTL_API=3 /opt/k8s/bin/etcdctl \
    --endpoints=https://${node_ip}:2379 \
    --cacert=/etc/kubernetes/cert/ca.pem \
    --cert=/etc/etcd/cert/etcd.pem \
    --key=/etc/etcd/cert/etcd-key.pem endpoint health
  done
```

预期输出：

```
https://172.27.129.105:2379 is healthy: successfully committed propos
al: took = 2.192932ms
https://172.27.129.111:2379 is healthy: successfully committed propos
al: took = 3.546896ms
https://172.27.129.112:2379 is healthy: successfully committed propos
al: took = 3.013667ms
```

输出均为 `healthy` 时表示集群服务正常。

zhangjun 最后更新： 2018-10-18 04:04:02

tags: flanneld

05.部署 flannel 网络

kubernetes 要求集群内各节点(包括 master 节点)能通过 Pod 网段互联互通。flannel 使用 vxlan 技术为各节点创建一个可以互通的 Pod 网络，使用的端口为 UDP 8472，需要开放该端口（如公有云 AWS 等）。

flannel 第一次启动时，从 etcd 获取 Pod 网段信息，为本节点分配一个未使用的 `/24` 段地址，然后创建 `flannel.1`（也可能是其它名称，如 flannel1 等） 接口。

flannel 将分配的 Pod 网段信息写入 `/run/flannel/docker` 文件，docker 后续使用这个文件中的环境变量设置 `docker0` 网桥。

下载和分发 flanneld 二进制文件

到 https://github.com/coreos/flannel/releases 页面下载最新版本的发布包：

```
mkdir flannel
wget https://github.com/coreos/flannel/releases/download/v0.10.0/flan
nel-v0.10.0-linux-amd64.tar.gz
tar -xzvf flannel-v0.10.0-linux-amd64.tar.gz -C flannel
```

分发 flanneld 二进制文件到集群所有节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp  flannel/{flanneld,mk-docker-opts.sh} k8s@${node_ip}:/opt/k8s
/bin/
    ssh k8s@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

创建 flannel 证书和私钥

flannel 从 etcd 集群存取网段分配信息，而 etcd 集群启用了双向 x509 证书认证，所以需要为 flanneld 生成证书和私钥。

创建证书签名请求：

```
cat > flanneld-csr.json <<EOF
{
  "CN": "flanneld",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "4Paradigm"
    }
  ]
}
EOF
```

- 该证书只会被 kubectl 当做 client 证书使用，所以 hosts 字段为空；

生成证书和私钥：

```
cfssl gencert -ca=/etc/kubernetes/cert/ca.pem \
  -ca-key=/etc/kubernetes/cert/ca-key.pem \
  -config=/etc/kubernetes/cert/ca-config.json \
  -profile=kubernetes flanneld-csr.json | cfssljson -bare flanneld
ls flanneld*pem
```

将生成的证书和私钥分发到所有节点（master 和 worker）：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /etc/flanneld/cert && chown -R k8s
/etc/flanneld"
    scp flanneld*.pem k8s@${node_ip}:/etc/flanneld/cert
  done
```

向 etcd 写入集群 Pod 网段信息

注意：本步骤只需执行一次。

```
source /opt/k8s/bin/environment.sh
etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/cert/ca.pem \
  --cert-file=/etc/flanneld/cert/flanneld.pem \
  --key-file=/etc/flanneld/cert/flanneld-key.pem \
  set ${FLANNEL_ETCD_PREFIX}/config '{"Network":"'${CLUSTER_CIDR}'",
"SubnetLen": 24, "Backend": {"Type": "vxlan"}}'
```

- flanneld 当前版本 **(v0.10.0)** 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} 必须是 /16 段地址，必须与 kube-controller-manager 的 `--cluster-cidr` 参数值一致；

创建 flanneld 的 systemd unit 文件

```
source /opt/k8s/bin/environment.sh
export IFACE=eth0
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=/opt/k8s/bin/flanneld \\
  -etcd-cafile=/etc/kubernetes/cert/ca.pem \\
  -etcd-certfile=/etc/flanneld/cert/flanneld.pem \\
  -etcd-keyfile=/etc/flanneld/cert/flanneld-key.pem \\
  -etcd-endpoints=${ETCD_ENDPOINTS} \\
  -etcd-prefix=${FLANNEL_ETCD_PREFIX} \\
  -iface=${IFACE}
ExecStartPost=/opt/k8s/bin/mk-docker-opts.sh -k DOCKER_NETWORK_OPTION
S -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 `/run/flannel/docker` 文件，后续 docker 启动时使用这个文件中的环境变量配置 docker0 网桥；
- flanneld 使用系统缺省路由所在的接口与其它节点通信，对于有多个网络接口（如内网和公网）的节点，可以用 `-iface` 参数指定通信接口，如上面的 eth0 接口;
- flanneld 运行时需要 root 权限；

完整 unit 见 flanneld.service

分发 flanneld systemd unit 文件到所有节点

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp flanneld.service root@${node_ip}:/etc/systemd/system/
  done
```

启动 flanneld 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable
flanneld && systemctl restart flanneld"
  done
```

检查启动结果

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh k8s@${node_ip} "systemctl status flanneld|grep Active"
  done
```

确保状态为 `active (running)`，否则查看日志，确认原因：

```
$ journalctl -u flanneld
```

检查分配给各 flanneld 的 Pod 网段信息

查看集群 Pod 网段(/16)：

```
source /opt/k8s/bin/environment.sh
etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/cert/ca.pem \
  --cert-file=/etc/flanneld/cert/flanneld.pem \
  --key-file=/etc/flanneld/cert/flanneld-key.pem \
  get ${FLANNEL_ETCD_PREFIX}/config
```

输出：

```
{"Network":"172.30.0.0/16", "SubnetLen": 24, "Backend": {"Type": "vxlan"}}
```

查看已分配的 Pod 子网段列表(/24):

```
source /opt/k8s/bin/environment.sh
etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/cert/ca.pem \
  --cert-file=/etc/flanneld/cert/flanneld.pem \
  --key-file=/etc/flanneld/cert/flanneld-key.pem \
  ls ${FLANNEL_ETCD_PREFIX}/subnets
```

输出：

```
/kubernetes/network/subnets/172.30.81.0-24
/kubernetes/network/subnets/172.30.29.0-24
/kubernetes/network/subnets/172.30.39.0-24
```

查看某一 Pod 网段对应的节点 IP 和 flannel 接口地址:

```
source /opt/k8s/bin/environment.sh
etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/cert/ca.pem \
  --cert-file=/etc/flanneld/cert/flanneld.pem \
  --key-file=/etc/flanneld/cert/flanneld-key.pem \
  get ${FLANNEL_ETCD_PREFIX}/subnets/172.30.81.0-24
```

输出：

```
{"PublicIP":"172.27.129.105","BackendType":"vxlan","BackendData":{"VtepMAC":"12:21:93:9e:b1:eb"}}
```

验证各节点能通过 Pod 网段互通

在各节点上部署 flannel 后，检查是否创建了 flannel 接口(名称可能为 flannel0、flannel.0、flannel.1 等)：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "/usr/sbin/ip addr show flannel.1|grep -w inet"
  done
```

输出：

```
inet 172.30.81.0/32 scope global flannel.1
inet 172.30.29.0/32 scope global flannel.1
inet 172.30.39.0/32 scope global flannel.1
```

在各节点上 ping 所有 flannel 接口 IP，确保能通：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "ping -c 1 172.30.81.0"
    ssh ${node_ip} "ping -c 1 172.30.29.0"
    ssh ${node_ip} "ping -c 1 172.30.39.0"
  done
```

zhangjun　　最后更新： 2018-10-18 04:04:02

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

06-0.部署 master 节点

kubernetes master 节点运行如下组件：

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

kube-scheduler 和 kube-controller-manager 可以以集群模式运行，通过 leader 选举产生一个工作进程，其它进程处于阻塞模式。

对于 kube-apiserver，可以运行多个实例（本文档是 3 实例），但对其它组件需要提供统一的访问地址，该地址需要高可用。本文档使用 keepalived 和 haproxy 实现 kube-apiserver VIP 高可用和负载均衡。

下载最新版本的二进制文件

从 `CHANGELOG` 页面 下载 server tarball 文件。

```
wget https://dl.k8s.io/v1.10.4/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所有 master 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp server/bin/* k8s@${node_ip}:/opt/k8s/bin/
    ssh k8s@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

zhangjun　　最后更新： 2018-10-18 04:04:02

06-1.部署高可用组件

本文档讲解使用 keepalived 和 haproxy 实现 kube-apiserver 高可用的步骤：

- keepalived 提供 kube-apiserver 对外服务的 VIP；
- haproxy 监听 VIP，后端连接所有 kube-apiserver 实例，提供健康检查和负载均衡功能；

运行 keepalived 和 haproxy 的节点称为 LB 节点。由于 keepalived 是一主多备运行模式，故至少两个 LB 节点。

本文档复用 master 节点的三台机器，haproxy 监听的端口(8443) 需要与 kube-apiserver 的端口 6443 不同，避免冲突。

keepalived 在运行过程中周期检查本机的 haproxy 进程状态，如果检测到 haproxy 进程异常，则触发重新选主的过程，VIP 将飘移到新选出来的主节点，从而实现 VIP 的高可用。

所有组件（如 kubeclt、apiserver、controller-manager、scheduler 等）都通过 VIP 和 haproxy 监听的 8443 端口访问 kube-apiserver 服务。

安装软件包

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "yum install -y keepalived haproxy"
  done
```

配置和下发 **haproxy** 配置文件

haproxy 配置文件：

```
cat > haproxy.cfg <<EOF
global
    log /dev/log    local0
    log /dev/log    local1 notice
    chroot /var/lib/haproxy
    stats socket /var/run/haproxy-admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon
    nbproc 1

defaults
    log     global
    timeout connect 5000
    timeout client  10m
    timeout server  10m

listen  admin_stats
    bind 0.0.0.0:10080
    mode http
    log 127.0.0.1 local0 err
    stats refresh 30s
    stats uri /status
    stats realm welcome login\ Haproxy
    stats auth admin:123456
    stats hide-version
    stats admin if TRUE

listen kube-master
    bind 0.0.0.0:8443
    mode tcp
    option tcplog
    balance source
    server 172.27.129.105 172.27.129.105:6443 check inter 2000 fall 2
 rise 2 weight 1
    server 172.27.129.111 172.27.129.111:6443 check inter 2000 fall 2
 rise 2 weight 1
    server 172.27.129.112 172.27.129.112:6443 check inter 2000 fall 2
 rise 2 weight 1
EOF
```

- haproxy 在 10080 端口输出 status 信息；
- haproxy 监听所有接口的 8443 端口，该端口与环境变量 ${KUBE_APISERVER} 指

定的端口必须一致；
- server 字段列出所有 kube-apiserver 监听的 IP 和端口；

下发 haproxy.cfg 到所有 master 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp haproxy.cfg root@${node_ip}:/etc/haproxy
  done
```

起 haproxy 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl restart haproxy"
  done
```

检查 haproxy 服务状态

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status haproxy|grep Active"
  done
```

确保状态为 `active (running)`，否则查看日志，确认原因：

```
journalctl -u haproxy
```

检查 haproxy 是否监听 8443 端口：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "netstat -lnpt|grep haproxy"
  done
```

确保输出类似于:

```
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               L
ISTEN      120583/haproxy
```

配置和下发 **keepalived** 配置文件

keepalived 是一主（master）多备（backup）运行模式，故有两种类型的配置文件。master 配置文件只有一份，backup 配置文件视节点数目而定，对于本文档而言，规划如下：

- master: 172.27.129.105
- backup：172.27.129.111、172.27.129.112

master 配置文件：

```
source /opt/k8s/bin/environment.sh
cat  > keepalived-master.conf <<EOF
global_defs {
    router_id lb-master-105
}

vrrp_script check-haproxy {
    script "killall -0 haproxy"
    interval 5
    weight -30
}

vrrp_instance VI-kube-master {
    state MASTER
    priority 120
    dont_track_primary
    interface ${VIP_IF}
    virtual_router_id 68
    advert_int 3
    track_script {
        check-haproxy
    }
    virtual_ipaddress {
        ${MASTER_VIP}
    }
}
EOF
```

- VIP 所在的接口（interface ${VIP_IF}）为 `eth0` ；
- 使用 `killall -0 haproxy` 命令检查所在节点的 haproxy 进程是否正常。如果异常则将权重减少（-30）,从而触发重新选主过程；
- router_id、virtual_router_id 用于标识属于该 HA 的 keepalived 实例，如果有多套 keepalived HA，则必须各不相同；

backup 配置文件：

```
source /opt/k8s/bin/environment.sh
cat  > keepalived-backup.conf <<EOF
global_defs {
    router_id lb-backup-105
}

vrrp_script check-haproxy {
    script "killall -0 haproxy"
    interval 5
    weight -30
}

vrrp_instance VI-kube-master {
    state BACKUP
    priority 110
    dont_track_primary
    interface ${VIP_IF}
    virtual_router_id 68
    advert_int 3
    track_script {
        check-haproxy
    }
    virtual_ipaddress {
        ${MASTER_VIP}
    }
}
EOF
```

- VIP 所在的接口（interface ${VIP_IF}）为 `eth0` ；
- 使用 `killall -0 haproxy` 命令检查所在节点的 haproxy 进程是否正常。如果异常则将权重减少（-30）,从而触发重新选主过程；
- router_id、virtual_router_id 用于标识属于该 HA 的 keepalived 实例，如果有多套 keepalived HA，则必须各不相同；
- priority 的值必须小于 master 的值；

下发 keepalived 配置文件

下发 master 配置文件：

```
scp keepalived-master.conf root@172.27.129.105:/etc/keepalived/keepal
ived.conf
```

下发 backup 配置文件：

```
scp keepalived-backup.conf root@172.27.129.111:/etc/keepalived/keepal
ived.conf
scp keepalived-backup.conf root@172.27.129.112:/etc/keepalived/keepal
ived.conf
```

起 keepalived 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl restart keepalived"
  done
```

检查 keepalived 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status keepalived|grep Active"
  done
```

确保状态为 `active (running)`，否则查看日志，确认原因：

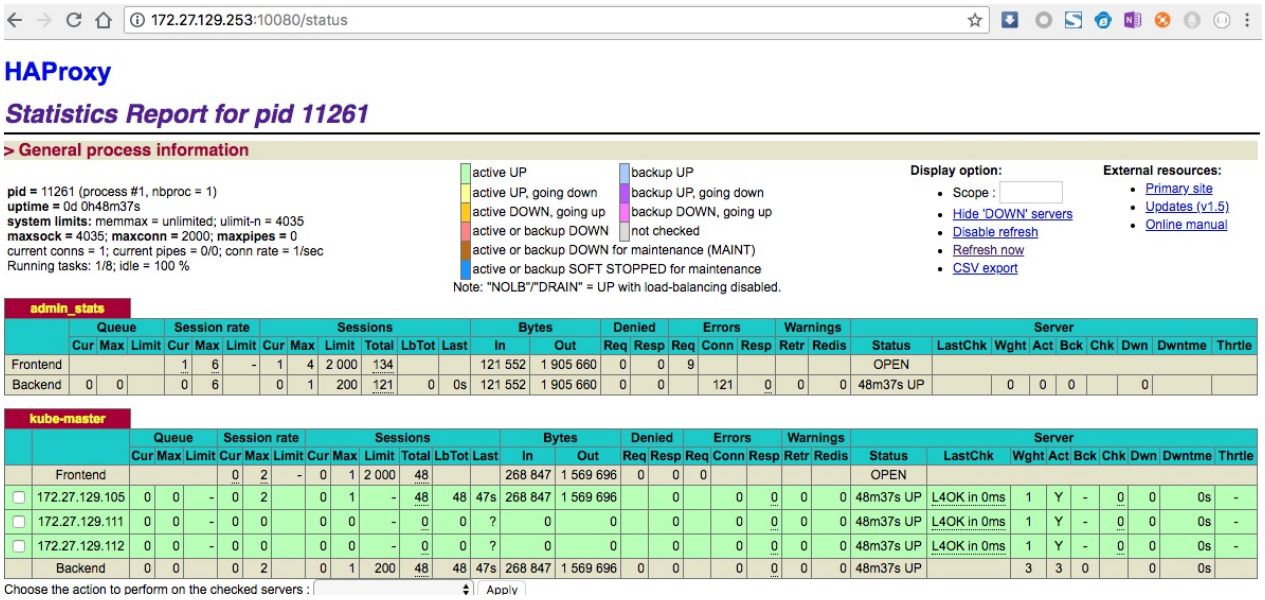
```
journalctl -u keepalived
```

查看 VIP 所在的节点，确保可以 ping 通 VIP：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "/usr/sbin/ip addr show ${VIP_IF}"
    ssh ${node_ip} "ping -c 1 ${MASTER_VIP}"
  done
```

查看 haproxy 状态页面

浏览器访问 ${MASTER_VIP}:10080/status 地址，查看 haproxy 状态页面：



图片 - *haproxy*

zhangjun　　最后更新： 2018-10-18 04:04:02

tags: master, kube-apiserver

06-1.部署 kube-apiserver 组件

本文档讲解使用 keepalived 和 haproxy 部署一个 3 节点高可用 master 集群的步骤，对应的 LB VIP 为环境变量 ${MASTER_VIP}。

准备工作

下载最新版本的二进制文件、安装和配置 flanneld 参考：06-0.部署master节点.md

创建 kubernetes 证书和私钥

创建证书签名请求：

```
source /opt/k8s/bin/environment.sh
cat > kubernetes-csr.json <<EOF
{
  "CN": "kubernetes",
  "hosts": [
    "127.0.0.1",
    "172.27.129.105",
    "172.27.129.111",
    "172.27.129.112",
    "${MASTER_VIP}",
    "${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": "4Paradigm"
    }
  ]
}
EOF
```

- hosts 字段指定授权使用该证书的 **IP** 或域名列表，这里列出了 VIP 、apiserver 节点 IP、kubernetes 服务 IP 和域名；
- 域名最后字符不能是 `.` (如不能为 `kubernetes.default.svc.cluster.local.`)，否则解析时失败，提示： `x509: cannot parse dnsName "kubernetes.default.svc.cluster.local."` ；
- 如果使用非 `cluster.local` 域名，如 `opsnull.com` ，则需要修改域名列表中的最后两个域名
  为： `kubernetes.default.svc.opsnull` 、 `kubernetes.default.svc.opsnull.com`

- kubernetes 服务 IP 是 apiserver 自动创建的，一般是 `--service-cluster-ip-range` 参数指定的网段的第一个**IP**，后续可以通过如下命令获取：

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

生成证书和私钥：

```
cfssl gencert -ca=/etc/kubernetes/cert/ca.pem \
  -ca-key=/etc/kubernetes/cert/ca-key.pem \
  -config=/etc/kubernetes/cert/ca-config.json \
  -profile=kubernetes kubernetes-csr.json | cfssljson -bare kubernetes
ls kubernetes*pem
```

将生成的证书和私钥文件拷贝到 master 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /etc/kubernetes/cert/ && sudo chown -R k8s /etc/kubernetes/cert/"
    scp kubernetes*.pem k8s@${node_ip}:/etc/kubernetes/cert/
  done
```

- k8s 账户可以读写 /etc/kubernetes/cert/ 目录；

创建加密配置文件

```
source /opt/k8s/bin/environment.sh
cat > encryption-config.yaml <<EOF
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: ${ENCRYPTION_KEY}
      - identity: {}
EOF
```

将加密配置文件拷贝到 master 节点的 `/etc/kubernetes` 目录下：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp encryption-config.yaml root@${node_ip}:/etc/kubernetes/
  done
```

替换后的 encryption-config.yaml 文件：encryption-config.yaml

创建 kube-apiserver systemd unit 模板文件

```
source /opt/k8s/bin/environment.sh
cat > kube-apiserver.service.template <<EOF
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target

[Service]
ExecStart=/opt/k8s/bin/kube-apiserver \\
  --enable-admission-plugins=Initializers,NamespaceLifecycle,NodeRest
riction,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota
\\
  --anonymous-auth=false \\
```

```
  --experimental-encryption-provider-config=/etc/kubernetes/encryptio
n-config.yaml \\
  --advertise-address=##NODE_IP## \\
  --bind-address=##NODE_IP## \\
  --insecure-port=0 \\
  --authorization-mode=Node,RBAC \\
  --runtime-config=api/all \\
  --enable-bootstrap-token-auth \\
  --service-cluster-ip-range=${SERVICE_CIDR} \\
  --service-node-port-range=${NODE_PORT_RANGE} \\
  --tls-cert-file=/etc/kubernetes/cert/kubernetes.pem \\
  --tls-private-key-file=/etc/kubernetes/cert/kubernetes-key.pem \\
  --client-ca-file=/etc/kubernetes/cert/ca.pem \\
  --kubelet-client-certificate=/etc/kubernetes/cert/kubernetes.pem \\
  --kubelet-client-key=/etc/kubernetes/cert/kubernetes-key.pem \\
  --service-account-key-file=/etc/kubernetes/cert/ca-key.pem \\
  --etcd-cafile=/etc/kubernetes/cert/ca.pem \\
  --etcd-certfile=/etc/kubernetes/cert/kubernetes.pem \\
  --etcd-keyfile=/etc/kubernetes/cert/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/log/kube-apiserver-audit.log \\
  --event-ttl=1h \\
  --alsologtostderr=true \\
  --logtostderr=false \\
  --log-dir=/var/log/kubernetes \\
  --v=2
Restart=on-failure
RestartSec=5
Type=notify
User=k8s
LimitNOFILE=65536

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

- `--experimental-encryption-provider-config`：启用加密特性；
- `--authorization-mode=Node,RBAC`： 开启 Node 和 RBAC 授权模式，拒绝未授

权的请求；

- `--enable-admission-plugins`：启用 `ServiceAccount` 和 `NodeRestriction`；
- `--service-account-key-file`：签名 ServiceAccount Token 的公钥文件，kube-controller-manager 的 `--service-account-private-key-file` 指定私钥文件，两者配对使用；
- `--tls-*-file`：指定 apiserver 使用的证书、私钥和 CA 文件。`--client-ca-file` 用于验证 client (kue-controller-manager、kube-scheduler、kubelet、kube-proxy 等)请求所带的证书；
- `--kubelet-client-certificate`、`--kubelet-client-key`：如果指定，则使用 https 访问 kubelet APIs；需要为证书对应的用户(上面 kubernetes*.pem 证书的用户为 kubernetes) 用户定义 RBAC 规则，否则访问 kubelet API 时提示未授权；
- `--bind-address`：不能为 `127.0.0.1`，否则外界不能访问它的安全端口 6443；
- `--insecure-port=0`：关闭监听非安全端口(8080)；
- `--service-cluster-ip-range`：指定 Service Cluster IP 地址段；
- `--service-node-port-range`：指定 NodePort 的端口范围；
- `--runtime-config=api/all=true`：启用所有版本的 APIs，如 autoscaling/v2alpha1；
- `--enable-bootstrap-token-auth`：启用 kubelet bootstrap 的 token 认证；
- `--apiserver-count=3`：指定集群运行模式，多台 kube-apiserver 会通过 leader 选举产生一个工作节点，其它节点处于阻塞状态；
- `User=k8s`：使用 k8s 账户运行；

为各节点创建和分发 kube-apiserver systemd unit 文件

替换模板文件中的变量，为各节点创建 systemd unit 文件：

```
source /opt/k8s/bin/environment.sh
for (( i=0; i < 3; i++ ))
  do
    sed -e "s/##NODE_NAME##/${NODE_NAMES[i]}/" -e "s/##NODE_IP##/${NODE_IPS[i]}/" kube-apiserver.service.template > kube-apiserver-${NODE_IPS[i]}.service
  done
ls kube-apiserver*.service
```

