生产环境 Kubernetes 集群部署

版本	描述	时间	编写
V1.0	Kubernetes 集群部署方案	2019-03-14	迷城

目　录

1 前言

概述

本文档详细的描述了 Kubenetes1.13.6 生产环境集群部署以及监控、存储、容灾、网络规划、故障处理操作，通过故障处理，可以及时解决 Kubenetes 系统在运行时发生的故障，使 Kubenetes 系统能够正常运行。

读者对象

本文档主要适用系统维护工程师操作，维护人员必须具备以下经验和技能：

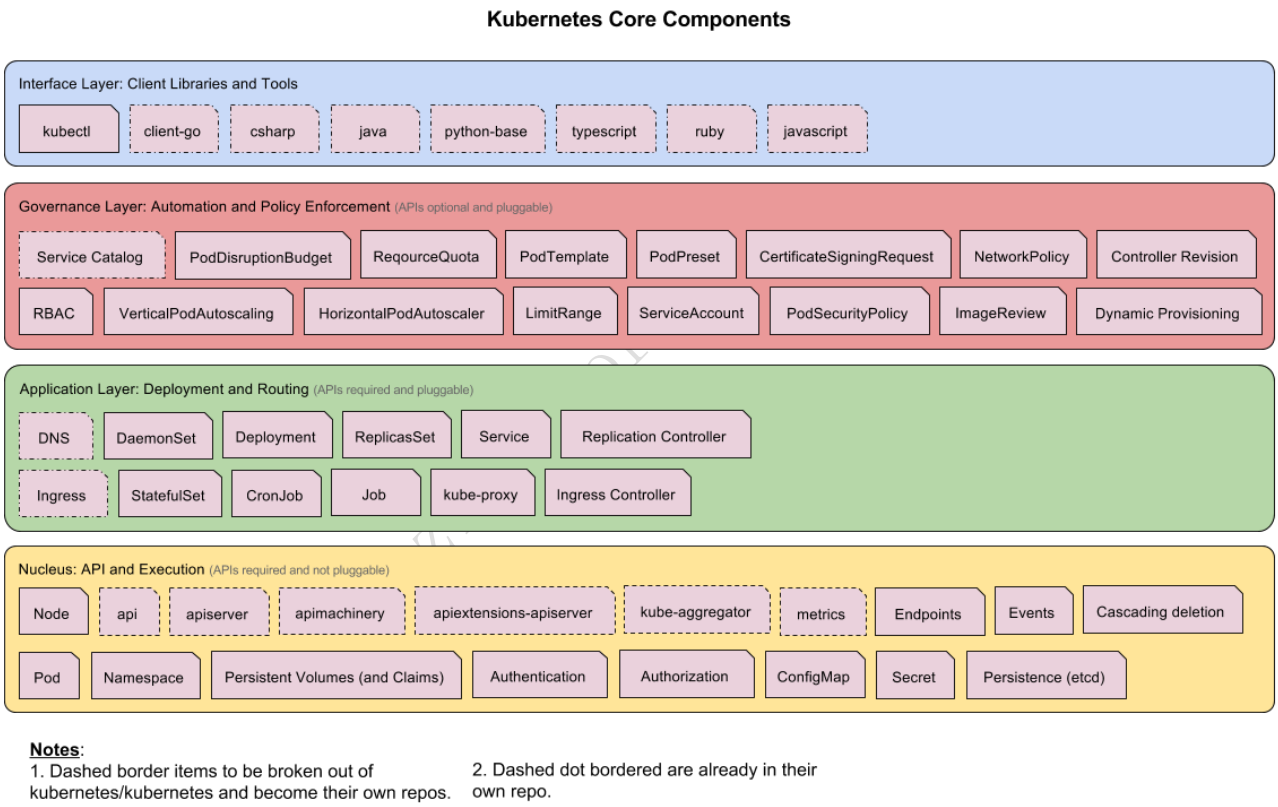
- 熟悉当前网络的组网和相关网元的版本信息。
- 有 kubernetes 维护经验，熟悉 docker、kubernetes 的日常操作维护方式。

网络规划

请自行根据公司业务进行规划

2 架构说明

2.1 系统架构概述



Kubernetes 主要由以下几个核心组件组成：

- etcd 保存了整个集群的状态；
- kube-apiserver 提供了资源操作的唯一入口，并提供认证、授权、访问控制、API 注册和发现等机制；
- kube-controller-manager 负责维护集群的状态，比如故障检测、自动扩展、滚动更新等；
- kube-scheduler 负责资源的调度，按照预定的调度策略将 Pod 调度到相应的机器上；
- kubelet 负责维持容器的生命周期，同时也负责 Volume（CVI）和网络（CNI）的管理；

- Container runtime 负责镜像管理以及 Pod 和容器的真正运行（CRI），默认的容器运行时为 Docker；
- kube-proxy 负责为 Service 提供 cluster 内部的服务发现和负载均衡；

除了核心组件，还有一些推荐的 Add-ons：

- kube-dns 负责为整个集群提供 DNS 服务
- Ingress Controller 为服务提供外网入口
- Heapster 提供资源监控
- Dashboard 提供 GUI
- Federation 提供跨可用区的集群
- Fluentd-elasticsearch 提供集群日志采集、存储与查询

2.2 安装前准备

2.2.1 获取软件及工具

表 1 软件包下载地址

软件包名称	说明	下载路径
CentOS-7-x86_64-Minimal-1810.iso	Centos7 操作系统镜像	✧ https://mirrors.aliyun.com/centos/7/isos/x86_64/CentOS-7-x86_64-Minimal-1810.iso
kubernetes-server-linux-amd64.tar.gz	Kubernetes master 节点的安装包，请获取所有包后解压。	✧ 访问 https://github.com/kubernetes，进入“kubernets > releases ”。选择对应版本后，下载安装包。
kubernetes-node-linux-amd64.tar.gz	Kubernetes Node 节点的安装包。	✧ https://dl.k8s.io/v1.13.6/kubernetes-node-linux-amd64.tar.gz
etcd-v3.3.10-linux-amd64.tar.gz	etcd 是一个开源的分布式键值存储，为 Container Linux 集群提供共享配置和服务发现。	✧ https://github.com/etcd-io/etcd/releases/download/v3.3.10/etcd-v3.3.10-linux-amd64.tar.gz
flannel-v0.11.0-linux-amd64.tar.gz	Container Network CNI plugin	✧ https://github.com/coreos/flannel/releases/download/v0.11.0/flannel-v0.11.0-linux-amd64.tar.gz

2.2.2 安装规划

Pod 分配 IP 段： 10.244.0.0/16
ClusterIP 地址： 10.99.0.0/16
CoreDns 地址： 10.99.110.110
统一安装路径： /data/apps/

主机名 (HostName)	IP 地址	角色 (Role)	组件 Component	集群 IP (Vip)
K8S-PROD-MASTER-A1	10.211.18.4	Master	Kube-apiserver 、 kube-controller-manager、kube-scheduler、etcd、kube-proxy、docker、flanel、keepalived、haproxy	10.211.18.10
K8S-PROD-MASTER-A2	10.211.18.5	Master		
K8S-PROD-MASTER-A3	10.211.18.6	Master		
K8S-PROD-NODE-A1	10.211.18.11	Node	Kubelet 、 kube-proxy 、 docker 、 flannel、node_exporter	不涉及
K8S-PROD-NODE-A2	10.211.18.12	Node		不涉及
K8S-PROD-NODE-A3	10.211.18.13	Node		不涉及
K8S-PROD-NODE-A4	10.211.18.14	Node		不涉及
K8S-PROD-LB-A1	10.211.18.50	Ingress	Keepalived Ingress	10.211.18.100
K8S-PROD-LB-B1	10.211.18.51	Ingress		
K8S-PROD-REGISTR-A1	10.211.18.61	Registr		
K8S-PROD-REGISTR-B1	10.211.18.62	Registr		
Ceph 集群	/	ceph		

2.2.3 修改主机名/关闭 selinux

操作步骤：

步骤 1 以 root 用户登录所有节点。

步骤 2 执行以下命令修改主机名、关闭 selinux。

```
[root@localhost ~]#  hostnamectl  --static set-hostname  K8S-PROD-NODE-A1

#关闭 Selinux

[root@localhost ~]#  sed -i 's/SELINUX=.*/SELINUX=disabled/g'
/etc/selinux/config
```

2.2.4 升级系统内核/同步系统时间

注意： Overlay 需要内核版本在 3.12+，所以在安装完 centos7.6 之后要进行内核升级

步骤 1 以 root 用户登录所有节点。

步骤 2 安装时间同步软件

```
[root@K8S-PROD-MASTER-A1 ~]# yum install chrony –y
```

步骤 3 修改/etc/chrony.conf 配置文件， 增加如下内容。

server time.aliyun.com iburst

步骤 4 安装必备软件

```
yum install wget vim gcc git lrzsz net-tools tcpdump telnet  rsync -y
```

✧ 启动 chronyd

```
[root@K8S-PROD-MASTER-A1 ~]# systemctl  start chronyd
```

✧ 调整内核参数

```
cat >  /etc/sysctl.d/k8s.conf  << EOF
net.ipv4.tcp_keepalive_time = 600
net.ipv4.tcp_keepalive_intvl = 30
net.ipv4.tcp_keepalive_probes = 10
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
net.ipv4.neigh.default.gc_stale_time = 120
net.ipv4.conf.all.rp_filter = 0
net.ipv4.conf.default.rp_filter = 0
net.ipv4.conf.default.arp_announce = 2
net.ipv4.conf.lo.arp_announce = 2
```

```
net.ipv4.conf.all.arp_announce = 2
```

```
net.ipv4.ip_forward = 1
net.ipv4.tcp_max_tw_buckets = 5000
net.ipv4.tcp_syncookies = 1
```

```
net.ipv4.tcp_max_syn_backlog = 1024
net.ipv4.tcp_synack_retries = 2

net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-arptables = 1
net.netfilter.nf_conntrack_max = 2310720
fs.inotify.max_user_watches=89100
fs.may_detach_mounts = 1
fs.file-max = 52706963
fs.nr_open = 52706963
EOF
执行以下命令使修改生效
sysctl --system
modprobe br_netfilter
```

2.2.5 加载 ipvs 模块

```
cat << EOF > /etc/sysconfig/modules/ipvs.modules
#!/bin/bash
ipvs_modules_dir="/usr/lib/modules/\`uname -
r\`/kernel/net/netfilter/ipvs"
for i in \`ls \$ipvs_modules_dir | sed -r 's#(.*).ko.xz#\1#'\`; do
/sbin/modinfo -F filename \$i &> /dev/null
if [ \$? -eq 0 ]; then
/sbin/modprobe \$i
fi
done
EOF

chmod +x /etc/sysconfig/modules/ipvs.modules
bash /etc/sysconfig/modules/ipvs.modules
```


✧ 导入 elrepo Key

步骤 1 导入 KEY

```
[root@K8S-PROD-MASTER-A1~]# rpm -import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
```

步骤 2 安装 elrepo 的 yum 源

```
[root@K8S-PROD-MASTER-A1 ~]#  rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-2.el7.elrepo.noarch.rpm

Retrieving http://www.elrepo.org/elrepo-release-7.0-2.el7.elrepo.noarch.rpm
Retrieving http://elrepo.org/elrepo-release-7.0-3.el7.elrepo.noarch.rpm
Preparing...                          ################################
[100%]
Updating / installing...
   1:elrepo-release-7.0-3.el7.elrepo  ################################
[100%]
```

✧ 使用以下命令列出可用的内核相关包

步骤 3 执行如下命令查看可用内核

yum --disablerepo="*" --enablerepo="elrepo-kernel" list available

```
elrepo-kernel
elrepo-kernel/primary_db
Available Packages
kernel-lt.x86_64                                                4.4.176-1.el7.elrepo
kernel-lt-devel.x86_64                                          4.4.176-1.el7.elrepo
kernel-lt-doc.noarch                                            4.4.176-1.el7.elrepo
kernel-lt-headers.x86_64                                        4.4.176-1.el7.elrepo
kernel-lt-tools.x86_64                                          4.4.176-1.el7.elrepo
kernel-lt-tools-libs.x86_64                                     4.4.176-1.el7.elrepo
kernel-lt-tools-libs-devel.x86_64                               4.4.176-1.el7.elrepo
kernel-ml.x86_64                                                5.0.1-1.el7.elrepo
kernel-ml-devel.x86_64                                          5.0.1-1.el7.elrepo
kernel-ml-doc.noarch                                            5.0.1-1.el7.elrepo
kernel-ml-headers.x86_64                                        5.0.1-1.el7.elrepo
kernel-ml-tools.x86_64                                          5.0.1-1.el7.elrepo
kernel-ml-tools-libs.x86_64                                     5.0.1-1.el7.elrepo
kernel-ml-tools-libs-devel.x86_64                               5.0.1-1.el7.elrepo
perf.x86_64                                                     5.0.1-1.el7.elrepo
python-perf.x86_64                                              5.0.1-1.el7.elrepo
```

步骤 4 安装 4.4.176 内核

```
[root@K8S-PROD-MASTER-A1 ~]# yum --enablerepo=elrepo-kernel install kernel-lt kernel-lt-devel -y
```

步骤 5　更改内核默认启动顺序

```
[root@K8S-PROD-MASTER-A1 ~]# grub2-set-default 0
[root@K8S-PROD-MASTER-A1 ~]# reboot
```

步骤 6　验证内核是否升级成功

```
[root@K8S-PROD-LB-A1 ~]# uname -rp
4.4.176-1.el7.elrepo.x86_64 x86_64
```

2.2.6 上传软件包

参考表 1 将需上传的软件包或者脚本上传到 Master 和 Node 节点的指定目录下

待上传包/脚本	说明	上传路径
kubernetes-server-linux-amd64.tar.gz	Kubernetes Master 节点所需安装包。	/opt/software
kubernetes-node-linux-amd64.tar.gz	Kubernetes Node 节点所需安装包。	/opt/software
etcd-v3.3.10-linux-amd64.tar.gz	Etcd 键值存储数据库	/opt/software
flannel-v0.11.0-linux-amd64.tar.gz	CNI 网络插件	/opt/software

2.2.7 防火墙放行端口

Master Node Inbound			
Protocol	Port Range	Source	Purpose
TCP	6443 8443	Worker Node,API Requests	Kubernetes API Server
UDP	8285	Master & Worker Nodes	Flannel overlay network – udp backend
UDP	8472	Master & Worker Nodes	Flannel overlay network – vxlan backend
Worker Node Inbound			
TCP	10250	Master Nodes	Worker node Kubelet API for exec and logs.
TCP	10255	Heapster	Worker node read-only Kubelet API.
TCP	30000-40000	External Application Consumers	Default port range for external service ports. Typically, these ports would need to be exposed to external load-balancers, or other external consumers of the application itself.
UDP	8285	Master & Worker Nodes	flannel overlay network - udp backend. This is the default network configuration (only required if using flannel)
UDP	8472	Master & Worker Nodes	flannel overlay network - vxlan backend (only required if using flannel)
TCP	179	Worket Nodes	Calico BGP network (only required if the BGP backend is used)
Etcd Node Inbound			
TCP	2379-2380	Master Nodes	etcd server client API
TCP	2379-2380	Worker Nodes	etcd server client API (only required if using flannel or Calico).
Ingress			
TCP	80	Ingress Nodes	http
TCP	443	Ingress Nodes	https
TCP	8080	Ingress Nodes	other

2.3 安装 Docker

步骤 1　安装存储驱动

```
yum install -y yum-utils \
device-mapper-persistent-data \
lvm2
```

步骤 2　添加 YUM 仓库

```
yum-config-manager --add-repo
https://mirrors.aliyun.com/docker-ce/linux/centos/docker-
ce.repo
```

