**# == 重要信息 ==**

# TLS Bootstrapping 使用的 Token，可以使用命令 head -c 16 /dev/urandom | od -An -t x | tr -d ' ' 生成
BOOTSTRAP_TOKEN="fb51fd6014cb3963f9e77fe65e0813b9"
# 最好使用 主机未用的网段 来定义服务网段和 Pod 网段
# 服务网段 (Service CIDR），部署前路由不可达，部署后集群内使用IP:Port可达
SERVICE_CIDR="10.254.0.0/16"
# POD 网段 (Cluster CIDR），部署前路由不可达，**部署后**路由可达(ovs保证)
# 此网络也叫kubernetes cluster network
CLUSTER_CIDR="10.66.192.0/20"
# 每个docker主机的子网，属于上面的POD网段
SUBNET_LEN= 26
# 服务端口范围 (NodePort Range)
NODE_PORT_RANGE="30000-32767"
# etcd 集群服务地址列表
ETCD_ENDPOINTS="https://10.66.8.170:2379,https://10.66.8.171:2379,https://10.66.8.172:2379"
# 网络配置前缀
KUBERNETES_CLUSTER_NETWORK_PREFIX="/kubernetes/network"
# kubernetes 服务 IP (一般是 SERVICE_CIDR 中第一个IP)
export CLUSTER_KUBERNETES_SVC_IP="10.254.0.1"
# 集群 DNS 服务 IP (从 SERVICE_CIDR 中预分配)
CLUSTER_DNS_SVC_IP="10.254.0.2"
# 集群 DNS 域名
CLUSTER_DNS_DOMAIN="wukong.local."

**# == IP分配 ==**

10.66.8.170 etcd1
10.66.8.171 etcd2
10.66.8.172 etcd3

10.66.8.168 keepalived/haproxy
10.66.8.169 keepalived/haproxy
10.66.8.201 VIP

10.66.8.168 apiserver/controller-manager/scheduler
10.66.8.169 apiserver/controller-manager/scheduler
10.66.8.201 VIP

10.66.0.10  kubelet/kube-proxy/docker/ovs
10.66.0.11  kubelet/kube-proxy/docker/ovs
10.66.0.12  kubelet/kube-proxy/docker/ovs
10.66.0.18  kubelet/kube-proxy/docker/ovs

**# ==基础部分==**
**# 以下部分在所有机器上执行**
**# 系统初始化**
timedatectl set-timezone Asia/Shanghai
setenforce 0
sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config

# 配置coredump
echo "/apps/core-%e-%s-%u-%g-%p-%t" > /proc/sys/kernel/core_pattern
echo "kernel.core_pattern = /apps/core-%e-%s-%u-%g-%p-%t" >> /etc/sysctl.conf

wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo
#wget -O /etc/yum.repos.d/epel-7.repo  http://mirrors.aliyun.com/repo/epel-7.repo

systemctl stop NetworkManager
systemctl disable NetworkManager
systemctl stop firewalld
systemctl disable firewalld
systemctl stop postfix
systemctl disable postfix

iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X

yum -y install telnet lsof python2-pip

**# ==配置证书==**
**# 配置证书**

**# 此部分只需要生成一次即可，然后拷贝到所有k8s集群的所有机器的/etc/kubernetes/ssl目录下**

```
mkdir -p /apps/soft/cfssl/; cd /apps/soft/cfssl/

# 下载cfssl工具
wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
chmod +x cfssl_linux-amd64
mv cfssl_linux-amd64 cfssl
wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
chmod +x cfssljson_linux-amd64
mv cfssljson_linux-amd64 cfssljson
wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64
chmod +x cfssl-certinfo_linux-amd64
mv cfssl-certinfo_linux-amd64 cfssl-certinfo

export PATH=/apps/soft/cfssl/:$PATH

mkdir ssl;cd ssl
cfssl print-defaults config > config.json
cfssl print-defaults csr > csr.json

# 配置ca和profile文件
# expiry，为证书的过期时间
cat >ca-config.json<<EOF
{
    "signing": {
        "default": {
            "expiry": "87600h"
        },
        "profiles": {
            "kubernetes": {
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth",
                    "client auth"
                ],
                "expiry": "87600h"
            }
        }
```

```
    }
}
EOF

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

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

# 配置kubernetes证书
# hosts：允许使用证书的IP地址,如果kube-apiserer部署了HA环境，需要将vip地址加入到下面列表
# etcd的主机地址必须要存在下面的hosts中，否则会出现认证失败
cat >kubernetes-csr.json<<EOF
{
    "CN": "kubernetes",
    "hosts": [
        "127.0.0.1",
        "10.66.8.168",
        "10.66.8.169",
        "10.66.8.170",
        "10.66.8.171",
        "10.66.8.172",
```

```
        "10.66.8.201",
        "10.254.0.1",
        "kubernetes",
        "kubernetes.default",
        "kubernetes.default.svc",
        "kubernetes.default.svc.cluster",
        "kubernetes.default.svc.cluster.local"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "BeiJing",
            "L": "BeiJing",
            "O": "k8s",
            "OU": "System"
        }
    ]
}
EOF
# 生成kubernetes证书
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes
kubernetes-csr.json | cfssljson -bare kubernetes

# 配置admin证书
cat >admin-csr.json<<EOF
{
    "CN": "admin",
    "hosts": [],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
```

```
            "ST": "BeiJing",
            "L": "BeiJing",
            "O": "system:masters",
            "OU": "System"
        }
    ]
}
EOF
# 生成admin证书
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes
admin-csr.json | cfssljson -bare admin

配置kube-proxy证书
cat >kube-proxy-csr.json  <<EOF
{
    "CN": "system:kube-proxy",
    "hosts": [],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "BeiJing",
            "L": "BeiJing",
            "O": "k8s",
            "OU": "System"
        }
    ]
}
EOF
#生成kube-proxy证书
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes
kube-proxy-csr.json | cfssljson -bare kube-proxy

# 颁发证书
mkdir -p /etc/kubernetes/ssl
cp *.pem /etc/kubernetes/ssl
```