- NODE_NAMES 和 NODE_IPS 为相同长度的 bash 数组，分别为节点名称和对应的 IP；

分发生成的 systemd unit 文件：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /var/log/kubernetes && chown -R k8s
 /var/log/kubernetes"
    scp kube-apiserver-${node_ip}.service root@${node_ip}:/etc/system
d/system/kube-apiserver.service
  done
```

- 必须先创建日志目录；
- 文件重命名为 kube-apiserver.service;

替换后的 unit 文件：kube-apiserver.service

启动 kube-apiserver 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable
kube-apiserver && systemctl restart kube-apiserver"
  done
```

检查 kube-apiserver 运行状态

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status kube-apiserver |grep 'Activ
e:'"
  done
```

确保状态为 `active (running)`，否则到 master 节点查看日志，确认原因：

```
journalctl -u kube-apiserver
```

打印 kube-apiserver 写入 etcd 的数据

```
source /opt/k8s/bin/environment.sh
ETCDCTL_API=3 etcdctl \
    --endpoints=${ETCD_ENDPOINTS} \
    --cacert=/etc/kubernetes/cert/ca.pem \
    --cert=/etc/etcd/cert/etcd.pem \
    --key=/etc/etcd/cert/etcd-key.pem \
    get /registry/ --prefix --keys-only
```

检查集群信息

```
$ kubectl cluster-info
Kubernetes master is running at https://172.27.129.253:8443

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

$ kubectl get all --all-namespaces
NAMESPACE   NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP
   PORT(S)   AGE
default     service/kubernetes   ClusterIP   10.254.0.1   <none>
   443/TCP   35m

$ kubectl get componentstatuses
NAME                 STATUS      MESSAGE
                                                                ERROR
controller-manager   Unhealthy   Get http://127.0.0.1:10252/healthz:
dial tcp 127.0.0.1:10252: getsockopt: connection refused
scheduler            Unhealthy   Get http://127.0.0.1:10251/healthz:
dial tcp 127.0.0.1:10251: getsockopt: connection refused
etcd-1               Healthy     {"health":"true"}
etcd-0               Healthy     {"health":"true"}
etcd-2               Healthy     {"health":"true"}
```

注意：

1. 如果执行 kubectl 命令式时输出如下错误信息，则说明使用的 `~/.kube/config` 文件不对，请切换到正确的账户后再执行该命令：

   ```
   The connection to the server localhost:8080 was refused - did you specify the right host or port?
   ```

2. 执行 `kubectl get componentstatuses` 命令时，apiserver 默认向 127.0.0.1 发送请求。当 controller-manager、scheduler 以集群模式运行时，有可能和 kube-apiserver 不在一台机器上，这时 controller-manager 或 scheduler 的状态为 Unhealthy，但实际上它们工作正常。

检查 kube-apiserver 监听的端口

```
$ sudo netstat -lnpt|grep kube
tcp        0      0 172.27.129.105:6443     0.0.0.0:*               L
ISTEN      13075/kube-apiserve
```

- 6443: 接收 https 请求的安全端口，对所有请求做认证和授权；
- 由于关闭了非安全端口，故没有监听 8080；

授予 kubernetes 证书访问 kubelet API 的权限

在执行 kubectl exec、run、logs 等命令时，apiserver 会转发到 kubelet。这里定义 RBAC 规则，授权 apiserver 调用 kubelet API。

```
$ kubectl create clusterrolebinding kube-apiserver:kubelet-apis --clu
sterrole=system:kubelet-api-admin --user kubernetes
```

参考

1. 关于证书域名最后字符不能是 `.` 的问题，实际和 Go 的版本有关，1.9 不支持这种类型的证书：https://github.com/kubernetes/ingress-nginx/issues/2188

zhangjun　　最后更新： 2018-10-18 04:04:02

tags: master, kube-controller-manager

06-3.部署高可用 kube-controller-manager 集群

本文档介绍部署高可用 kube-controller-manager 集群的步骤。

该集群包含 3 个节点，启动后将通过竞争选举机制产生一个 leader 节点，其它节点为阻塞状态。当 leader 节点不可用后，剩余节点将再次进行选举产生新的 leader 节点，从而保证服务的可用性。

为保证通信安全，本文档先生成 x509 证书和私钥，kube-controller-manager 在如下两种情况下使用该证书：

1. 与 kube-apiserver 的安全端口通信时;
2. 在安全端口(https，10252) 输出 prometheus 格式的 metrics；

准备工作

下载最新版本的二进制文件、安装和配置 flanneld 参考：06-0.部署master节点.md

创建 kube-controller-manager 证书和私钥

创建证书签名请求：

```
cat > kube-controller-manager-csr.json <<EOF
{
    "CN": "system:kube-controller-manager",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "hosts": [
      "127.0.0.1",
      "172.27.129.105",
      "172.27.129.111",
      "172.27.129.112"
    ],
    "names": [
      {
        "C": "CN",
        "ST": "BeiJing",
        "L": "BeiJing",
        "O": "system:kube-controller-manager",
        "OU": "4Paradigm"
      }
    ]
}
EOF
```

- hosts 列表包含所有 kube-controller-manager 节点 IP；
- CN 为 system:kube-controller-manager、O 为 system:kube-controller-manager，kubernetes 内置的 ClusterRoleBindings system:kube-controller-manager 赋予 kube-controller-manager 工作所需的权限。

生成证书和私钥：

```
cfssl gencert -ca=/etc/kubernetes/cert/ca.pem \
  -ca-key=/etc/kubernetes/cert/ca-key.pem \
  -config=/etc/kubernetes/cert/ca-config.json \
  -profile=kubernetes kube-controller-manager-csr.json | cfssljson -b
are kube-controller-manager
```

将生成的证书和私钥分发到所有 master 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-controller-manager*.pem k8s@${node_ip}:/etc/kubernetes/c
ert/
  done
```

创建和分发 kubeconfig 文件

kubeconfig 文件包含访问 apiserver 的所有信息，如 apiserver 地址、CA 证书和自身使用的证书；

```
source /opt/k8s/bin/environment.sh
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/cert/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-controller-manager.kubeconfig

kubectl config set-credentials system:kube-controller-manager \
  --client-certificate=kube-controller-manager.pem \
  --client-key=kube-controller-manager-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-controller-manager.kubeconfig

kubectl config set-context system:kube-controller-manager \
  --cluster=kubernetes \
  --user=system:kube-controller-manager \
  --kubeconfig=kube-controller-manager.kubeconfig

kubectl config use-context system:kube-controller-manager --kubeconfig
=kube-controller-manager.kubeconfig
```

分发 kubeconfig 到所有 master 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-controller-manager.kubeconfig k8s@${node_ip}:/etc/kubern
etes/
  done
```

创建和分发 kube-controller-manager systemd unit 文件

```
source /opt/k8s/bin/environment.sh
cat > kube-controller-manager.service <<EOF
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
ExecStart=/opt/k8s/bin/kube-controller-manager \\
  --port=0 \\
  --secure-port=10252 \\
  --bind-address=127.0.0.1 \\
  --kubeconfig=/etc/kubernetes/kube-controller-manager.kubeconfig \\
  --service-cluster-ip-range=${SERVICE_CIDR} \\
  --cluster-name=kubernetes \\
  --cluster-signing-cert-file=/etc/kubernetes/cert/ca.pem \\
  --cluster-signing-key-file=/etc/kubernetes/cert/ca-key.pem \\
  --experimental-cluster-signing-duration=8760h \\
  --root-ca-file=/etc/kubernetes/cert/ca.pem \\
  --service-account-private-key-file=/etc/kubernetes/cert/ca-key.pem
\\
  --leader-elect=true \\
  --feature-gates=RotateKubeletServerCertificate=true \\
  --controllers=*,bootstrapsigner,tokencleaner \\
  --horizontal-pod-autoscaler-use-rest-clients=true \\
  --horizontal-pod-autoscaler-sync-period=10s \\
  --tls-cert-file=/etc/kubernetes/cert/kube-controller-manager.pem \\
  --tls-private-key-file=/etc/kubernetes/cert/kube-controller-manager
-key.pem \\
  --use-service-account-credentials=true \\
  --alsologtostderr=true \\
  --logtostderr=false \\
  --log-dir=/var/log/kubernetes \\
  --v=2
Restart=on
Restart=on-failure
RestartSec=5
User=k8s

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

- `--port=0`：关闭监听 http /metrics 的请求，同时 `--address` 参数无效，`--bind-address` 参数有效；

- `--secure-port=10252` 、 `--bind-address=0.0.0.0` ：在所有网络接口监听 10252 端口的 https /metrics 请求；
- `--kubeconfig` ：指定 kubeconfig 文件路径，kube-controller-manager 使用它连接和验证 kube-apiserver；
- `--cluster-signing-*-file` ：签名 TLS Bootstrap 创建的证书；
- `--experimental-cluster-signing-duration` ：指定 TLS Bootstrap 证书的有效期；
- `--root-ca-file` ：放置到容器 ServiceAccount 中的 CA 证书，用来对 kube-apiserver 的证书进行校验；
- `--service-account-private-key-file` ：签名 ServiceAccount 中 Token 的私钥文件，必须和 kube-apiserver 的 `--service-account-key-file` 指定的公钥文件配对使用；
- `--service-cluster-ip-range` ：指定 Service Cluster IP 网段，必须和 kube-apiserver 中的同名参数一致；
- `--leader-elect=true` ：集群运行模式，启用选举功能；被选为 leader 的节点负责处理工作，其它节点为阻塞状态；
- `--feature-gates=RotateKubeletServerCertificate=true` ：开启 kublet server 证书的自动更新特性；
- `--controllers=*,bootstrapsigner,tokencleaner` ：启用的控制器列表，tokencleaner 用于自动清理过期的 Bootstrap token；
- `--horizontal-pod-autoscaler-*` ：custom metrics 相关参数，支持 autoscaling/v2alpha1；
- `--tls-cert-file` 、 `--tls-private-key-file` ：使用 https 输出 metrics 时使用的 Server 证书和秘钥；
- `--use-service-account-credentials=true` :
- `User=k8s` ：使用 k8s 账户运行；

kube-controller-manager 不对请求 https metrics 的 Client 证书进行校验，故不需要指定 `--tls-ca-file` 参数，而且该参数已被淘汰。

完整 unit 见 kube-controller-manager.service

分发 systemd unit 文件到所有 master 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-controller-manager.service root@${node_ip}:/etc/systemd/
system/
  done
```

kube-controller-manager 的权限

ClusteRole: system:kube-controller-manager 的权限很小，只能创建 secret、serviceaccount 等资源对象，各 controller 的权限分散到 ClusterRole system:controller:XXX 中。

需要在 kube-controller-manager 的启动参数中添加 `--use-service-account-credentials=true` 参数，这样 main controller 会为各 controller 创建对应的 ServiceAccount XXX-controller。

内置的 ClusterRoleBinding system:controller:XXX 将赋予各 XXX-controller ServiceAccount 对应的 ClusterRole system:controller:XXX 权限。

启动 kube-controller-manager 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /var/log/kubernetes && chown -R k8s
 /var/log/kubernetes"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable
kube-controller-manager && systemctl restart kube-controller-manager"
  done
```

- 必须先创建日志目录；

检查服务运行状态

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh k8s@${node_ip} "systemctl status kube-controller-manager|grep
 Active"
  done
```

确保状态为 `active (running)`，否则查看日志，确认原因：

```
$ journalctl -u kube-controller-manager
```

查看输出的 metric

注意：以下命令在 kube-controller-manager 节点上执行。

kube-controller-manager 监听 10252 端口，接收 https 请求：

```
$ sudo netstat -lnpt|grep kube-controll
tcp        0      0 127.0.0.1:10252         0.0.0.0:*               L
ISTEN      18377/kube-controll
```

```
$ curl -s --cacert /etc/kubernetes/cert/ca.pem https://127.0.0.1:1025
2/metrics |head
# HELP ClusterRoleAggregator_adds Total number of adds handled by wor
kqueue: ClusterRoleAggregator
# TYPE ClusterRoleAggregator_adds counter
ClusterRoleAggregator_adds 3
# HELP ClusterRoleAggregator_depth Current depth of workqueue: Cluste
rRoleAggregator
# TYPE ClusterRoleAggregator_depth gauge
ClusterRoleAggregator_depth 0
# HELP ClusterRoleAggregator_queue_latency How long an item stays in
workqueueClusterRoleAggregator before being requested.
# TYPE ClusterRoleAggregator_queue_latency summary
ClusterRoleAggregator_queue_latency{quantile="0.5"} 57018
ClusterRoleAggregator_queue_latency{quantile="0.9"} 57268
```

- curl --cacert CA 证书用来验证 kube-controller-manager https server 证书；

测试 kube-controller-manager 集群的高可用

停掉一个或两个节点的 kube-controller-manager 服务，观察其它节点的日志，看是否获取了 leader 权限。

查看当前的 leader

```
$ kubectl get endpoints kube-controller-manager --namespace=kube-syst
em  -o yaml
apiVersion: v1
kind: Endpoints
metadata:
  annotations:
    control-plane.alpha.kubernetes.io/leader: '{"holderIdentity":"kub
e-node2_084534e2-6cc4-11e8-a418-5254001f5b65","leaseDurationSeconds":
15,"acquireTime":"2018-06-10T15:40:33Z","renewTime":"2018-06-10T16:19
:08Z","leaderTransitions":12}'
  creationTimestamp: 2018-06-10T13:59:42Z
  name: kube-controller-manager
  namespace: kube-system
  resourceVersion: "4540"
  selfLink: /api/v1/namespaces/kube-system/endpoints/kube-controller-
manager
  uid: 862cc048-6cb6-11e8-96fa-525400ba84c6
```

可见，当前的 leader 为 kube-node2 节点。

参考

1. 关于 controller 权限和 use-service-account-credentials 参数：https://github.com/kubernetes/kubernetes/issues/48208
2. kublet 认证和授权：https://kubernetes.io/docs/admin/kubelet-authentication-authorization/#kubelet-authorization

zhangjun　　最后更新： 2018-10-18 04:04:02

tags: master, kube-scheduler

06-3.部署高可用 kube-scheduler 集群

本文档介绍部署高可用 kube-scheduler 集群的步骤。

该集群包含 3 个节点，启动后将通过竞争选举机制产生一个 leader 节点，其它节点为阻塞状态。当 leader 节点不可用后，剩余节点将再次进行选举产生新的 leader 节点，从而保证服务的可用性。

为保证通信安全，本文档先生成 x509 证书和私钥，kube-scheduler 在如下两种情况下使用该证书：

1. 与 kube-apiserver 的安全端口通信;
2. 在安全端口(https，10251) 输出 prometheus 格式的 metrics；

准备工作

下载最新版本的二进制文件、安装和配置 flanneld 参考：06-0.部署master节点.md

创建 kube-scheduler 证书和私钥

创建证书签名请求：

```
cat > kube-scheduler-csr.json <<EOF
{
    "CN": "system:kube-scheduler",
    "hosts": [
      "127.0.0.1",
      "172.27.129.105",
      "172.27.129.111",
      "172.27.129.112"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
      {
        "C": "CN",
        "ST": "BeiJing",
        "L": "BeiJing",
        "O": "system:kube-scheduler",
        "OU": "4Paradigm"
      }
    ]
}
EOF
```

- hosts 列表包含所有 kube-scheduler 节点 IP；
- CN 为 system:kube-scheduler、O 为 system:kube-scheduler，kubernetes 内置的 ClusterRoleBindings system:kube-scheduler 将赋予 kube-scheduler 工作所需的权限。

生成证书和私钥：

```
cfssl gencert -ca=/etc/kubernetes/cert/ca.pem \
  -ca-key=/etc/kubernetes/cert/ca-key.pem \
  -config=/etc/kubernetes/cert/ca-config.json \
  -profile=kubernetes kube-scheduler-csr.json | cfssljson -bare kube-scheduler
```

创建和分发 **kubeconfig** 文件

kubeconfig 文件包含访问 apiserver 的所有信息，如 apiserver 地址、CA 证书和自身使用的证书；

```
source /opt/k8s/bin/environment.sh
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/cert/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-scheduler.kubeconfig

kubectl config set-credentials system:kube-scheduler \
  --client-certificate=kube-scheduler.pem \
  --client-key=kube-scheduler-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-scheduler.kubeconfig

kubectl config set-context system:kube-scheduler \
  --cluster=kubernetes \
  --user=system:kube-scheduler \
  --kubeconfig=kube-scheduler.kubeconfig

kubectl config use-context system:kube-scheduler --kubeconfig=kube-sc
heduler.kubeconfig
```

- 上一步创建的证书、私钥以及 kube-apiserver 地址被写入到 kubeconfig 文件中；

分发 kubeconfig 到所有 master 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-scheduler.kubeconfig k8s@${node_ip}:/etc/kubernetes/
  done
```

创建和分发 kube-scheduler systemd unit 文件

```
cat > kube-scheduler.service <<EOF
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
ExecStart=/opt/k8s/bin/kube-scheduler \\
  --address=127.0.0.1 \\
  --kubeconfig=/etc/kubernetes/kube-scheduler.kubeconfig \\
  --leader-elect=true \\
  --alsologtostderr=true \\
  --logtostderr=false \\
  --log-dir=/var/log/kubernetes \\
  --v=2
Restart=on-failure
RestartSec=5
User=k8s

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

- `--address`：在 127.0.0.1:10251 端口接收 http /metrics 请求；kube-scheduler 目前还不支持接收 https 请求；
- `--kubeconfig`：指定 kubeconfig 文件路径，kube-scheduler 使用它连接和验证 kube-apiserver；
- `--leader-elect=true`：集群运行模式，启用选举功能；被选为 leader 的节点负责处理工作，其它节点为阻塞状态；
- `User=k8s`：使用 k8s 账户运行；

完整 unit 见 kube-scheduler.service。

分发 systemd unit 文件到所有 master 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-scheduler.service root@${node_ip}:/etc/systemd/system/
  done
```

启动 kube-scheduler 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /var/log/kubernetes && chown -R k8s
 /var/log/kubernetes"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable
kube-scheduler && systemctl restart kube-scheduler"
  done
```

- 必须先创建日志目录；

检查服务运行状态

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh k8s@${node_ip} "systemctl status kube-scheduler|grep Active"
  done
```

确保状态为 `active (running)`，否则查看日志，确认原因：

```
journalctl -u kube-scheduler
```

查看输出的 metric

注意：以下命令在 kube-scheduler 节点上执行。

kube-scheduler 监听 10251 端口，接收 http 请求：

```
$ sudo netstat -lnpt|grep kube-sche
tcp        0      0 127.0.0.1:10251         0.0.0.0:*               L
ISTEN      23783/kube-schedule
```

```
$ curl -s http://127.0.0.1:10251/metrics |head
# HELP apiserver_audit_event_total Counter of audit events generated
and sent to the audit backend.
# TYPE apiserver_audit_event_total counter
apiserver_audit_event_total 0
# HELP go_gc_duration_seconds A summary of the GC invocation duration
s.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 9.7715e-05
go_gc_duration_seconds{quantile="0.25"} 0.000107676
go_gc_duration_seconds{quantile="0.5"} 0.00017868
go_gc_duration_seconds{quantile="0.75"} 0.000262444
go_gc_duration_seconds{quantile="1"} 0.001205223
```

测试 kube-scheduler 集群的高可用

随便找一个或两个 master 节点，停掉 kube-scheduler 服务，看其它节点是否获取了 leader 权限（systemd 日志）。

查看当前的 leader

```
$ kubectl get endpoints kube-scheduler --namespace=kube-system  -o ya
ml
apiVersion: v1
kind: Endpoints
metadata:
  annotations:
    control-plane.alpha.kubernetes.io/leader: '{"holderIdentity":"kub
e-node3_61f34593-6cc8-11e8-8af7-5254002f288e","leaseDurationSeconds":
15,"acquireTime":"2018-06-10T16:09:56Z","renewTime":"2018-06-10T16:20
:54Z","leaderTransitions":1}'
  creationTimestamp: 2018-06-10T16:07:33Z
  name: kube-scheduler
  namespace: kube-system
  resourceVersion: "4645"
  selfLink: /api/v1/namespaces/kube-system/endpoints/kube-scheduler
  uid: 62382d98-6cc8-11e8-96fa-525400ba84c6
```

可见，当前的 leader 为 kube-node3 节点。

zhangjun　　最后更新：2018-10-18 04:04:02

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

07-0.部署 worker 节点

kubernetes work 节点运行如下组件：

- docker
- kubelet
- kube-proxy

安装和配置 flanneld

参考 05-部署flannel网络.md

安装依赖包

CentOS:

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "yum install -y epel-release"
    ssh root@${node_ip} "yum install -y conntrack ipvsadm ipset jq ip
tables curl sysstat libseccomp && /usr/sbin/modprobe ip_vs "
  done
```

Ubuntu:

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "apt-get install -y conntrack ipvsadm ipset j
q iptables curl sysstat libseccomp && /usr/sbin/modprobe ip_vs "
  done
```

zhangjun 最后更新： 2018-10-18 04:04:02

tags: worker, docker

07-1.部署 docker 组件

docker 是容器的运行环境，管理它的生命周期。kubelet 通过 Container Runtime Interface (CRI) 与 docker 进行交互。

安装依赖包

参考 07-0.部署worker节点.md

下载和分发 docker 二进制文件

到 https://download.docker.com/linux/static/stable/x86_64/ 页面下载最新发布包：

```
wget https://download.docker.com/linux/static/stable/x86_64/docker-18
.03.1-ce.tgz
tar -xvf docker-18.03.1-ce.tgz
```

分发二进制文件到所有 worker 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp docker/docker*  k8s@${node_ip}:/opt/k8s/bin/
    ssh k8s@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

创建和分发 systemd unit 文件

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

[Service]
Environment="PATH=/opt/k8s/bin:/bin:/sbin:/usr/bin:/usr/sbin"
EnvironmentFile=-/run/flannel/docker
ExecStart=/opt/k8s/bin/dockerd --log-level=error $DOCKER_NETWORK_OPTI
ONS
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
EOF
```

- EOF 前后有双引号，这样 bash 不会替换文档中的变量，如 $DOCKER_NETWORK_OPTIONS；
- 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 `/run/flannel/docker` 文件中，dockerd 启动前读取该文件中的环境变量 `DOCKER_NETWORK_OPTIONS` ，然后设置 docker0 网桥网段；
- 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)；
- docker 需要以 root 用于运行；
- 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**

```
/sbin/iptables -P FORWARD ACCEPT
```

完整 unit 见 docker.service

分发 systemd unit 文件到所有 worker 机器:

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp docker.service root@${node_ip}:/etc/systemd/system/
  done
```

配置和分发 docker 配置文件

使用国内的仓库镜像服务器以加快 pull image 的速度，同时增加下载的并发数 (需要重启 dockerd 生效)：

```
cat > docker-daemon.json <<EOF
{
    "registry-mirrors": ["https://hub-mirror.c.163.com", "https://doc
ker.mirrors.ustc.edu.cn"],
    "max-concurrent-downloads": 20
}
EOF
```

分发 docker 配置文件到所有 work 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p  /etc/docker/"
    scp docker-daemon.json root@${node_ip}:/etc/docker/daemon.json
  done
```

启动 docker 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl stop firewalld && systemctl disabl
e firewalld"
    ssh root@${node_ip} "/usr/sbin/iptables -F && /usr/sbin/iptables
-X && /usr/sbin/iptables -F -t nat && /usr/sbin/iptables -X -t nat"
    ssh root@${node_ip} "/usr/sbin/iptables -P FORWARD ACCEPT"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable
docker && systemctl restart docker"
    ssh root@${node_ip} 'for intf in /sys/devices/virtual/net/docker0
/brif/*; do echo 1 > $intf/hairpin_mode; done'
    ssh root@${node_ip} "sudo sysctl -p /etc/sysctl.d/kubernetes.conf"

  done
```