步骤 3　列出所有版本的 Docker CE

```
#  yum list docker-ce --showduplicates | sort -r
docker-ce.x86_64          18.06.3.ce-3.el7
docker-ce.x86_64          18.06.2.ce-3.el7
```

步骤 4　安装特定版本

```
# yum install docker-ce-<VERSION_STRING> docker-ce-cli-
<VERSION_STRING> containerd.io


# yum install docker-ce-18.06.2.ce-3.el7  containerd.io -y
```

步骤 5　修改 docker 服务配置文件

```
 vi /usr/lib/systemd/system/docker.service   #增加如下参数

[Service]
# kill only the docker process, not all processes in the
cgroup
KillMode=process
```

步骤 6　配置 docker 参数

```
# mkdir /etc/docker
# vi /etc/docker/daemon.json

{
   "log-level": "warn",
   "selinux-enabled": false,
   "insecure-registries": [
      "10.211.18.61:10500",
      "10.211.18.62:10500"
   ],
   "registry-mirrors": [
      "https://3laho3y3.mirror.aliyuncs.com"
   ],
   "default-shm-size": "128M",
   "data-root": "/data/docker",
```

```
    "max-concurrent-downloads": 10,
    "max-concurrent-uploads": 5,
    "oom-score-adjust": -1000,
    "debug": false,
    "live-restore": true,
    "exec-opts": [
        "native.cgroupdriver=cgroupfs"
    ]
}
```

2.3.2 参数说明

参数名称	描述
log-level	日志级别[error|warn|info|debug]。
insecure-registries	配置私有镜像仓库，多个地址以“,”隔开。
registry-mirrors	默认拉取镜像仓库地址
max-concurrent-downloads	最大下载镜像数量
max-concurrent-uploads	最大上传镜像数量
live-restore	Docker 停止时保持容器继续运行，取值[true|false]
native.cgroupdriver	Docker 存储驱动
data-root	设置 docker 存储路径， 默认为/var/lib/docker
详细参数请参考官方文档： https://docs.docker.com/engine/reference/commandline/dockerd/#/linux-configuration-file	

2.3.3 启动 docker

```
# systemctl  enable docker
# systemctl  start  docker
```

2.4 安装 Cfssl

步骤 1 在 K8S-PROD-MASTER-A1 节点操作

```
wget -O /bin/cfssl https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
```

```
wget -O /bin/cfssljson
https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
wget -O /bin/cfssl-certinfo https://pkg.cfssl.org/R1.2/cfssl-
certinfo_linux-amd64
for cfssl in `ls /bin/cfssl*`;do chmod +x $cfssl;done;
```

2.5 安装 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。

步骤 1 在 K8S-PROD-MASTER-A1 节点制作证书文件

```
# mkdir -pv $HOME/etcd-ssl && cd $HOME/etcd-ssl
```

1. 生成 CA 证书，expiry 为证书过期时间(10 年)

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

2. 生成 CA 证书请求文件， ST/L/字段可自行修改

```
cat > etcd-ca-csr.json << EOF
{
  "CN": "etcd",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
```

```
      "ST": "ZheJiang",
      "L": "HangZhou",
      "O": "etcd",
      "OU": "Etcd Security"
    }
  ]
}
EOF
```

3. 生成证书请求文件，ST/L/字段可自行修改

```
cat > etcd-csr.json << EOF
{
    "CN": "etcd",
    "hosts": [
      "127.0.0.1",
      "10.211.18.4",
      "10.211.18.5",
      "10.211.18.6",
      "10.211.18.10"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "ZheJiang",
            "L": "HangZhou",
            "O": "etcd",
            "OU": "Etcd Security"
        }
    ]
}
EOF
```

注意： hosts 字段内为 etcd 主机 ip， 请根据业务 ip 自行修改。

步骤 2 拷贝证书文件到 etcd 节点（在 K8S-PROD-MASTER-A1 节点操作）

```
[root@K8S-PROD-MASTER-A1 etcd-ssl]# cfssl gencert -initca etcd-ca-csr.json | cfssljs
on -bare etcd-ca

2019/03/12 13:25:46 [INFO] generating a new CA key and certificate from CSR
2019/03/12 13:25:46 [INFO] generate received request
2019/03/12 13:25:46 [INFO] received CSR
2019/03/12 13:25:46 [INFO] generating key: rsa-2048
2019/03/12 13:25:47 [INFO] encoded CSR
```

```
2019/03/12 13:25:47 [INFO] signed certificate with serial number 7773066816391912455
45791647825506612900706535555

[root@K8S-PROD-MASTER-A1 etcd-ssl]# cfssl gencert -ca=etcd-ca.pem -ca-key=etcd-ca-ke
y.pem -config=ca-config.json -profile=kubernetes etcd-csr.json | cfssljson -bare etc
d
2019/03/12 13:25:57 [INFO] generate received request
2019/03/12 13:25:57 [INFO] received CSR
2019/03/12 13:25:57 [INFO] generating key: rsa-2048
2019/03/12 13:25:57 [INFO] encoded CSR
2019/03/12 13:25:57 [INFO] signed certificate with serial number 1470673449416436957
72357706461558238128357623444
2019/03/12 13:25:57 [WARNING] This certificate lacks a "hosts" field. This makes it u
nsuitable for
websites. For more information see the Baseline Requirements for the Issuance and Man
agement
of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum (https://cabfor
um.org);
specifically, section 10.2.3 ("Information Requirements").
```

步骤 3　分别在 etcd 节点创建以下目录

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# mkdir -pv /data/apps/etcd/{ssl,bin,etc,data}
```

步骤 4　分发证书文件

```
[root@K8S-PROD-MASTER-A1]# cd /root/etcd-ssl
[root@K8S-PROD-MASTER-A1]# mkdir -pv /data/apps/etcd/ssl
[root@K8S-PROD-MASTER-A1]# cp etcd*.pem /data/apps/etcd/ssl
[root@K8S-PROD-MASTER-A1]# scp -r /data/apps/etcd 10.211.18.5:/data/apps/etcd
[root@K8S-PROD-MASTER-A1]# scp -r /data/apps/etcd 10.211.18.6:/data/apps/etcd
```

步骤 5　解压 etcd-v3.3.12-linux-amd64.tar.gz

```
[root@K8S-PROD-MASTER-A1 software]# tar zxf etcd-v3.3.12-linux-amd64.tar.gz
[root@K8S-PROD-MASTER-A1 software]# mv etcd-v3.3.12-linux-amd64/etcd* /data/apps/etc
d/bin/
[root@K8S-PROD-MASTER-A1 software]# cd /data/apps/etcd/
```

步骤 6　配置系统服务

```
[root@K8S-PROD-MASTER-A1 etcd]# vi /usr/lib/systemd/system/etcd.service
[Unit]
```

```
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
Type=notify
WorkingDirectory=/data/apps/etcd/
EnvironmentFile=-/data/apps/etcd/etc/etcd.conf
User=etcd
# set GOMAXPROCS to number of processors
ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /data/apps/etcd/bin/etcd --name=\"${ETCD
_NAME}\" --data-dir=\"${ETCD_DATA_DIR}\" --listen-client-urls=\"${ETCD_LISTEN_CLIENT
_URLS}\""
Restart=on-failure
LimitNOFILE=65536

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

步骤 7 修改配置文件，红色字段内容请根据业务规划自行修改。

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cat << EOF > /data/apps/etcd/etc/etcd.conf
[Member]
#ETCD_CORS=""
ETCD_NAME="K8S-PROD-ETCD-A1"
ETCD_DATA_DIR="/data/apps/etcd/data/default.etcd"
#ETCD_WAL_DIR=""
ETCD_LISTEN_PEER_URLS="https://10.211.18.4:2380"
ETCD_LISTEN_CLIENT_URLS="https://127.0.0.1:2379,https://10.211.18.4:2379"
#ETCD_MAX_SNAPSHOTS="5"
#ETCD_MAX_WALS="5"
#ETCD_SNAPSHOT_COUNT="100000"
#ETCD_HEARTBEAT_INTERVAL="100"
#ETCD_ELECTION_TIMEOUT="1000"
#ETCD_QUOTA_BACKEND_BYTES="0"
#ETCD_MAX_REQUEST_BYTES="1572864"
#ETCD_GRPC_KEEPALIVE_MIN_TIME="5s"
#ETCD_GRPC_KEEPALIVE_INTERVAL="2h0m0s"
#ETCD_GRPC_KEEPALIVE_TIMEOUT="20s"
#
```

```
#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://10.211.18.4:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://127.0.0.1:2379,https://10.211.18.4:2379"
#ETCD_DISCOVERY=""
#ETCD_DISCOVERY_FALLBACK="proxy"
#ETCD_DISCOVERY_PROXY=""
#ETCD_DISCOVERY_SRV=""
ETCD_INITIAL_CLUSTER="K8S-PROD-ETCD-A1=https://10.211.18.4:2380,K8S-PROD-ETCD-A2=ht
tps://10.211.18.5:2380,K8S-PROD-ETCD-A3=https://10.211.18.6:2380"
ETCD_INITIAL_CLUSTER_TOKEN="BigBoss"
#ETCD_INITIAL_CLUSTER_STATE="new"
#ETCD_STRICT_RECONFIG_CHECK="true"
#ETCD_ENABLE_V2="true"
#
#[Proxy]
#ETCD_PROXY="off"
#ETCD_PROXY_FAILURE_WAIT="5000"
#ETCD_PROXY_REFRESH_INTERVAL="30000"
#ETCD_PROXY_DIAL_TIMEOUT="1000"
#ETCD_PROXY_WRITE_TIMEOUT="5000"
#ETCD_PROXY_READ_TIMEOUT="0"
#
#[Security]
ETCD_CERT_FILE="/data/apps/etcd/ssl/etcd.pem"
ETCD_KEY_FILE="/data/apps/etcd/ssl/etcd-key.pem"
#ETCD_CLIENT_CERT_AUTH="false"
ETCD_TRUSTED_CA_FILE="/data/apps/etcd/ssl/etcd-ca.pem"
#ETCD_AUTO_TLS="false"
ETCD_PEER_CERT_FILE="/data/apps/etcd/ssl/etcd.pem"
ETCD_PEER_KEY_FILE="/data/apps/etcd/ssl/etcd-key.pem"
#ETCD_PEER_CLIENT_CERT_AUTH="false"
ETCD_PEER_TRUSTED_CA_FILE="/data/apps/etcd/ssl/etcd-ca.pem"
#ETCD_PEER_AUTO_TLS="false"
#
[Logging]
ETCD_DEBUG="false"
#ETCD_LOG_PACKAGE_LEVELS=""
ETCD_LOG_OUTPUT="default"
#
#[Unsafe]
#ETCD_FORCE_NEW_CLUSTER="false"
#
#[Version]
```

```
#ETCD_VERSION="false"
#ETCD_AUTO_COMPACTION_RETENTION="0"
#
#[Profiling]
#ETCD_ENABLE_PPROF="false"
#ETCD_METRICS="basic"
#
#[Auth]
#ETCD_AUTH_TOKEN="simple"
EOF
```

步骤 8　启动服务

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# useradd -r etcd && chown etcd.etcd -R /data/apps/
etcd
[root@K8S-PROD-MASTER-A1 k8s-ssl]# systemctl enable etcd
[root@K8S-PROD-MASTER-A1 k8s-ssl]# systemctl start etcd
[root@K8S-PROD-MASTER-A1 k8s-ssl]# systemctl status etcd

● etcd.service - Etcd Server
   Loaded: loaded (/usr/lib/systemd/system/etcd.service; enabled; vendor preset: dis
abled)
   Active: active (running) since Tue 2019-03-12 17:11:30 CST; 49ms ago
```

将 etcd 配置文件、系统服务拷贝到其他 etcd 节点，根据实际情况修改对应字段信息。

步骤 9　设置环境变量

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# echo "PATH=$PATH:/data/apps/etcd/bin/" >> /etc/pr
ofile.d/etcd.sh
[root@K8S-PROD-MASTER-A1 k8s-ssl]# chmod +x /etc/profile.d/etcd.sh
[root@K8S-PROD-MASTER-A1 k8s-ssl]# source  /etc/profile.d/etcd.sh
[root@K8S-PROD-MASTER-A1 k8s-ssl]# etcdctl --endpoints "https://10.211.18.4:2379,htt
ps://10.211.18.4:2379,https://10.211.18.4:2379"   --ca-file=/data/apps/etcd/ssl/etcd
-ca.pem  --cert-file=/data/apps/etcd/ssl/etcd.pem   --key-file=/data/apps/etcd/ssl/e
tcd-key.pem  member list

46b51f26084e3e7e: name=K8S-PROD-ETCD-A2 peerURLs=https://10.211.18.5:2380 clientURLs
=https://10.211.18.5:2379,https://127.0.0.1:2379 isLeader=true
74974148d0145708: name=K8S-PROD-ETCD-A1 peerURLs=https://10.211.18.4:2380 clientURLs
=https://10.211.18.4:2379,https://127.0.0.1:2379 isLeader=false
```

```
b69515ada606e488: name=K8S-PROD-ETCD-A3 peerURLs=https://10.211.18.6:2380 clientURLs
=https://10.211.18.6:2379,https://127.0.0.1:2379 isLeader=false
```

2.6 安装 Kubernetes 组件

2.6.1 生成集群 CA 证书文件

步骤 1 配置 CA 证书。

1. 使用 root 用户，登录 K8S-PROD-MASTER-A1 节点。
   - 在家目录创建用于生成证书临时目录。
   - mkdir $HOME/k8s-ssl && cd $HOME/k8s-ssl
2. 配置 ca 证书信息， ST/L/expiry 字段可自行修改。

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cat > ca-csr.json << EOF
{
  "CN": "kubernetes",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "HangZhou",
      "L": "ZheJiang",
      "O": "k8s",
      "OU": "System"
    }
  ],
  "ca": {
     "expiry": "87600h"
  }
}
EOF
```

3. 执行 gencert 命令生成证书文件。

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cfssl gencert -initca ca-csr.json | cfssljson -ba
re ca
```

```
2019/03/12 17:38:18 [INFO] generating a new CA key and certificate from CSR
2019/03/12 17:38:19 [INFO] generate received request
2019/03/12 17:38:19 [INFO] received CSR
2019/03/12 17:38:19 [INFO] generating key: rsa-2048
2019/03/12 17:38:19 [INFO] encoded CSR
2019/03/12 17:38:19 [INFO] signed certificate with serial number 3712936686668588279
7351131966790776531962875 1223

[root@K8S-PROD-MASTER-A1 k8s-ssl]# ls -l ca*.pem
-rw------- 1 root root 1679 Mar 12 17:38 ca-key.pem
-rw-r--r-- 1 root root 1363 Mar 12 17:38 ca.pem
```

2.6.2 配置 kube-apiserver 证书

注意： 如需自定义集群域名， 请修改 cluster | cluster.local 字段为自定义域名

步骤 1 配置 kube-apiserver 证书信息

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cat > kube-apiserver-csr.json << EOF
{
    "CN": "kube-apiserver",
    "hosts": [
      "127.0.0.1",
      "10.211.18.4",
      "10.211.18.5",
      "10.211.18.6",
      "10.211.18.10",
      "10.99.0.1",
      "kubernetes",
      "kubernetes.default",
      "kubernetes.default.svc",
      "kubernetes.default.svc.ziji",
      "kubernetes.default.svc.ziji.work"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
```

```
            "ST": "HangZhou",
            "L": "ZheJiang",
            "O": "k8s",
            "OU": "System"
        }
    ]
}
EOF
```

步骤 2 执行 cfssl 生成 kube-apiserver 证书

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -conf
ig=/root/etcd-ssl/ca-config.json -profile=kubernetes kube-apiserver-csr.json | cfssl
json -bare kube-apiserver

2019/03/12 22:01:33 [INFO] generate received request
2019/03/12 22:01:33 [INFO] received CSR
2019/03/12 22:01:33 [INFO] generating key: rsa-2048
2019/03/12 22:01:34 [INFO] encoded CSR
2019/03/12 22:01:34 [INFO] signed certificate with serial number 3866948564046855946
521753099962302080772965681 38
2019/03/12 22:01:34 [WARNING] This certificate lacks a "hosts" field. This makes it u
nsuitable for
websites. For more information see the Baseline Requirements for the Issuance and Man
agement
of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum (https://cabfor
um.org);
specifically, section 10.2.3 ("Information Requirements").