**# ==部署ETCD==**

**# IP地址**

10.66.8.170 etcd1

10.66.8.171 etcd2

10.66.8.172 etcd3

**# 复制之前安装的证书到本地**

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

# **安装etcd**

# **etcd一般需要3台服务器，本手册也是采用3台服务器**

# 复制证书在所有的etcd的server上

mkdir -p /apps/soft/etcd/var

mkdir -p /apps/soft/etcd/bin

wget https://github.com/coreos/etcd/releases/download/v3.1.5/etcd-v3.1.5-linux-amd64.tar.gz

tar -xvf etcd-v3.1.5-linux-amd64.tar.gz

mv etcd-v3.1.5-linux-amd64/etcd* /apps/soft/etcd/bin

# 配置环境

export PATH=/apps/soft/etcd/bin:$PATH

echo 'export PATH=/apps/soft/etcd/bin:$PATH' >> /etc/profile

# 配置etcd service

# --initial-cluster=etcd-host0 为整个etcd集群的IP地址和端口

# 剩下出现IP地址的地方，全部替换成本机IP地址

cat >/etc/systemd/system/etcd.service<<EOF

[Unit]

Description=Etcd Server

After=network.target

After=network-online.target

Wants=network-online.target

Documentation=https://github.com/coreos

[Service]

Type=notify

WorkingDirectory=/apps/soft/etcd/var/

```
EnvironmentFile=-/etc/etcd/etcd.conf
ExecStart=/apps/soft/etcd/bin/etcd \\
  --name=etcd-host2 \\
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \\
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \\
  --peer-cert-file=/etc/kubernetes/ssl/kubernetes.pem \\
  --peer-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \\
  --trusted-ca-file=/etc/kubernetes/ssl/ca.pem \\
  --peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \\
  --initial-advertise-peer-urls=https://10.66.8.172:2380 \\
  --listen-peer-urls=https://10.66.8.172:2380 \\
  --listen-client-urls=https://10.66.8.172:2379,http://127.0.0.1:2379 \\
  --advertise-client-urls=https://10.66.8.172:2379 \\
  --initial-cluster-token=etcd-cluster-0 \\
  --initial-cluster=etcd-host0=https://10.66.8.170:2380,etcd-
host1=https://10.66.8.171:2380,etcd-host2=https://10.66.8.172:2380 \\
  --initial-cluster-state=new \\
  --data-dir=/apps/soft/etcd/var/
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOF

# 启动etcd服务
systemctl daemon-reload
systemctl enable etcd
systemctl start etcd
systemctl status etcd

# 配置kubernetes容器的网段
# 此部分只需要配置一次即可
# 此处需要使用etcd，如果etcd没有安装，可以等到etcd安装后再进行下面的部分
#  /kubernetes/network/config kubernetes的网络的配置路径，
# 此路径将被generate_subnet.py脚本和
# /etc/sysconfig/ovs_config文件共同使用
# {"Network":"10.66.192.0/20", "Gateway":"10.66.192.1", "SubnetLen": 25}
```

# +以上分别为： kubernetes 集群网络段，网关，每个子网段
# 需要注意，**超网的网关默认为网段内第一个地址**，为了避免docker0使用此地址，
# + **默认第一个子网不使用**，从第二个子网开始才使用
ETCDCTL_API=3 etcdctl --endpoints=https://10.66.8.170:2379 --cacert=/etc/kubernetes/ssl/ca.pem --cert=/etc/kubernetes/ssl/kubernetes.pem --key=/etc/kubernetes/ssl/kubernetes-key.pem put  /kubernetes/network/config '{"Network":"10.66.192.0/20", "Gateway":"10.66.192.1", "SubnetLen": 25}'

## # ==kubernetes master高可用环境==

**# IP地址**

10.66.8.168  keepalived/haproxy
10.66.8.169  keepalived/haproxy
10.66.8.201  VIP

**# 复制之前安装的证书到本地**

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

**# 如果不部署高可用环境，可以忽略此部分**
**# 此部分只在master/slave节点上部署**
# kubernets 的apiserver是一个无状态的服务，为了实现高可用架构，需要实现一个反向代理，
# 而controller-manager和scheduler是有状态的服务，需要选举出一个master服务。
# master 10.66.8.168
# slave 10.66.8.169

**#  安装包**

# 在master和salve上执行下面命令
yum -y install keepalived haproxy
systemctl enable keepalived
systemctl enable haproxy
# 配置非本地IP地址也可以绑定监听
echo 'net.ipv4.ip_nonlocal_bind = 1'>>/etc/sysctl.conf
sysctl -p