- 关闭 firewalld(centos7)/ufw(ubuntu16.04)，否则可能会重复创建 iptables 规则；
- 清理旧的 iptables rules 和 chains 规则；
- 开启 docker0 网桥下虚拟网卡的 hairpin 模式;

检查服务运行状态

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh k8s@${node_ip} "systemctl status docker|grep Active"
  done
```

确保状态为 `active (running)`，否则查看日志，确认原因：

```
$ journalctl -u docker
```

检查 docker0 网桥

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh k8s@${node_ip} "/usr/sbin/ip addr show flannel.1 && /usr/sbin
/ip addr show docker0"
  done
```

确认各 work 节点的 docker0 网桥和 flannel.1 接口的 IP 处于同一个网段中(如下 172.30.39.0 和 172.30.39.1)：

```
3: flannel.1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueu
e state UNKNOWN group default
    link/ether ce:2f:d6:53:e5:f3 brd ff:ff:ff:ff:ff:ff
    inet 172.30.39.0/32 scope global flannel.1
      valid_lft forever preferred_lft forever
    inet6 fe80::cc2f:d6ff:fe53:e5f3/64 scope link
      valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueu
e state DOWN group default
    link/ether 02:42:bf:65:16:5c brd ff:ff:ff:ff:ff:ff
    inet 172.30.39.1/24 brd 172.30.39.255 scope global docker0
      valid_lft forever preferred_lft forever
```

zhangjun　　最后更新： 2018-10-18 04:04:02

tags: worker, kubelet

07-2.部署 kubelet 组件

kublet 运行在每个 worker 节点上，接收 kube-apiserver 发送的请求，管理 Pod 容器，执行交互式命令，如 exec、run、logs 等。

kublet 启动时自动向 kube-apiserver 注册节点信息，内置的 cadvisor 统计和监控节点的资源使用情况。

为确保安全，本文档只开启接收 https 请求的安全端口，对请求进行认证和授权，拒绝未授权的访问(如 apiserver、heapster)。

下载和分发 kubelet 二进制文件

参考 06-0.部署master节点.md

安装依赖包

参考 07-0.部署worker节点.md

创建 kubelet bootstrap kubeconfig 文件

```
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"

    # 创建 token
    export BOOTSTRAP_TOKEN=$(kubeadm token create \
      --description kubelet-bootstrap-token \
      --groups system:bootstrappers:${node_name} \
      --kubeconfig ~/.kube/config)

    # 设置集群参数
    kubectl config set-cluster kubernetes \
      --certificate-authority=/etc/kubernetes/cert/ca.pem \
      --embed-certs=true \
      --server=${KUBE_APISERVER} \
      --kubeconfig=kubelet-bootstrap-${node_name}.kubeconfig

    # 设置客户端认证参数
    kubectl config set-credentials kubelet-bootstrap \
      --token=${BOOTSTRAP_TOKEN} \
      --kubeconfig=kubelet-bootstrap-${node_name}.kubeconfig

    # 设置上下文参数
    kubectl config set-context default \
      --cluster=kubernetes \
      --user=kubelet-bootstrap \
      --kubeconfig=kubelet-bootstrap-${node_name}.kubeconfig

    # 设置默认上下文
    kubectl config use-context default --kubeconfig=kubelet-bootstrap-
${node_name}.kubeconfig
  done
```

- 证书中写入 Token 而非证书，证书后续由 controller-manager 创建。

查看 kubeadm 为各节点创建的 token：

```
$ kubeadm token list --kubeconfig ~/.kube/config
TOKEN                     TTL       EXPIRES                     USAGE
S                   DESCRIPTION               EXTRA GROUPS
k0s2bj.7nvw1zi1nalyz4gz   23h       2018-06-14T15:14:31+08:00   authe
ntication,signing   kubelet-bootstrap-token   system:bootstrappers:ku
be-node1
mkus5s.vilnjk3kutei600l   23h       2018-06-14T15:14:32+08:00   authe
ntication,signing   kubelet-bootstrap-token   system:bootstrappers:ku
be-node3
zkiem5.0m4xhw0jc8r466nk   23h       2018-06-14T15:14:32+08:00   authe
ntication,signing   kubelet-bootstrap-token   system:bootstrappers:ku
be-node2
```

- 创建的 token 有效期为 1 天，超期后将不能再被使用，且会被 kube-controller-manager 的 tokencleaner 清理(如果启用该 controller 的话)；
- kube-apiserver 接收 kubelet 的 bootstrap token 后，将请求的 user 设置为 system:bootstrap:，group 设置为 system:bootstrappers；

各 token 关联的 Secret：

```
$ kubectl get secrets  -n kube-system
NAME                     TYPE                                  DATA
    AGE
bootstrap-token-k0s2bj   bootstrap.kubernetes.io/token         7
    1m
bootstrap-token-mkus5s   bootstrap.kubernetes.io/token         7
    1m
bootstrap-token-zkiem5   bootstrap.kubernetes.io/token         7
    1m
default-token-99st7      kubernetes.io/service-account-token   3
    2d
```

分发 bootstrap kubeconfig 文件到所有 worker 节点

```
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"
    scp kubelet-bootstrap-${node_name}.kubeconfig k8s@${node_name}:/e
tc/kubernetes/kubelet-bootstrap.kubeconfig
  done
```

创建和分发 kubelet 参数配置文件

从 v1.10 开始，kubelet 部分参数需在配置文件中配置，`kubelet --help` 会提示：

```
DEPRECATED: This parameter should be set via the config file specifie
d by the Kubelet's --config flag
```

创建 kubelet 参数配置模板文件：

```
source /opt/k8s/bin/environment.sh
cat > kubelet.config.json.template <<EOF
{
  "kind": "KubeletConfiguration",
  "apiVersion": "kubelet.config.k8s.io/v1beta1",
  "authentication": {
    "x509": {
      "clientCAFile": "/etc/kubernetes/cert/ca.pem"
    },
    "webhook": {
      "enabled": true,
      "cacheTTL": "2m0s"
    },
    "anonymous": {
      "enabled": false
    }
  },
  "authorization": {
    "mode": "Webhook",
    "webhook": {
      "cacheAuthorizedTTL": "5m0s",
      "cacheUnauthorizedTTL": "30s"
    }
  },
  "address": "##NODE_IP##",
  "port": 10250,
  "readOnlyPort": 0,
  "cgroupDriver": "cgroupfs",
  "hairpinMode": "promiscuous-bridge",
  "serializeImagePulls": false,
  "featureGates": {
    "RotateKubeletClientCertificate": true,
    "RotateKubeletServerCertificate": true
  },
  "clusterDomain": "${CLUSTER_DNS_DOMAIN}",
  "clusterDNS": ["${CLUSTER_DNS_SVC_IP}"]
}
EOF
```

- address：API 监听地址，不能为 127.0.0.1，否则 kube-apiserver、heapster 等不能调用 kubelet 的 API；
- readOnlyPort=0：关闭只读端口(默认 10255)，等效为未指定；
- authentication.anonymous.enabled：设置为 false，不允许匿名访问 10250 端口；

- authentication.x509.clientCAFile：指定签名客户端证书的 CA 证书，开启 HTTP 证书认证；
- authentication.webhook.enabled=true：开启 HTTPs bearer token 认证；
- 对于未通过 x509 证书和 webhook 认证的请求(kube-apiserver 或其他客户端)，将被拒绝，提示 Unauthorized；
- authroization.mode=Webhook：kubelet 使用 SubjectAccessReview API 查询 kube-apiserver 某 user、group 是否具有操作资源的权限(RBAC)；
- featureGates.RotateKubeletClientCertificate、featureGates.RotateKubeletServerCertificate：自动 rotate 证书，证书的有效期取决于 kube-controller-manager 的 --experimental-cluster-signing-duration 参数；
- 需要 root 账户运行；

为各节点创建和分发 kubelet 配置文件：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    sed -e "s/##NODE_IP##/${node_ip}/" kubelet.config.json.template >
 kubelet.config-${node_ip}.json
    scp kubelet.config-${node_ip}.json root@${node_ip}:/etc/kubernete
s/kubelet.config.json
  done
```

替换后的 kubelet.config.json 文件：kubelet.config.json

创建和分发 kubelet systemd unit 文件

创建 kubelet systemd unit 文件模板：

```
cat > kubelet.service.template <<EOF
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=docker.service
Requires=docker.service

[Service]
WorkingDirectory=/var/lib/kubelet
ExecStart=/opt/k8s/bin/kubelet \\
  --bootstrap-kubeconfig=/etc/kubernetes/kubelet-bootstrap.kubeconfig
 \\
  --cert-dir=/etc/kubernetes/cert \\
  --kubeconfig=/etc/kubernetes/kubelet.kubeconfig \\
  --config=/etc/kubernetes/kubelet.config.json \\
  --hostname-override=##NODE_NAME## \\
  --pod-infra-container-image=registry.access.redhat.com/rhel7/pod-in
frastructure:latest \\
  --allow-privileged=true \\
  --alsologtostderr=true \\
  --logtostderr=false \\
  --log-dir=/var/log/kubernetes \\
  --v=2
Restart=on-failure
RestartSec=5

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

- 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；
- `--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；
- K8S approve kubelet 的 csr 请求后，在 `--cert-dir` 目录创建证书和私钥文件，然后写入 `--kubeconfig` 文件；

替换后的 unit 文件：kubelet.service

为各节点创建和分发 kubelet systemd unit 文件：

```
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"
    sed -e "s/##NODE_NAME##/${node_name}/" kubelet.service.template >
 kubelet-${node_name}.service
    scp kubelet-${node_name}.service root@${node_name}:/etc/systemd/s
ystem/kubelet.service
  done
```

Bootstrap Token Auth 和授予权限

kublet 启动时查找配置的 --kubeletconfig 文件是否存在，如果不存在则使用 --bootstrap-kubeconfig 向 kube-apiserver 发送证书签名请求 (CSR)。

kube-apiserver 收到 CSR 请求后，对其中的 Token 进行认证（事先使用 kubeadm 创建的 token），认证通过后将请求的 user 设置为 system:bootstrap:，group 设置为 system:bootstrappers，这一过程称为 Bootstrap Token Auth。

默认情况下，这个 user 和 group 没有创建 CSR 的权限，kubelet 启动失败，错误日志如下：

```
$ sudo journalctl -u kubelet -a |grep -A 2 'certificatesigningrequest
s'
May 06 06:42:36 kube-node1 kubelet[26986]: F0506 06:42:36.314378   26
986 server.go:233] failed to run Kubelet: cannot create certificate s
igning request: certificatesigningrequests.certificates.k8s.io is for
bidden: User "system:bootstrap:lemy40" cannot create certificatesigni
ngrequests.certificates.k8s.io at the cluster scope
May 06 06:42:36 kube-node1 systemd[1]: kubelet.service: Main process
exited, code=exited, status=255/n/a
May 06 06:42:36 kube-node1 systemd[1]: kubelet.service: Failed with r
esult 'exit-code'.
```

解决办法是：创建一个 clusterrolebinding，将 group system:bootstrappers 和 clusterrole system:node-bootstrapper 绑定：

```
$ kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=s
ystem:node-bootstrapper --group=system:bootstrappers
```

启动 kubelet 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /var/lib/kubelet"
    ssh root@${node_ip} "/usr/sbin/swapoff -a"
    ssh root@${node_ip} "mkdir -p /var/log/kubernetes && chown -R k8s
 /var/log/kubernetes"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable
kubelet && systemctl restart kubelet"
  done
```

- 关闭 swap 分区，否则 kubelet 会启动失败；
- 必须先创建工作和日志目录；

```
$ journalctl -u kubelet |tail
Jun 13 16:05:40 kube-node2 kubelet[22343]: I0613 16:05:40.388242   22
343 feature_gate.go:226] feature gates: &{{} map[RotateKubeletServerC
ertificate:true RotateKubeletClientCertificate:true]}
Jun 13 16:05:40 kube-node2 kubelet[22343]: I0613 16:05:40.394342   22
343 mount_linux.go:211] Detected OS with systemd
Jun 13 16:05:40 kube-node2 kubelet[22343]: W0613 16:05:40.394494   22
343 cni.go:171] Unable to update cni config: No networks found in /et
c/cni/net.d
Jun 13 16:05:40 kube-node2 kubelet[22343]: I0613 16:05:40.399508   22
343 server.go:376] Version: v1.10.4
Jun 13 16:05:40 kube-node2 kubelet[22343]: I0613 16:05:40.399583   22
343 feature_gate.go:226] feature gates: &{{} map[RotateKubeletServerC
ertificate:true RotateKubeletClientCertificate:true]}
Jun 13 16:05:40 kube-node2 kubelet[22343]: I0613 16:05:40.399736   22
343 plugins.go:89] No cloud provider specified.
Jun 13 16:05:40 kube-node2 kubelet[22343]: I0613 16:05:40.399752   22
343 server.go:492] No cloud provider specified: "" from the config fi
le: ""
Jun 13 16:05:40 kube-node2 kubelet[22343]: I0613 16:05:40.399777   22
343 bootstrap.go:58] Using bootstrap kubeconfig to generate TLS clien
t cert, key and kubeconfig file
Jun 13 16:05:40 kube-node2 kubelet[22343]: I0613 16:05:40.446068   22
343 csr.go:105] csr for this node already exists, reusing
Jun 13 16:05:40 kube-node2 kubelet[22343]: I0613 16:05:40.453761   22
343 csr.go:113] csr for this node is still valid
```

kubelet 启动后使用 --bootstrap-kubeconfig 向 kube-apiserver 发送 CSR 请求，当这个 CSR 被 approve 后，kube-controller-manager 为 kubelet 创建 TLS 客户端证书、私钥和 --kubeletconfig 文件。

注意：kube-controller-manager 需要配置 `--cluster-signing-cert-file` 和 `--cluster-signing-key-file` 参数，才会为 TLS Bootstrap 创建证书和私钥。

```
$ kubectl get csr
NAME                                                   AGE       REQU
ESTOR                 CONDITION
node-csr-QzuuQiuUfcSdp3j5W4B2UOuvQ_n9aTNHAlrLzVFiqrk   43s       syst
em:bootstrap:zkiem5   Pending
node-csr-oVbPmU-ikVknpynwu0Ckz_MvkAO_F1j0hmbcDa__sGA   27s       syst
em:bootstrap:mkus5s   Pending
node-csr-u0E1-ugxgotO_9FiGXo8DkD6a7-ew8sX2qPE6KPS2IY   13m       syst
em:bootstrap:k0s2bj   Pending

$ kubectl get nodes
No resources found.
```

- 三个 work 节点的 csr 均处于 pending 状态；

approve kubelet CSR 请求

可以手动或自动 approve CSR 请求。推荐使用自动的方式，因为从 v1.8 版本开始，可以自动轮转approve csr 后生成的证书。

手动 approve CSR 请求

查看 CSR 列表：

```
$ kubectl get csr
NAME                                                   AGE       REQU
ESTOR                 CONDITION
node-csr-QzuuQiuUfcSdp3j5W4B2UOuvQ_n9aTNHAlrLzVFiqrk   43s       syst
em:bootstrap:zkiem5   Pending
node-csr-oVbPmU-ikVknpynwu0Ckz_MvkAO_F1j0hmbcDa__sGA   27s       syst
em:bootstrap:mkus5s   Pending
node-csr-u0E1-ugxgotO_9FiGXo8DkD6a7-ew8sX2qPE6KPS2IY   13m       syst
em:bootstrap:k0s2bj   Pending
```

approve CSR：

```
$ kubectl certificate approve node-csr-QzuuQiuUfcSdp3j5W4B2UOuvQ_n9aTNHAlrLzVFiqrk
certificatesigningrequest.certificates.k8s.io "node-csr-QzuuQiuUfcSdp3j5W4B2UOuvQ_n9aTNHAlrLzVFiqrk" approved
```

查看 Approve 结果：

```
$ kubectl describe  csr node-csr-QzuuQiuUfcSdp3j5W4B2UOuvQ_n9aTNHAlrLzVFiqrk
Name:               node-csr-QzuuQiuUfcSdp3j5W4B2UOuvQ_n9aTNHAlrLzVFiqrk
Labels:             <none>
Annotations:        <none>
CreationTimestamp:  Wed, 13 Jun 2018 16:05:04 +0800
Requesting User:    system:bootstrap:zkiem5
Status:             Approved
Subject:
         Common Name:    system:node:kube-node2
         Serial Number:
         Organization:   system:nodes
Events:  <none>
```

- `Requesting User`：请求 CSR 的用户，kube-apiserver 对它进行认证和授权；
- `Subject`：请求签名的证书信息；
- 证书的 CN 是 system:node:kube-node2， Organization 是 system:nodes，kube-apiserver 的 Node 授权模式会授予该证书的相关权限；

自动 approve CSR 请求

创建三个 ClusterRoleBinding，分别用于自动 approve client、renew client、renew server 证书：

```
cat > csr-crb.yaml <<EOF
 # Approve all CSRs for the group "system:bootstrappers"
 kind: ClusterRoleBinding
 apiVersion: rbac.authorization.k8s.io/v1
 metadata:
   name: auto-approve-csrs-for-group
 subjects:
```

```
 - kind: Group
   name: system:bootstrappers
   apiGroup: rbac.authorization.k8s.io
 roleRef:
   kind: ClusterRole
   name: system:certificates.k8s.io:certificatesigningrequests:nodecl
ient
   apiGroup: rbac.authorization.k8s.io
---
 # To let a node of the group "system:nodes" renew its own credential
s
 kind: ClusterRoleBinding
 apiVersion: rbac.authorization.k8s.io/v1
 metadata:
   name: node-client-cert-renewal
 subjects:
 - kind: Group
   name: system:nodes
   apiGroup: rbac.authorization.k8s.io
 roleRef:
   kind: ClusterRole
   name: system:certificates.k8s.io:certificatesigningrequests:selfno
declient
   apiGroup: rbac.authorization.k8s.io
---
# A ClusterRole which instructs the CSR approver to approve a node re
questing a
# serving cert matching its client cert.
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: approve-node-server-renewal-csr
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/selfnodeserver"]
  verbs: ["create"]
---
 # To let a node of the group "system:nodes" renew its own server cre
dentials
 kind: ClusterRoleBinding
 apiVersion: rbac.authorization.k8s.io/v1
 metadata:
   name: node-server-cert-renewal
 subjects:
 - kind: Group
   name: system:nodes
```

```
    apiGroup: rbac.authorization.k8s.io
  roleRef:
    kind: ClusterRole
    name: approve-node-server-renewal-csr
    apiGroup: rbac.authorization.k8s.io
EOF
```

- auto-approve-csrs-for-group：自动 approve node 的第一次 CSR； 注意第一次 CSR 时，请求的 Group 为 system:bootstrappers；
- node-client-cert-renewal：自动 approve node 后续过期的 client 证书，自动生成的证书 Group 为 system:nodes;
- node-server-cert-renewal：自动 approve node 后续过期的 server 证书，自动生成的证书 Group 为 system:nodes;

生效配置：

```
$ kubectl apply -f csr-crb.yaml
```

查看 kublet 的情况

等待一段时间(1-10 分钟)，三个节点的 CSR 都被自动 approve：

```
$ kubectl get csr
NAME                                                   AGE       REQU
ESTOR                 CONDITION
csr-98h25                                              6m        syst
em:node:kube-node2    Approved,Issued
csr-lb5c9                                              7m        syst
em:node:kube-node3    Approved,Issued
csr-m2hn4                                              14m       syst
em:node:kube-node1    Approved,Issued
node-csr-7q7i0q4MF_K2TSEJj16At4CJFLlJkHIqei6nMIAaJCU   28m       syst
em:bootstrap:k0s2bj   Approved,Issued
node-csr-ND77wk2P8k2lHBtgBaObiyYw0uz1Um7g2pRvveMF-c4   35m       syst
em:bootstrap:mkus5s   Approved,Issued
node-csr-Nysmrw55nnM48NKwEJuiuCGmZoxouK4N8jiEHBtLQso   6m        syst
em:bootstrap:zkiem5   Approved,Issued
node-csr-QzuuQiuUfcSdp3j5W4B2UOuvQ_n9aTNHAlrLzVFiqrk   1h        syst
em:bootstrap:zkiem5   Approved,Issued
node-csr-oVbPmU-ikVknpynwu0Ckz_MvkAO_F1j0hmbcDa__sGA   1h        syst
em:bootstrap:mkus5s   Approved,Issued
node-csr-u0E1-ugxgotO_9FiGXo8DkD6a7-ew8sX2qPE6KPS2IY   1h        syst
em:bootstrap:k0s2bj   Approved,Issued
```

所有节点均 ready：

```
$ kubectl get nodes
NAME         STATUS    ROLES     AGE       VERSION
kube-node1   Ready     <none>    18m       v1.10.4
kube-node2   Ready     <none>    10m       v1.10.4
kube-node3   Ready     <none>    11m       v1.10.4
```

kube-controller-manager 为各 node 生成了 kubeconfig 文件和公私钥：

```
$ ls -l /etc/kubernetes/kubelet.kubeconfig
-rw------- 1 root root 2293 Jun 13 17:07 /etc/kubernetes/kubelet.kube
config

$ ls -l /etc/kubernetes/cert/|grep kubelet
-rw-r--r-- 1 root root 1046 Jun 13 17:07 kubelet-client.crt
-rw------- 1 root root  227 Jun 13 17:07 kubelet-client.key
-rw------- 1 root root 1334 Jun 13 17:07 kubelet-server-2018-06-13-17
-07-45.pem
lrwxrwxrwx 1 root root   58 Jun 13 17:07 kubelet-server-current.pem ->
 /etc/kubernetes/cert/kubelet-server-2018-06-13-17-07-45.pem
```

- kubelet-server 证书会周期轮转；

kubelet 提供的 API 接口

kublet 启动后监听多个端口，用于接收 kube-apiserver 或其它组件发送的请求：

```
$ sudo netstat -lnpt|grep kubelet
tcp        0      0 172.27.129.111:4194     0.0.0.0:*               L
ISTEN      2490/kubelet
tcp        0      0 127.0.0.1:10248         0.0.0.0:*               L
ISTEN      2490/kubelet
tcp        0      0 172.27.129.111:10250    0.0.0.0:*               L
ISTEN      2490/kubelet
```

- 4194: cadvisor http 服务；
- 10248: healthz http 服务；
- 10250: https API 服务；注意：未开启只读端口 10255；

例如执行 `kubectl ec -it nginx-ds-5rmws -- sh` 命令时，kube-apiserver 会向 kubelet 发送如下请求：

```
POST /exec/default/nginx-ds-5rmws/my-nginx?command=sh&input=1&output=
1&tty=1
```

kubelet 接收 10250 端口的 https 请求：

- /pods、/runningpods
- /metrics、/metrics/cadvisor、/metrics/probes

- /spec
- /stats、/stats/container
- /logs
- /run/、"/exec/", "/attach/", "/portForward/", "/containerLogs/" 等管理；

详情参考：https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/server/server.go#L434:3

由于关闭了匿名认证，同时开启了 webhook 授权，所有访问 10250 端口 https API 的请求都需要被认证和授权。

预定义的 ClusterRole system:kubelet-api-admin 授予访问 kubelet 所有 API 的权限：