[root@K8S-PROD-MASTER-A1 k8s-ssl]# ls -l kube-apiserver*.pem
-rw------- 1 root root 1675 Mar 12 22:01 kube-apiserver-key.pem
-rw-r--r-- 1 root root 1679 Mar 12 22:01 kube-apiserver.pem
```

2.6.3 配置 kube-controller-manager 证书

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。

步骤 1 创建证书和私钥

注意：hosts 列表包含所有 kube-controller-manager 节点 IP

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cat > kube-controller-manager-csr.json << EOF
{
    "CN": "system:kube-controller-manager",
    "hosts": [
      "127.0.0.1",
      "10.211.18.4",
      "10.211.18.5",
      "10.211.18.6",
      "10.211.18.10"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "HangZhou",
            "L": "ZheJiang",
            "O": "system:kube-controller-manager",
            "OU": "System"
        }
    ]
}
EOF
```

步骤 2 执行 cfssl 生成 kube-controller-manager 证书

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -conf
ig=ca-config.json -profile=kubernetes kube-controller-manager-csr.json | cfssljson -
bare kube-controller-manager

[root@K8S-PROD-MASTER-A1 k8s-ssl]# ls -l kube-controller-manager*.pem
-rw------- 1 root root 1679 Mar 13 16:51 kube-controller-manager-key.pem
-rw-r--r-- 1 root root 1521 Mar 13 16:51 kube-controller-manager.pem
```

2.6.4 配置 kube-scheduler 证书

步骤 1 创建证书和私钥

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cat > kube-scheduler-csr.json << EOF
{
    "CN": "system:kube-scheduler",
    "hosts": [
      "127.0.0.1",
      "10.211.18.4",
      "10.211.18.5",
      "10.211.18.6",
      "10.211.18.10"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "HangZhou",
            "L": "ZheJiang",
            "O": "system:kube-scheduler",
            "OU": "System"
        }
    ]
}
EOF
```

步骤 2 执行 cfssl 生成 kube-scheduler 证书

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -conf
ig=ca-config.json -profile=kubernetes kube-scheduler-csr.json | cfssljson -bare kube
-scheduler

2019/03/13 18:34:17 [INFO] generate received request
2019/03/13 18:34:17 [INFO] received CSR
2019/03/13 18:34:17 [INFO] generating key: rsa-2048
2019/03/13 18:34:18 [INFO] encoded CSR
```

```
2019/03/13 18:34:18 [INFO] signed certificate with serial number 4035533526251696355
37829194764174077828018057798
2019/03/13 18:34:18 [WARNING] This certificate lacks a "hosts" field. This makes it u
nsuitable for
websites. For more information see the Baseline Requirements for the Issuance and Man
agement
of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum (https://cabfor
um.org);
specifically, section 10.2.3 ("Information Requirements").

[root@K8S-PROD-MASTER-A1 k8s-ssl]# ls -l kube-scheduler*.pem
-rw------- 1 root root 1679 Mar 13 18:34 kube-scheduler-key.pem
-rw-r--r-- 1 root root 1497 Mar 13 18:34 kube-scheduler.pem
```

2.6.5 配置 kube-proxy 证书

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 kube-proxy 使用 ipvs 模式。

步骤 1 创建 kube-proxy 证书

该证书只会被 kube-proxy 当做 client 证书使用，所以 hosts 字段为空。

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

步骤 2　生成 kube-proxy 证书

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -conf
ig=ca-config.json -profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-pro
xy

2019/03/13 17:27:13 [INFO] generate received request
2019/03/13 17:27:13 [INFO] received CSR
……

[root@K8S-PROD-MASTER-A1 k8s-ssl]# ls -l kube-proxy*.pem
-rw------- 1 root root 1675 Mar 13 17:27 kube-proxy-key.pem
-rw-r--r-- 1 root root 1428 Mar 13 17:27 kube-proxy.pem
```

2.6.6 配置 admin 证书

为集群组件 kubelet、kubectl 配置 admin TLS 认证证书，具有访问 kubernetes 所有 api 的权限。

步骤 1　创建 admin 证书文件

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cat > admin-csr.json << EOF
{
    "CN": "admin",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "HangZhou",
            "L": "ZheJiang",
            "O": "system:masters",
            "OU": "System"
        }
    ]
}
EOF
```

步骤 2　生成 admin 证书

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -conf
ig=ca-config.json -profile=kubernetes admin-csr.json | cfssljson -bare admin

2019/03/13 18:25:57 [INFO] generate received request
2019/03/13 18:25:57 [INFO] received CSR
2019/03/13 18:25:57 [INFO] generating key: rsa-2048
2019/03/13 18:25:57 [INFO] encoded CSR
2019/03/13 18:25:57 [INFO] signed certificate with serial number 1252266061910239555
197363347594079348583942509 16
2019/03/13 18:25:57 [WARNING] This certificate lacks a "hosts" field. This makes it u
nsuitable for
websites. For more information see the Baseline Requirements for the Issuance and Man
agement
of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum (https://cabfor
um.org);
specifically, section 10.2.3 ("Information Requirements").

[root@K8S-PROD-MASTER-A1 k8s-ssl]# ls -l admin*.pem
-rw------- 1 root root 1679 Mar 13 18:25 admin-key.pem
-rw-r--r-- 1 root root 1407 Mar 13 18:25 admin.pem
```

2.6.7 分发证书文件

提示： node 节点只需要 ca、kube-proxy、kubelet 证书，不需要拷贝 kube-controller-manager、 kube-schedule、kube-apiserver 证书

步骤 1 创建证书存放目录

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# mkdir -pv /data/apps/kubernetes/{pki,log,etc,cert
s}
mkdir: created directory ‘/data/apps/kubernetes’
mkdir: created directory ‘/data/apps/kubernetes/pki’
```

步骤 2 拷贝证书文件到 apiserver、master 节点

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cp ca*.pem admin*.pem kube-proxy*.pem kube-schedu
ler*.pem kube-controller-manager*.pem kube-apiserver*.pem /data/apps/kubernetes/pki/

[root@K8S-PROD-MASTER-A1 k8s-ssl]# rsync  -avzP /data/apps/kubernetes 10.211.18.4/da
ta/apps/
[root@K8S-PROD-MASTER-A1 k8s-ssl]# rsync  -avzP /data/apps/kubernetes 10.211.18.5/da
ta/apps/
```



```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# rsync  -avzP /data/apps/kubernetes 10.211.18.6/da
ta/apps/
```

2.6.8 部署 Master 节点

步骤 1 解压 server 包并拷贝文件到其他 master 节点

```
[root@K8S-PROD-MASTER-A1 k8s-ssl]# cd /opt/software/
[root@K8S-PROD-MASTER-A1 software]# ls -l
total 517804
drwxr-xr-x 3 1000 1000       96 Mar 12 16:11 etcd-v3.3.12-linux-amd64
-rw-r--r-- 1 root root  11350736 Mar 11 08:42 etcd-v3.3.12-linux-amd64.tar.gz
-rw-r--r-- 1 root root   9565743 Mar 11 08:44 flannel-v0.11.0-linux-amd64.tar.gz
-rw-r--r-- 1 root root  91418745 Mar 11 08:51 kubernetes-node-linux-amd64.tar.gz
-rw-r--r-- 1 root root 417885230 Mar 11 09:10 kubernetes-server-linux-amd64.tar.gz

[root@K8S-PROD-MASTER-A1 software]# tar zxf kubernetes-server-linux-amd64.tar.gz
[root@K8S-PROD-MASTER-A1 software]# rsync -avzP kubernetes/server   10.211.18.4:/dat
a/apps/kubernetes/
[root@K8S-PROD-MASTER-A1 software]# rsync -avzP kubernetes/server   10.211.18.4:/dat
a/apps/kubernetes/
[root@K8S-PROD-MASTER-A1 software]# rsync -avzP kubernetes/server   10.211.18.4:/dat
a/apps/kubernetes/
```

步骤 2 配置环境变量，安装 docker 命令补全

```
[root@K8S-PROD-MASTER-A1 software]# yum install  bash-completion  -y
[root@K8S-PROD-MASTER-A1 software]# cat > /etc/profile.d/kubernetes.sh << EOF
K8S_HOME=/data/apps/kubernetes
export PATH=\$K8S_HOME/server/bin:\$PATH
source <(kubectl completion bash)
EOF

[root@K8S-PROD-MASTER-A1 software]# source /etc/profile.d/kubernetes.sh

[root@K8S-PROD-MASTER-A1 ~]# kubectl version
Client Version: version.Info{Major:"1", Minor:"13", GitVersion:"v1.13.6", GitCommit:
"c27b913fddd1a6c480c229191a087698aa92f0b1", GitTreeState:"clean", BuildDate:"2019-02
-28T13:37:52Z", GoVersion:"go1.11.5", Compiler:"gc", Platform:"linux/amd64"}
```



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

2.6.8.1 配置 TLS Bootstrapping

步骤 1 生成 token

```
[root@K8S-PROD-MASTER-A1 software]# export BOOTSTRAP_TOKEN=$(head -c 16 /dev/urandom
 | od -An -t x | tr -d ' ')
[root@K8S-PROD-MASTER-A1 software]# cat > /data/apps/kubernetes/token.csv << EOF
${BOOTSTRAP_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap"
EOF
```

2.6.8.2 创建 kubelet bootstrapping kubeconfig

设置 kube-apiserver 访问地址， 后面需要对 kube-apiserver 配置高可用集群， 这里设置 apiserver 浮动 IP。 KUBE_APISERVER=浮动 IP

```
[root@K8S-PROD-MASTER-A1 software]# cd /data/apps/kubernetes/
# export KUBE_APISERVER="https://10.211.18.10:8443"

# kubectl config set-cluster kubernetes \
--certificate-authority=/data/apps/kubernetes/pki/ca.pem \
--embed-certs=true \
--server=${KUBE_APISERVER} \
--kubeconfig=kubelet-bootstrap.kubeconfig
Cluster "kubernetes" set.

# kubectl config set-credentials kubelet-bootstrap \
--token=${BOOTSTRAP_TOKEN} \
--kubeconfig=kubelet-bootstrap.kubeconfig
User "kubelet-bootstrap" set.

# kubectl config set-context default \
--cluster=kubernetes \
--user=kubelet-bootstrap \
--kubeconfig=kubelet-bootstrap.kubeconfig
Context "default" created.

# kubectl config use-context default --kubeconfig=kubelet-bootstrap.kueconfig
Switched to context "default".
```

2.6.8.3 创建 kube-controller-manager kubeconfig

```
[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl config set-cluster kubernetes \
--certificate-authority=/data/apps/kubernetes/pki/ca.pem \
--embed-certs=true \
--server=${KUBE_APISERVER} \
--kubeconfig=kube-controller-manager.kuconfig

[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl config set-credentials kube-controller
-manager \
--client-certificate=/data/apps/kubernetes/pki/kube-controller-manager.pem \
--client-key=/data/apps/kubernetes/pki/kube-controller-manager-key.pem \
--embed-certs=true \
--kubeconfig=kube-controller-manager.kubeconfig

[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl config set-context default \
--cluster=kubernetes \
--user=kube-controller-manager \
--kubeconfig=kube-controller-manager.kubeconfig

[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl config use-context default --kubeconfi
g=kube-controller-manager.kubeconfig
```

2.6.8.4 创建 kube-scheduler kubeconfig

```
kubectl config set-cluster kubernetes \
--certificate-authority=/data/apps/kubernetes/pki/ca.pem \
--embed-certs=true \
--server=${KUBE_APISERVER} \
--kubeconfig=kube-scheduler.kubeconfig

kubectl config set-credentials kube-scheduler \
--client-certificate=/data/apps/kubernetes/pki/kube-scheduler.pem \
--client-key=/data/apps/kubernetes/pki/kube-scheduler-key.pem \
--embed-certs=true \
--kubeconfig=kube-scheduler.kubeconfig
```

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

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

2.6.8.5 创建 kube-proxy kubeconfig

```
kubectl config set-cluster kubernetes \
--certificate-authority=/data/apps/kubernetes/pki/ca.pem \
--embed-certs=true \
--server=${KUBE_APISERVER} \
--kubeconfig=kube-proxy.kubeconfig

kubectl config set-credentials kube-proxy \
--client-certificate=/data/apps/kubernetes/pki/kube-proxy.pem \
--client-key=/data/apps/kubernetes/pki/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
```

2.6.8.6 创建 admin kubeconfig

```
kubectl config set-cluster kubernetes \
--certificate-authority=/data/apps/kubernetes/pki/ca.pem \
--embed-certs=true \
--server=${KUBE_APISERVER} \
--kubeconfig=admin.conf

kubectl config set-credentials admin \
--client-certificate=/data/apps/kubernetes/pki/admin.pem \
```

```
--client-key=/data/apps/kubernetes/pki/admin-key.pem \
--embed-certs=true \
--kubeconfig=admin.conf

kubectl config set-context default \
--cluster=kubernetes \
--user=admin \
--kubeconfig=admin.conf

kubectl config use-context default --kubeconfig=admin.conf
```

2.6.8.7 分发 kubelet/kube-proxy 配置文件

步骤 1 分发配置文件到 node 节点

```
[root@K8S-PROD-MASTER-A1 kubernetes]# rsync  -avz --exclude=kube-scheduler.kubeconfig --exclude=kube-controller-manager.kubeconfig --exclude=admin.conf --exclude=token.csv etc 10.211.18.11:/data/apps/kubernetes

sending incremental file list
etc/
etc/kube-proxy.kubeconfig
etc/kubelet-bootstrap.kubeconfig

sent 5,842 bytes  received 58 bytes  1,311.11 bytes/sec
total size is 8,482  speedup is 1.44
```

步骤 2 分发配置文件到其他 master 节点

2.6.8.8 配置 kube-apiserver

kube-apiserver 是 Kubernetes 最重要的核心组件之一，主要提供以下的功能

- 提供集群管理的 REST API 接口，包括认证授权、数据校验以及集群状态变更等
- 提供其他模块之间的数据交互和通信的枢纽（其他模块通过 API Server 查询或修改数据，只有 API Server 才直接操作 etcd）

```
[root@K8S-PROD-MASTER-A1 kubernetes]# cd /data/apps/kubernetes/pki/
```

```
[root@K8S-PROD-MASTER-A1 pki]# openssl genrsa -out /data/apps/kubernetes/pki/sa.key
2048
Generating RSA private key, 2048 bit long modulus
..............+++
......+++
e is 65537 (0x10001)
[root@K8S-PROD-MASTER-A1 pki]# openssl rsa -in /data/apps/kubernetes/pki/sa.key -pub
out -out /data/apps/kubernetes/pki/sa.pub
writing RSA key
[root@K8S-PROD-MASTER-A1 pki]# ls -l sa.*
-rw-r--r-- 1 root root 1679 Mar 14 11:49 sa.key
-rw-r--r-- 1 root root  451 Mar 14 11:49 sa.pub
```