**# 配置keepalived**

# 在master上执行下面操作
# 10.66.8.201 这个apiserver的VIP地址
# interface eth vip地址绑定的接口
# unicast_src_ip 10.66.8.169  本机的发送心跳数据的接口

```
# priority 99 数字大的为master
cat>/etc/keepalived/keepalived.conf<<EOF
global_defs {
   notification_email {
   }
}

vrrp_script check_keepalived {
   script "/etc/haproxy/check_haproxy.sh"
   interval 2
   rise 2
   fall 2
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 198
    priority 99
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111.xxx.2
    }
    unicast_src_ip 10.66.8.168
    virtual_ipaddress {
        10.66.8.201
    }
    track_script {
        check_keepalived
    }
}
EOF

# 在salve执行下面命令
cat>/etc/keepalived/keepalived.conf<<EOF
global_defs {
   notification_email {
   }
```

```
}

vrrp_script check_keepalived {
   script "/etc/haproxy/check_haproxy.sh"
   interval 2
   rise 2
   fall 2
}

vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 198
    priority 101
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111.xxx.2
    }
    unicast_src_ip 10.66.8.169
    virtual_ipaddress {
        10.66.8.201
    }
    track_script {
        check_keepalived
    }
}
EOF

# 检测haproxy脚本
cat > /etc/haproxy/check_haproxy.sh <<EOF
#!/bin/bash
if [ \$(ps -C haproxy --no-header | wc -l) -eq 0 ]; then
     systemctl start haproxy
     exit 1
fi
EOF
chmod 755 /etc/haproxy/check_haproxy.sh
```

**# HAproxy 配置**

```
# 配置证书
cat /etc/kubernetes/ssl/admin.pem /etc/kubernetes/ssl/admin-key.pem
>/etc/kubernetes/ssl/k8s-admin.crt

# 替换haproxy启动配置
cat > /etc/systemd/system/haproxy.service <<EOF
[Unit]
Description=HAProxy Load Balancer
After=syslog.target network.target

[Service]
EnvironmentFile=/etc/sysconfig/haproxy
ExecStart=/usr/sbin/haproxy-systemd-wrapper -f /etc/haproxy/haproxy.cfg -p
/run/haproxy.pid $OPTIONS
ExecReload=/bin/kill -USR2 $MAINPID
KillMode=mixed
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF

# 配置apiserver代理，本配置仅仅代理443端口，使用tcp模式
# bind 10.66.8.201:6443  配置一个vip地址监听6443，
# 是因为10.254.0.1的service ip地址会被dnat为vip:6443端口
# server bjsh66-0-10 10.66.0.10:6443 真实的apiserver主机名和IP地址，有多少条就配置多少行
cat >/etc/haproxy/haproxy.cfg<<EOF
global
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     36000
    user        haproxy
    group       haproxy
    daemon
    stats socket /var/lib/haproxy/stats
```

```
defaults
    mode                    tcp
    log                     global
    option                  tcplog
    option                  dontlognull
    option                  redispatch
    retries                 3
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    #timeout check           10s
    timeout tunnel          2h
    maxconn                 36000

userlist usersfor_k8s
    user admin insecure-password Admin12345

frontend apiserver_vip_https
    bind 10.66.8.201:6443
    default_backend https_server

frontend apiserver_ha_https
    bind 0.0.0.0:443
    default_backend https_server

backend https_server
    option srvtcpka
    balance roundrobin
    server bjsh66-8-168 10.66.8.168:6443  check inter 10s weight 10
    server bjsh66-8-169 10.66.8.169:6443  check inter 10s weight 10

listen apiserver_ha_http
    bind 0.0.0.0:80
    mode http
    option httplog
    option forwardfor
    option redispatch
    option httpchk
```

```
    acl AuthOkay_k8s http_auth(usersfor_k8s)
    http-request auth realm KubernetesAdmin if !AuthOkay_k8s

    balance roundrobin
    server bjsh66-8-168 10.66.8.168:6443  check inter 10s weight 10 ca-file
/etc/kubernetes/ssl/ca.pem ssl crt /etc/kubernetes/ssl/k8s-admin.crt
    server bjsh66-8-169 10.66.8.169:6443  check inter 10s weight 10 ca-file
/etc/kubernetes/ssl/ca.pem ssl crt /etc/kubernetes/ssl/k8s-admin.crt

listen  admin_stats
       bind  0.0.0.0:8082
       log  global
       mode  http
       maxconn  10
       stats  enable
       #Hide  HAPRoxy version, a necessity for any public-facing site
       stats  hide-version
       stats  refresh 30s
       stats  show-node
       stats  realm Haproxy\ Statistics
       stats  auth admin:Admin12345
       stats  uri /haproxy?stats
EOF

# 配置日志记录
# 添加记录haproxy的配置进入rsyslog
cat > /etc/rsyslog.d/haproxy.conf<<EOF
\$ModLoad imudp
\$UDPServerRun 514
local2.*                       /var/log/haproxy.log
EOF

修改/etc/sysconfig/rsyslog文件
sed -i -e 's/SYSLOGD_OPTIONS=""/SYSLOGD_OPTIONS="-r -m 0 -c 2"/'
/etc/sysconfig/rsyslog

修复logrotate截断日志导致syslog丢日志bug
sed -i -e 's/syslogd.pid/rsyslogd.pid/' /etc/logrotate.d/syslog
```

# 重启rsyslogd服务
systemctl restart rsyslog

**# 启动服务**
systemctl start haproxy
systemctl enable haproxy
systemctl start keepalived
systemctl enable keepalived

**# ==创建kubernetes证书和配置文件==**
**# 以下部分只需要配置一次**

**# 下载kubernete安装包**
**# 如果在kubernetes其他的服务器上，本步操作可以不用做，因为node和server都包含kubectl命令**
wget https://dl.k8s.io/v1.6.13/kubernetes-client-linux-amd64.tar.gz
tar -zxf kubernetes-client-linux-amd64.tar.gz
cp kubernetes/client /apps/soft/kubernetes -rf
export PATH=/apps/soft/kubernetes/bin:$PATH