```
$ kubectl describe clusterrole system:kubelet-api-admin
Name:         system:kubelet-api-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources      Non-Resource URLs  Resource Names  Verbs
  ---------      -----------------  --------------  -----
  nodes          []                 []              [get list watch p
roxy]
  nodes/log      []                 []              [*]
  nodes/metrics  []                 []              [*]
  nodes/proxy    []                 []              [*]
  nodes/spec     []                 []              [*]
  nodes/stats    []                 []              [*]
```

kublet api 认证和授权

kublet 配置了如下认证参数：

- authentication.anonymous.enabled：设置为 false，不允许匿名访问 10250 端口；
- authentication.x509.clientCAFile：指定签名客户端证书的 CA 证书，开启 HTTPs 证书认证；
- authentication.webhook.enabled=true：开启 HTTPs bearer token 认证；

同时配置了如下授权参数：

- authroization.mode=Webhook：开启 RBAC 授权；

kubelet 收到请求后，使用 clientCAFile 对证书签名进行认证，或者查询 bearer token 是否有效。如果两者都没通过，则拒绝请求，提示 Unauthorized：

```
$ curl -s --cacert /etc/kubernetes/cert/ca.pem https://172.27.129.111
:10250/metrics
Unauthorized

$ curl -s --cacert /etc/kubernetes/cert/ca.pem -H "Authorization: Bea
rer 123456" https://172.27.129.111:10250/metrics
Unauthorized
```

通过认证后，kubelet 使用 SubjectAccessReview API 向 kube-apiserver 发送请求，查询证书或 token 对应的 user、group 是否有操作资源的权限(RBAC)；

证书认证和授权：

```
$ # 权限不足的证书；
$ curl -s --cacert /etc/kubernetes/cert/ca.pem --cert /etc/kubernetes
/cert/kube-controller-manager.pem --key /etc/kubernetes/cert/kube-con
troller-manager-key.pem https://172.27.129.111:10250/metrics
Forbidden (user=system:kube-controller-manager, verb=get, resource=no
des, subresource=metrics)

$ # 使用部署 kubectl 命令行工具时创建的、具有最高权限的 admin 证书；
$ curl -s --cacert /etc/kubernetes/cert/ca.pem --cert ./admin.pem --k
ey ./admin-key.pem https://172.27.129.111:10250/metrics|head
# HELP apiserver_client_certificate_expiration_seconds Distribution o
f the remaining lifetime on the certificate used to authenticate a re
quest.
# TYPE apiserver_client_certificate_expiration_seconds histogram
apiserver_client_certificate_expiration_seconds_bucket{le="0"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="21600"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="43200"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="86400"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="172800"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="345600"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="604800"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="2.592e+06"}
 0
```

- `--cacert`、`--cert`、`--key` 的参数值必须是文件路径，如上面的 `./admin.pem` 不能省略 `./`，否则返回 `401 Unauthorized`；

bear token 认证和授权：

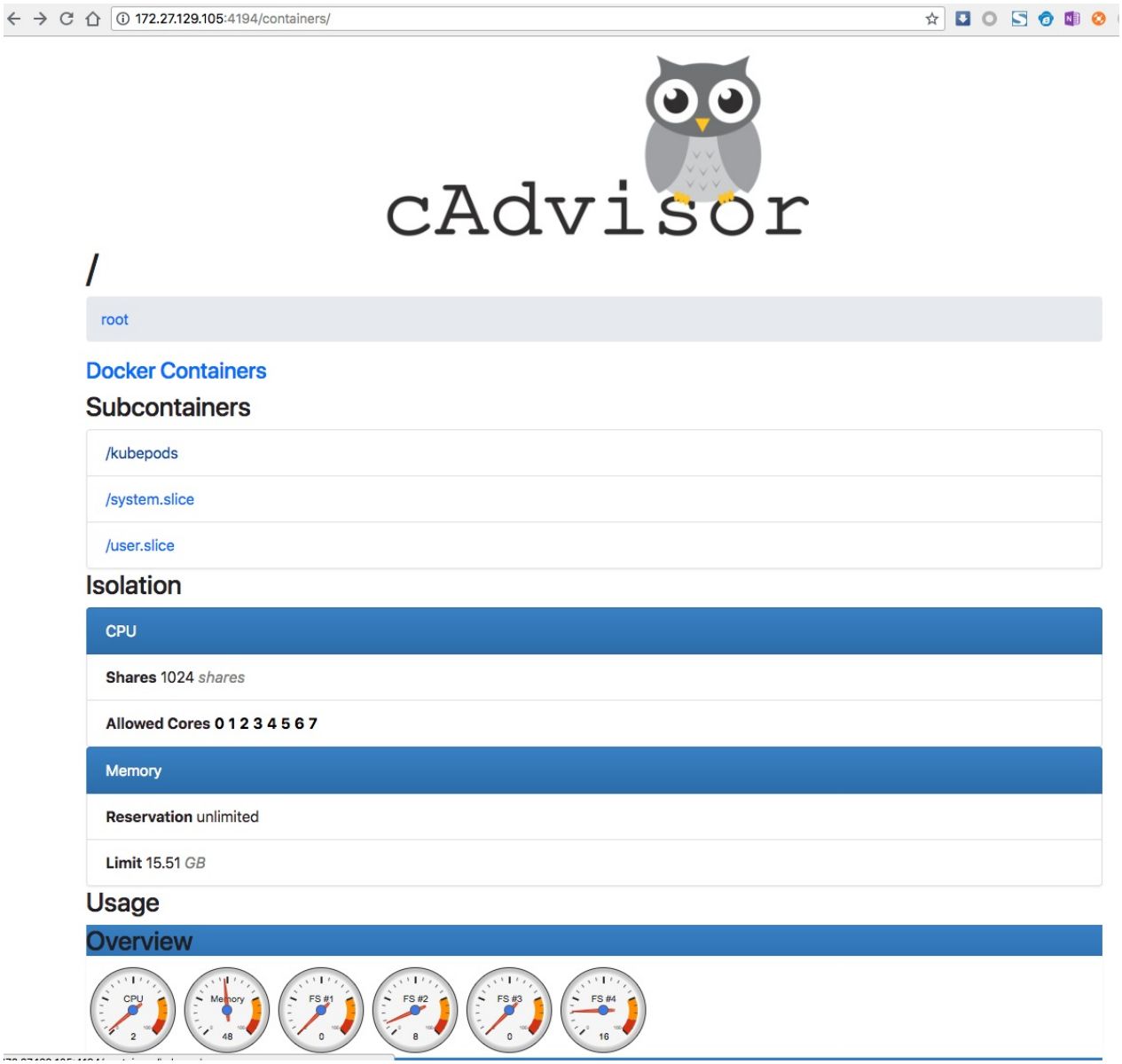
创建一个 ServiceAccount，将它和 ClusterRole system:kubelet-api-admin 绑定，从而具有调用 kubelet API 的权限：

```
kubectl create sa kubelet-api-test
kubectl create clusterrolebinding kubelet-api-test --clusterrole=syst
em:kubelet-api-admin --serviceaccount=default:kubelet-api-test
SECRET=$(kubectl get secrets | grep kubelet-api-test | awk '{print $1
}')
TOKEN=$(kubectl describe secret ${SECRET} | grep -E '^token' | awk '{
print $2}')
echo ${TOKEN}

$ curl -s --cacert /etc/kubernetes/cert/ca.pem -H "Authorization: Bea
rer ${TOKEN}" https://172.27.129.111:10250/metrics|head
# HELP apiserver_client_certificate_expiration_seconds Distribution o
f the remaining lifetime on the certificate used to authenticate a re
quest.
# TYPE apiserver_client_certificate_expiration_seconds histogram
apiserver_client_certificate_expiration_seconds_bucket{le="0"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="21600"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="43200"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="86400"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="172800"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="345600"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="604800"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="2.592e+06"}
 0
```

cadvisor 和 metrics

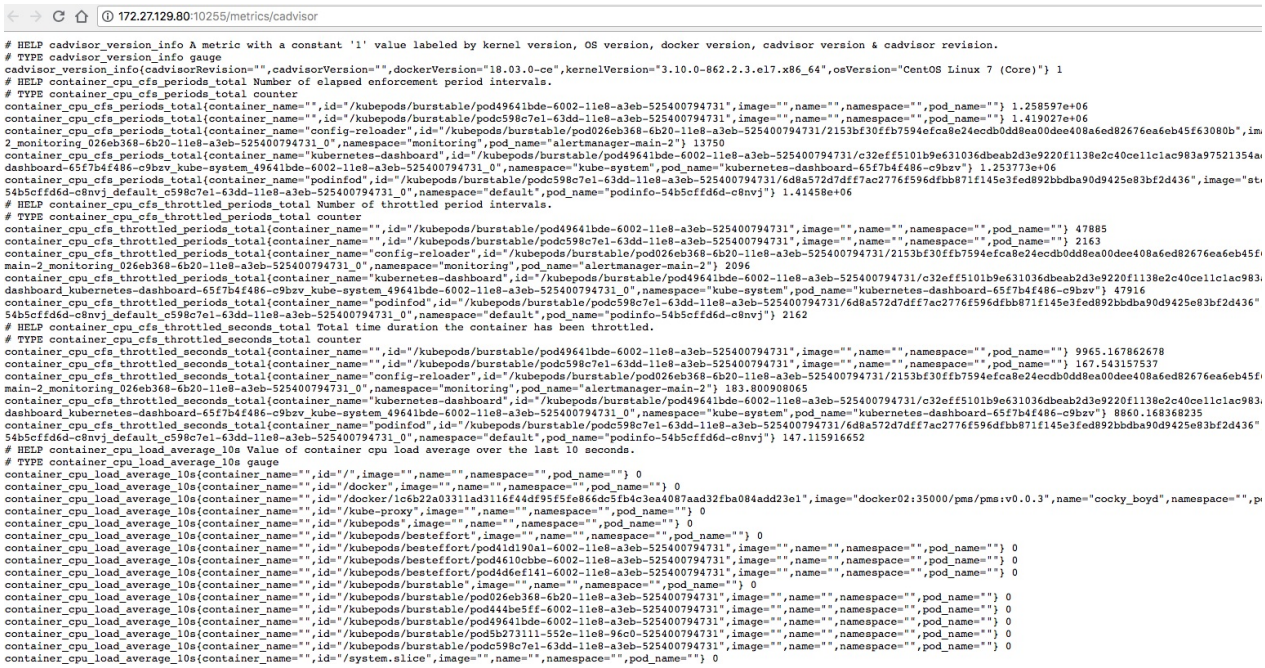
cadvisor 统计所在节点各容器的资源(CPU、内存、磁盘、网卡)使用情况，分别在自己的 http web 页面(4194 端口)和 10250 以 promehteus metrics 的形式输出。

浏览器访问 http://172.27.129.105:4194/containers/ 可以查看到 cadvisor 的监控页面：



图片 - *cadvisor-home*

浏览器访问 https://172.27.129.80:10250/metrics 和 https://172.27.129.80:10250/metrics/cadvisor 分别返回 kublet 和 cadvisor 的 metrics。



图片 - *cadvisor-metrics*

注意：

- kublet.config.json 设置 authentication.anonymous.enabled 为 false，不允许匿名证书访问 10250 的 https 服务；
- 参考A.浏览器访问kube-apiserver安全端口.md，创建和导入相关证书，然后访问上面的 10250 端口；

获取 **kublet** 的配置

从 kube-apiserver 获取各 node 的配置：

```
$ source /opt/k8s/bin/environment.sh
$ # 使用部署 kubectl 命令行工具时创建的、具有最高权限的 admin 证书；
$ curl -sSL --cacert /etc/kubernetes/cert/ca.pem --cert ./admin.pem --key ./admin-key.pem ${KUBE_APISERVER}/api/v1/nodes/kube-node1/proxy/configz | jq \
  '.kubeletconfig|.kind="KubeletConfiguration"|.apiVersion="kubelet.config.k8s.io/v1beta1"'
{
  "syncFrequency": "1m0s",
  "fileCheckFrequency": "20s",
  "httpCheckFrequency": "20s",
  "address": "172.27.129.80",
```

```
    "port": 10250,
    "readOnlyPort": 10255,
    "authentication": {
      "x509": {},
      "webhook": {
        "enabled": false,
        "cacheTTL": "2m0s"
      },
      "anonymous": {
        "enabled": true
      }
    },
    "authorization": {
      "mode": "AlwaysAllow",
      "webhook": {
        "cacheAuthorizedTTL": "5m0s",
        "cacheUnauthorizedTTL": "30s"
      }
    },
    "registryPullQPS": 5,
    "registryBurst": 10,
    "eventRecordQPS": 5,
    "eventBurst": 10,
    "enableDebuggingHandlers": true,
    "healthzPort": 10248,
    "healthzBindAddress": "127.0.0.1",
    "oomScoreAdj": -999,
    "clusterDomain": "cluster.local.",
    "clusterDNS": [
      "10.254.0.2"
    ],
    "streamingConnectionIdleTimeout": "4h0m0s",
    "nodeStatusUpdateFrequency": "10s",
    "imageMinimumGCAge": "2m0s",
    "imageGCHighThresholdPercent": 85,
    "imageGCLowThresholdPercent": 80,
    "volumeStatsAggPeriod": "1m0s",
    "cgroupsPerQOS": true,
    "cgroupDriver": "cgroupfs",
    "cpuManagerPolicy": "none",
    "cpuManagerReconcilePeriod": "10s",
    "runtimeRequestTimeout": "2m0s",
    "hairpinMode": "promiscuous-bridge",
    "maxPods": 110,
    "podPidsLimit": -1,
    "resolvConf": "/etc/resolv.conf",
```

```
    "cpuCFSQuota": true,
    "maxOpenFiles": 1000000,
    "contentType": "application/vnd.kubernetes.protobuf",
    "kubeAPIQPS": 5,
    "kubeAPIBurst": 10,
    "serializeImagePulls": false,
    "evictionHard": {
      "imagefs.available": "15%",
      "memory.available": "100Mi",
      "nodefs.available": "10%",
      "nodefs.inodesFree": "5%"
    },
    "evictionPressureTransitionPeriod": "5m0s",
    "enableControllerAttachDetach": true,
    "makeIPTablesUtilChains": true,
    "iptablesMasqueradeBit": 14,
    "iptablesDropBit": 15,
    "featureGates": {
      "RotateKubeletClientCertificate": true,
      "RotateKubeletServerCertificate": true
    },
    "failSwapOn": true,
    "containerLogMaxSize": "10Mi",
    "containerLogMaxFiles": 5,
    "enforceNodeAllocatable": [
      "pods"
    ],
    "kind": "KubeletConfiguration",
    "apiVersion": "kubelet.config.k8s.io/v1beta1"
}
```

或者参考代码中的注释：https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/apis/kubeletconfig/v1beta1/types.go

参考

1. kubelet 认证和授权：https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet-authentication-authorization/

zhangjun　　最后更新：2018-10-18 04:04:02

tags: worker, kube-proxy

07-3.部署 kube-proxy 组件

kube-proxy 运行在所有 worker 节点上，，它监听 apiserver 中 service 和 Endpoint 的变化情况，创建路由规则来进行服务负载均衡。

本文档讲解部署 kube-proxy 的部署，使用 ipvs 模式。

下载和分发 kube-proxy 二进制文件

参考 06-0.部署master节点.md

安装依赖包

各节点需要安装 `ipvsadm` 和 `ipset` 命令，加载 `ip_vs` 内核模块。

参考 07-0.部署worker节点.md

创建 kube-proxy 证书

创建证书签名请求：

```
cat > kube-proxy-csr.json <<EOF
{
  "CN": "system:kube-proxy",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "4Paradigm"
    }
  ]
}
EOF
```

- CN：指定该证书的 User 为 `system:kube-proxy` ；
- 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；
- 该证书只会被 kube-proxy 当做 client 证书使用，所以 hosts 字段为空；

生成证书和私钥：

```
cfssl gencert -ca=/etc/kubernetes/cert/ca.pem \
  -ca-key=/etc/kubernetes/cert/ca-key.pem \
  -config=/etc/kubernetes/cert/ca-config.json \
  -profile=kubernetes  kube-proxy-csr.json | cfssljson -bare kube-pro
xy
```

创建和分发 **kubeconfig** 文件

```
source /opt/k8s/bin/environment.sh
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/cert/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-proxy.kubeconfig

kubectl config set-credentials kube-proxy \
  --client-certificate=kube-proxy.pem \
  --client-key=kube-proxy-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-proxy.kubeconfig

kubectl config set-context default \
  --cluster=kubernetes \
  --user=kube-proxy \
  --kubeconfig=kube-proxy.kubeconfig

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

- `--embed-certs=true`：将 ca.pem 和 admin.pem 证书内容嵌入到生成的 kubectl-proxy.kubeconfig 文件中(不加时，写入的是证书文件路径)；

分发 kubeconfig 文件：

```
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"
    scp kube-proxy.kubeconfig k8s@${node_name}:/etc/kubernetes/
  done
```

创建 kube-proxy 配置文件

从 v1.10 开始，kube-proxy 部分参数可以配置文件中配置。可以使用 `--write-config-to` 选项生成该配置文件，或者参考 kubeproxyconfig 的类型定义源文件：https://github.com/kubernetes/kubernetes/blob/master/pkg/proxy/apis/kubeproxyconfig/types.go

创建 kube-proxy config 文件模板：

```
cat >kube-proxy.config.yaml.template <<EOF
apiVersion: kubeproxy.config.k8s.io/v1alpha1
bindAddress: ##NODE_IP##
clientConnection:
  kubeconfig: /etc/kubernetes/kube-proxy.kubeconfig
clusterCIDR: ${CLUSTER_CIDR}
healthzBindAddress: ##NODE_IP##:10256
hostnameOverride: ##NODE_NAME##
kind: KubeProxyConfiguration
metricsBindAddress: ##NODE_IP##:10249
mode: "ipvs"
EOF
```

- `bindAddress`：监听地址；
- `clientConnection.kubeconfig`：连接 apiserver 的 kubeconfig 文件；
- `clusterCIDR`：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；
- `hostnameOverride`：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 ipvs 规则；
- `mode`：使用 ipvs 模式；

为各节点创建和分发 kube-proxy 配置文件：

```
source /opt/k8s/bin/environment.sh
for (( i=0; i < 3; i++ ))
  do
    echo ">>> ${NODE_NAMES[i]}"
    sed -e "s/##NODE_NAME##/${NODE_NAMES[i]}/" -e "s/##NODE_IP##/${NO
DE_IPS[i]}/" kube-proxy.config.yaml.template > kube-proxy-${NODE_NAME
S[i]}.config.yaml
    scp kube-proxy-${NODE_NAMES[i]}.config.yaml root@${NODE_NAMES[i]}
:/etc/kubernetes/kube-proxy.config.yaml
  done
```

替换后的 kube-proxy.config.yaml 文件：kube-proxy.config.yaml

创建和分发 kube-proxy systemd unit 文件

```
source /opt/k8s/bin/environment.sh
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=/opt/k8s/bin/kube-proxy \\
  --config=/etc/kubernetes/kube-proxy.config.yaml \\
  --alsologtostderr=true \\
  --logtostderr=false \\
  --log-dir=/var/log/kubernetes \\
  --v=2
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

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

替换后的 unit 文件：kube-proxy.service

分发 kube-proxy systemd unit 文件：

```
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"
    scp kube-proxy.service root@${node_name}:/etc/systemd/system/
  done
```

启动 kube-proxy 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /var/lib/kube-proxy"
    ssh root@${node_ip} "mkdir -p /var/log/kubernetes && chown -R k8s
 /var/log/kubernetes"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable
kube-proxy && systemctl restart kube-proxy"
  done
```

- 必须先创建工作和日志目录；

检查启动结果

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh k8s@${node_ip} "systemctl status kube-proxy|grep Active"
  done
```

确保状态为 `active (running)`，否则查看日志，确认原因：

```
journalctl -u kube-proxy
```

查看监听端口和 metrics

```
[k8s@kube-node1 ~]$ sudo netstat -lnpt|grep kube-prox
tcp        0      0 172.27.129.105:10249    0.0.0.0:*               L
ISTEN      16847/kube-proxy
tcp        0      0 172.27.129.105:10256    0.0.0.0:*               L
ISTEN      16847/kube-proxy
```

- 10249：http prometheus metrics port;
- 10256：http healthz port;

查看 ipvs 路由规则

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "/usr/sbin/ipvsadm -ln"
  done
```

预期输出：

```
>>> 172.27.129.105
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.254.0.1:443 rr persistent 10800
  -> 172.27.129.105:6443          Masq    1      0          0
>>> 172.27.129.111
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.254.0.1:443 rr persistent 10800
  -> 172.27.129.105:6443          Masq    1      0          0
>>> 172.27.129.112
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.254.0.1:443 rr persistent 10800
  -> 172.27.129.105:6443          Masq    1      0          0
```

可见将所有到 kubernetes cluster ip 443 端口的请求都转发到 kube-apiserver 的 6443 端口；

zhangjun　　最后更新： 2018-10-18 04:04:02

tags: verify

08.验证集群功能

本文档使用 daemonset 验证 master 和 worker 节点是否工作正常。

检查节点状态

```
$ kubectl get nodes
NAME         STATUS    ROLES     AGE       VERSION
kube-node1   Ready     <none>    3h        v1.10.4
kube-node2   Ready     <none>    3h        v1.10.4
kube-node3   Ready     <none>    3h        v1.10.4
```

都为 Ready 时正常。

创建测试文件

```
$ cat > nginx-ds.yml <<EOF
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
EOF
```

执行定义文件

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

检查各 Node 上的 Pod IP 连通性

```
$ kubectl get pods  -o wide|grep nginx-ds
nginx-ds-dbn97   1/1       Running   0          2m        172.30.29.2
   kube-node2
nginx-ds-rk777   1/1       Running   0          2m        172.30.81.2
   kube-node1
nginx-ds-tr9g5   1/1       Running   0          2m        172.30.39.2
   kube-node3
```

可见，nginx-ds 的 Pod IP 分别是 `172.30.39.2`、`172.30.81.2`、`172.30.29.2`，在所有 Node 上分别 ping 这三个 IP，看是否连通：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "ping -c 1 172.30.39.2"
    ssh ${node_ip} "ping -c 1 172.30.81.2"
    ssh ${node_ip} "ping -c 1 172.30.29.2"
  done
```

检查服务 IP 和端口可达性

```
$ kubectl get svc |grep nginx-ds
nginx-ds     NodePort    10.254.254.228   <none>        80:8900/TCP
 4m
```

可见：

- Service Cluster IP：10.254.254.228
- 服务端口：80
- NodePort 端口：8900

在所有 Node 上 curl Service IP：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "curl 10.254.254.228"
  done
```

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：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "curl ${node_ip}:8900"
  done
```

预期输出 nginx 欢迎页面内容。

zhangjun　　最后更新： 2018-10-18 04:04:02

09-0.部署集群插件

插件是集群的附件组件，丰富和完善了集群的功能。

- 09-1.coredns
- 09-2.Dashboard
- 09-3.Heapster (influxdb、grafana)
- 09-4.Metrics Server
- 09-5.EFK (elasticsearch、fluentd、kibana)

zhangjun 最后更新： 2018-10-18 04:04:02

tags: addons, dns, coredns

09-1.部署 coredns 插件

修改配置文件

将下载的 kubernetes-server-linux-amd64.tar.gz 解压后，再解压其中的 kubernetes-src.tar.gz 文件。

coredns 对应的目录是：`cluster/addons/dns`。

```
$ pwd
/opt/k8s/kubernetes/cluster/addons/dns

$ cp coredns.yaml.base coredns.yaml
$ diff coredns.yaml.base coredns.yaml
61c61
<         kubernetes __PILLAR__DNS__DOMAIN__ in-addr.arpa ip6.arpa {
---
>         kubernetes cluster.local. in-addr.arpa ip6.arpa {
153c153
<   clusterIP: __PILLAR__DNS__SERVER__
---
>   clusterIP: 10.254.0.2
```

创建 coredns