分发文件到其他 apiserver 节点

```
[root@K8S-PROD-MASTER-A1 pki]# scp -r /data/apps/kubernetes/pki/sa.* 10.211.18.5:/da
ta/apps/kubernetes/pki/
[root@K8S-PROD-MASTER-A1 pki]# scp -r /data/apps/kubernetes/pki/sa.* 10.211.18.6:/da
ta/apps/kubernetes/pki/

[root@K8S-PROD-MASTER-A1 kubernetes]# scp -r /data/apps/kubernetes/etc 10.211.18.5:/
data/apps/kubernetes/
[root@K8S-PROD-MASTER-A1 kubernetes]# scp -r /data/apps/kubernetes/etc 10.211.18.6:/
data/apps/kubernetes/
```

2.6.8.8.1 配置 apiserver 系统服务

步骤 1 分别在 10.211.18.4、10.211.18.5、10.211.18.6 三台主机执行如下命令。

```
cat > /usr/lib/systemd/system/kube-apiserver.service << EOF
[Unit]
Description=Kubernetes API Service
Documentation=https://github.com/kubernetes/kubernetes
After=network.target

[Service]
EnvironmentFile=-/data/apps/kubernetes/etc/kube-apiserver.conf
ExecStart=/data/apps/kubernetes/server/bin/kube-apiserver \\
\$KUBE_LOGTOSTDERR \\
```



```
\$KUBE_LOG_LEVEL \\
\$KUBE_ETCD_ARGS \\
\$KUBE_API_ADDRESS \\
\$KUBE_SERVICE_ADDRESSES \\
\$KUBE_ADMISSION_CONTROL \\
\$KUBE_APISERVER_ARGS
Restart=on-failure
Type=notify
LimitNOFILE=65536

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

步骤 2　创建 apiserver 配置文件

```
[root@K8S-PROD-MASTER-A1 etc]# cat >/data/apps/kubernetes/etc/kube-apiserver.conf <<
EOF
KUBE_API_ADDRESS="--advertise-address=10.211.18.4"

KUBE_ETCD_ARGS="--etcd-servers=https://10.211.18.7:2379,https://10.211.18.3:2379,ht
tps://10.211.18.14:2379 \
--etcd-cafile=/data/apps/etcd/ssl/etcd-ca.pem \
--etcd-certfile=/data/apps/etcd/ssl/etcd.pem \
--etcd-keyfile=/data/apps/etcd/ssl/etcd-key.pem"

KUBE_LOGTOSTDERR="--logtostderr=false"
KUBE_LOG_LEVEL=" --log-dir=/data/apps/kubernetes/log/  --v=2 \
--audit-log-maxage=7 \
--audit-log-maxbackup=10  \
--audit-log-maxsize=100 \
--audit-log-path=/data/apps/kubernetes/log/kubernetes.audit --event-ttl=12h"


KUBE_SERVICE_ADDRESSES="--service-cluster-ip-range=10.99.0.0/16"
KUBE_ADMISSION_CONTROL="--enable-admission-plugins=NamespaceLifecycle,LimitRanger,S
erviceAccount,DefaultStorageClass,DefaultTolerationSeconds,MutatingAdmissionWebhoo
k,ValidatingAdmissionWebhook,ResourceQuota,PersistentVolumeClaimResize,PodPreset"

KUBE_APISERVER_ARGS="--storage-backend=etcd3 \
--apiserver-count=3 \
```

```
--endpoint-reconciler-type=lease \
--runtime-config=api/all,settings.k8s.io/v1alpha1=true,admissionregistration.k8s.io
/v1beta1 \
--enable-admission-plugins=Initializers \
--allow-privileged=true \
--authorization-mode=Node,RBAC \
--enable-bootstrap-token-auth=true \
--token-auth-file=/data/apps/kubernetes/etc/token.csv \
--service-node-port-range=30000-40000 \
--tls-cert-file=/data/apps/kubernetes/pki/kube-apiserver.pem \
--tls-private-key-file=/data/apps/kubernetes/pki/kube-apiserver-key.pem \
--client-ca-file=/data/apps/kubernetes/pki/ca.pem \
--service-account-key-file=/data/apps/kubernetes/pki/sa.pub \
--enable-swagger-ui=false \
--secure-port=6443 \
--kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname \
--anonymous-auth=false \
--kubelet-client-certificate=/data/apps/kubernetes/pki/admin.pem \
--kubelet-client-key=/data/apps/kubernetes/pki/admin-key.pem "
EOF
```


步骤 3 分别启动三台 apiserver

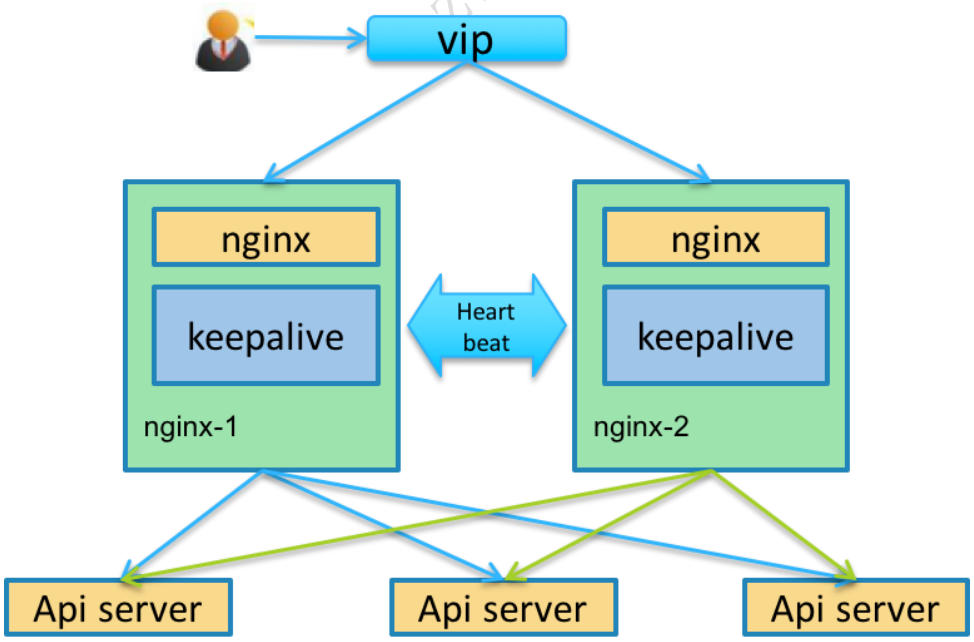
```
[root@K8S-PROD-MASTER-A1 etc]# mkdir -p /data/apps/kubernetes/log
[root@K8S-PROD-MASTER-A1 etc]# systemctl daemon-reload
[root@K8S-PROD-MASTER-A1 etc]# systemctl enable kube-apiserver
Created symlink from /etc/systemd/system/multi-user.target.wants/kube-apiserver.serv
ice to /usr/lib/systemd/system/kube-apiserver.service.

[root@K8S-PROD-MASTER-A1 etc]# systemctl start kube-apiserver
[root@K8S-PROD-MASTER-A1 etc]# systemctl status kube-apiserver
```

访问测试

出现一下内容说明搭建成功：

```
curl -k https://10.211.18.4:6443

{
"kind": "Status",
"apiVersion": "v1",
"metadata": {

},
"status": "Failure",
"message": "Unauthorized",
"reason": "Unauthorized",
"code": 401
}
```

2.6.9 配置 apiserver 高可用部署



2.6.9.9 安装相关软件包

```
10.211.18.4   apiserver-0
10.211.18.5   apiserver-1
10.211.18.6   apiserver-2

分别在 apiserver 集群节点安装 haproxy、keepalived
yum install haproxy keepalived -y
```

2.6.9.10 配置 haproxy

步骤 1 在 Master-A1 执行如下命令

```
[root@K8S-PROD-MASTER-A1 ~]# cd /etc/haproxy/
[root@K8S-PROD-MASTER-A1 haproxy]# cp haproxy.cfg{,.bak}
[root@K8S-PROD-MASTER-A1 haproxy]# > haproxy.cfg
```

依次在其他 apiserver 节点创建 haproxy.cfg 配置文件

```
[root@K8S-PROD-MASTER-A1 haproxy]# vi haproxy.cfg

global
    log 127.0.0.1 local2
    chroot /var/lib/haproxy
    pidfile /var/run/haproxy.pid
    maxconn 4000
    user haproxy
    group haproxy
    daemon
    stats socket /var/lib/haproxy/stats

defaults
    mode tcp
    log global
    retries 3
    option tcplog
    option httpclose
    option dontlognull
    option abortonclose
```

```
    option redispatch
    maxconn 32000
    timeout connect 5000ms
    timeout client 2h
    timeout server 2h

listen stats
    mode http
    bind :10086
    stats enable
    stats uri /admin?stats
    stats auth admin:admin
    stats admin if TRUE

frontend k8s_apiserver
    bind *:8443
    mode tcp
    default_backend https_sri

backend https_sri
    balance roundrobin
    server apiserver1_10_211_18_4 10.211.18.4:6443 check inter 2000 fall 2 rise 2 wei
ght 100
    server apiserver2_10_211_18_5 10.211.18.5:6443 check inter 2000 fall 2 rise 2 wei
ght 100
    server apiserver3_10_211_18_6 10.211.18.6:6443 check inter 2000 fall 2 rise 2 wei
ght 100
```

步骤 2 在 Master-A2 执行如下命令

```
[root@K8S-PROD-MASTER-A2 ~]# cd /etc/haproxy/
[root@K8S-PROD-MASTER-A2 haproxy]# cp haproxy.cfg{,.bak}
[root@K8S-PROD-MASTER-A2 haproxy]# > haproxy.cfg
```

```
 [root@K8S-PROD-MASTER-A2 haproxy]# vi haproxy.cfg
global
    log 127.0.0.1 local2
    chroot /var/lib/haproxy
    pidfile /var/run/haproxy.pid
    maxconn 4000
```

```
    user haproxy
    group haproxy
    daemon
    stats socket /var/lib/haproxy/stats

defaults
    mode tcp
    log global
    retries 3
    option tcplog
    option httpclose
    option dontlognull
    option abortonclose
    option redispatch
    maxconn 32000
    timeout connect 5000ms
    timeout client 2h
    timeout server 2h

listen stats
    mode http
    bind :10086
    stats enable
    stats uri /admin?stats
    stats auth admin:admin
    stats admin if TRUE

frontend k8s_apiserver
    bind *:8443
    mode tcp
    default_backend https_sri

backend https_sri
    balance roundrobin
    server apiserver1_10_211_18_4 10.211.18.4:6443 check inter 2000 fall 2 rise 2 wei
ght 100
    server apiserver2_10_211_18_5 10.211.18.5:6443 check inter 2000 fall 2 rise 2 wei
ght 100
     server apiserver3_10_211_18_6 10.211.18.6:6443 check inter 2000 fall 2 rise 2 we
ight 100
```

步骤 3　在 Master-A3 执行如下命令

```
[root@K8S-PROD-MASTER-A3 ~]# cd /etc/haproxy/
[root@K8S-PROD-MASTER-A3 haproxy]# cp haproxy.cfg{,.bak}
[root@K8S-PROD-MASTER-A3 haproxy]# > haproxy.cfg
```

```
[root@K8S-PROD-MASTER-A3 haproxy]# vi haproxy.cfg

global
    log 127.0.0.1 local2
    chroot /var/lib/haproxy
    pidfile /var/run/haproxy.pid
    maxconn 4000
    user haproxy
    group haproxy
    daemon
    stats socket /var/lib/haproxy/stats

defaults
    mode tcp
    log global
    retries 3
    option tcplog
    option httpclose
    option dontlognull
    option abortonclose
    option redispatch
    maxconn 32000
    timeout connect 5000ms
    timeout client 2h
    timeout server 2h

listen stats
    mode http
    bind :10086
    stats enable
    stats uri /admin?stats
    stats auth admin:admin
    stats admin if TRUE

frontend k8s_apiserver
    bind *:8443
    mode tcp
```

```
    default_backend https_sri

backend https_sri
    balance roundrobin
    server apiserver1_10_211_18_4 10.211.18.4:6443 check inter 2000 fall 2 rise 2 wei
ght 100
    server apiserver2_10_211_18_5 10.211.18.5:6443 check inter 2000 fall 2 rise 2 wei
ght 100
    server apiserver3_10_211_18_6 10.211.18.6:6443 check inter 2000 fall 2 rise 2 wei
ght 100
```

2.6.9.11 配置 keepalived

```
[root@K8S-PROD-MASTER-A1 haproxy]# cd /etc/keepalived/
[root@K8S-PROD-MASTER-A1 keepalived]# cp keepalived.conf{,.bak}
[root@K8S-PROD-MASTER-A1 keepalived]#  > keepalived.conf
```

步骤 1　Master-A1 节点

```
[root@K8S-PROD-MASTER-A1 keepalived]# vi keepalived.conf

! Configuration File for keepalived
global_defs {
    notification_email {
        test@test.com
    }
    notification_email_from admin@test.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_MASTER_APISERVER
}

vrrp_script check_haproxy {
    script "/etc/keepalived/check_haproxy.sh"
    interval 3
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 60
    priority 100
```

```
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    unicast_peer {
    10.211.18.4
    10.211.18.5
    10.211.18.6
    }
    virtual_ipaddress {
        10.211.18.10/24 label ens33:0
    }

    track_script {
        check_haproxy
    }
}
```

注意：

其他两个节点 state MASTER 字段需改为 state BACKUP， priority 100 需要分别设置 90 80

```
[root@K8S-PROD-MASTER-A2 haproxy]# cd /etc/keepalived/
[root@K8S-PROD-MASTER-A2 keepalived]# cp keepalived.conf{,.bak}
[root@K8S-PROD-MASTER-A2 keepalived]#  > keepalived.conf
```

```
[root@K8S-PROD-MASTER-A2 keepalived]# vi keepalived.conf

! Configuration File for keepalived
global_defs {
    notification_email {
        test@test.com
    }
    notification_email_from admin@test.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_MASTER_APISERVER
}

vrrp_script check_haproxy {
```

```
    script "/etc/keepalived/check_haproxy.sh"
    interval 3
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 60
    priority 90
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    unicast_peer {
    10.211.18.4
    10.211.18.5
    10.211.18.6
    }
    virtual_ipaddress {
        10.211.18.10/24 label ens33:0
    }

    track_script {
        check_haproxy
    }
}
```

创建 keepalived 配置文件

```
[root@K8S-PROD-MASTER-A3 haproxy]# cd /etc/keepalived/
[root@K8S-PROD-MASTER-A3 keepalived]# cp keepalived.conf{,.bak}
[root@K8S-PROD-MASTER-A3 keepalived]#  > keepalived.conf
```

```
[root@K8S-PROD-MASTER-A3 keepalived]# vi keepalived.conf

! Configuration File for keepalived
global_defs {
    notification_email {
        test@test.com
    }
```

```
    notification_email_from admin@test.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_MASTER_APISERVER
}

vrrp_script check_haproxy {
    script "/etc/keepalived/check_haproxy.sh"
    interval 3
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 60
    priority 80
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    unicast_peer {
    10.211.18.4
    10.211.18.5
    10.211.18.6
    }
    virtual_ipaddress {
        10.211.18.10/24 label ens33:0
    }

    track_script {
        check_haproxy
    }
}
```

所有节点(Master1~Master3)需要配置检查脚本（check_haproxy.sh），当 haproxy 挂掉后自动停止 keepalived

```
[root@K8S-PROD-MASTER-A1 keepalived]# vi /etc/keepalived/check_haproxy.sh

#!/bin/bash
```

```
flag=$(systemctl status haproxy &> /dev/null;echo $?)

if [[ $flag != 0 ]];then
echo "haproxy is down,close the keepalived"
systemctl stop keepalived
fi
```

步骤 2　修改系统服务

```
[root@K8S-PROD-MASTER-A1 keepalived]#  vi /usr/lib/systemd/system/keepalived.service