**# 创建kubeconfig文件**
# 创建 kubectl kubeconfig 文件
# --server制定apiserver的地址，如果apiserver是HA模式，需要指定vip地址和端口
kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/ssl/ca.pem --embed-certs=true --server=https://10.66.8.201:6443
kubectl config set-credentials admin --client-certificate=/etc/kubernetes/ssl/admin.pem --embed-certs=true --client-key=/etc/kubernetes/ssl/admin-key.pem
kubectl config set-context kubernetes --cluster=kubernetes --user=admin
kubectl config use-context kubernetes

# 创建 TLS Bootstrapping Token
**#** kube-apiserver 为客户端生成 TLS 证书的 TLS Bootstrapping 功能，这样就不需要为每个客户端生成证书了（该功能目前仅支持 kubelet ）
**#** 11c15d659e66635a09727acb05955749 bootstrapping token
# 生成token BOOTSTRAP_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')
cat > token.csv <<EOF
11c15d659e66635a09727acb05955749,kubelet-bootstrap,10001,"system:kubelet-bootstrap"

EOF

# 创建 kubelet bootstrapping kubeconfig 文件
# --server=10.66.8.201 是apiserver地址，如果apiserver是HA模式，需要指定vip地址和端口
# --token 上面创建的token字符串
kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/ssl/ca.pem --embed-certs=true --server=https://10.66.8.201:6443 --kubeconfig=bootstrap.kubeconfig
kubectl config set-credentials kubelet-bootstrap --token=11c15d659e66635a09727acb05955749 --kubeconfig=bootstrap.kubeconfig
kubectl config set-context default --cluster=kubernetes --user=kubelet-bootstrap --kubeconfig=bootstrap.kubeconfig
kubectl config use-context default --kubeconfig=bootstrap.kubeconfig

# 创建 kube-proxy kubeconfig 文件
# --server=10.66.8.201 是apiserver地址，如果apiserver是HA模式，需要指定vip地址和端口
kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/ssl/ca.pem --embed-certs=true --server=https://10.66.8.201:6443 --kubeconfig=kube-proxy.kubeconfig
kubectl config set-credentials kube-proxy --client-certificate=/etc/kubernetes/ssl/kube-proxy.pem --client-key=/etc/kubernetes/ssl/kube-proxy-key.pem --embed-certs=true --kubeconfig=kube-proxy.kubeconfig
kubectl config set-context default --cluster=kubernetes --user=kube-proxy --kubeconfig=kube-proxy.kubeconfig
kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig

**# 分发到所有的kubernetes机器上**
cp bootstrap.kubeconfig kube-proxy.kubeconfig token.csv  /etc/kubernetes/

## # ==kubernetes master安装==

# IP地址
10.66.8.168  apiserver/controller-manager/scheduler
10.66.8.169  apiserver/controller-manager/scheduler
10.66.8.201  VIP

wget https://dl.k8s.io/v1.6.13/kubernetes-server-linux-amd64.tar.gz
tar -zxf kubernetes-server-linux-amd64.tar.gz
cp kubernetes/server /apps/soft/kubernetes -rf
export PATH=/apps/soft/kubernetes/bin:$PATH
echo 'export PATH=/apps/soft/kubernetes/bin:$PATH' >> /etc/profile

**# 复制之前创建好的kubernetes的证书配置文件到本地**

cp bootstrap.kubeconfig kube-proxy.kubeconfig token.csv  /etc/kubernetes/

**# 此部分，如果部署高可用环境，在每个master/slave上都需要部署**

**# 部署apiserver**

# 配置kube-apiserver.service

# --service-cluster-ip-range 配置kubernetes service ip 的网段

# --service-node-port-range 服务绑定物理网卡的端口范围

# --advertise-address=10.66.8.201 **如果配置高可用环境，这里需要配置VIP地址**

# --bind-address=10.66.8.168 绑定本地物理网卡的地址

```
mkdir -p /apps/logs/kubernetes/
cat >/etc/systemd/system/kube-apiserver.service<<EOF
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
User=root
ExecStart=/apps/soft/kubernetes/bin/kube-apiserver \\
  --admission-
control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota
 \\
  --advertise-address=10.66.8.201 \\
  --allow-privileged=true \\
  --apiserver-count=1 \\
  --audit-log-maxage=30 \\
  --audit-log-maxbackup=3 \\
  --audit-log-maxsize=100 \\
  --audit-log-path=/apps/logs/kubernetes/audit.log \\
  --authorization-mode=RBAC \\
  --bind-address=10.66.8.168 \\
  --client-ca-file=/etc/kubernetes/ssl/ca.pem \\
  --enable-swagger-ui=true \\
  --etcd-cafile=/etc/kubernetes/ssl/ca.pem \\
```

```
  --etcd-certfile=/etc/kubernetes/ssl/kubernetes.pem \\
  --etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem \\
  --etcd-servers=https://10.66.8.170:2379,https://10.66.8.171:2379,https://10.66.8.172:2379 \\
  --event-ttl=12h \\
  --kubelet-https=true \\
  --insecure-bind-address=10.66.8.168 \\
  --runtime-config=rbac.authorization.k8s.io/v1alpha1 \\
  --service-account-key-file=/etc/kubernetes/ssl/ca-key.pem \\
  --service-cluster-ip-range=10.254.0.0/16 \\
  --service-node-port-range=30001-32767 \\
  --tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem \\
  --tls-private-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \\
  --experimental-bootstrap-token-auth \\
  --token-auth-file=/etc/kubernetes/token.csv \\
  --event-ttl=12h0m0s \\
  --v=5
Restart=on-failure
RestartSec=5
Type=notify
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOF

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

# **安装kubernetes-controller**