```
$ kubectl create -f coredns.yaml
```

检查 coredns 功能

```
$ kubectl get all -n kube-system
NAME                            READY     STATUS    RESTARTS   AGE
pod/coredns-77c989547b-6l6jr    1/1       Running   0          3m
pod/coredns-77c989547b-d9lts    1/1       Running   0          3m

NAME              TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
    AGE
service/coredns   ClusterIP   10.254.0.2   <none>        53/UDP,53/TC
P   3m

NAME                     DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
  AGE
deployment.apps/coredns   2         2         2            2
  3m

NAME                                  DESIRED   CURRENT   READY     AG
E
replicaset.apps/coredns-77c989547b    2         2         2         3m
```

新建一个 Deployment

```
$ cat > my-nginx.yaml <<EOF
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
EOF
$ kubectl create -f my-nginx.yaml
```

Export 该 Deployment, 生成 `my-nginx` 服务：

```
$ kubectl expose deploy my-nginx
service "my-nginx" exposed

$ kubectl get services --all-namespaces |grep my-nginx
default       my-nginx     ClusterIP   10.254.242.255   <none>
 80/TCP          9s
```

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 `10.254.242.255`

```
$ cat > pod-nginx.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
EOF

$ 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. 4
pd.io
options ndots:5

root@nginx:/#  ping my-nginx
PING my-nginx.default.svc.cluster.local (10.254.242.255): 48 data byt
es
56 bytes from 10.254.242.255: icmp_seq=0 ttl=64 time=0.115 ms
^C--- my-nginx.default.svc.cluster.local ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.115/0.115/0.115/0.000 ms


root@nginx:/# ping my-nginx
PING my-nginx.default.svc.cluster.local (10.254.63.136): 48 data byte
```

```
s
^C--- my-nginx.default.svc.cluster.local ping statistics ---
4 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
56 bytes from 10.254.0.1: icmp_seq=0 ttl=64 time=0.097 ms
56 bytes from 10.254.0.1: icmp_seq=1 ttl=64 time=0.123 ms
^C--- kubernetes.default.svc.cluster.local ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.097/0.110/0.123/0.000 ms

root@nginx:/# ping coredns.kube-system.svc.cluster.local
PING coredns.kube-system.svc.cluster.local (10.254.0.2): 48 data byte
s
56 bytes from 10.254.0.2: icmp_seq=0 ttl=64 time=0.129 ms
^C--- coredns.kube-system.svc.cluster.local ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.129/0.129/0.129/0.000 ms
```

参考

https://community.infoblox.com/t5/Community-Blog/CoreDNS-for-Kubernetes-Service-Discovery/ba-p/8187 https://coredns.io/2017/03/01/coredns-for-kubernetes-service-discovery-take-2/ https://www.cnblogs.com/boshen-hzb/p/7511432.html https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/dns

zhangjun 最后更新： 2018-10-18 04:04:02

tags: addons, dashboard

09-2.部署 dashboard 插件

修改配置文件

将下载的 kubernetes-server-linux-amd64.tar.gz 解压后，再解压其中的 kubernetes-src.tar.gz 文件。

dashboard 对应的目录是：`cluster/addons/dashboard`。

```
$ pwd
/opt/k8s/kubernetes/cluster/addons/dashboard

$ cp dashboard-controller.yaml{,.orig}

$ diff dashboard-controller.yaml{,.orig}
33c33
<         image: siriuszg/kubernetes-dashboard-amd64:v1.8.3
---
>         image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.8.3

$ cp dashboard-service.yaml{,.orig}

$ 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；

执行所有定义文件

```
$ ls *.yaml
dashboard-configmap.yaml  dashboard-controller.yaml  dashboard-rbac.y
aml  dashboard-secret.yaml  dashboard-service.yaml

$ kubectl create -f  .
```

查看分配的 NodePort

```
$ kubectl get deployment kubernetes-dashboard  -n kube-system
NAME                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   A
GE
kubernetes-dashboard   1         1         1            1           2
m

$ kubectl --namespace kube-system get pods -o wide
NAME                                    READY     STATUS    RESTARTS
  AGE       IP            NODE
coredns-77c989547b-6l6jr                1/1       Running   0
  58m       172.30.39.3   kube-node3
coredns-77c989547b-d9lts                1/1       Running   0
  58m       172.30.81.3   kube-node1
kubernetes-dashboard-65f7b4f486-wgc6j   1/1       Running   0
  2m        172.30.81.5   kube-node1

$ kubectl get services kubernetes-dashboard -n kube-system
NAME                   TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S
)        AGE
kubernetes-dashboard   NodePort   10.254.96.204   <none>        443:8
607/TCP   2m
```

- NodePort 8607 映射到 dashboard pod 443 端口；

dashboard 的 --authentication-mode 支持 token、basic，默认为 token。如果使用 basic，则 kube-apiserver 必须配置 '--authorization-mode=ABAC' 和 '--basic-auth-file' 参数。

查看 dashboard 支持的命令行参数

```
$ kubectl exec --namespace kube-system -it kubernetes-dashboard-65f7b
4f486-wgc6j  -- /dashboard --help
2018/06/13 15:17:44 Starting overwatch
Usage of /dashboard:
      --alsologtostderr                  log to standard error as we
ll as files
      --apiserver-host string            The address of the Kubernet
es Apiserver to connect to in the format of protocol://address:port,
e.g., http://localhost:8080. If not specified, the assumption is that
```

```
 the binary runs inside a Kubernetes cluster and local discovery is a
ttempted.
      --authentication-mode stringSlice   Enables authentication opti
ons that will be reflected on login screen. Supported values: token,
basic. Default: token.Note that basic option should only be used if a
piserver has '--authorization-mode=ABAC' and '--basic-auth-file' flag
s set. (default [token])
      --auto-generate-certificates        When set to true, Dashboard
 will automatically generate certificates used to serve HTTPS. Defaul
t: false.
      --bind-address ip                   The IP address on which to
serve the --secure-port (set to 0.0.0.0 for all interfaces). (default
 0.0.0.0)
      --default-cert-dir string           Directory path containing '
--tls-cert-file' and '--tls-key-file' files. Used also when auto-gene
rating certificates flag is set. (default "/certs")
      --disable-settings-authorizer       When enabled, Dashboard set
tings page will not require user to be logged in and authorized to ac
cess settings page.
      --enable-insecure-login             When enabled, Dashboard log
in view will also be shown when Dashboard is not served over HTTPS. D
efault: false.
      --heapster-host string              The address of the Heapster
 Apiserver to connect to in the format of protocol://address:port, e.
g., http://localhost:8082. If not specified, the assumption is that t
he binary runs inside a Kubernetes cluster and service proxy will be
used.
      --insecure-bind-address ip          The IP address on which to
serve the --port (set to 0.0.0.0 for all interfaces). (default 127.0.
0.1)
      --insecure-port int                 The port to listen to for i
ncoming HTTP requests. (default 9090)
      --kubeconfig string                 Path to kubeconfig file wit
h authorization and master location information.
      --log_backtrace_at traceLocation    when logging hits line file
:N, emit a stack trace (default :0)
      --log_dir string                    If non-empty, write log fil
es in this directory
      --logtostderr                       log to standard error inste
ad of files
      --metric-client-check-period int    Time in seconds that define
s how often configured metric client health check should be run. Defa
ult: 30 seconds. (default 30)
      --port int                          The secure port to listen t
o for incoming HTTPS requests. (default 8443)
      --stderrthreshold severity          logs at or above this thres
```

```
hold go to stderr (default 2)
      --system-banner string              When non-empty displays mes
sage to Dashboard users. Accepts simple HTML tags. Default: ''.
      --system-banner-severity string     Severity of system banner.
Should be one of 'INFO|WARNING|ERROR'. Default: 'INFO'. (default "INF
O")
      --tls-cert-file string              File containing the default
 x509 Certificate for HTTPS.
      --tls-key-file string               File containing the default
 x509 private key matching --tls-cert-file.
      --token-ttl int                     Expiration time (in seconds)
 of JWE tokens generated by dashboard. Default: 15 min. 0 - never exp
ires (default 900)
  -v, --v Level                           log level for V logs
      --vmodule moduleSpec                comma-separated list of pat
tern=N settings for file-filtered logging
command terminated with exit code 2
$
```

访问 dashboard

为了集群安全，从 1.7 开始，dashboard 只允许通过 https 访问，如果使用 kube proxy 则必须监听 localhost 或 127.0.0.1，对于 NodePort 没有这个限制，但是仅建议在开发环境中使用。

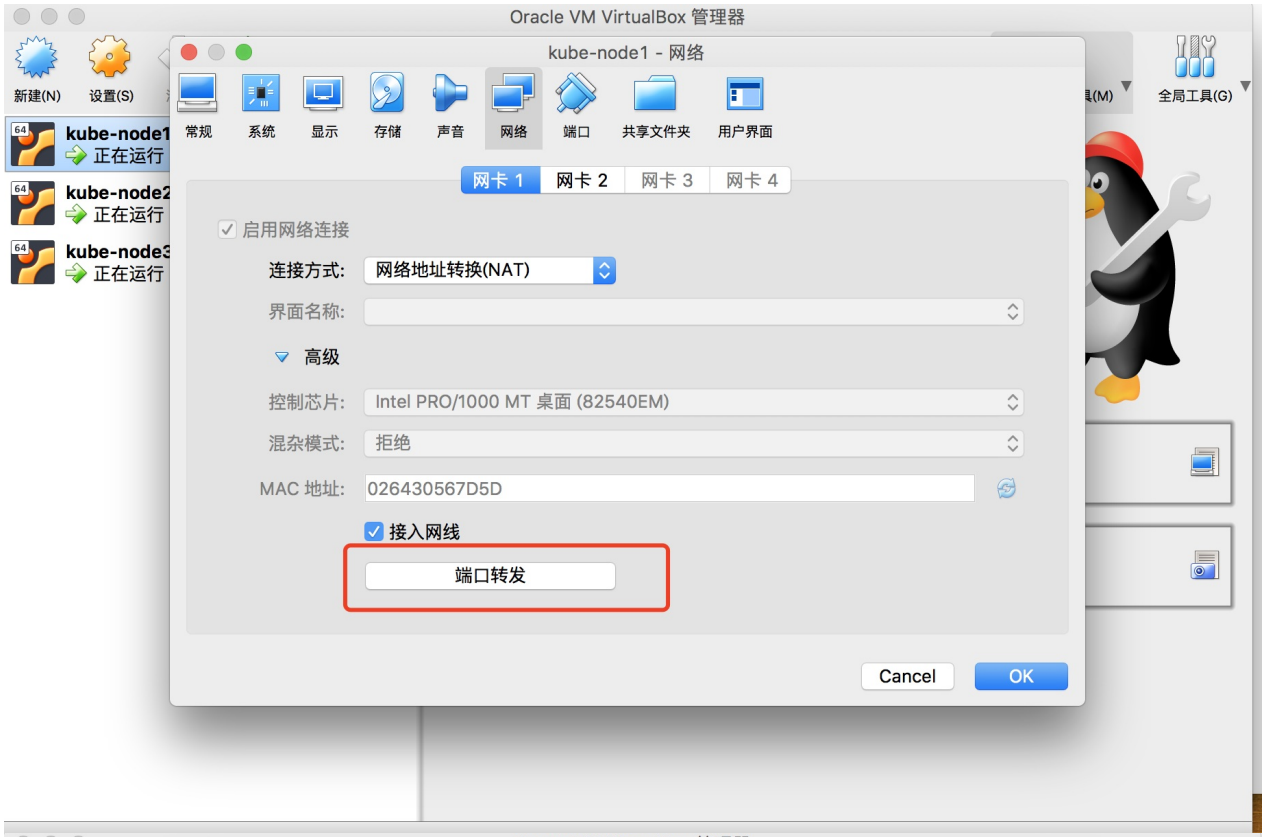
对于不满足这些条件的登录访问，在登录成功后浏览器不跳转，始终停在登录界面。

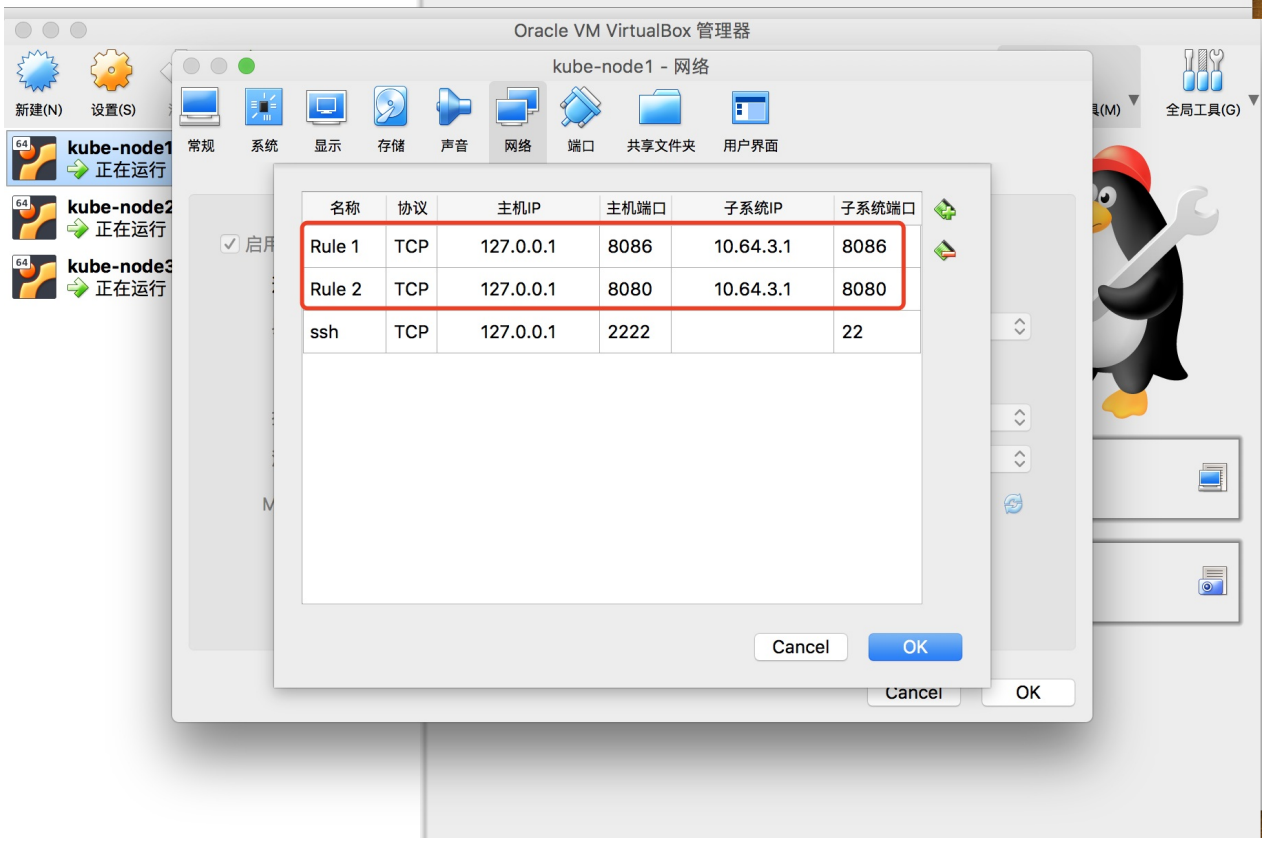
参考： https://github.com/kubernetes/dashboard/wiki/Accessing-Dashboard---1.7.X-and-above https://github.com/kubernetes/dashboard/issues/2540

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

如果使用了 VirtualBox，需要启用 VirtualBox 的 ForworadPort 功能将虚机监听的端口和 Host 的本地端口绑定。

可以在 Vagrant 的配置中指定这些端口转发规则，对于正在运行的虚机，也可以通过 VirtualBox 的界面进行配置：





通过 kubectl proxy 访问 dashboard

启动代理：

```
$ kubectl proxy --address='localhost' --port=8086 --accept-hosts='^*$'

Starting to serve on 127.0.0.1:8086
```

- --address 必须为 localhost 或 127.0.0.1；
- 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://127.0.0.1:8086/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy`

通过 kube-apiserver 访问 dashboard

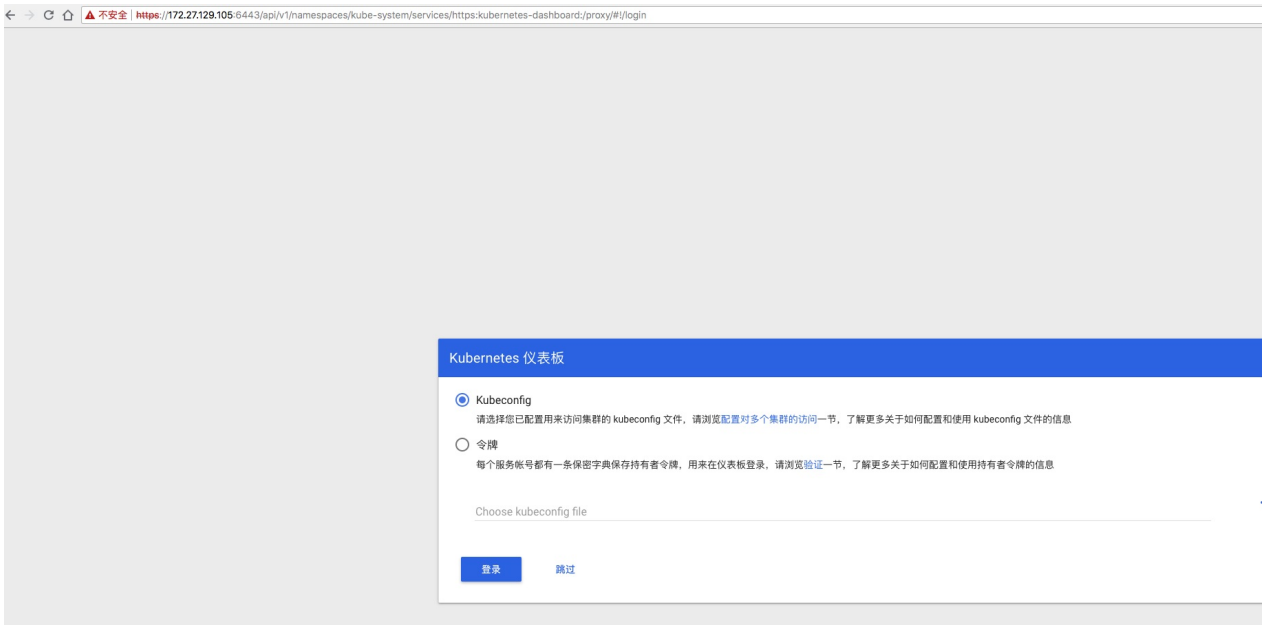
获取集群服务地址列表：

```
$ kubectl cluster-info
Kubernetes master is running at https://172.27.129.105:6443
CoreDNS is running at https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/coredns:dns/proxy
kubernetes-dashboard is running at https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy

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

必须通过 kube-apiserver 的安全端口(https)访问 dashbaord，访问时浏览器需要使用自定义证书，否则会被 kube-apiserver 拒绝访问。

创建和导入自定义证书的步骤，参考：A.浏览器访问kube-apiserver安全端口

浏览器访问 URL： `https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/` 对于 virtuabox 做了端口映射： `http://127.0.0.1:6443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/`



图片 - *dashboard-login*

创建登录 Dashboard 的 token 和 kubeconfig 配置文件

上面提到，Dashboard 默认只支持 token 认证，所以如果使用 KubeConfig 文件，需要在该文件中指定 token，不支持使用 client 证书认证。

创建登录 token

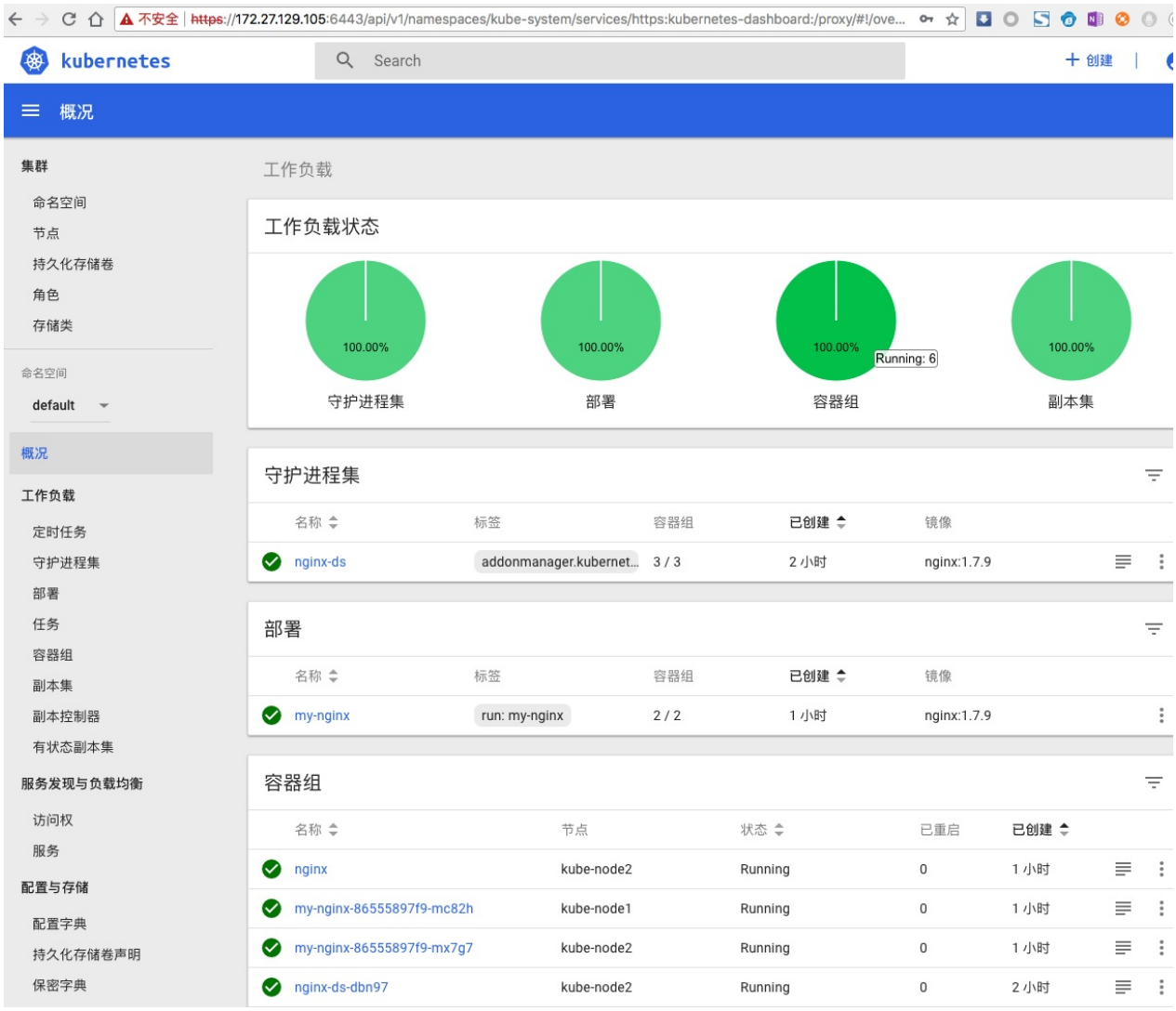
```
kubectl create sa dashboard-admin -n kube-system
kubectl create clusterrolebinding dashboard-admin --clusterrole=clust
er-admin --serviceaccount=kube-system:dashboard-admin
ADMIN_SECRET=$(kubectl get secrets -n kube-system | grep dashboard-ad
min | awk '{print $1}')
DASHBOARD_LOGIN_TOKEN=$(kubectl describe secret -n kube-system ${ADMI
N_SECRET} | grep -E '^token' | awk '{print $2}')
echo ${DASHBOARD_LOGIN_TOKEN}
```

使用输出的 token 登录 Dashboard。

创建使用 token 的 KubeConfig 文件