[Unit]
Description=LVS and VRRP High Availability Monitor
After=syslog.target network-online.target
Requires=haproxy.service    #增加该字段
[Service]
Type=forking
PIDFile=/var/run/keepalived.pid
KillMode=process
EnvironmentFile=-/etc/sysconfig/keepalived
ExecStart=/usr/sbin/keepalived $KEEPALIVED_OPTIONS
ExecReload=/bin/kill -HUP $MAINPID

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

2.6.9.12 firewalld 放行 VRRP 协议

```
firewall-cmd --direct --permanent --add-rule ipv4 filter INPUT 0 --in-interface ens3
3 --destination 224.0.0.18 --protocol vrrp -j ACCEPT
firewall-cmd --reload
```

注意：　--in-interface ens33 请修改为实际网卡名称

登录 apiserver 节点启动以下服务

```
[root@K8S-PROD-MASTER-A1 haproxy]# systemctl enable haproxy
```

```
Created symlink from /etc/systemd/system/multi-user.target.wants/haproxy.service to
/usr/lib/systemd/system/haproxy.service.


[root@K8S-PROD-MASTER-A1 haproxy]# systemctl  start haproxy
[root@K8S-PROD-MASTER-A1 haproxy]# systemctl  status haproxy
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/usr/lib/systemd/system/haproxy.service; enabled; vendor preset:
disabled)
   Active: active (running) since Mon 2019-04-29 10:19:21 CST; 6h ago
 Main PID: 5304 (haproxy-systemd)
   CGroup: /system.slice/haproxy.service
           ├─5304 /usr/sbin/haproxy-systemd-wrapper -f /etc/haproxy/haproxy.cfg -p
/run/haproxy.pid
           ├─5308 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
 -Ds
           └─5344 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
 -Ds

Apr 29 10:19:21 K8S-PROD-MASTER-A1 systemd[1]: Started HAProxy Load Balancer.
Apr 29 10:19:21 K8S-PROD-MASTER-A1 haproxy-systemd-wrapper[5304]: haproxy-systemd-wr
apper: executing /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -
Ds
```

```
[root@K8S-PROD-MASTER-A1 keepalived]# systemctl enable keepalived
Created symlink from /etc/systemd/system/multi-user.target.wants/keepalived.service
to /usr/lib/systemd/system/keepalived.service.

[root@K8S-PROD-MASTER-A1 keepalived]# systemctl  start keepalived
[root@K8S-PROD-MASTER-A1 keepalived]# systemctl  status keepalived
● keepalived.service - LVS and VRRP High Availability Monitor
   Loaded: loaded (/usr/lib/systemd/system/keepalived.service; enabled; vendor prese
t: disabled)
   Active: active (running) since Mon 2019-04-29 10:19:28 CST; 4h 2min ago
 Main PID: 5355 (keepalived)
   CGroup: /system.slice/keepalived.service
           ├─5355 /usr/sbin/keepalived -D
           ├─5356 /usr/sbin/keepalived -D
           └─5357 /usr/sbin/keepalived -D
```

```
Apr 29 10:21:32 K8S-PROD-MASTER-A1 Keepalived_vrrp[5357]: Sending gratuitous ARP on ens33 for 10.211.18.10
Apr 29 10:21:32 K8S-PROD-MASTER-A1 Keepalived_vrrp[5357]: Sending gratuitous ARP on ens33 for 10.211.18.10
Apr 29 10:21:32 K8S-PROD-MASTER-A1 Keepalived_vrrp[5357]: Sending gratuitous ARP on ens33 for 10.211.18.10
Apr 29 10:21:32 K8S-PROD-MASTER-A1 Keepalived_vrrp[5357]: Sending gratuitous ARP on ens33 for 10.211.18.10
Apr 29 10:21:37 K8S-PROD-MASTER-A1 Keepalived_vrrp[5357]: Sending gratuitous ARP on ens33 for 10.211.18.10
Apr 29 10:21:37 K8S-PROD-MASTER-A1 Keepalived_vrrp[5357]: VRRP_Instance(VI_1) Sending/queueing gratuitous ARPs on ens33 for 10.211.18.10
```

执行 ip a 命令， 查看浮动 IP

```
3: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state U
    link/ether 00:0c:29:db:37:ff brd ff:ff:ff:ff:ff:ff
    inet 10.211.18.4/24 brd 10.211.18.255 scope global noprefixroute ens33
       valid_lft forever preferred_lft forever
    inet 10.211.18.10/24 scope global secondary ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::1b38:c17f:bf53:ab32/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

http://10.211.18.10:10086/admin?stats 登录 haproxy，查看服务是否正常

HAProxy version 1.5.18, released 2016/05/10

Statistics Report for pid 5344

> General process information

pid = 5344 (process #1, nbproc = 1)
uptime = 0d 6h19m28s
system limits: memmax = unlimited; ulimit-n = 8035
maxsock = 8035; maxconn = 4000; maxpipes = 0
current conns = 11; current pipes = 0/0; conn rate = 2/sec
Running tasks: 2/19; idle = 96 %

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings			
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk
Frontend				2	2	-	2	2	3 000	4			948	19 156	0	0	1					OPEN	
Backend	0	0		0	1		0	1	300	1	0	0s	948	19 156	0	0		1	0	0	0	6h19m UP	

k8s_apiserver

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings			
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastC
Frontend				0	20	-	9	25	2 000	160			23 451 881	137 529 786	0	0	0					OPEN	

https_sri

		Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings			
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	
☐	apiserver1_10_211_18_4	0	0	-	0	6		1	8	-	42	42	1h29m	5 743 058	25 628 181		0		0	0	0	0	6h18m UP	L
☐	apiserver2_10_211_18_5	0	0	-	0	10		5	14	-	79	79	1h29m	16 372 877	101 930 312		0		0	0	0	0	6h19m UP	L
☐	apiserver3_10_211_18_6	0	0	-	0	7		3	7	-	39	39	1h29m	1 335 946	9 971 293		0		0	0	0	0	6h18m UP	L
	Backend	0	0		0	20		9	25	200	160	160	1h29m	23 451 881	137 529 786	0	0		0	0	0	0	6h19m UP	

Choose the action to perform on the checked servers : Apply

2.6.10 配置启动 kube-controller-manager

kube-controller-manager 负责维护集群的状态，比如故障检测、自动扩展、滚动更新等

在启动时设置 --leader-elect=true 后，controller manager 会使用多节点选主的方式选择主节点。只有主节点才会调用 StartControllers() 启动所有控制器，而其他从节点则仅执行选主算法。

步骤 2 执行如下创建系统服务文件

```
[root@K8S-PROD-MASTER-A1 software]#  cat > /usr/lib/systemd/system/kube-controller-manager.service << EOF
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes

[Service]
EnvironmentFile=-/data/apps/kubernetes/etc/kube-controller-manager.conf
ExecStart=/data/apps/kubernetes/server/bin/kube-controller-manager \
$KUBE_LOGTOSTDERR \
$KUBE_LOG_LEVEL \
$KUBECONFIG \
$KUBE_CONTROLLER_MANAGER_ARGS

Restart=always
RestartSec=10s

#Restart=on-failure
LimitNOFILE=65536

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

步骤 3 创建 kube-controller-manager.conf 配置文件

```
[root@K8S-PROD-MASTER-A2 etc]# cat > /data/apps/kubernetes/etc/kube-controller-manager.conf <<EOF
KUBE_LOGTOSTDERR="--logtostderr=false"
KUBE_LOG_LEVEL="--v=2 --log-dir=/data/apps/kubernetes/log/"
```

```
KUBECONFIG="--kubeconfig=/data/apps/kubernetes/etc/kube-controller-manager.kubeconf
ig"

KUBE_CONTROLLER_MANAGER_ARGS="--bind-address=127.0.0.1 \
--cluster-cidr=10.99.0.0/16 \
--cluster-name=kubernetes \
--cluster-signing-cert-file=/data/apps/kubernetes/pki/ca.pem \
--cluster-signing-key-file=/data/apps/kubernetes/pki/ca-key.pem \
--service-account-private-key-file=/data/apps/kubernetes/pki/sa.key \
--root-ca-file=/data/apps/kubernetes/pki/ca.pem \
--leader-elect=true \
--use-service-account-credentials=true \
--node-monitor-grace-period=10s \
--pod-eviction-timeout=10s \
--allocate-node-cidrs=true \
--controllers=*,bootstrapsigner,tokencleaner \
--horizontal-pod-autoscaler-use-rest-clients=true \
--experimental-cluster-signing-duration=87600h0m0s \
--feature-gates=RotateKubeletServerCertificate=true""
EOF
```

步骤 4　启动 kube-controller-manager

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

systemctl status kube-controller-manager
```

```
[root@K8S-PROD-MASTER-A2 etc]# systemctl status kube-controller-manager
● kube-controller-manager.service
   Loaded: loaded (/usr/lib/systemd/system/kube-controller-manager.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2019-05-22 17:15:02 CST; 17h ago
 Main PID: 10018 (kube-controller)
    Tasks: 6
   Memory: 16.3M
   CGroup: /system.slice/kube-controller-manager.service
           └─10018 /data/apps/kubernetes/server/bin/kube-controller-manager --logtostderr=false --v=2 --log-dir=/data/apps/kubernetes/log/ -

May 22 17:15:02 K8S-PROD-MASTER-A2 systemd[1]: Stopped kube-controller-manager.service.
May 22 17:15:02 K8S-PROD-MASTER-A2 systemd[1]: Started kube-controller-manager.service.
```

配置 kubectl

```
[root@K8S-PROD-MASTER-A1 pki]# rm -rf $HOME/.kube
[root@K8S-PROD-MASTER-A1 pki]# mkdir -p $HOME/.kube
[root@K8S-PROD-MASTER-A1 pki]# cp  /data/apps/kubernetes/etc/admin.conf  $HOME/.kube
/config
```

```
[root@K8S-PROD-MASTER-A1 pki]# sudo chown $(id -u):$(id -g) $HOME/.kube/config
[root@K8S-PROD-MASTER-A1 pki]# kubectl get node
```

查看各个组件的状态

```
kubectl get componentstatuses
```

```
[root@K8S-PROD-MASTER-A1 pki]# kubectl get componentstatuses
NAME                 STATUS    MESSAGE   ERROR
controller-manager Healthy   ok
scheduler           Healthy   ok
etcd-1              Healthy {"health": "true"}
etcd-0              Healthy {"health": "true"}
etcd-2              Healthy {"health": "true"}
```

配置 kubelet 使用 bootstrap

```
[root@K8S-PROD-MASTER-A1 pki]#  kubectl create clusterrolebinding kubelet-bootstrap
\
--clusterrole=system:node-bootstrapper \
--user=kubelet-bootstrap
```

2.6.11 配置启动 kube-scheduler

kube-scheduler 负责分配调度 Pod 到集群内的节点上，它监听 kube-apiserver，查询还未分配 Node 的 Pod，然后根据调度策略为这些 Pod 分配节点。按照预定的调度策略将 Pod 调度到相应的机器上（更新 Pod 的 NodeName 字段）。

步骤 1 执行如下创建系统服务文件

```
[root@K8S-PROD-MASTER-A2 etc]# cat > /usr/lib/systemd/system/kube-scheduler.service
<< EOF
[Unit]
Description=Kubernetes Scheduler Plugin
Documentation=https://github.com/kubernetes/kubernetes
```

```
[Service]
EnvironmentFile=-/data/apps/kubernetes/etc/kube-scheduler.conf
ExecStart=/data/apps/kubernetes/server/bin/kube-scheduler \
$KUBE_LOGTOSTDERR \
$KUBE_LOG_LEVEL \
$KUBECONFIG \
$KUBE_SCHEDULER_ARGS
Restart=on-failure
LimitNOFILE=65536

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

步骤 2 创建 kube-scheduler.conf 配置文件

```
[root@K8S-PROD-MASTER-A2 etc]#  cat > /data/apps/kubernetes/etc/kube-scheduler.conf
<< EOF
KUBE_LOGTOSTDERR="--logtostderr=false"
KUBE_LOG_LEVEL="--v=2 --log-dir=/data/apps/kubernetes/log/"

KUBECONFIG="--kubeconfig=/data/apps/kubernetes/etc/kube-scheduler.kubeconfig"
KUBE_SCHEDULER_ARGS="--leader-elect=true --address=127.0.0.1"
EOF
```

步骤 3 启动 kube-scheduler， 并设置服务开机自启动

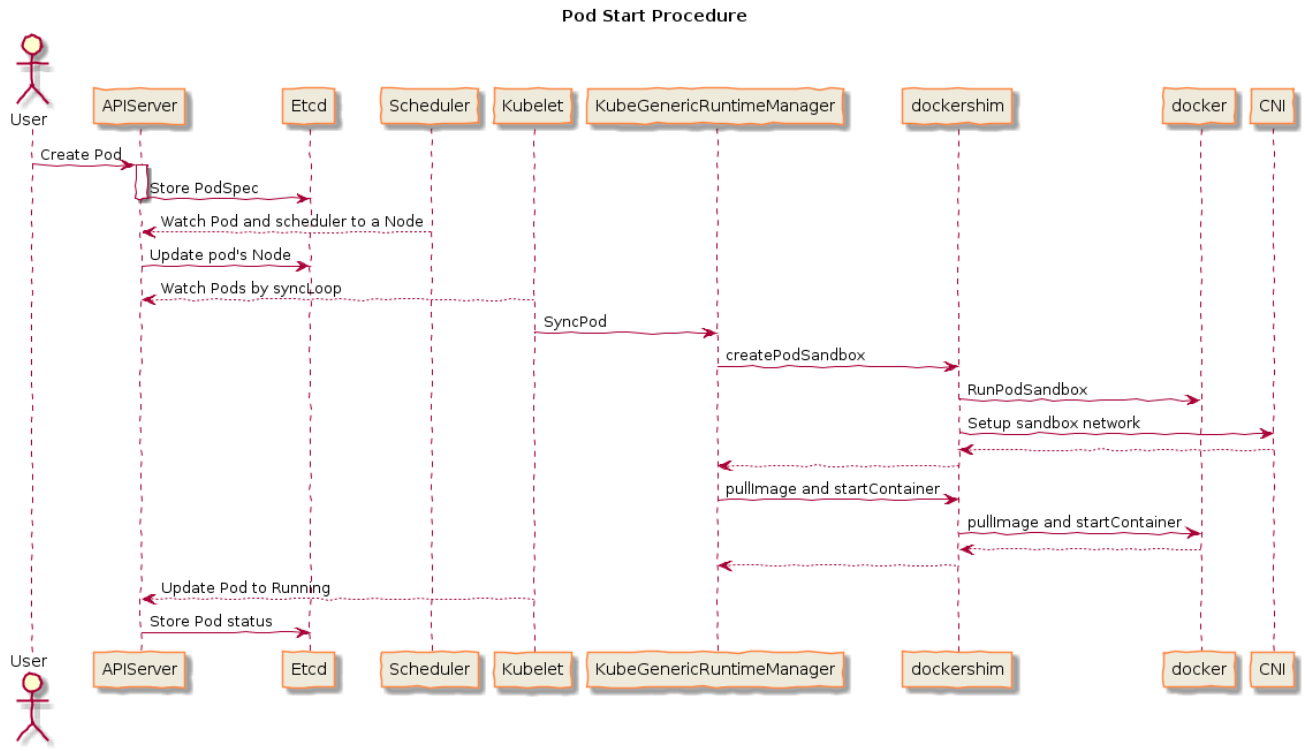
```
systemctl daemon-reload
systemctl enable kube-scheduler
systemctl start kube-scheduler
systemctl  status kube-scheduler
```

```
[root@K8S-PROD-MASTER-A2 etc]# systemctl  status kube-scheduler
● kube-scheduler.service - Kubernetes Scheduler Plugin
   Loaded: loaded (/usr/lib/systemd/system/kube-scheduler.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2019-05-22 17:15:35 CST; 17h ago
     Docs: https://github.com/kubernetes/kubernetes
 Main PID: 10166 (kube-scheduler)
    Tasks: 8
   Memory: 18.5M
   CGroup: /system.slice/kube-scheduler.service
           └─10166 /data/apps/kubernetes/server/bin/kube-scheduler --logtostderr=false --v=2 --log-dir=/data/apps/kubernetes/log/ --
```