```
#  --cluster-cidr 配置kubernetes网络的地址段，一般为一个超网
# --service-cluster-ip-range 配置kubernetes service ip 的网段
# --leader-elect=true 部署多台机器组成的 master 集群时选举产生一处于工作状态的 kube-controller-manager 进程
# --master= 如果是HA模式，此处需要配置为127.0.0.1，因为apiserver和controller-mananger安装在同一台服务器上
cat >/etc/systemd/system/kube-controller-manager.service<<EOF
[Unit]
```

```
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=/apps/soft/kubernetes/bin/kube-controller-manager \\
  --address=127.0.0.1 \\
  --allocate-node-cidrs=true \\
  --cluster-cidr=10.66.192.0/20 \\
  --cluster-name=kubernetes \\
  --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem \\
  --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem \\
  --leader-elect=true \\
  --master=http://127.0.0.1:8080 \\
  --root-ca-file=/etc/kubernetes/ssl/ca.pem \\
  --service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem \\
  --service-cluster-ip-range=10.254.0.0/16 \\
  --pod-eviction-timeout=2m0s \\
  --v=5
Restart=on-failure
RestartSec=5
#Type=notify

[Install]
WantedBy=multi-user.target
EOF

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

**# 安装kube-scheduler**

```
# --address 值必须为 127.0.0.1 ，因为当前 kube-apiserver 期望 scheduler 和 controller-manager 在同一台机器
# --leader-elect=true 部署多台机器组成的 master 集群时选举产生一处于工作状态的 kube-controller-manager 进程
```

```
cat >/etc/systemd/system/kube-scheduler.service <<EOF
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=/apps/soft/kubernetes/bin/kube-scheduler \\
  --leader-elect=true \\
  --master=http://127.0.0.1:8080 \\
  --address=127.0.0.1 \\
  --v=5
Restart=on-failure
RestartSec=5
#Type=notify

[Install]
WantedBy=multi-user.target
EOF

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

**查看集群状态**

```
kubectl get cs
```

## # ==部署kubernetes node==

**# IP地址**

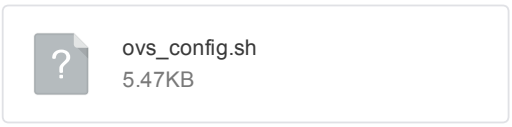
```
10.66.0.10  kubelet/kube-proxy/docker/ovs
10.66.0.11  kubelet/kube-proxy/docker/ovs
10.66.0.12  kubelet/kube-proxy/docker/ovs
10.66.0.18  kubelet/kube-proxy/docker/ovs
```

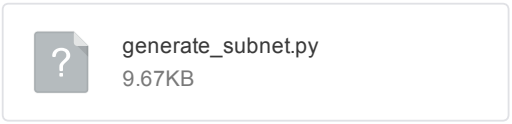
## # == 部署node的网络和docker ==

**# 部署网络**

**# 脚本**

**# 本脚本需要复制到/apps/soft/kubernetes/bin/目录下, 并赋予执行权限**




mkdir -p /apps/soft/kubernetes/bin/

**复制文件**

chmod 755 /apps/soft/kubernetes/bin/generate_subnet.py

chmod 755 /apps/soft/kubernetes/bin/ovs_config.sh

**# 安装新内核**

# 因需要使用overlay2和xfs dtype功能，已经修复一些bug

# + xfs bug: input/output error

# + docker bug:  kernel crash in xfs_vm_writepage - kernel BUG at fs/xfs/xfs_aops.c:1062!

wget http://repos.lax-noc.com/elrepo/archive/kernel/el7/x86_64/RPMS/kernel-lt-4.4.105-1.el7.elrepo.x86_64.rpm

rpm -ivh kernel-lt-4.4.105-1.el7.elrepo.x86_64.rpm

# 将新内核设为默认启动

grub2-set-default 0

**# 格式化xfs 开启d_type支持**

# 将apps目录中的数据备份

cp -rfa /apps /apps.bak

# df  -h 查看/apps目录挂载的设备，然后umount

# 需要结束掉heka进程，因为heka进程文件在/apps目录下

umount /apps

# 格式化

mkfs.xfs -f -n ftype=1 /dev/sda5

# 格式化磁盘后，磁盘的源UUID会改版，需要查找新的uuid，ls /dev/disk/by-uuid/ -lh

# 然后修改/etc/fstab文件，替换UUID，然后需要重启系统才能生效

ls -l /dev/disk/by-uuid |grep sda5

vim /etc/fstab

# 恢复数据

```
mv /apps.bak/*  /apps/

# 重启
reboot

# 复制etcdctl工具
mkdir -p /apps/soft/etcd/bin/
# 从etcd的机器上复制etcdctl工具到本地的/apps/soft/etcd/bin
chmod 755 /apps/soft/etcd/bin/etcdctl

# 配置环境
export PATH=/apps/soft/etcd/bin:$PATH
echo 'export PATH=/apps/soft/etcd/bin:$PATH' >> /etc/profile

# 安装etcd3模块
# generate_subnet.py脚本需要安装etcd3模块
# 如果pip更新慢，可以使用阿里云的代理
mkdir -p ~/.pip
cat > ~/.pip/pip.conf <<EOF
[global]
index-url = http://mirrors.aliyun.com/pypi/simple/