```
source /opt/k8s/bin/environment.sh
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/cert/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=dashboard.kubeconfig

# 设置客户端认证参数，使用上面创建的 Token
kubectl config set-credentials dashboard_user \
  --token=${DASHBOARD_LOGIN_TOKEN} \
  --kubeconfig=dashboard.kubeconfig

# 设置上下文参数
kubectl config set-context default \
  --cluster=kubernetes \
  --user=dashboard_user \
  --kubeconfig=dashboard.kubeconfig

# 设置默认上下文
kubectl config use-context default --kubeconfig=dashboard.kubeconfig
```

用生成的 dashboard.kubeconfig 登录 Dashboard。



图片 - *images/dashboard.png*

由于缺少 Heapster 插件，当前 dashboard 不能展示 Pod、Nodes 的 CPU、内存等统计数据和图表；

参考

https://github.com/kubernetes/dashboard/wiki/Access-control
https://github.com/kubernetes/dashboard/issues/2558
https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/

zhangjun 最后更新： 2018-10-18 04:04:02

tags: addons, heapster

09-3.部署 heapster 插件

Heapster是一个收集者，将每个Node上的cAdvisor的数据进行汇总，然后导到第三方工具(如InfluxDB)。

Heapster 是通过调用 kubelet 的 http API 来获取 cAdvisor 的 metrics 数据的。

由于 kublet 只在 10250 端口接收 https 请求，故需要修改 heapster 的 deployment 配置。同时，需要赋予 kube-system:heapster ServiceAccount 调用 kubelet API 的权限。

下载 heapster 文件

到 heapster release 页面 下载最新版本的 heapster

```
wget https://github.com/kubernetes/heapster/archive/v1.5.3.tar.gz
tar -xzvf v1.5.3.tar.gz
mv v1.5.3.tar.gz heapster-1.5.3.tar.gz
```

官方文件目录： `heapster-1.5.3/deploy/kube-config/influxdb`

修改配置

```
$ cd heapster-1.5.3/deploy/kube-config/influxdb
$ cp grafana.yaml{,.orig}
$ diff grafana.yaml.orig grafana.yaml
16c16
<         image: gcr.io/google_containers/heapster-grafana-amd64:v4.4
.3
---
>         image: wanghkkk/heapster-grafana-amd64-v4.4.3:v4.4.3
67c67
<   # type: NodePort
---
>   type: NodePort
```

- 开启 NodePort；

```
$ cp heapster.yaml{,.orig}
$ diff heapster.yaml.orig heapster.yaml
23c23
<         image: gcr.io/google_containers/heapster-amd64:v1.5.3
---
>         image: fishchen/heapster-amd64:v1.5.3
27c27
<         - --source=kubernetes:https://kubernetes.default
---
>         - --source=kubernetes:https://kubernetes.default?kubeletHtt
ps=true&kubeletPort=10250
```

- 由于 kubelet 只在 10250 监听 https 请求，故添加相关参数；

```
$ cp influxdb.yaml{,.orig}
$ diff influxdb.yaml.orig influxdb.yaml
16c16
<         image: gcr.io/google_containers/heapster-influxdb-amd64:v1.
3.3
---
>         image: fishchen/heapster-influxdb-amd64:v1.3.3
```

执行所有定义文件

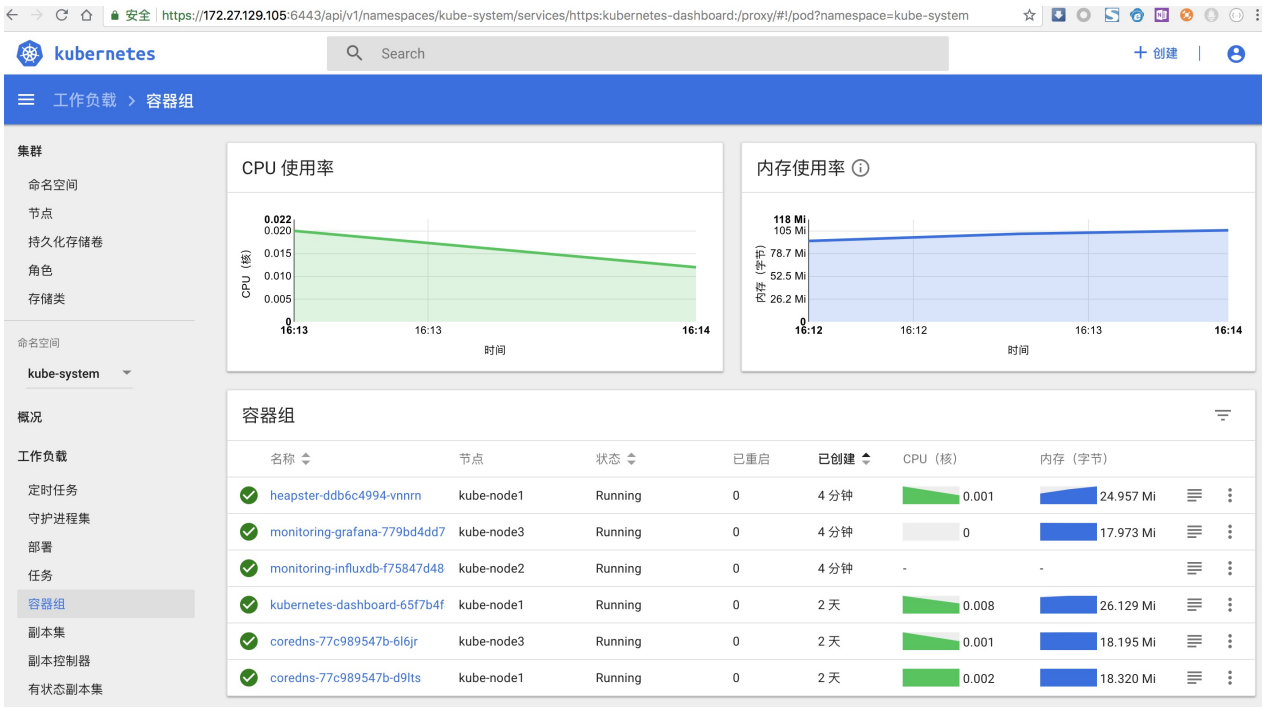
```
$ pwd
/opt/k8s/heapster-1.5.2/deploy/kube-config/influxdb
$ ls *.yaml
grafana.yaml  heapster.yaml  influxdb.yaml
$ kubectl create -f  .

$ cd ../rbac/
$ pwd
/opt/k8s/heapster-1.5.2/deploy/kube-config/rbac
$ ls
heapster-rbac.yaml
$ cp heapster-rbac.yaml{,.orig}
$ diff heapster-rbac.yaml.orig heapster-rbac.yaml
12a13,26
> ---
> kind: ClusterRoleBinding
> apiVersion: rbac.authorization.k8s.io/v1beta1
> metadata:
>   name: heapster-kubelet-api
> roleRef:
>   apiGroup: rbac.authorization.k8s.io
>   kind: ClusterRole
>   name: system:kubelet-api-admin
> subjects:
> - kind: ServiceAccount
>   name: heapster
>   namespace: kube-system
>

$ kubectl create -f heapster-rbac.yaml
```

- 将 serviceAccount kube-system:heapster 与 ClusterRole system:kubelet-api-admin 绑定，授予它调用 kubelet API 的权限；

检查执行结果

```
$ kubectl get pods -n kube-system | grep -E 'heapster|monitoring'
heapster-ddb6c4994-vnnrn                    1/1       Running   0
  1m
monitoring-grafana-779bd4dd7b-xqkgk         1/1       Running   0
  1m
monitoring-influxdb-f75847d48-2lnz6         1/1       Running   0
  1m
```

检查 kubernets dashboard 界面，可以正确显示各 Nodes、Pods 的 CPU、内存、负载等统计数据和图表：



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://172.27.129.105:6443
 CoreDNS is running at https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/coredns:dns/proxy
 Heapster is running at https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/heapster/proxy
 kubernetes-dashboard is running at https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy
 monitoring-grafana is running at https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/monitoring-grafana/proxy
 monitoring-influxdb is running at https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/monitoring-influxdb/proxy

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

浏览器访问 URL： `https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/monitoring-grafana/proxy` 对于 virtuabox 做了端口映射： `http://127.0.0.1:8080/api/v1/namespaces/kube-system/services/monitoring-grafana/proxy`

2. 通过 kubectl proxy 访问：

   创建代理

```
 kubectl proxy --address='172.27.129.105' --port=8086 --accept-hosts='^*$'
 Starting to serve on 172.27.129.80:8086
```

   浏览器访问 URL： `http://172.27.129.105:8086/api/v1/namespaces/kube-system/services/monitoring-grafana/proxy/?orgId=1` 对于 virtuabox 做了端口映射： `http://127.0.0.1:8086/api/v1/namespaces/kube-system/services/monitoring-grafana/proxy/?orgId=1`

3. 通过 NodePort 访问：

```
$ kubectl get svc -n kube-system|grep -E 'monitoring|heapster'
heapster               ClusterIP   10.254.58.136    <none>
 80/TCP          47m
monitoring-grafana     NodePort    10.254.28.196    <none>
 80:8452/TCP     47m
monitoring-influxdb    ClusterIP   10.254.138.164   <none>
 8086/TCP        47m
```

- grafana 监听 NodePort 8452；

  浏览器访问 URL：`http://172.27.129.105:8452/?orgId=1`

图片 - *grafana*

参考：

1. 配置 heapster：https://github.com/kubernetes/heapster/blob/master/docs/source-configuration.md

zhangjun　　最后更新：2018-10-18 04:04:02

tags: addons, metrics, metrics-server

09-4.部署 metrics-server 插件

创建 metrics-server 使用的证书

创建 metrics-server 证书签名请求:

```
cat > metrics-server-csr.json <<EOF
{
  "CN": "aggregator",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "4Paradigm"
    }
  ]
}
EOF
```

- 注意： CN 名称为 aggregator，需要与 kube-apiserver 的 --requestheader-allowed-names 参数配置一致；

生成 metrics-server 证书和私钥：

```
cfssl gencert -ca=/etc/kubernetes/cert/ca.pem \
  -ca-key=/etc/kubernetes/cert/ca-key.pem  \
  -config=/etc/kubernetes/cert/ca-config.json  \
  -profile=kubernetes metrics-server-csr.json | cfssljson -bare metri
cs-server
```

将生成的证书和私钥文件拷贝到 kube-apiserver 节点：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp metrics-server*.pem k8s@${node_ip}:/etc/kubernetes/cert/
  done
```

修改 kubernetes 控制平面组件的配置以支持 metrics-server

kube-apiserver

添加如下配置参数：

```
--requestheader-client-ca-file=/etc/kubernetes/cert/ca.pem
--requestheader-allowed-names=""
--requestheader-extra-headers-prefix="X-Remote-Extra-"
--requestheader-group-headers=X-Remote-Group
--requestheader-username-headers=X-Remote-User
--proxy-client-cert-file=/etc/kubernetes/cert/metrics-server.pem
--proxy-client-key-file=/etc/kubernetes/cert/metrics-server-key.pem
--runtime-config=api/all=true
```

- `--requestheader-XXX`、`--proxy-client-XXX` 是 kube-apiserver 的 aggregator layer 相关的配置参数，metrics-server & HPA 需要使用；
- `--requestheader-client-ca-file`：用于签名 `--proxy-client-cert-file` 和 `--proxy-client-key-file` 指定的证书；在启用了 metric aggregator 时使用；
- 如果 --requestheader-allowed-names 不为空，则--proxy-client-cert-file 证书的 CN 必须位于 allowed-names 中，默认为 aggregator;

如果 kube-apiserver 机器没有运行 kube-proxy，则还需要添加 `--enable-aggregator-routing=true` 参数；

关于 `--requestheader-XXX` 相关参数，参考：

- https://github.com/kubernetes-incubator/apiserver-builder/blob/master/docs/concepts/auth.md
- https://docs.bitnami.com/kubernetes/how-to/configure-autoscaling-custom-metrics/

注意：requestheader-client-ca-file 指定的 CA 证书，必须具有 client auth and server auth；

kube-controllr-manager

添加如下配置参数：

--horizontal-pod-autoscaler-use-rest-clients=true

用于配置 HPA 控制器使用 REST 客户端获取 metrics 数据。

整体架构

图片 - *k8s-hpa.png*

修改插件配置文件配置文件

metrics-server 插件位于 kubernetes 的 `cluster/addons/metrics-server/` 目录下。

修改 metrics-server-deployment 文件：

```
$ cp metrics-server-deployment.yaml{,.orig}
$ diff metrics-server-deployment.yaml.orig metrics-server-deployment.
yaml
51c51
<         image: mirrorgooglecontainers/metrics-server-amd64:v0.2.1
---
>         image: k8s.gcr.io/metrics-server-amd64:v0.2.1
54c54
<         - --source=kubernetes.summary_api:''
---
>         - --source=kubernetes.summary_api:https://kubernetes.defaul
t?kubeletHttps=true&kubeletPort=10250
60c60
<         image: siriuszg/addon-resizer:1.8.1
---
>         image: k8s.gcr.io/addon-resizer:1.8.1
```

- metrics-server 的参数格式与 heapster 类似。由于 kubelet 只在 10250 监听 https 请求，故添加相关参数；

授予 kube-system:metrics-server ServiceAccount 访问 kubelet API 的权限：

```
$ cat auth-kubelet.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metrics-server:system:kubelet-api-admin
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:kubelet-api-admin
subjects:
- kind: ServiceAccount
  name: metrics-server
  namespace: kube-system
```

- 新建一个 ClusterRoleBindings 定义文件，授予相关权限；

创建 metrics-server

```
$ pwd
/opt/k8s/kubernetes/cluster/addons/metrics-server
$ ls -l *.yaml
-rw-rw-r-- 1 k8s k8s  398 Jun  5 07:17 auth-delegator.yaml
-rw-rw-r-- 1 k8s k8s  404 Jun 16 18:02 auth-kubelet.yaml
-rw-rw-r-- 1 k8s k8s  419 Jun  5 07:17 auth-reader.yaml
-rw-rw-r-- 1 k8s k8s  393 Jun  5 07:17 metrics-apiservice.yaml
-rw-rw-r-- 1 k8s k8s 2640 Jun 16 17:54 metrics-server-deployment.yaml
-rw-rw-r-- 1 k8s k8s  336 Jun  5 07:17 metrics-server-service.yaml
-rw-rw-r-- 1 k8s k8s  801 Jun  5 07:17 resource-reader.yaml
$ kubectl create -f .
```

查看运行情况

```
$ kubectl get pods -n kube-system |grep metrics-server
metrics-server-v0.2.1-7486f5bd67-v95q2   2/2       Running   0
   45s

$ kubectl get svc -n kube-system|grep metrics-server
metrics-server         ClusterIP   10.254.115.120   <none>        443
/TCP         1m
```

查看 metrcs-server 输出的 metrics

metrics-server 输出的
APIs：https://github.com/kubernetes/community/blob/master/contributors/design-proposals/instrumentation/resource-metrics-api.md

1. 通过 kube-apiserver 或 kubectl proxy 访问：

   https://172.27.129.105:6443/apis/metrics.k8s.io/v1beta1/nodes
   https://172.27.129.105:6443/apis/metrics.k8s.io/v1beta1/nodes/
   https://172.27.129.105:6443/apis/metrics.k8s.io/v1beta1/pods
   https://172.27.129.105:6443/apis/metrics.k8s.io/v1beta1/namespace//pods/

2. 直接使用 kubectl 命令访问：

   kubectl get --raw apis/metrics.k8s.io/v1beta1/nodes kubectl get --raw apis/metrics.k8s.io/v1beta1/pods kubectl get --raw apis/metrics.k8s.io/v1beta1/nodes/ kubectl get --raw apis/metrics.k8s.io/v1beta1/namespace//pods/

```
$ kubectl get --raw "/apis/metrics.k8s.io/v1beta1" | jq .
{
  "kind": "APIResourceList",
  "apiVersion": "v1",
  "groupVersion": "metrics.k8s.io/v1beta1",
  "resources": [
    {
      "name": "nodes",
      "singularName": "",
      "namespaced": false,
      "kind": "NodeMetrics",
      "verbs": [
        "get",
        "list"
```

```
      ]
    },
    {
      "name": "pods",
      "singularName": "",
      "namespaced": true,
      "kind": "PodMetrics",
      "verbs": [
        "get",
        "list"
      ]
    }
  ]
}

$ kubectl get --raw "/apis/metrics.k8s.io/v1beta1/nodes" | jq .
{
  "kind": "NodeMetricsList",
  "apiVersion": "metrics.k8s.io/v1beta1",
  "metadata": {
    "selfLink": "/apis/metrics.k8s.io/v1beta1/nodes"
  },
  "items": [
    {
      "metadata": {
        "name": "kube-node3",
        "selfLink": "/apis/metrics.k8s.io/v1beta1/nodes/kube-node3",
        "creationTimestamp": "2018-06-16T10:24:03Z"
      },
      "timestamp": "2018-06-16T10:23:00Z",
      "window": "1m0s",
      "usage": {
        "cpu": "133m",
        "memory": "1115728Ki"
      }
    },
    {
      "metadata": {
        "name": "kube-node1",
        "selfLink": "/apis/metrics.k8s.io/v1beta1/nodes/kube-node1",
        "creationTimestamp": "2018-06-16T10:24:03Z"
      },
      "timestamp": "2018-06-16T10:23:00Z",
      "window": "1m0s",
      "usage": {
        "cpu": "221m",
```

```
        "memory": "6799908Ki"
      }
    },
    {
      "metadata": {
        "name": "kube-node2",
        "selfLink": "/apis/metrics.k8s.io/v1beta1/nodes/kube-node2",
        "creationTimestamp": "2018-06-16T10:24:03Z"
      },
      "timestamp": "2018-06-16T10:23:00Z",
      "window": "1m0s",
      "usage": {
        "cpu": "76m",
        "memory": "1130180Ki"
      }
    }
  ]
}
```

- /apis/metrics.k8s.io/v1beta1/nodes 和 /apis/metrics.k8s.io/v1beta1/pods 返回的 usage 包含 CPU 和 Memory；

参考：

1. https://kubernetes.feisky.xyz/zh/addons/metrics.html
2. metrics-server RBAC：https://github.com/kubernetes-incubator/metrics-server/issues/40
3. metrics-server 参数：https://github.com/kubernetes-incubator/metrics-server/issues/25
4. https://kubernetes.io/docs/tasks/debug-application-cluster/core-metrics-pipeline/

zhangjun 最后更新： 2018-10-18 04:04:02

tags: addons, EFK, fluentd, elasticsearch, kibana

09-5.部署 EFK 插件

EFK 对应的目录：`kubernetes/cluster/addons/fluentd-elasticsearch`

```
$ cd /opt/k8s/kubernetes/cluster/addons/fluentd-elasticsearch
$ ls *.yaml
es-service.yaml  es-statefulset.yaml  fluentd-es-configmap.yaml  flue
ntd-es-ds.yaml  kibana-deployment.yaml  kibana-service.yaml
```

修改定义文件

```
$ cp es-statefulset.yaml{,.orig}
$ diff es-statefulset.yaml{,.orig}
76c76
<       - image: longtds/elasticsearch:v5.6.4
---
>       - image: k8s.gcr.io/elasticsearch:v5.6.4

$ cp fluentd-es-ds.yaml{,.orig}
$ diff fluentd-es-ds.yaml{,.orig}
79c79
<         image: netonline/fluentd-elasticsearch:v2.0.4
---
>         image: k8s.gcr.io/fluentd-elasticsearch:v2.0.4
```

给 Node 设置标签

DaemonSet `fluentd-es` 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    ROLES     AGE       VERSION
kube-node1    Ready     <none>    3d        v1.10.4
kube-node2    Ready     <none>    3d        v1.10.4
kube-node3    Ready     <none>    3d        v1.10.4

$ kubectl label nodes kube-node3 beta.kubernetes.io/fluentd-ds-ready=
true
node "kube-node3" labeled
```

执行定义文件

```
$ pwd
/opt/k8s/kubernetes/cluster/addons/fluentd-elasticsearch
$ ls *.yaml
es-service.yaml  es-statefulset.yaml  fluentd-es-configmap.yaml  flue
ntd-es-ds.yaml  kibana-deployment.yaml  kibana-service.yaml
$ kubectl create -f .
```

检查执行结果

```
$ kubectl get pods -n kube-system -o wide|grep -E 'elasticsearch|flue
ntd|kibana'
elasticsearch-logging-0                 1/1       Running   0
   5m        172.30.81.7   kube-node1
elasticsearch-logging-1                 1/1       Running   0
   2m        172.30.39.8   kube-node3
fluentd-es-v2.0.4-hntfp                 1/1       Running   0
   5m        172.30.39.6   kube-node3
kibana-logging-7445dc9757-pvpcv         1/1       Running   0
   5m        172.30.39.7   kube-node3

$ kubectl get service  -n kube-system|grep -E 'elasticsearch|kibana'
elasticsearch-logging   ClusterIP   10.254.50.198    <none>        92
00/TCP        5m
kibana-logging          ClusterIP   10.254.255.190   <none>        56
01/TCP        5m
```

kibana Pod 第一次启动时会用较长时间**(0-20**分钟**)**来优化和 Cache 状态页面，可以 tailf 该 Pod 的日志观察进度：