2.6.12 部署 Node 节点

2.6.12.13 配置 kubelet

kubelet 负责维持容器的生命周期，同时也负责 Volume（CVI）和网络（CNI）的管理；每个节点上都运行一个 kubelet 服务进程，默认监听 10250 端口，接收并执行 master 发来的指令，管理 Pod 及 Pod 中的容器。每个 kubelet 进程会在 API Server 上注册节点自身信息，定期向 master 节点汇报节点的资源使用情况，并通过 cAdvisor/metric-server 监控节点和容器的资源。



配置并启动 kubelet， flanneld (master 与 node 节点都需要安装)

在 Master 节点配置 kubelet

```
[root@K8S-PROD-MASTER-A1 pki]# cat > /usr/lib/systemd/system/kubelet.service << EOF
[Unit]
Description=Kubernetes Kubelet Server
Documentation=https://github.com/kubernetes/kubernetes
After=docker.service
Requires=docker.service

[Service]
```

```
EnvironmentFile=-/data/apps/kubernetes/etc/kubelet.conf
ExecStart=/data/apps/kubernetes/node/bin/kubelet \\
\$KUBE_LOGTOSTDERR \\
\$KUBE_LOG_LEVEL \\
\$KUBELET_CONFIG \\
\$KUBELET_HOSTNAME \\
\$KUBELET_POD_INFRA_CONTAINER \\
\$KUBELET_ARGS
Restart=on-failure

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

创建 kubelet 配置文件

```
[root@K8S-PROD-MASTER-A1 pki]# cat > /data/apps/kubernetes/etc/kubelet.conf << EOF
KUBE_LOGTOSTDERR="--logtostderr=false"
KUBE_LOG_LEVEL="--v=2 --log-dir=/data/apps/kubernetes/log/"

KUBELET_HOSTNAME="--hostname-override=10.211.18.4"
KUBELET_POD_INFRA_CONTAINER="--pod-infra-container-image=registry.cn-hangzhou.aliyu
ncs.com/google_containers/pause-amd64:3.1"
KUBELET_CONFIG="--config=/data/apps/kubernetes/etc/kubelet-config.yml"
KUBELET_ARGS="--bootstrap-kubeconfig=/data/apps/kubernetes/etc/kubelet-bootstrap.ku
beconfig --kubeconfig=/data/apps/kubernetes/etc/kubelet.kubeconfig --cert-dir=/data/
apps/kubernetes/pki --feature-gates=RotateKubeletClientCertificate=true"
EOF
```

```
[root@K8S-PROD-MASTER-A1 pki]# cat > /data/apps/kubernetes/etc/kubelet-config.yml <<
 EOF
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
address: 10.211.18.4
port: 10250
cgroupDriver: cgroupfs
clusterDNS:
  - 10.99.110.110
clusterDomain: ziji.work.
hairpinMode: promiscuous-bridge
maxPods: 200
failSwapOn: false
```

```
imageGCHighThresholdPercent: 90
imageGCLowThresholdPercent: 80
imageMinimumGCAge: 5m0s
serializeImagePulls: false
authentication:
  x509:
    clientCAFile: /data/apps/kubernetes/pki/ca.pem
  anonymous:
    enbaled: false
  webhook:
    enbaled: false
EOF
```

启动 kubelet

```
[root@K8S-PROD-MASTER-A1 pki]# systemctl daemon-reload
[root@K8S-PROD-MASTER-A1 pki]# systemctl enable kubelet
[root@K8S-PROD-MASTER-A1 pki]# systemctl restart kubelet
[root@K8S-PROD-MASTER-A1 pki]# systemctl  status kubelet
● kubelet.service - Kubernetes Kubelet Server
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset:
disabled)
   Active: active (running) since Mon 2019-04-29 18:35:33 CST; 13s ago
     Docs: https://github.com/kubernetes/kubernetes
 Main PID: 29401 (kubelet)
    Tasks: 8
   Memory: 13.0M
   CGroup: /system.slice/kubelet.service
           └─29401 /data/apps/kubernetes/server/bin/kubelet --logtostderr=true --v=
0 --config=/data/apps/kubernetes/etc/kubelet-config.yml --hostname-override=10.211.1
8.4 --pod-infra-container-i...

Apr 29 18:35:33 K8S-PROD-MASTER-A1 systemd[1]: kubelet.service holdoff time over, sc
heduling restart.
Apr 29 18:35:33 K8S-PROD-MASTER-A1 systemd[1]: Stopped Kubernetes Kubelet Server.
Apr 29 18:35:33 K8S-PROD-MASTER-A1 systemd[1]: Started Kubernetes Kubelet Server.
Apr 29 18:35:33 K8S-PROD-MASTER-A1 kubelet[29401]: I0429 18:35:33.694991   29401 ser
ver.go:407] Version: v1.13.4
Apr 29 18:35:33 K8S-PROD-MASTER-A1 kubelet[29401]: I0429 18:35:33.695245   29401 plu
gins.go:103] No cloud provider specified.
Apr 29 18:35:33 K8S-PROD-MASTER-A1 kubelet[29401]: E0429 18:35:33.699058   29401 boo
tstrap.go:184] Unable to read existing bootstrap client config: invalid configuratio
n: [unable to read client-...
```

在 Node 节点配置 kubelet

```
[root@K8S-PROD-NODE-A1 etc]#  cd software/
[root@K8S-PROD-NODE-A1 etc]#  tar zxf kubernetes-node-linux-amd64.tar.gz
[root@K8S-PROD-NODE-A1 etc]#  mv kubernetes/node  /data/apps/kubernetes/
```

```
[root@K8S-PROD-NODE-A1 etc]#  cat > /etc/systemd/system/kubelet.service << EOF
[Unit]
Description=Kubernetes Kubelet Server
Documentation=https://github.com/kubernetes/kubernetes
After=docker.service
Requires=docker.service

[Service]
EnvironmentFile=-/data/apps/kubernetes/etc/kubelet.conf
ExecStart=/data/apps/kubernetes/node/bin/kubelet \
$KUBE_LOGTOSTDERR \
$KUBE_LOG_LEVEL \
$KUBELET_CONFIG \
$KUBELET_HOSTNAME \
$KUBELET_POD_INFRA_CONTAINER \
$KUBELET_ARGS
Restart=on-failure

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

```
[root@K8S-PROD-NODE-A1 etc]# cat > /data/apps/kubernetes/etc/kubelet << EOF
KUBE_LOGTOSTDERR="--logtostderr=false"
KUBE_LOG_LEVEL="--v=2 --log-dir=/data/apps/kubernetes/log/"

KUBELET_HOSTNAME="--hostname-override=10.211.18.11"
KUBELET_POD_INFRA_CONTAINER="--pod-infra-container-image=registry.cn-hangzhou.aliyu
ncs.com/google_containers/pause-amd64:3.1"
KUBELET_CONFIG="--config=/data/apps/kubernetes/etc/kubelet-config.yml"
```

```
KUBELET_ARGS="--bootstrap-kubeconfig=/data/apps/kubernetes/etc/kubelet-bootstrap.ku
beconfig --kubeconfig=/data/apps/kubernetes/etc/kubelet.kubeconfig --cert-dir=/data/
apps/kubernetes/pki"
EOF
```

```
[root@K8S-PROD-NODE-A1 etc]# cat > /data/apps/kubernetes/etc/kubelet-config.yml << E
OF
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
address: 10.211.18.11
port: 10250
cgroupDriver: cgroupfs
clusterDNS:
  - 10.99.110.110
clusterDomain: ziji.work.
hairpinMode: promiscuous-bridge
maxPods: 200
failSwapOn: false
imageGCHighThresholdPercent: 90
imageGCLowThresholdPercent: 80
imageMinimumGCAge: 5m0s
serializeImagePulls: false
authentication:
  x509:
    clientCAFile: /data/apps/kubernetes/pki/ca.pem
  anonymous:
    enbaled: false
  webhook:
    enbaled: false
EOF
```

node1 启动 kubelet

```
[root@K8S-PROD-NODE-A1 etc]# systemctl daemon-reload
[root@K8S-PROD-NODE-A1 etc]# systemctl enable kubelet
[root@K8S-PROD-NODE-A1 etc]# systemctl restart kubelet
[root@K8S-PROD-NODE-A1 etc]# systemctl status kubelet
● kubelet.service - Kubernetes Kubelet Server
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset:
disabled)
   Active: active (running) since Fri 2019-03-22 11:55:33 CST; 14s ago
     Docs: https://github.com/kubernetes/kubernetes
```



```
 Main PID: 11124 (kubelet)
    Tasks: 8
   Memory: 13.1M
   CGroup: /system.slice/kubelet.service
           └─11124 /data/apps/kubernetes/node/bin/kubelet --logtostderr=true --v=0
--config=/data/apps/kubernetes/etc/kubelet-config.yml --hostname-override=10.211.18.
11 --pod-infra-container-im...

Mar 22 11:55:33 K8S-PROD-NODE-A1 systemd[1]: kubelet.service holdoff time over, sche
duling restart.
Mar 22 11:55:33 K8S-PROD-NODE-A1 systemd[1]: Stopped Kubernetes Kubelet Server.
Mar 22 11:55:33 K8S-PROD-NODE-A1 systemd[1]: Started Kubernetes Kubelet Server.
Mar 22 11:55:33 K8S-PROD-NODE-A1 kubelet[11124]: I0322 11:55:33.941658   11124 serve
r.go:407] Version: v1.13.4
Mar 22 11:55:33 K8S-PROD-NODE-A1 kubelet[11124]: I0322 11:55:33.942154   11124 plugi
ns.go:103] No cloud provider specified.
Mar 22 11:55:33 K8S-PROD-NODE-A1 kubelet[11124]: E0322 11:55:33.965594   11124 boots
trap.go:184] Unable to read existing bootstrap client config:
```

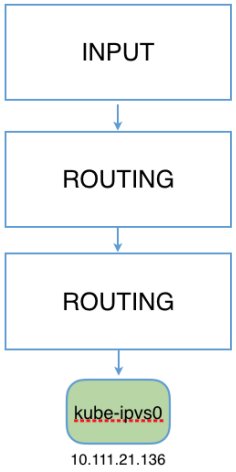

在 node 上操作

重复以上步骤， 修改 kubelet-config.yml address:地址为 node 节点 ip, --hostname-override= 为 node ip 地址

2.6.12.14 配置 kube-proxy

kube-proxy 负责为 Service 提供 cluster 内部的服务发现和负载均衡；

每台机器上都运行一个 kube-proxy 服务，它监听 API server 中 service 和 endpoint 的变化情况，并通过 ipvs/iptables 等来为服务配置负载均衡（仅支持 TCP 和 UDP）。




```
$ ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.111.21.136:80 rr persistent 10800
  -> 192.168.23.130:80            Masq    1      0          0
  -> 192.168.23.134:80            Masq    1      0          0
```

所有节点都要配置 kube-proxy

注意：使用 ipvs 模式时，需要预先在每台 Node 上加载内核模块 nf_conntrack_ipv4, ip_vs, ip_vs_rr, ip_vs_wrr, ip_vs_sh 等。

master 节点操作

```
安装 conntrack-tools
[root@K8S-PROD-MASTER-A1]#  yum install -y conntrack-tools ipvsadm ipset conntrack l
ibseccomp
```

创建服务启动文件

```
[root@K8S-PROD-MASTER-A1]# cat > /usr/lib/systemd/system/kube-proxy.service << EOF
[Unit]
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/kubernetes/kubernetes
After=network.target

[Service]
EnvironmentFile=-/data/apps/kubernetes/etc/kube-proxy.conf
ExecStart=/data/apps/kubernetes/server/bin/kube-proxy \
$KUBE_LOGTOSTDERR \
$KUBE_LOG_LEVEL \
$KUBECONFIG \
$KUBE_PROXY_ARGS
```

```
Restart=on-failure
LimitNOFILE=65536
KillMode=process

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

启用 ipvs 主要就是把 kube-proxy 的--proxy-mode 配置选项修改为 ipvs,并且要启用--masquerade-all，使用 iptables 辅助 ipvs 运行。

```
[root@K8S-PROD-MASTER-A1]# cat > /data/apps/kubernetes/etc/kube-proxy.conf << EOF
KUBE_LOGTOSTDERR="--logtostderr=false"
KUBE_LOG_LEVEL="--v=2 --log-dir=/data/apps/kubernetes/log/"

KUBECONFIG="--kubeconfig=/data/apps/kubernetes/etc/kube-proxy.kubeconfig"
KUBE_PROXY_ARGS="--proxy-mode=ipvs --masquerade-all=true --cluster-cidr=10.99.0.0/16
"
EOF
```

启动 kube-proxy 并设置为开机自启动

```
[root@K8S-PROD-MASTER-A1 kubernetes]# systemctl daemon-reload
[root@K8S-PROD-MASTER-A1 kubernetes]# systemctl enable kube-proxy
[root@K8S-PROD-MASTER-A1 kubernetes]# systemctl restart kube-proxy
[root@K8S-PROD-MASTER-A1 kubernetes]# systemctl status kube-proxy
● kube-proxy.service - Kubernetes Kube-Proxy Server
   Loaded: loaded (/usr/lib/systemd/system/kube-proxy.service; enabled; vendor prese
t: disabled)
   Active: active (running) since Mon 2019-04-29 10:19:21 CST; 23h ago
     Docs: https://github.com/kubernetes/kubernetes
 Main PID: 5310 (kube-proxy)
    Tasks: 0
   Memory: 29.1M
   CGroup: /system.slice/kube-proxy.service
           ‣ 5310 /data/apps/kubernetes/server/bin/kube-proxy --logtostderr=false --v
=0 --kubeconfig=/data/apps/kubernetes/etc/kube-proxy.conf --proxy-mode=ipvs --masque
rade-all=true
```

在所有的 node 上操作

```
安装 conntrack-tools
[root@K8S-PROD-NODE-A1 kubernetes]#   yum install -y conntrack-tools ipvsadm ipset c
onntrack libseccomp
```

创建服务启动文件

```
[root@K8S-PROD-NODE-A1 kubernetes]# cat > /usr/lib/systemd/system/kube-proxy.service
 << EOF
[Unit]
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/kubernetes/kubernetes
After=network.target

[Service]
EnvironmentFile=-/data/apps/kubernetes/etc/kube-proxy.conf
ExecStart=/data/apps/kubernetes/server/bin/kube-proxy \\
      \$KUBE_LOGTOSTDERR \\
      \$KUBE_LOG_LEVEL \\
      \$KUBECONFIG \\
      \$KUBE_PROXY_ARGS
Restart=on-failure
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOF


[root@K8S-PROD-NODE-A1 kubernetes]# cat > /data/apps/kubernetes/etc/kube-proxy.conf
<< EOF
KUBE_LOGTOSTDERR="--logtostderr=false"
KUBE_LOG_LEVEL="--v=2 --log-dir=/data/apps/kubernetes/log/"

KUBECONFIG="--kubeconfig=/data/apps/kubernetes/etc/kube-proxy.conf"
KUBE_PROXY_ARGS="--proxy-mode=ipvs --masquerade-all=true --cluster-cidr=10.99.0.0/16
"
EOF
```

启动 kube-proxy 并设置为开机自启动