[install]
trusted-host=mirrors.aliyun.com
EOF

pip install etcd3

# 安装open vswitch
yum -y install make gcc openssl-devel autoconf automake rpm-build redhat-rpm-config
yum -y install python-devel openssl-devel kernel-devel kernel-debug-devel libtool wget
bridge-utils
mkdir -p ~/rpmbuild/SOURCES
cd ~; wget http://openvswitch.org/releases/openvswitch-2.5.0.tar.gz
cp openvswitch-2.5.0.tar.gz ~/rpmbuild/SOURCES/
tar xfz openvswitch-2.5.0.tar.gz
sed 's/openvswitch-kmod, //g' openvswitch-2.5.0/rhel/openvswitch.spec > openvswitch-
2.5.0/rhel/openvswitch_no_kmod.spec
rpmbuild -bb --nocheck ~/openvswitch-2.5.0/rhel/openvswitch_no_kmod.spec
```

```
yum -y localinstall ~/rpmbuild/RPMS/x86_64/openvswitch-2.5.0-1.x86_64.rpm

systemctl enable openvswitch.service
systemctl start openvswitch.service

# 安装docker
wget https://download.docker.com/linux/centos/docker-ce.repo -O
/etc/yum.repos.d/docker-ce.repo
yum -y install docker-ce-17.06.2.ce-1.el7.centos.x86_64

# 配置docker
# 因为使用了ovs为网络解决方案，因此需要关闭masq和iptables转发
# --ip-masq=false --iptables=false
# --userland-proxy=true 允许容器内访问另一个容器，通过宿主机映射的port
# --insecure-registry 配置容器的仓库地址
# --data-root docker数据的存放地
cat >/etc/sysconfig/docker<<EOF
DOCKER_OPTS="--registry-mirror=https://registry.docker-cn.com --live-restore=true --ip-
masq=false --iptables=false  --log-level=info --userland-proxy=true --log-driver=json-file --
log-opt=max-size=100m --log-opt=max-file=5"
DOCKER_STORAGE_OPTIONS="--data-root=/apps/docker --storage-driver=overlay2"
INSECURE_REGISTRY="--insecure-registry=reg.qianbao-inc.com"
EOF

# ExecStartPre=/apps/soft/kubernetes/bin/generate_subnet.py
"\$GENERATE_SUBNET_ARGS"
# + 从etcd获取kubernetes cluster network的超网地址，然后在生成子网地址
# ExecStartPost=/apps/soft/kubernetes/bin/ovs-config.sh "\$OVS_CONFIG_ARGS"
#  + 配置ovs、docker、路由等等的配置，整个kubernetes的网络都在此处配置
# ExecStopPost=/usr/sbin/sysctl -w net.ipv4.conf.docker0.send_redirects=0
# ExecStopPost=/usr/sbin/sysctl -w net.ipv4.conf.all.send_redirects=0
# + 以上两行主要是关闭icmp redirect
cat >/etc/systemd/system/docker.service<<EOF
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target firewalld.service
Wants=network-online.target
```

```
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker

EnvironmentFile=/etc/sysconfig/docker
EnvironmentFile=/etc/sysconfig/ovs_config
EnvironmentFile=-/etc/sysconfig/kubernets_cluster_network

ExecStartPre=/apps/soft/kubernetes/bin/generate_subnet.py "\$GENERATE_SUBNET_ARGS"
ExecStart=/usr/bin/dockerd \$INSECURE_REGISTRY \$DOCKER_STORAGE_OPTIONS
\$DOCKER_OPTS \$DOCKER_OPT_BIP
ExecStartPost=/apps/soft/kubernetes/bin/ovs_config.sh "\$OVS_CONFIG_ARGS"
ExecStopPost=/usr/sbin/sysctl -w net.ipv4.conf.docker0.send_redirects=0
ExecStopPost=/usr/sbin/sysctl -w net.ipv4.conf.all.send_redirects=0
ExecReload=/bin/kill -s HUP  \$MAINPID
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
# Uncomment TasksMax if your systemd version supports it.
# Only systemd 226 and above support this version.
#TasksMax=infinity
TimeoutStartSec=0
# set delegate yes so that systemd does not reset the cgroups of docker containers
Delegate=yes
# kill only the docker process, not all processes in the cgroup
KillMode=process
# restart the docker process if it exits prematurely
Restart=on-failure
RestartSec=5
#StartLimitBurst=3#
#StartLimitInterval=60s

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

# 配置docker的环境变量，为ovs的相关配置
# iface 管理接口的地址， -addrs 为etcd的服务器地址+端口列表，可以配置多个地址，用逗号分隔
# vip_addr 如果本机上也部署了kubernetes master ，并且是HA模式，那么这里需要配置vip地址
# manage_gateway 管理网络的默认网关地址，也就是eth2端口，默认网关是主机地址最后一位是1
# + 例如：10.66.0.11 它的默认网关就是10.66.0.1，如果要改变，则需要修改此参数
# etcd_endpoints etcd的endpoints地址
# bond_iface 为运行trunk的网卡，跑业务流量
**# vlan_id  vlan id，kubernetes网络的vlan号**
cat >/etc/sysconfig/ovs_config << EOF
GENERATE_SUBNET_ARGS="-iface eth2 -addrs
10.66.8.170:2379,10.66.8.171:2379,10.66.8.172:2379"
OVS_CONFIG_ARGS="--bond_iface bond0 --vlan_id 1237 --manage_iface eth2 --
etcd_endpoints https://10.66.8.170:2379,https://10.66.8.171:2379,https://10.66.8.172:2379"
EOF