```
[k8s@kube-node1 fluentd-elasticsearch]$ kubectl logs kibana-logging-7
445dc9757-pvpcv -n kube-system -f
{"type":"log","@timestamp":"2018-06-16T11:36:18Z","tags":["info","opt
imize"],"pid":1,"message":"Optimizing and caching bundles for graph,
ml, kibana, stateSessionStorageRedirect, timelion and status_page. Th
is may take a few minutes"}
{"type":"log","@timestamp":"2018-06-16T11:40:03Z","tags":["info","opt
imize"],"pid":1,"message":"Optimization of bundles for graph, ml, kib
ana, stateSessionStorageRedirect, timelion and status_page complete i
n 224.57 seconds"}
```

注意：只有当的 Kibana pod 启动完成后，才能查看 kibana dashboard，否则会提示 refuse。

访问 kibana

1. 通过 kube-apiserver 访问：

   ```
    $ kubectl cluster-info|grep -E 'Elasticsearch|Kibana'
    Elasticsearch is running at https://172.27.129.105:6443/api/v1/n
   amespaces/kube-system/services/elasticsearch-logging/proxy
    Kibana is running at https://172.27.129.105:6443/api/v1/namespac
   es/kube-system/services/kibana-logging/proxy
   ```

   浏览器访问 URL： `https://172.27.129.105:6443/api/v1/namespaces/kube-system/services/kibana-logging/proxy` 对于 virtuabox 做了端口映射： `http://127.0.0.1:8080/api/v1/namespaces/kube-system/services/kibana-logging/proxy`

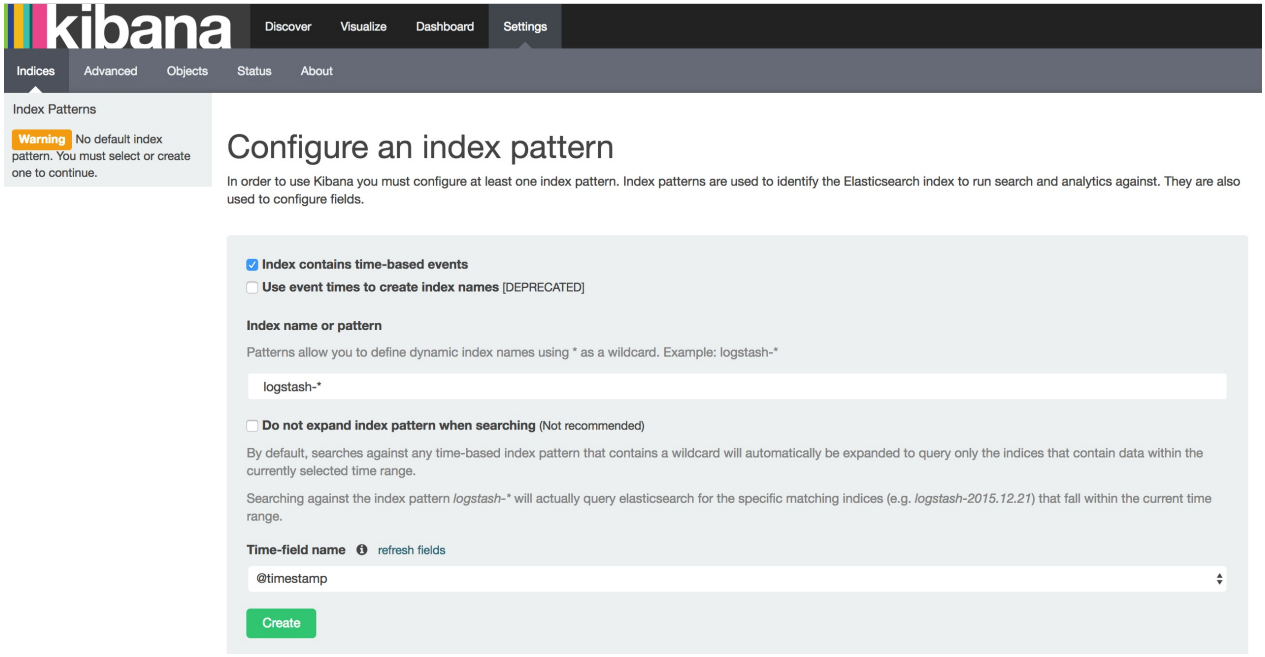
2. 通过 kubectl proxy 访问：

   创建代理

   ```
    $ kubectl proxy --address='172.27.129.105' --port=8086 --accept-
   hosts='^*$'
    Starting to serve on 172.27.129.80:8086
   ```

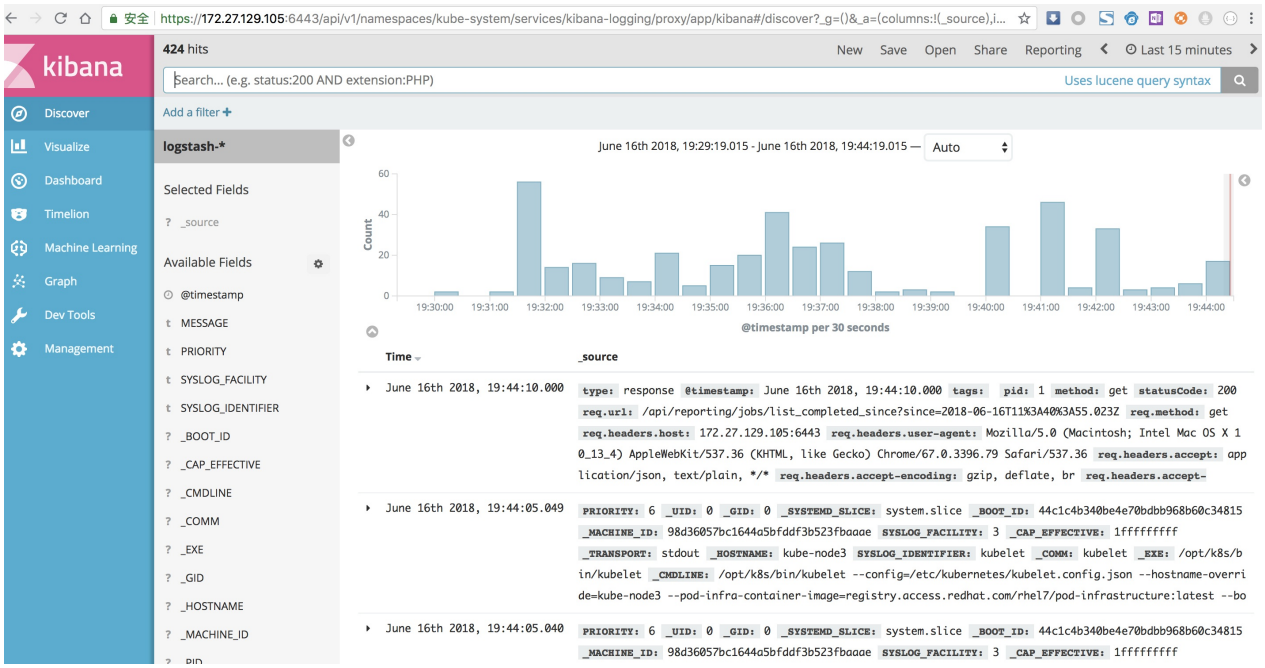
浏览器访问 URL：`http://172.27.129.105:8086/api/v1/namespaces/kube-system/services/kibana-logging/proxy` 对于 virtuabox 做了端口映射：`http://127.0.0.1:8086/api/v1/namespaces/kube-system/services/kibana-logging/proxy`

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；



图片 - *es-home*

zhangjun 最后更新： 2018-10-18 04:04:02

tags: registry, ceph

10.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: 172.27.132.66
- docker registry: 172.27.132.67

部署 ceph RGW 节点

```
$ ceph-deploy rgw create 172.27.132.66 # 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 --acce
ss=full --secret=secretkey --key-type=swift
$
```

创建 demo:swift 子账号的 sercret key

```
$ radosgw-admin key create --subuser=demo:swift --key-type=swift --ge
n-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": "5Y1B1SIJ2YHKEHO5U36B",
            "secret_key": "nrIvtPqUj7pUlccLYPuR3ntVzIa50DToIpe7xFjT"
        }
    ],
    "swift_keys": [
        {
            "user": "demo:swift",
            "secret_key": "ttQcU1O17DFQ4I9xzKqwgUe7WIYYX99zhcIfU9vb"
        }
    ],
    "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": []
}
```

- `ttQcU1O17DFQ4I9xzKqwgUe7WIYYX99zhcIfU9vb` 为子账号 demo:swift 的 secret key；

创建 docker registry

创建 registry 使用的 x509 证书

```
$ mkdir -p registry/{auth,certs}
$ cat > registry-csr.json <<EOF
{
  "CN": "registry",
  "hosts": [
      "127.0.0.1",
      "172.27.132.67"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "4Paradigm"
    }
  ]
}
EOF
$ cfssl gencert -ca=/etc/kubernetes/cert/ca.pem \
    -ca-key=/etc/kubernetes/cert/ca-key.pem \
    -config=/etc/kubernetes/cert/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 Baisc 认证文件

```
$ docker run --entrypoint htpasswd registry:2 -Bbn foo foo123  > regi
stry/auth/htpasswd
$ cat  registry/auth/htpasswd
foo:$2y$05$iZaM45Jxlcg0DJKXZMggLOibAsHLGybyU.CgU9AHqWcVDyBjiScN.
```

配置 registry 参数

```
export RGW_AUTH_URL="http://172.27.132.66:7480/auth/v1"
export RGW_USER="demo:swift"
export RGW_SECRET_KEY="ttQcU1O17DFQ4I9xzKqwgUe7WIYYX99zhcIfU9vb"
cat > config.yml << EOF
# https://docs.docker.com/registry/configuration/#list-of-configurati
on-options
version: 0.1
log:
  level: info
  fromatter: 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
[k8s@kube-node1 cert]$ cp config.yml registry
[k8s@kube-node1 cert]$ scp -r registry 172.27.132.67:/opt/k8s
```

- storage.swift 指定后端使用 swfit 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

```
ssh k8s@172.27.132.67
$ docker run -d -p 8000:8000 --privileged \
    -v /opt/k8s/registry/auth/:/auth \
    -v /opt/k8s/registry/certs:/certs \
    -v /opt/k8s/registry/config.yml:/etc/docker/registry/config.yml \
    --name registry registry:2
```

- 执行该 docker run 命令的机器 IP 为 172.27.132.67；

向 registry push image

将签署 registry 证书的 CA 证书拷贝到 `/etc/docker/certs.d/172.27.132.67:8000` 目录下

```
[k8s@kube-node1 cert]$ sudo mkdir -p /etc/docker/certs.d/172.27.132.6
7:8000
[k8s@kube-node1 cert]$ sudo cp /etc/kubernetes/cert/ca.pem /etc/docke
r/certs.d/172.27.132.67:8000/ca.crt
```

登陆私有 registry

```
$ docker login 172.27.132.67:8000
Username: foo
Password:
Login Succeeded
```

登陆信息被写入 `~/.docker/config.json` 文件

```
$ cat ~/.docker/config.json
{
        "auths": {
                "172.27.132.67:8000": {
                        "auth": "Zm9vOmZvbzEyMw=="
                }
        }
}
```

将本地的 image 打上私有 registry 的 tag

```
$ docker tag prom/node-exporter:v0.16.0 172.27.132.67:8000/prom/node-
exporter:v0.16.0
$ docker images |grep pause
prom/node-exporter:v0.16.0                              latest
     f9d5de079539        2 years ago         239.8 kB
172.27.132.67:8000/prom/node-exporter:v0.16.0
latest              f9d5de079539        2 years ago         239.8 kB
```

将 image push 到私有 registry

```
$ docker push 172.27.132.67:8000/prom/node-exporter:v0.16.0
The push refers to a repository [172.27.132.67:8000/prom/node-exporte
r:v0.16.0]
5f70bf18a086: Pushed
e16a89738269: Pushed
latest: digest: sha256:9a6b437e896acad3f5a2a8084625fdd4177b2e7124ee94
3af642259f2f283359 size: 916
```

查看 ceph 上是否已经有 push 的 pause 容器文件

```
[k8s@kube-node1 ~]$ rados lspools
rbd
cephfs_data
cephfs_metadata
.rgw.root
k8s
default.rgw.control
default.rgw.meta
default.rgw.log
default.rgw.buckets.index
default.rgw.buckets.data

[k8s@kube-node1 ~]$  rados --pool default.rgw.buckets.data ls|grep no
de-exporter
1f3f02c4-fe58-4626-992b-c6c0fe4c8acf.34107.1_files/docker/registry/v2
/repositories/prom/node-exporter/_layers/sha256/cdb7590af5f064887f3d6
008d46be65e929c74250d747813d85199e04fc70463/link
1f3f02c4-fe58-4626-992b-c6c0fe4c8acf.34107.1_files/docker/registry/v2
/repositories/prom/node-exporter/_manifests/revisions/sha256/55302581
333c43d540db0e144cf9e7735423117a733cdec27716d87254221086/link
1f3f02c4-fe58-4626-992b-c6c0fe4c8acf.34107.1_files/docker/registry/v2
/repositories/prom/node-exporter/_manifests/tags/v0.16.0/current/link
1f3f02c4-fe58-4626-992b-c6c0fe4c8acf.34107.1_files/docker/registry/v2
/repositories/prom/node-exporter/_manifests/tags/v0.16.0/index/sha256
/55302581333c43d540db0e144cf9e7735423117a733cdec27716d87254221086/lin
k
1f3f02c4-fe58-4626-992b-c6c0fe4c8acf.34107.1_files/docker/registry/v2
/repositories/prom/node-exporter/_layers/sha256/224a21997e8ca8514d42e
b2ed98b19a7ee2537bce0b3a26b8dff510ab637f15c/link
1f3f02c4-fe58-4626-992b-c6c0fe4c8acf.34107.1_files/docker/registry/v2
/repositories/prom/node-exporter/_layers/sha256/528dda9cf23d0fad80347
749d6d06229b9a19903e49b7177d5f4f58736538d4e/link
1f3f02c4-fe58-4626-992b-c6c0fe4c8acf.34107.1_files/docker/registry/v2
/repositories/prom/node-exporter/_layers/sha256/188af75e2de0203eac7c6
e982feff45f9c340eaac4c7a0f59129712524fa2984/link
```

私有 **registry** 的运维操作

查询私有镜像中的 **images**

```
[k8s@kube-node1 ~]$ curl  --user foo:foo123 --cacert /etc/docker/certs.d/172.27.132.67\:8000/ca.crt https://172.27.132.67:8000/v2/_catalog
{"repositories":["prom/node-exporter"]}
```

查询某个镜像的 tags 列表

```
[k8s@kube-node1 ~]$  curl  --user foo:foo123 --cacert /etc/docker/certs.d/172.27.132.67\:8000/ca.crt https://172.27.132.67:8000/v2/prom/node-exporter/tags/list
{"name":"prom/node-exporter","tags":["v0.16.0"]}
```

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`：

```
[k8s@kube-node1 ~]$ curl -v -H "Accept: application/vnd.docker.distribution.manifest.v2+json" --user foo:foo123 --cacert /etc/docker/certs.d/172.27.132.67\:8000/ca.crt https://172.27.132.67:8000/v2/prom/node-exporter/manifests/v0.16.0
* About to connect() to 172.27.132.67 port 8000 (#0)
*   Trying 172.27.132.67...
* Connected to 172.27.132.67 (172.27.132.67) port 8000 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
*   CAfile: /etc/docker/certs.d/172.27.132.67:8000/ca.crt
  CApath: none
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*       subject: CN=registry,OU=4Paradigm,O=k8s,L=BeiJing,ST=BeiJing,C=CN
*       start date: Jul 05 12:52:00 2018 GMT
*       expire date: Jul 02 12:52:00 2028 GMT
*       common name: registry
*       issuer: CN=kubernetes,OU=4Paradigm,O=k8s,L=BeiJing,ST=BeiJing,C=CN
* Server auth using Basic with user 'foo'
> GET /v2/prom/node-exporter/manifests/v0.16.0 HTTP/1.1
```

```
> Authorization: Basic Zm9vOmZvbzEyMw==
> User-Agent: curl/7.29.0
> Host: 172.27.132.67:8000
> Accept: application/vnd.docker.distribution.manifest.v2+json
>
< HTTP/1.1 200 OK
< Content-Length: 949
< Content-Type: application/vnd.docker.distribution.manifest.v2+json
< Docker-Content-Digest: sha256:55302581333c43d540db0e144cf9e77354231
17a733cdec27716d87254221086
< Docker-Distribution-Api-Version: registry/2.0
< Etag: "sha256:55302581333c43d540db0e144cf9e7735423117a733cdec27716d
87254221086"
< X-Content-Type-Options: nosniff
< Date: Fri, 06 Jul 2018 06:18:41 GMT
<
{
   "schemaVersion": 2,
   "mediaType": "application/vnd.docker.distribution.manifest.v2+json"
,
   "config": {
      "mediaType": "application/vnd.docker.container.image.v1+json",
      "size": 3511,
      "digest": "sha256:188af75e2de0203eac7c6e982feff45f9c340eaac4c7a
0f59129712524fa2984"
   },
   "layers": [
      {
         "mediaType": "application/vnd.docker.image.rootfs.diff.tar.g
zip",
         "size": 2392417,
         "digest": "sha256:224a21997e8ca8514d42eb2ed98b19a7ee2537bce0
b3a26b8dff510ab637f15c"
      },
      {
         "mediaType": "application/vnd.docker.image.rootfs.diff.tar.g
zip",
         "size": 560703,
         "digest": "sha256:cdb7590af5f064887f3d6008d46be65e929c74250d
747813d85199e04fc70463"
      },
      {
         "mediaType": "application/vnd.docker.image.rootfs.diff.tar.g
zip",
         "size": 5332460,
         "digest": "sha256:528dda9cf23d0fad80347749d6d06229b9a19903e4
```

```
9b7177d5f4f58736538d4e"
      }
   ]
```

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 foo:foo123 --cacert /etc/docker/certs.d/172.
27.132.67\:8000/ca.crt https://172.27.132.67:8000/v2/prom/node-export
er/manifests/sha256:68effe31a4ae8312e47f54bec52d1fc925908009ce7e6f734
e1b54a4169081c5
$
```

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 foo:foo123 --cacert /etc/docker/certs.d/172.
27.132.67\:8000/ca.crt https://172.27.132.67:8000/v2/prom/node-export
er/blobs/sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7
c22955b46d4
$ curl -X DELETE  --cacert /etc/docker/certs.d/172.27.132.67\:8000/ca
.crt https://172.27.132.67:8000/v2/prom/node-exporter/blobs/sha256:04
176c8b224aa0eb9942af765f66dae866f436e75acef028fe44b8a98e045515
$
```

常见问题

login 失败 416

执行 http://docs.ceph.com/docs/master/install/install-ceph-gateway/ 里面的 s3 test.py 程序失败：

[k8s@kube-node1 cert]$ python s3test.py Traceback (most recent call last): File "s3test.py", line 12, in bucket = conn.create_bucket('my-new-bucket') File "/usr/lib/python2.7/site-packages/boto/s3/connection.py", line 625, in create_bucket response.status, response.reason, body) boto.exception.S3ResponseError: S3ResponseError: 416 Requested Range Not Satisfiable

解决版办法：

1. 在管理节点上修改 ceph.conf
2. ceph-deploy config push kube-node1 kube-node2 kube-node3
3. systemctl restart 'ceph-mds@kube-node3.service' systemctl restart ceph-osd@0 systemctl restart 'ceph-mon@kube-node1.service' systemctl restart 'ceph-mgr@kube-node1.service'

For anyone who is hitting this issue set default pg_num and pgp_num to lower value(8 for example), or set mon_max_pg_per_osd to a high value in ceph.conf radosgw-admin doesn' throw proper error when internal pool creation fails, hence the upper level error which is very confusing.

https://tracker.ceph.com/issues/21497

login 失败 503

[root@kube-node1 ~]# docker login 172.27.132.67:8000 Username: foo Password: Error response from daemon: login attempt to https://172.27.132.67:8000/v2/ failed with status: 503 Service Unavailable

原因： docker run 缺少 --privileged 参数；

zhangjun 最后更新： 2018-10-18 04:04:02

tags: registry, harbor

11.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/1.21.2/doc
ker-compose-Linux-x86_64
$ mv ~/docker-compose-Linux-x86_64 /opt/k8s/bin/docker-compose
$ chmod a+x  /opt/k8s/bin/docker-compose
$ export PATH=/opt/k8s/bin:$PATH
$
```

从 harbor 发布页面下载最新的 harbor 离线安装包

```
$ wget  --continue https://storage.googleapis.com/harbor-releases/rel
ease-1.5.0/harbor-offline-installer-v1.5.1.tgz
$ tar -xzvf harbor-offline-installer-v1.5.1.tgz
$
```

导入 docker images

导入离线安装包中 harbor 相关的 docker images：

```
$ cd harbor
$ docker load -i harbor.v1.5.1.tar.gz
$
```

创建 harbor nginx 服务器使用的 x509 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": "4Paradigm"
    }
  ]
}
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/cert/ca.pem \
  -ca-key=/etc/kubernetes/cert/ca-key.pem \
  -config=/etc/kubernetes/cert/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

```
$ cp harbor.cfg{,.bak}
$ vim harbor.cfg
$ diff harbor.cfg{,.bak}
7c7
< hostname = 172.27.129.81
---
> hostname = reg.mydomain.com
11c11
< ui_url_protocol = https
---
> ui_url_protocol = http
23,24c23,24
< ssl_cert =  /etc/harbor/ssl/harbor.pem
< ssl_cert_key = /etc/harbor/ssl/harbor-key.pem
---
> ssl_cert = /data/cert/server.crt
> ssl_cert_key = /data/cert/server.key

$ cp prepare{,.bak}
$ vim prepare
$ diff prepare{,.bak}
453a454
>         print("%s %w", args, kw)
490c491
<     empty_subj = "/"
---
>     empty_subj = "/C=/ST=/L=/O=/CN=/"
```

- 需要修改 prepare 脚本的 empyt_subj 参数，否则后续 install 时出错退出：

  Fail to generate key file: ./common/config/ui/private_key.pem, cert file: ./common/config/registry/root.crt

参考：https://github.com/vmware/harbor/issues/2920

加载和启动 harbor 镜像

```
$ sudo mkdir /data
$ sudo chmod 777 /var/run/docker.sock /data
$ sudo apt-get install python
$ ./install.sh

[Step 0]: checking installation environment ...