```
[root@K8S-PROD-NODE-A1 kubernetes]# systemctl daemon-reload
[root@K8S-PROD-NODE-A1 kubernetes]# systemctl enable kube-proxy
[root@K8S-PROD-NODE-A1 kubernetes]# systemctl restart kube-proxy
[root@K8S-PROD-NODE-A1 kubernetes]# systemctl status kube-proxy
● kube-proxy.service - Kubernetes Kube-Proxy Server
   Loaded: loaded (/usr/lib/systemd/system/kube-proxy.service; enabled; vendor prese
t: disabled)
   Active: active (running) since Tue 2019-03-19 17:50:53 CST; 3 days ago
     Docs: https://github.com/kubernetes/kubernetes
 Main PID: 16909 (kube-proxy)
    Tasks: 0
   Memory: 8.5M
   CGroup: /system.slice/kube-proxy.service
           ‣ 16909 /data/apps/kubernetes/node/bin/kube-proxy --logtostderr=false --v=
0 --kubeconfig=/data/apps/kubernetes/etc/kube-proxy.conf --proxy-mode=ipvs --masquer
ade-all=true --cluster-cid...
```

2.6.13 通过证书验证添加各个节点

步骤 1 在 master 节点操作

```
[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl  get csr
NAME                                                  AGE   REQUESTOR           CONDITIO
N
node-csr-NT3oojJY6_VRvkkNKQMEBZyK6BZoOnEbKCqcEvjnqco   16h   kubelet-bootstrap   Pen
ding
node-csr-O8Zzno_W7X7QLt7EqdaVvvXfg0RS3AaSbdoOzUO821M   16h   kubelet-bootstrap   Pen
ding
node-csr-gXAsSMLxfcmdwaOeu8swemiqFQOcrZaPw85yfcguNlA   16h   kubelet-bootstrap   Pen
ding
node-csr-sjskQXhS3v8ZS7Wi0rjSdGocdTsMw_zHWL8quCATR5c   16h   kubelet-bootstrap   Pen
ding
```

步骤 2 通过验证并添加进集群

```
[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl get csr | awk '/node/{print $1}' | xar
gs kubectl certificate approve
```



```
[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl get csr | awk '/node/{print $1}' | xargs kubectl certificate approve
certificatesigningrequest.certificates.k8s.io/node-csr-NT3oojJY6_VRvkkNKQMEBZyK6BZoOnEbKCqcEvjnqco approved
certificatesigningrequest.certificates.k8s.io/node-csr-O8Zzno_W7X7QLt7EqdaVvvXfg0RS3AaSbdoOzUO821M approved
certificatesigningrequest.certificates.k8s.io/node-csr-gXAsSMLxfcmdwaOeu8swemiqFQOcrZaPw85yfcguNlA approved
certificatesigningrequest.certificates.k8s.io/node-csr-sjskQXhS3v8ZS7Wi0rjSdGocdTsMw_zHWL8quCATR5c approved
[root@K8S-PROD-MASTER-A1 kubernetes]#
```

```
###单独执行命令例子：
kubectl certificate approve node-csr-O8Zzno_W7X7QLt7EqdaVvvXfg0RS3AaSbdoOzUO821M
```

```
#查看节点
#此时节点状态为 NotReady，因为还没有配置网络

[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl  get nodes
NAME           STATUS     ROLES    AGE   VERSION
10.211.18.11   Ready      <none>   41s   v1.13.6
10.211.18.4    Ready      <none>   41s   v1.13.6
10.211.18.5    Ready      <none>   41s   v1.13.6
10.211.18.50   NotReady   <none>   41s   v1.13.6
10.211.18.6    Ready      <none>   41s   v1.13.6
```

```
# 在 node 节点查看生成的文件

[root@K8S-PROD-NODE-A1 kubernetes]# ls -l /data/apps/kubernetes/etc/kubelet.conf
-rw------- 1 root root 2315 Apr 30 10:02 /data/apps/kubernetes/etc/kubelet.conf

[root@K8S-PROD-NODE-A1 kubernetes]# ls -l /data/apps/kubernetes/pki/kubelet*
-rw------- 1 root root 1277 Apr 30 10:02 /data/apps/kubernetes/pki/kubelet-client-20
19-04-30-10-02-49.pem
lrwxrwxrwx 1 root root   64 Apr 30 10:02 /data/apps/kubernetes/pki/kubelet-client-cu
rrent.pem -> /data/apps/kubernetes/pki/kubelet-client-2019-04-30-10-02-49.pem
-rw-r--r-- 1 root root 2177 Mar 19 11:46 /data/apps/kubernetes/pki/kubelet.crt
-rw------- 1 root root 1675 Mar 19 11:46 /data/apps/kubernetes/pki/kubelet.key
```

2.7 设置集群角色

```
在任意 master 节点操作， 为集群节点打 label
[root@K8S-PROD-MASTER-A1]#
kubectl label nodes 10.211.18.4 node-role.kubernetes.io/master=MASTER-A1
```

```
kubectl label nodes 10.211.18.5 node-role.kubernetes.io/master=MASTER-A2
kubectl label nodes 10.211.18.6 node-role.kubernetes.io/master=MASTER-A3

设置 192.168.1.11 - 14 lable 为 node
kubectl label nodes 10.211.18.11 node-role.kubernetes.io/node=NODE-A1
kubectl label nodes 10.211.18.12 node-role.kubernetes.io/node=NODE-A2
kubectl label nodes 10.211.18.13 node-role.kubernetes.io/node=NODE-A3
kubectl label nodes 10.211.18.14 node-role.kubernetes.io/node=NODE-A4

为 Ingress 边缘节点设置 label
kubectl label nodes 10.211.18.50 node-role.kubernetes.io/LB=LB-A1
kubectl label nodes 10.211.18.51 node-role.kubernetes.io/LB=LB-B1

设置 master 一般情况下不接受负载
kubectl taint nodes 10.211.18.4 node-role.kubernetes.io/master=MASTER-A1:NoSchedule
--overwrite
node/10.211.18.4 modified
kubectl taint nodes 10.211.18.5 node-role.kubernetes.io/master=MASTER-A2:NoSchedule
--overwrite
node/10.211.18.4 modified
kubectl taint nodes 10.211.18.6 node-role.kubernetes.io/master=MASTER-A3:NoSchedule
--overwrite
```

如何删除 lable ？ – overwrite

```
kubectl label nodes 10.211.18.50  node-role.kubernetes.io/LB- --overwrite
node/10.211.18.50 labeled
[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl  get nodes --show-labels
NAME           STATUS     ROLES    AGE   VERSION   LABELS
10.211.18.11   Ready      node     41d   v1.13.4   beta.kubernetes.io/arch=amd64,bet
a.kubernetes.io/os=linux,kubernetes.io/hostname=10.211.18.11,node-role.kubernetes.i
o/node=NODE-A1
10.211.18.4    Ready      master   41d   v1.13.4   beta.kubernetes.io/arch=amd64,bet
a.kubernetes.io/os=linux,kubernetes.io/hostname=10.211.18.4,node-role.kubernetes.io
/master=MASTER-A1
10.211.18.5    Ready      master   41d   v1.13.4   beta.kubernetes.io/arch=amd64,bet
a.kubernetes.io/os=linux,kubernetes.io/hostname=10.211.18.5,node-role.kubernetes.io
/master=MASTER-A2
10.211.18.50   NotReady   <none>   41d   v1.13.4   beta.kubernetes.io/arch=amd64,bet
a.kubernetes.io/os=linux,kubernetes.io/hostname=10.211.18.50
10.211.18.6    Ready      master   41d   v1.13.4   beta.kubernetes.io/arch=amd64,bet
a.kubernetes.io/os=linux,kubernetes.io/hostname=10.211.18.6,node-role.kubernetes.io
/master=MASTER-A3
```


查看节点 Roles，ROLES 已经标识出了 master 和 node

```
[root@K8S-PROD-MASTER-A1 kubernetes]# kubectl  get node
NAME           STATUS     ROLES    AGE   VERSION
10.211.18.11   Ready      node     41d   v1.13.6
10.211.18.4    Ready      master   41d   v1.13.6
10.211.18.5    Ready      master   41d   v1.13.6
10.211.18.50   NotReady   LB       41d   v1.13.6
10.211.18.6    Ready      master   41d   v1.13.6
```

2.8 配置网络插件

Master 和 node 节点

```
[root@K8S-PROD-NODE-A1 docker]# cd /opt/software/
[root@K8S-PROD-NODE-A1 software]# tar zxvf flannel-v0.11.0-linux-amd64.tar.gz
mv flanneld mk-docker-opts.sh  /data/apps/kubernetes/node/bin/
chmod +x /data/apps/kubernetes/node/bin/*
```


2.8.1 创建 flanneld.conf 配置文件

2.8.1.15 创建网络段

```
#在 etcd 集群执行如下命令， 为 docker 创建互联网段
etcdctl \
--ca-file=/data/apps/etcd/ssl/etcd-ca.pem --cert-file=/data/apps/etcd/ssl/etcd.pem --key-file=/data/apps/etcd/ssl/etcd-key.pem \
--endpoints="https://10.211.18.4:2379,https://10.211.18.5:2379,https://10.211.18.6:2379" \
set /coreos.com/network/config '{ "Network": "10.99.0.0/16", "Backend": {"Type": "vxlan"}}'
```

步骤 1 在 node 节点创建 etcd 证书存放路径， 并拷贝 etcd 证书到 Node 节点

```
[root@K8S-PROD-NODE-A1 software]# mkdir -p /data/apps/etcd
```

```
[root@K8S-PROD-MASTER-A1 kubernetes]# scp -r /data/apps/etcd/ssl 10.211.18.11:/data/
apps/etcd/
root@10.211.18.11's password:
etcd-ca-key.pem
                                                                      100% 1675   735.4
KB/s   00:00
etcd-ca.pem
                                                                     100% 1371   555.3K
B/s   00:00
etcd-key.pem
                                                                      100% 1675   581.8
KB/s   00:00
etcd.pem
```

步骤 2　创建 etcd 证书目录

```
[root@K8S-PROD-NODE-A1 kubernetes]# ls -l /data/apps/etcd/ssl/
total 16
-rw------- 1 root root 1675 Apr 30 11:01 etcd-ca-key.pem
-rw-r--r-- 1 root root 1371 Apr 30 11:01 etcd-ca.pem
-rw------- 1 root root 1675 Apr 30 11:01 etcd-key.pem
-rw-r--r-- 1 root root 1460 Apr 30 11:01 etcd.pem
```

步骤 3　创建 flannel 配置文件

```
cat > /data/apps/kubernetes/etc/flanneld.conf<< EOF
FLANNEL_OPTIONS="--etcd-endpoints=https://10.211.18.4:2379,https://10.211.18.5:237
9,https://10.211.18.6:2379 -etcd-cafile=/data/apps/etcd/ssl/etcd-ca.pem -etcd-certfi
le=/data/apps/etcd/ssl/etcd.pem -etcd-keyfile=/data/apps/etcd/ssl/etcd-key.pem"

EOF
```

2.8.1.16 创建系统服务

```
[root@K8S-PROD-NODE-A1 kubernetes]# cat > /usr/lib/systemd/system/flanneld.service <
< EOF
[Unit]
Description=Flanneld overlay address etcd agent
After=network-online.target network.target
Before=docker.service
```

```
[Service]
Type=notify
EnvironmentFile=/data/apps/kubernetes/etc/flanneld.conf
ExecStart=/data/apps/kubernetes/node/bin/flanneld --ip-masq $FLANNEL_OPTIONS
ExecStartPost=/data/apps/kubernetes/node/bin/mk-docker-opts.sh -k DOCKER_NETWORK_OPT
IONS -d /run/flannel/subnet.env
Restart=on-failure

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

注意： master 节点的 flanneld 服务配置文件/node/bin/ 需要改为/server/bin/

2.8.1.17 修改 docker.service 启动文件

添加子网配置文件

```
[root@K8S-PROD-NODE-A1 kubernetes]# vi /usr/lib/systemd/system/docker.service
# --graph 表示修改 docker 默认/var/lib/docker 存储路径为/data/docker ，需提前创建目录
EnvironmentFile=/run/flannel/subnet.env
ExecStart=/usr/bin/dockerd -H unix:// $DOCKER_NETWORK_OPTIONS $DOCKER_DNS_OPTIONS
```

修改 docker 服务启动文件，注入 dns 参数

```
dns 根据实际部署的 dns 服务来填写
[root@K8S-PROD-NODE-A1 kubernetes]# mkdir -p /usr/lib/systemd/system/docker.service.
d/
[root@K8S-PROD-NODE-A1 kubernetes]# vi /usr/lib/systemd/system/docker.service.d/dock
er-dns.conf
[Service]
Environment="DOCKER_DNS_OPTIONS=\
--dns 10.99.110.110 --dns 114.114.114.114 \
--dns-search default.svc.ziji.work --dns-search svc.ziji.work \
--dns-opt ndots:2 --dns-opt timeout:2 --dns-opt attempts:2"
```

启动 flanneld

```
systemctl  daemon-reload
systemctl  start flanneld
```

```
systemctl  restart docker
systemctl  status  flanneld
● flanneld.service - Flanneld overlay address etcd agent
   Loaded: loaded (/usr/lib/systemd/system/flanneld.service; enabled; vendor preset:
 disabled)
   Active: active (running) since Tue 2019-04-30 15:42:03 CST; 1min 29s ago
  Process: 17685 ExecStartPost=/data/apps/kubernetes/node/bin/mk-docker-opts.sh -k D
OCKER_NETWORK_OPTIONS -d /run/flannel/subnet.env (code=exited, status=0/SUCCESS)
 Main PID: 17668 (flanneld)
    Tasks: 8
   Memory: 10.7M
   CGroup: /system.slice/flanneld.service
           └─17668 /data/apps/kubernetes/node/bin/flanneld --ip-masq --etcd-endpoin
ts=https://10.211.18.4:2379,https://10.211.18.5:2379,https://10.211.18.6:2379 -etcd-
cafile=/data/apps/etcd/ssl/...

Apr 30 15:42:03 K8S-PROD-NODE-A1 flanneld[17668]: I0430 15:42:03.060859   17668 mai
n.go:351] Current network or subnet (10.99.0.0/16, 10.99.79.0/24) is not equal to pr
evious one (0.0...tables rules
Apr 30 15:42:03 K8S-PROD-NODE-A1 flanneld[17668]: I0430 15:42:03.098916   17668 ipta
bles.go:167] Deleting iptables rule: -s 0.0.0.0/0 -d 0.0.0.0/0 -j RETURN
Apr 30 15:42:03 K8S-PROD-NODE-A1 flanneld[17668]: I0430 15:42:03.100839   17668 ipta
bles.go:167] Deleting iptables rule: -s 0.0.0.0/0 ! -d 224.0.0.0/4 -j MASQUERADE
Apr 30 15:42:03 K8S-PROD-NODE-A1 flanneld[17668]: I0430 15:42:03.102994   17668 ipta
bles.go:167] Deleting iptables rule: ! -s 0.0.0.0/0 -d 0.0.0.0/0 -j RETURN
```