# 从master处拷贝证书和/root/.kube目录
cp kube.tgz /root/
cd /root; tar zxf kube.tgz
cp kubernetes.tgz /etc/
cd /etc; tar -zxf kubernetes.tgz

**# 需要先部署etcd、配置好kubernets网络、ovs的配置、复制脚本、拷贝证书，这些操作完成之后才能启动docker**
systemctl daemon-reload
systemctl enable docker
systemctl start docker

## # ==部署kubernetes的node部分 ==
**# 安装node**
wget https://dl.k8s.io/v1.6.13/kubernetes-node-linux-amd64.tar.gz
tar -zxf kubernetes-node-linux-amd64.tar.gz
cp kubernetes/node/bin /apps/soft/kubernetes/ -rf
export PATH=/apps/soft/kubernetes/bin:$PATH
echo 'export PATH=/apps/soft/kubernetes/bin:$PATH' >> /etc/profile

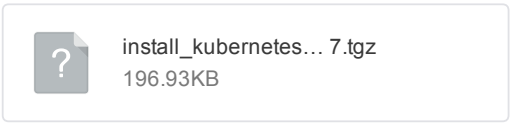
# **配置clusterrolebinding**
# **本操作只需要执行一次**

```
#  kubelet 启动时向 kube-apiserver 发送 TLS bootstrapping 请求，需要先将bootstrap token 文件中的 kubelet-bootstrap 用户赋予 system:node-bootstrapper角色，
# 然后 kublet 才有权限创建认证请求(certificatesigningrequests)：
kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-bootstrapper --user=kubelet-bootstrap

# 安装kubelet
# --address kubelet绑定的本机IP地址
# --hostname-override 设定本机的主机名
# --cluster_dns 指定dns的地址
# --cluster_domain 指定集群名称 默认是cluster.local
# --pod-infra-container-image 指定pod的下载地址，需要修改为内部的仓库，
# 默认是国外的仓库，经常出现无法下载的情况，reg.qianbao-inc.com/base/pod-infrastructure:latest
# 创建工作目录
mkdir -p /apps/soft/kubernetes/lib/kubelet

# 创建kubelet.service
cat >/etc/systemd/system/kubelet.service<<EOF
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=docker.service
Requires=docker.service

[Service]
WorkingDirectory=/apps/soft/kubernetes/lib/kubelet
ExecStart=/apps/soft/kubernetes/bin/kubelet \\
  --address=10.66.0.11 \\
  --hostname-override=10.66.0.11 \\
  --pod-infra-container-image=reg.qianbao-inc.com/base/pod-infrastructure:latest \\
  --experimental-bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig \\
  --kubeconfig=/etc/kubernetes/kubelet-kubeconfig \\
  --require-kubeconfig \\
  --cert-dir=/etc/kubernetes/ssl \\
  --container-runtime=docker \\
  --cluster_dns=10.254.0.2 \\
  --cluster_domain=cluster.local. \\
  --hairpin-mode promiscuous-bridge \\
```

```
  --allow-privileged=true \\
  --serialize-image-pulls=false \\
  --register-node=true \\
  --logtostderr=true \\
  --kube-reserved=memory=1Gi \\
  --v=5
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF

systemctl daemon-reload
systemctl enable kubelet
systemctl start kubelet

# csr授权
#  如果正常情况下，每个node都需要生成一个csr证书，授权给一个node，但是可以通过复制已近授权的node的证书文件，这样就不用每次都授权了，因此本操作执行一次就可以
# 执行下面的命令，找到Name下的证书名，例如csr-2b308
kubectl get csr
# 列出证书，然后执行授权动作
kubectl certificate approve csr-2b308

# 安装kube-proxy
cat >/etc/systemd/system/kube-proxy.service <<EOF
[Unit]
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=/apps/soft/kubernetes/bin/kube-proxy \\
  --bind-address=10.66.0.11 \\
  --cluster-cidr=10.254.0.0/16 \\
   --hostname-override=10.66.0.11 \\
```

```
  --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig \\
  --logtostderr=true \\
  #--log-dir=/var/log/kubernetes/ \\
  --v=5
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOF

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

**# ==部署辅助模块==**
**# 此部分只需要部署一次**
**# 此部分用到的yaml文件，基本上已经修改完成了，不需要修改，下面是对比出部分修改的地方。**
**# 以下需要的yaml文件均已经打包在本附件中**
**# 此部分使用diff对比两个配置文件的修改部分**




**# 安装kube-dns**

```
# 需要将 spec.clusterIP 设置为集群环境变量中变量 CLUSTER_DNS_SVC_IP值，
#  +这个 IP 需要和 kubelet 的 —cluster-dns 参数值一致

# 修改kubedns-svc.yaml
diff kubedns-svc.yaml.base kubedns-svc.yaml
30c30
< clusterIP: __PILLAR__DNS__SERVER__
---
> clusterIP: 10.254.0.2