Note: docker version: 18.03.0

Note: docker-compose version: 1.21.2

[Step 1]: loading Harbor images ...
Loaded image: vmware/clair-photon:v2.0.1-v1.5.1
Loaded image: vmware/postgresql-photon:v1.5.1
Loaded image: vmware/harbor-adminserver:v1.5.1
Loaded image: vmware/registry-photon:v2.6.2-v1.5.1
Loaded image: vmware/photon:1.0
Loaded image: vmware/harbor-migrator:v1.5.1
Loaded image: vmware/harbor-ui:v1.5.1
Loaded image: vmware/redis-photon:v1.5.1
Loaded image: vmware/nginx-photon:v1.5.1
Loaded image: vmware/mariadb-photon:v1.5.1
Loaded image: vmware/notary-signer-photon:v0.5.1-v1.5.1
Loaded image: vmware/harbor-log:v1.5.1
Loaded image: vmware/harbor-db:v1.5.1
Loaded image: vmware/harbor-jobservice:v1.5.1
Loaded image: vmware/notary-server-photon:v0.5.1-v1.5.1


[Step 2]: preparing environment ...
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.yml
```

```
Generated configuration file: ./common/config/db/env
Generated configuration file: ./common/config/jobservice/env
Generated configuration file: ./common/config/jobservice/config.yml
Generated configuration file: ./common/config/log/logrotate.conf
Generated configuration file: ./common/config/jobservice/config.yml
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 ... done
Creating redis              ... done
Creating harbor-adminserver ... done
Creating harbor-db          ... done
Creating registry           ... done
Creating harbor-ui          ... done
Creating harbor-jobservice  ... done
Creating nginx              ... done

✔ ----Harbor has been installed and started successfully.----

Now you should be able to visit the admin portal at https://172.27.12
9.81.
For more details, please visit https://github.com/vmware/harbor .
```

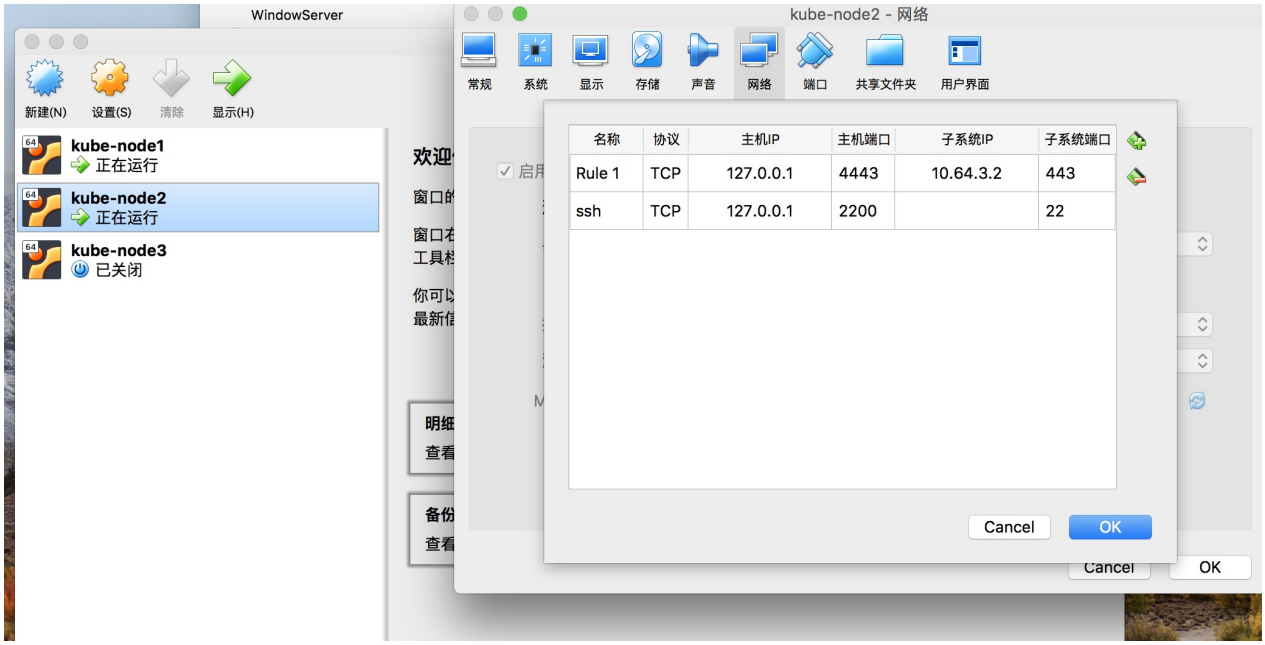
访问管理界面

确认所有组件都工作正常：

```
$ docker-compose  ps
       Name                     Command                  State
                              Ports
------------------------------------------------------------------------
------------------------------------------------------------------
harbor-adminserver   /harbor/start.sh                Up (healthy)
harbor-db            /usr/local/bin/docker-entr ...   Up (healthy)
3306/tcp
harbor-jobservice    /harbor/start.sh                Up
harbor-log           /bin/sh -c /usr/local/bin/ ...   Up (healthy)
127.0.0.1:1514->10514/tcp
harbor-ui            /harbor/start.sh                Up (healthy)
nginx                nginx -g daemon off;            Up (healthy)
0.0.0.0:443->443/tcp, 0.0.0.0:4443->4443/tcp, 0.0.0.0:80->80/tcp
redis                docker-entrypoint.sh redis ...   Up
6379/tcp
registry             /entrypoint.sh serve /etc/ ...   Up (healthy)
5000/tcp
```

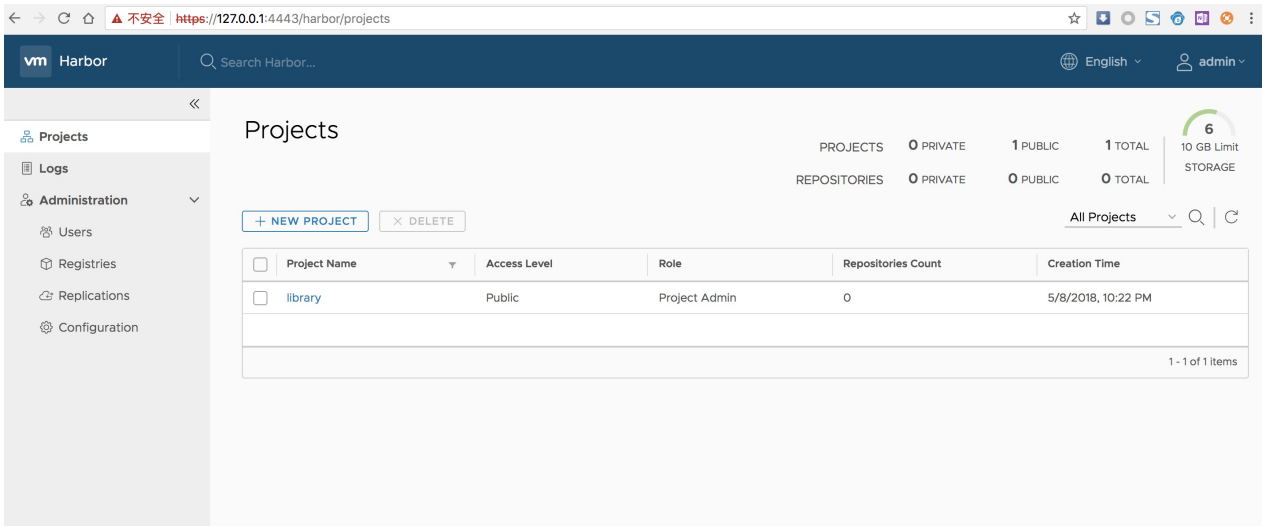
浏览器访问 `https://${NODE_IP}` ，示例的是 `https://172.27.129.81` ；

由于是在 virtualbox 虚机 kube-node2 中运行，所以需要做下端口转发，Vagrant 文件中已经指定 host 端口为 4443，也可以在 virtualbox 的 GUI 中直接添加端口转发：



图片 - *virtualbox-harbor*

浏览器访问 `https://127.0.0.1:443`，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

harbor 将日志打印到 /var/log/harbor 的相关目录下，使用 docker logs XXX 或 docker-compose logs XXX 将看不到容器的日志。

```
$ # 日志目录
$ ls /var/log/harbor
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/172.27.129.81` 目录下

```
$ sudo mkdir -p /etc/docker/certs.d/172.27.129.81
$ sudo cp /etc/kubernetes/cert/ca.pem /etc/docker/certs.d/172.27.129.
81/ca.crt
$
```

登陆 harbor

```
$ docker login 172.27.129.81
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
$ ./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.crt
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/admin.pem
Clearing the configuration file: ./common/config/nginx/cert/admin-key
.pem
Clearing the configuration file: ./common/config/nginx/nginx.conf
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.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.
$ sudo chmod -R 666 common ## 防止容器进程没有权限读取生成的配置
$ # 启动 harbor
$ docker-compose up -d
```

zhangjun 最后更新： 2018-10-18 04:04:02

tags: clean

12.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 umount
$ # 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}.service
$ # 删除程序文件
$ sudo rm -rf /opt/k8s/bin/*
$ # 删除证书文件
$ sudo rm -rf /etc/flanneld/cert /etc/kubernetes/cert
$
```

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-sch
eduler
$
```

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-man
ager,kube-scheduler}.service
$ # 删除程序文件
$ sudo rm -rf /opt/k8s/bin/{kube-apiserver,kube-controller-manager,ku
be-scheduler}
$ # 删除证书文件
$ sudo rm -rf /etc/flanneld/cert /etc/kubernetes/cert
$
```

清理 etcd 集群

停相关进程：

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

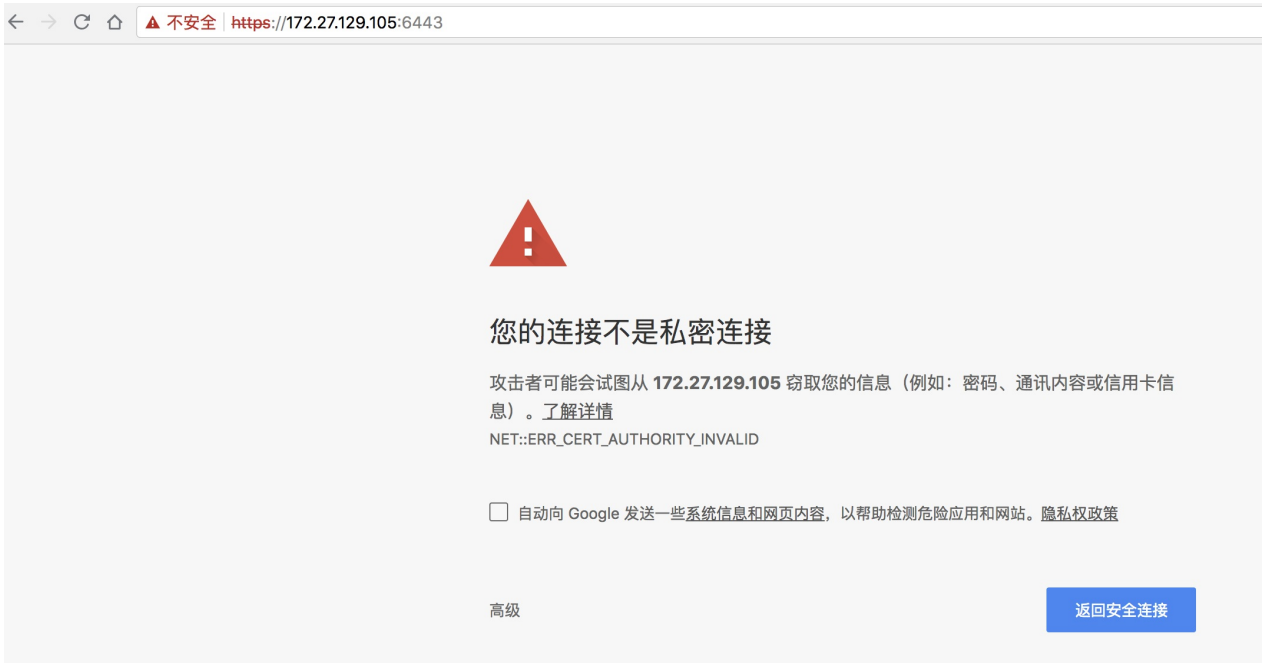
清理文件：

```
$ # 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 /opt/k8s/bin/etcd
$ # 删除 x509 证书文件
$ sudo rm -rf /etc/etcd/cert/*
$
```

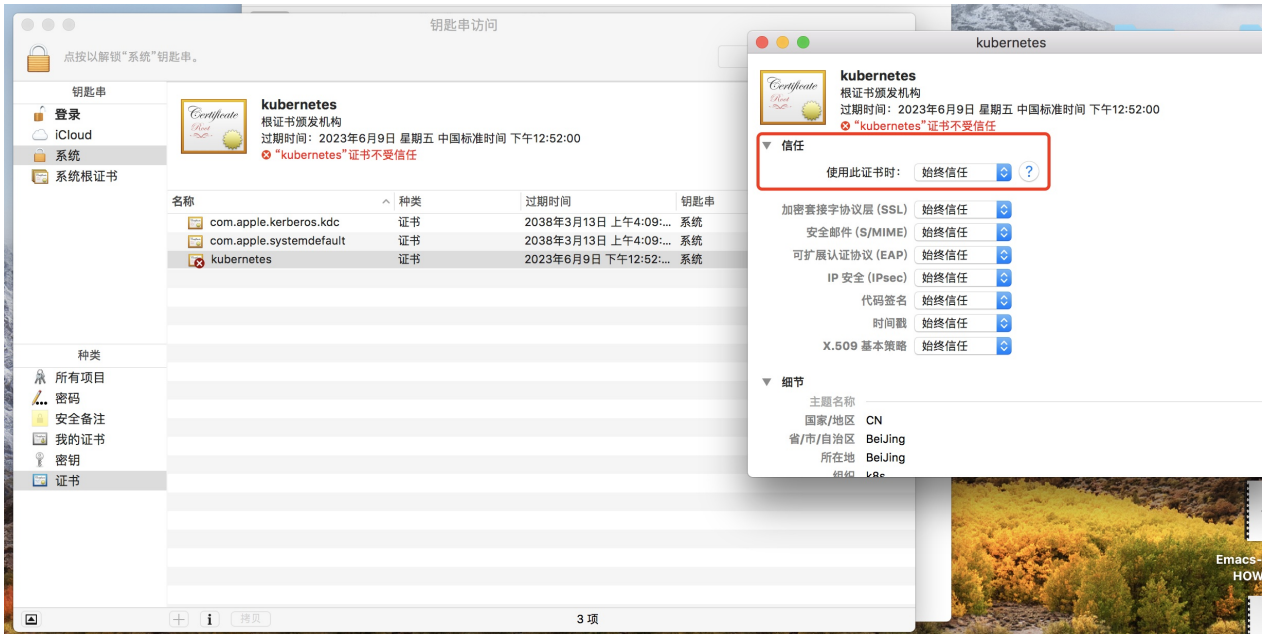
zhangjun 最后更新： 2018-10-18 04:04:02

A.浏览器访问 kube-apiserver 安全端口

浏览器访问 kube-apiserver 的安全端口 6443 时，提示证书不被信任：



这是因为 kube-apiserver 的 server 证书是我们创建的根证书 ca.pem 签名的，需要将根证书 ca.pem 导入操作系统，并设置永久信任。对于 Mac，操作如下：



对于win使用以下命令导入ca.perm

```
keytool -import -v -trustcacerts -alias appmanagement -file "PATH...\
\ca.pem" -storepass password -keystore cacerts
```

再次访问 https://172.27.129.105:6443/，已信任，但提示 401，未授权的访问：

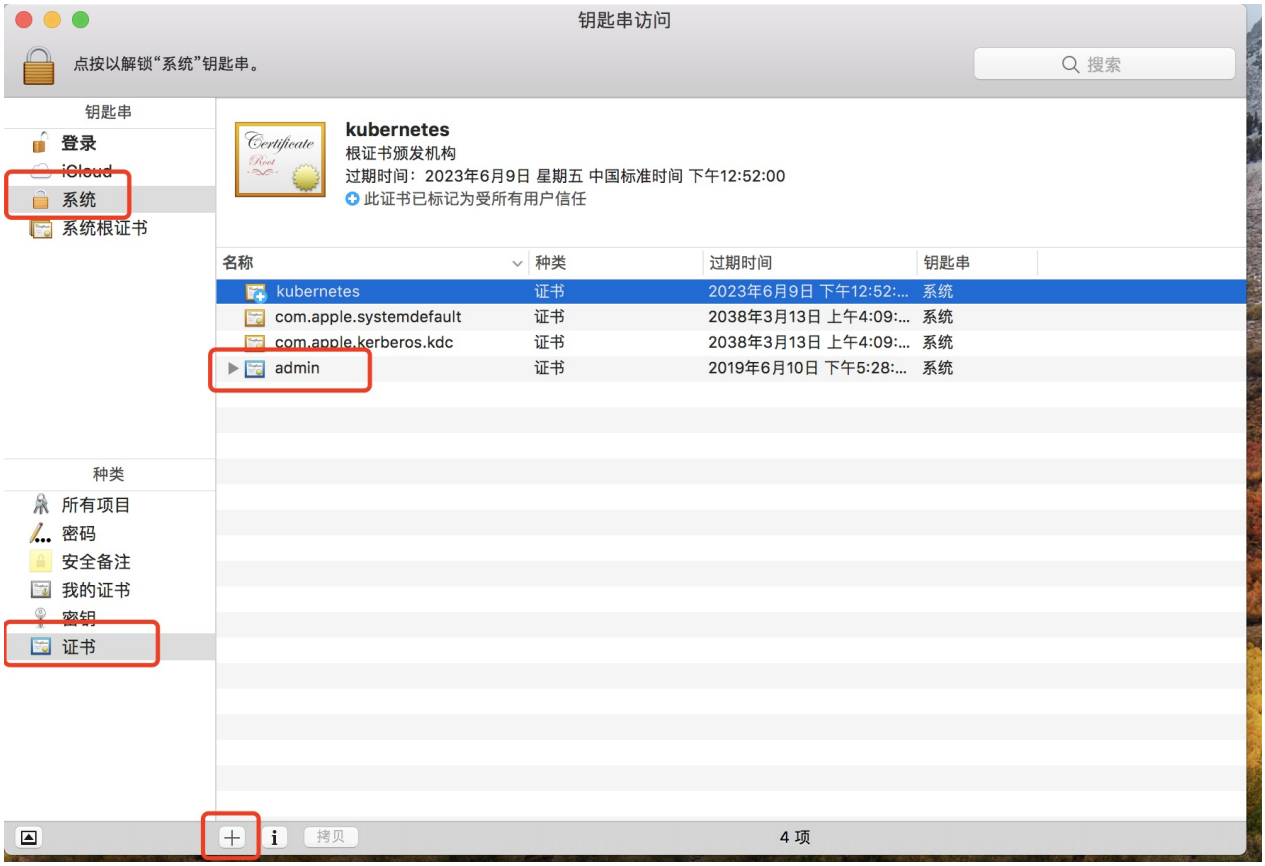
图片 - *ssl-success*

我们需要给浏览器生成一个 client 证书，访问 apiserver 的 6443 https 端口时使用。

这里使用部署 kubectl 命令行工具时创建的 admin 证书、私钥和上面的 ca 证书，创建一个浏览器可以使用 PKCS#12/PFX 格式的证书：

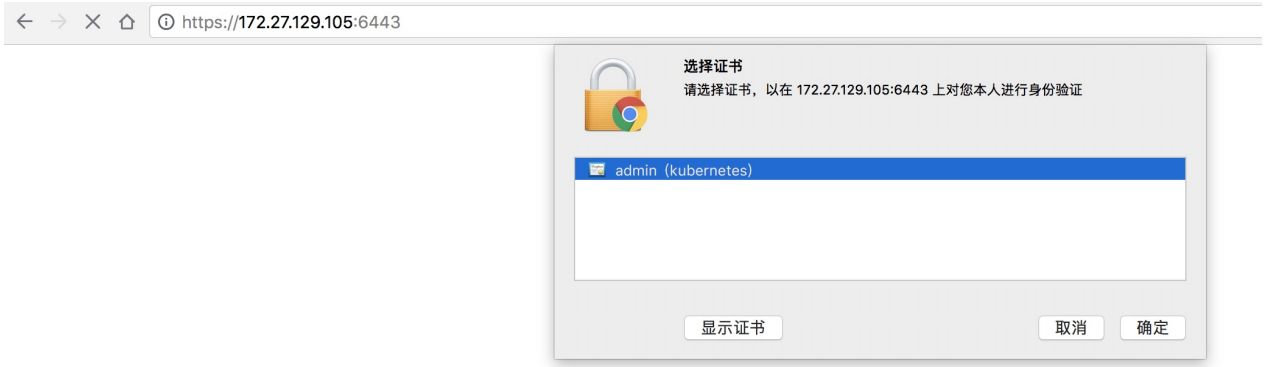
```
[k8s@kube-node1 ~]$ openssl pkcs12 -export -out admin.pfx -inkey admi
n-key.pem -in admin.pem -certfile ca.pem
```

将创建的 admin.pfx 导入到系统的证书中。对于 Mac，操作如下：



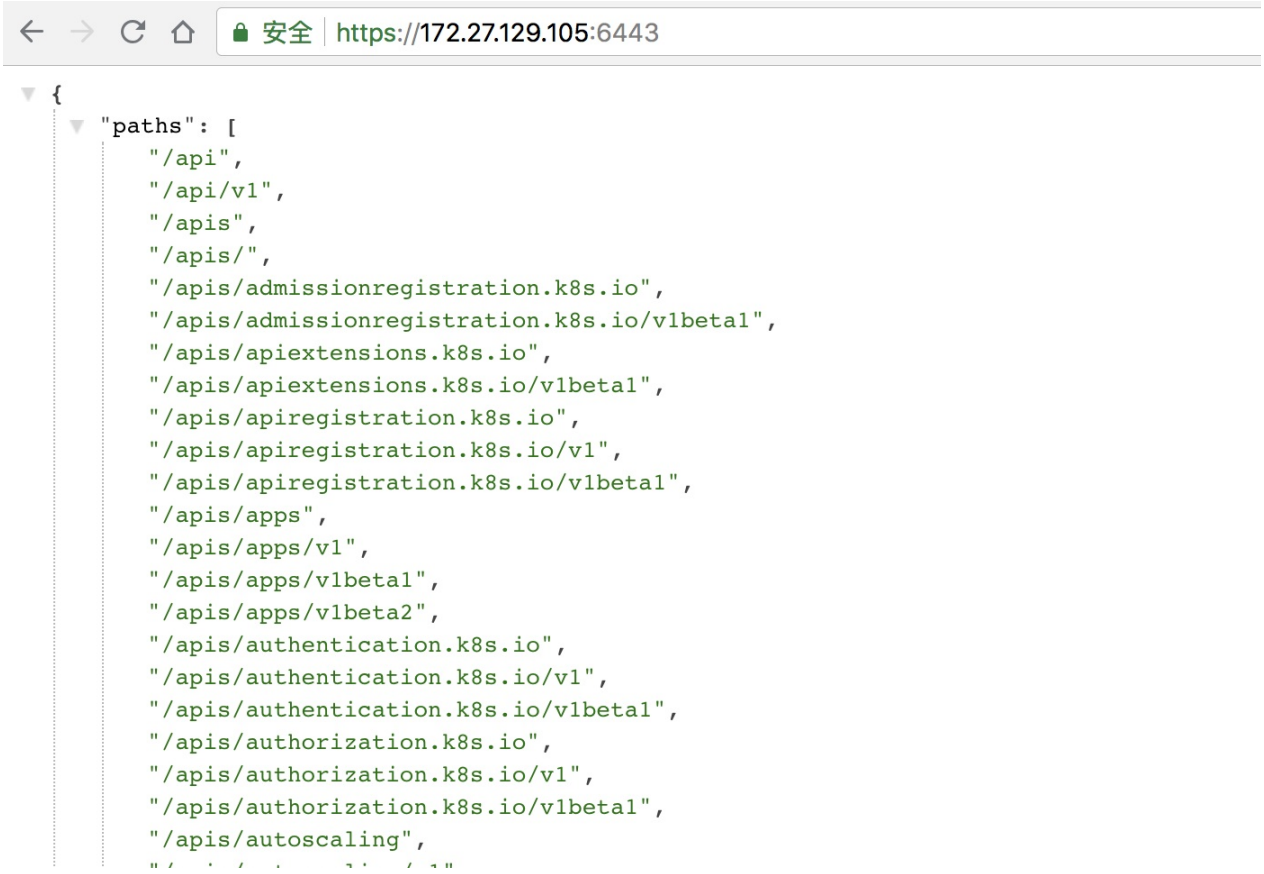
图片 - *admin-cert*

重启浏览器，再次访问 https://172.27.129.105:6443/，提示选择一个浏览器证书，这里选中上面导入的 admin.pfx：



图片 - *select-cert*

这一次，被授权访问 kube-apiserver 的安全端口：



图片 - *chrome-authored*

客户端选择证书的原理

1. 证书选择是在客户端和服务端 SSL/TLS 握手协商阶段商定的；
2. 服务端如果要求客户端提供证书，则在握手时会向客户端发送一个它接受的 CA 列表；
3. 客户端查找它的证书列表(一般是操作系统的证书，对于 Mac 为 keychain)，看有没有被 CA 签名的证书，如果有，则将它们提供给用户选择（证书的私钥）；
4. 用户选择一个证书私钥，然后客户端将使用它和服务端通信；

参考

- https://github.com/kubernetes/kubernetes/issues/31665
- https://www.sslshopper.com/ssl-converter.html
- https://stackoverflow.com/questions/40847638/how-chrome-browser-know-which-client-certificate-to-prompt-for-a-site

zhangjun　　最后更新： 2018-10-18 04:04:02

B.校验 TLS 证书

以校验 kubernetes 证书为例：

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=Kub
ernetes
        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=ku
bernetes
...
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authent
ication
            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:kubernete
s.default.svc, DNS:kubernetes.default.svc.cluster, DNS:kubernetes.def
ault.svc.cluster.local, IP Address:127.0.0.1, IP Address:10.64.3.7, I
P 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",
      "10.64.3.8",
      "10.66.3.86",
      "10.254.0.1"
    ],
    "not_before": "2017-04-05T05:36:00Z",
    "not_after": "2018-04-05T05:36:00Z",
    "sigalg": "SHA256WithRSA",
...
```

校验证书是否被 CA 证书签名

正确的情况：

```
$ openssl verify -CAfile /etc/kubernetes/cert/ca.pem /etc/kubernetes/
cert/kubernetes.pem
/etc/kubernetes/cert/kubernetes.pem: OK
```

失败的情况：

```
$ openssl verify -CAfile ca_wrong.pem /etc/kubernetes/cert/kubernetes
.pem
/etc/kubernetes/cert/kubernetes.pem: C = CN, ST = BeiJing, L = BeiJin
g, O = k8s, OU = 4Paradigm, CN = kubernetes
error 20 at 0 depth lookup:unable to get local issuer certificate
```

zhangjun　　最后更新： 2018-10-18 04:04:02

Tags

zhangjun 最后更新： 2018-10-18 04:04:02