# 修改kubedns-controller.yaml
# --domain 为集群环境文档 变量 CLUSTER_DNS_SVC_DOMAIN 的值；
```

```
#  + 使用系统已经做了 RoleBinding 的 kube-dns ServiceAccount，
#  + 该账户具有访问 kube-apiserver DNS 相关 API 的权限
diff kubedns-controller.yaml.base kubedns-controller.yaml
58c58
< image: gcr.io/google_containers/k8s-dns-kube-dns-amd64
:1.14.1
---
> image: zhaixigui/k8s-dns-kube-dns-amd64:v1.14.4
88c88
< - --domain=__PILLAR__DNS__DOMAIN__.
---
> - --domain=cluster.local.
92c92
< __PILLAR__FEDERATIONS__DOMAIN__MAP__
---
> #__PILLAR__FEDERATIONS__DOMAIN__MAP__
110c110
< image: gcr.io/google_containers/k8s-dns-dnsmasq-nannyamd64:
1.14.1
---
> image: zhaixigui/k8s-dns-dnsmasq-nanny-amd64:v1.14.4
129c129
< - --server=/__PILLAR__DNS__DOMAIN__/127.0.0.1#10053
---
> - --server=/cluster.local./127.0.0.1#10053
148c148
< image: gcr.io/google_containers/k8s-dns-sidecar-amd64:
1.14.1
---
> image: zhaixigui/k8s-dns-sidecar-amd64:v1.14.4
161,162c161,162
< - --probe=kubedns,127.0.0.1:10053,kubernetes.default.s
vc.__PILLAR__DNS__DOMAIN__,5,A
< - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.
__PILLAR__DNS__DOMAIN__,5,A
---
> - --probe=kubedns,127.0.0.1:10053,kubernetes.default.s
vc.cluster.local.,5,A
> - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.
```

cluster.local.,5,A

# 部署kubedns pods
# 进入install_kubernetes_yaml/kube-dns的yaml目录
kubectl create -f .

# **部署dashboard**
# 进入install_kubernetes_yaml/dashboard的yaml目录
**# 此部分使用diff对比两个配置文件的修改部分**
diff dashboard-service.yaml.orig dashboard-service.yaml
10a11
> type: NodePort

diff dashboard-controller.yaml
20a21
> serviceAccountName: dashboard
23c24
< image: gcr.io/google_containers/kubernetes-dashboard-a
md64:v1.6.0
---
> image: cokabug/kubernetes-dashboard-amd64:v1.6.0

# 执行部署命令
# 进入install_kubernetes_yaml/dashboard的yaml目录
kubectl create -f .

# 查看集群信息
kubectl cluster-info

# 执行完成后，可使用下面的连接访问dashboard
http://172.28.40.76:8080/api/v1/proxy/namespaces/kubesystem/
services/kubernetes-dashboard

# **部署监控**
# 进入install_kubernetes_yaml目录的heapster下
# 部署grafana
# 如果后续使用 kube-apiserver 或者 kubectl proxy 访问 grafana dashboard，

```
# +则必须将 GF_SERVER_ROOT_URL 设置为
/api/v1/proxy/namespaces/kubesystem/services/monitoring-grafana/ ，
# +否则后续访问grafana时访问时提示找不到
# +http://10.64.3.7:8086/api/v1/proxy/namespaces/kubesystem/services/monitoring-
grafana/api/dashboards/home 页面；
# value: /api/v1/proxy/namespaces/kube-system/services
#

# 部署heapstr
# heapster.yaml
# 在template下的spec下添加：
#  serviceAccountName: heapster
# config.toml文件需要修改
#  enabled = true

# influxdb 部署
# 进入install_kubernetes_yaml目录的heapster下
# 修改influxdb.yaml文件
# 在 volumeMounts添加如下：
#        volumeMounts:
#        - mountPath: /etc/
#          name: influxdb-config
# 在 volumes 添加如下：
#      volumes:
#      - name: influxdb-config
#        configMap:
#          name: influxdb-config

# 部署ingress
# 生成私有的d.k8s.me证书，用于dashboard
# 生成 CA 自签证书
mkdir cert && cd cert
openssl genrsa -out ca-key.pem 2048
openssl req -x509 -new -nodes -key ca-key.pem -days 10000 -out ca.pem -subj "/CN=kube-
ca"

# 编辑 openssl 配置
cp /etc/pki/tls/openssl.cnf .
vim openssl.cnf
```

```
# 主要修改如下
[req]
req_extensions = v3_req # 这行默认注释关着的 把注释删掉
# 下面配置是新增的
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = d.k8s.me
#DNS.2 = kibana.mritd.me

# 生成证书
openssl genrsa -out ingress-key.pem 2048
openssl req -new -key ingress-key.pem -out ingress.csr -subj "/CN=d.k8s.me" -config
openssl.cnf
openssl x509 -req -in ingress.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out
ingress.pem -days 365 -extensions v3_req -extfile openssl.cnf

# 创建secret
kubectl create secret tls d.k8s.me-secret --namespace=kube-system --key cert/ingress-
key.pem --cert cert/ingress.pem

# 安装ingress
# 进入/root/install_kubernetes_yaml/ingress
kubectl create -f .

# 访问dashboard的ingress
https://d.k8s.me


# 安装自研的watch服务
# 下载Wukong的代码到/apps/目录
# 复制ENV27的virtualENV环境到/usr/src下

# 安装supervisor
/usr/src/ENV27/bin/pip install supervisor==3.3.1
```

```
# 配置watch的supervisird
[group:wukong]
programs=wukong_watch

[program:wukong_watch]
command=/usr/src/ENV27/bin/python2.7
/apps/WuKong/src/wk_server/celery_wk/cluster_watch_nginx.py
user=root
startretries=300
autorestart=true
logfile=/apps/soft/supervisird/log/wukong/wukong_watch.log
stdout_logfile=/apps/soft/supervisird/log/wukong/wukong_watch.log
stderr_logfile=/apps/soft/supervisird/log/wukong/wukong_watch.err

# 启动
/usr/src/ENV27/bin/python2.7 /usr/src/ENV27/bin/supervisord

# 添加自启动
echo '/usr/src/ENV27/bin/python2.7 /usr/src/ENV27/bin/supervisord' >> /etc/rc.d/rc.local
```