2.9 配置 coredns

```
#10.99.110.110 是 kubelet 中配置的 dns
#安装 coredns

[root@K8S-PROD-MASTER-A1 ~]# cd /root && mkdir coredns && cd coredns
[root@K8S-PROD-MASTER-A1 coredns]# wget https://raw.githubusercontent.com/coredns/de
ployment/master/kubernetes/coredns.yaml.sed
[root@K8S-PROD-MASTER-A1 coredns]# wget https://raw.githubusercontent.com/coredns/de
ployment/master/kubernetes/deploy.sh
[root@K8S-PROD-MASTER-A1 coredns]# chmod +x deploy.sh

修改 coredns.yaml 文件，  增加红色字段内容
```



```
data:
  Corefile: |
    .:53 {
        errors
        health
        ready
        kubernetes ziji.work 10.99.0.0/16 {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf
        cache 20
        loop
        reload
        loadbalance
    }
```

```
[root@K8S-PROD-MASTER-A1 coredns]# ./deploy.sh -i 10.99.110.110 > coredns.yml

[root@K8S-PROD-MASTER-A1 coredns]# kubectl  apply -f coredns.yml
serviceaccount/coredns created
clusterrole.rbac.authorization.k8s.io/system:coredns created
clusterrolebinding.rbac.authorization.k8s.io/system:coredns created
configmap/coredns created
deployment.apps/coredns created
service/coredns created
```

```
#查看 coredns 是否运行正常

[root@K8S-PROD-MASTER-A1 coredns]# kubectl get svc,pods -n kube-system
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                  AGE
service/coredns   ClusterIP   10.99.110.110   <none>        53/UDP,53/TCP,9153/TCP
11m

NAME                          READY   STATUS    RESTARTS   AGE
pod/coredns-756d6db49-7rbfd   1/1     Running   0          11m
pod/coredns-756d6db49-jg797   1/1     Running   0          11m
```

2.10 测试 DNS 解析

创建一个 nginx 应用，测试应用和 dns 是否正常

```
[root@K8S-PROD-MASTER-A1 coredns]# cd /root && mkdir nginx && cd nginx
cat << EOF > nginx.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-test
spec:
  selector:
    app: nginx-test
  type: NodePort
  ports:
  - port: 80
    nodePort: 31000
    name: nginx-port
    targetPort: 80
    protocol: TCP

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-test
  template:
    metadata:
      name: nginx-test
      labels:
        app: nginx-test
    spec:
      containers:
      - name: nginx-test
        image: nginx
```



```
      ports:
      - containerPort: 80
EOF

[root@K8S-PROD-MASTER-A1 nginx]# kubectl apply -f nginx.yaml
```

2.10.1 创建一个 pod 用来测试 dns

```
[root@K8S-PROD-MASTER-A1 nginx]# kubectl run curl --image=radial/busyboxplus:curl -i
 --tty
kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a fu
ture version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.
If you don't see a command prompt, try pressing enter.
[ root@curl-66959f6557-tl5rn:/ ]$ nslookup  kubernetes
Server:    10.99.110.110
Address 1: 10.99.110.110 coredns.kube-system.svc.ziji.work

Name:      kubernetes
Address 1: 10.99.0.1 kubernetes.default.svc.ziji.work
```


2.11 部署 Dashboard

```
1、下载 dashboard 镜像
# 个人的镜像
registry.cn-hangzhou.aliyuncs.com/dnsjia/k8s:kubernetes-dashboard-amd64_v1.10.0

2、下载 yaml 文件
[root@K8S-PROD-MASTER-A1 dashboard]# curl -O   https://soft.8090st.com/kubernetes/da
shboard/kubernetes-dashboard.yaml
```

修改证书

在使用–auto-generate-certificates 自动生成证书后，访问 dashboard 报错：NET::ERR_CERT_INVALID

查看 dashboard 的日志提示证书未找到，为解决这个问题，将生成好的 dashboard.crt 和

dashboard.key
挂载到容器的/certs 下，然后重新发布 deployment 即可

CA 证书的生成可以参考如下配置

```
$cd ~
$mkdir certs

$ openssl genrsa -des3 -passout pass:x -out dashboard.pass.key 2048
...
$ openssl rsa -passin pass:x -in dashboard.pass.key -out dashboard.key
# Writing RSA key
$ rm dashboard.pass.key
$ openssl req -new -key dashboard.key -out dashboard.csr
...
Country Name (2 letter code) [AU]: US
...
A challenge password []:
...

Generate SSL certificate
The self-signed SSL certificate is generated from the dashboard.key private key and d
ashboard.csr files.

$ openssl x509 -req -sha256 -days 365 -in dashboard.csr -signkey dashboard.key -out d
ashboard.crt
注意： 默认生成证书到期时间为一年， 修改过期时间为 10 年  -days 3650
```

将创建的证书拷贝到其他 node 节点

```
这里我采取的是 hostPath 方式挂载,这个需要保证 dashboard 调度到的 node 上都要有这个文件；
其他挂载的方式可以参考[官网](https://kubernetes.io/docs/concepts/storage/volumes/)
修改 kubernetes-dashboard.yaml 为如下内容
volumes:
- name: kubernetes-dashboard-certs
# secret:
# secretName: kubernetes-dashboard-certs
hostPath:
path: /data/apps/kubernetes/certs
type: Directory
```



```
- name: tmp-volume
emptyDir: {}
```

```
#在 master 节点执行如下命令，部署 ui
[root@K8S-PROD-MASTER-A1 dashboard]# kubectl  apply -f kubernetes-dashboard.yaml
secret/kubernetes-dashboard-certs created
serviceaccount/kubernetes-dashboard created
role.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
deployment.apps/kubernetes-dashboard created
service/kubernetes-dashboard created
```

2.11.1 配置 Dashboard 令牌

```
[root@K8S-PROD-MASTER-A1 dashboard]# vi token.sh

#!/bin/bash

if kubectl get sa dashboard-admin -n kube-system &> /dev/null;then
echo -e "\033[33mWARNING: ServiceAccount dashboard-admin exist!\033[0m"
else
kubectl create sa dashboard-admin -n kube-system
kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --serv
iceaccount=kube-system:dashboard-admin
fi

[root@K8S-PROD-MASTER-A1 dashboard]# sh token.sh  #生成登录令牌

[root@K8S-PROD-MASTER-A1 dashboard]#  kubectl describe secret -n kube-system $(kubec
tl get secrets -n kube-system | grep dashboard-admin | cut -f1 -d ' ') | grep -E '^to
ken'


token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2N
vdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsIm
t1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJkYXNoYm9hcmQtYWRtaW4tdG9rZ
W4tYzlrNXMiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoi
ZGFzaGJvYXJkLWFkbWluIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQ
udWlkIjoiNDJkMjc1NTctZWZkMC0xMWU4LWFmNTgtMDAwYzI5OWJiNzhkIiwic3ViIjoic3lzdGVtOnNlcn
ZpY2VhY2NvdW50Omt1YmUtc3lzdGVtOmRhc2hib2FyZC1hZG1pbiJ9.HoEtcgtjMbM_DZ8J3w5xq_gZrr1M
```


-C5Axtt_PbGw39TbMqetsk1oCVNUdY5Hv_9z-liC-DBo2O-NO6IvPdrYBjgADwPBgc3fSjrZMfI8gDqwsKD
IVF6VXCzaMAy-QeqUh-zgoqZa93MdBaBlhGQXtLyx0kso8XMGQccPndnzjqRw_8gWXNX2Lt5vLkEDTYcBMk
qoGuwLJymQVtFVUwBHEHi9VIDgN4j5YV72ZDK320YgyS_nwjqwicyWpkDWq03yWhyJKyPGQ_Z8cylotCKr8
jFqxU7oEoX7lfu3SJA19C_ds5Ak0OJi7tMobI59APL-u8xdigvd0MZivsQS0AWDsA

2.11.2 登录 dashboard

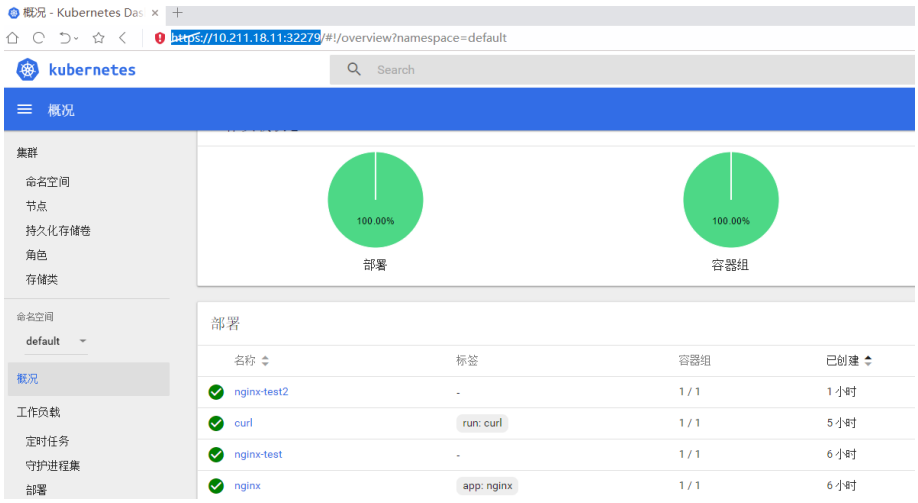
1、通过 node 节点 ip+端口号访问

```
[root@K8S-PROD-MASTER-A1 dashboard]#  kubectl get svc,pod -n kube-system  -o wide
```

```
[root@K8S-PROD-MASTER-A1 dashboard]#  kubectl get svc,pod -n kube-system  -o wide
NAME                            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                  AGE     SELECTOR
service/coredns                 ClusterIP   10.99.110.110   <none>        53/UDP,53/TCP,9153/TCP   6h24m   k8s-app=coredns
service/kubernetes-dashboard    NodePort    10.99.255.171   <none>        443:32279/TCP            18m     k8s-app=kubernetes-dashboard

NAME                                    READY   STATUS    RESTARTS   AGE     IP            NODE            NOMINATED NODE   READINES
pod/coredns-756d6db49-7rbfd             1/1     Running   0          6h24m   10.99.79.4    10.211.18.11    <none>           <none>
pod/coredns-756d6db49-jg797             1/1     Running   0          6h24m   10.99.79.3    10.211.18.11    <none>           <none>
pod/kubernetes-dashboard-5974995975-9cp76   1/1     Running   0          16m     10.99.79.8    10.211.18.11    <none>           <none>
[root@K8S-PROD-MASTER-A1 dashboard]#
```

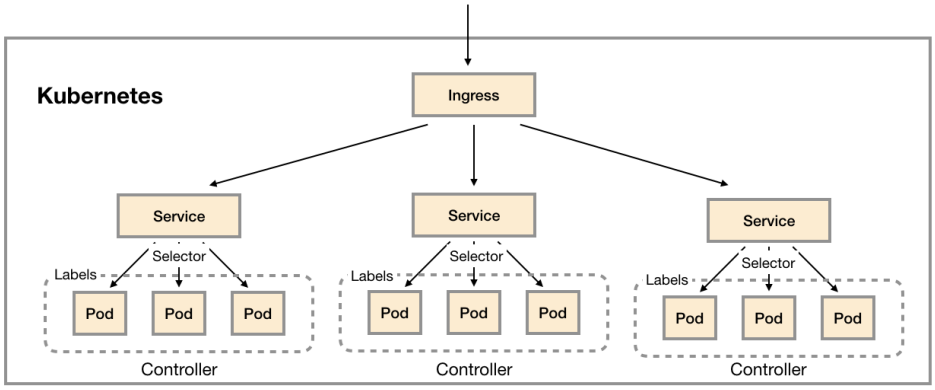
我们可以看到 dashboard pod 被调度到了 10.211.18.11 节点， 暴露端口为 32279

https://10.211.18.11:32279



2.12 配置 ingress(边缘节点)

什么叫边缘节点（Edge Node），所谓的边缘节点即集群内部用来向集群外暴露服务能力的节点，集群外部的服务通过该节点来调用集群内部的服务，边缘节点是集群内外交流的一个 Endpoint。



边缘节点要考虑两个问题

- 边缘节点的高可用，不能有单点故障，否则整个 kubernetes 集群将不可用
- 对外的一致暴露端口，即只能有一个外网访问 IP 和端口

对边缘节点打污点， 防止其他 Pod 被调度到 ingress 节点。

```
[root@K8S-PROD-MASTER-A1 ~]# kubectl  taint node 10.211.18.50 node-role.kubernetes.i
o/LB=LB-A1:NoSchedule
node/10.211.18.50 tainted

[root@K8S-PROD-MASTER-A1 ~]# kubectl  taint node 10.211.18.51 node-role.kubernetes.i
o/LB=LB-B1:NoSchedule
node/10.211.18.51 tainted


# 设置边缘节点 label，后面 pod 调度将根据 NodeSelector 选择对应的 Node 节点
[root@K8S-PROD-MASTER-A1 ~]# kubectl  label  nodes 10.211.18.50 edgenode=true
node/10.211.18.50 labeled
[root@K8S-PROD-MASTER-A1 ~]# kubectl  label  nodes 10.211.18.51 edgenode=true
```



```
node/10.211.18.51 labeled
```

```
[root@K8S-PROD-MASTER-A1 ~]# kubectl  describe  nodes  10.211.18.50
Name:               10.211.18.50
Roles:              LB
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    edgenode=true
                    kubernetes.io/hostname=10.211.18.50
                    node-role.kubernetes.io/LB=LB-A1
Annotations:        node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Tue, 19 Mar 2019 16:58:45 +0800
Taints:             edgenode=true:NoSchedule
Unschedulable:      false
```

这里以 DaemonSet 的方式在边缘节点 node 上启动一个 traefik，并使用 hostPort 的方式在 Node 节点监听 80 、443 端口

```
[root@K8S-PROD-MASTER-A1 ~]#  mkdir $HOME/traefik ; cd $HOME/traefik
```

2.12.1 创建 RBAC 文件

```
 [root@K8S-PROD-MASTER-A1 traefik]# vi traefik-rbac.yaml
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
rules:
  - apiGroups:
      - ""
    resources:
      - services
      - endpoints
      - secrets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - extensions
    resources:
      - ingresses
    verbs:
      - get
```



```
      - list
      - watch
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: traefik-ingress-controller
subjects:
- kind: ServiceAccount
  name: traefik-ingress-controller
  namespace: kube-system

[root@K8S-PROD-MASTER-A1 traefik]# kubectl  apply -f traefik-rbac.yaml
clusterrole.rbac.authorization.k8s.io/traefik-ingress-controller created
clusterrolebinding.rbac.authorization.k8s.io/traefik-ingress-controller created
```

2.12.2 创建 traefik.toml 配置文件

```
[root@K8S-PROD-MASTER-A1 traefik]# vi traefik-configmap.yaml
---
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: kube-system
  name: traefik-conf
data:
  traefik.toml: |
    insecureSkipVerify = true
    defaultEntryPoints = ["http","https"]
    [entryPoints]
      [entryPoints.ping]
      address=":8888"
      [entryPoints.ping.auth]
        [entryPoints.ping.auth.basic]
          users = [
           "admin:$apr1$ehrsakXa$zr4qevnn4t.gOV7J8Ia/y1",
           "test:$apr1$H6uskkkW$IgXLP6ewTrSuBkTrqE8wj/",
```



```
        ]
    [ping]
    entryPoint = "ping"

      [entryPoints.http]
      address = ":80"
        [entryPoints.http.redirect]
        #entryPoint = "https"
      [entryPoints.https]
      address = ":443"
        [entryPoints.https.tls]
        minVersion = "VersionTLS12"
        cipherSuites = [
          "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
          "TLS_RSA_WITH_AES_256_GCM_SHA384"
        ]

      [[entryPoints.https.tls.certificates]]
      CertFile = "/ssl/tls.crt"
      KeyFile = "/ssl/tls.key"


#以 configmap 方式创建 traefik.toml
[root@K8S-PROD-MASTER-A1 traefik]# kubectl  apply -f traefik-configmap.yaml
configmap/traefik-conf created
```

2.12.3 创建 secret

```
[root@K8S-PROD-MASTER-A1 traefik]# openssl req -x509 -nodes -days 365 -newkey rsa:20
48 -keyout tls.key -out tls.crt -subj "/CN=*.ziji.work"
Generating a 2048 bit RSA private key
...........+++
.......................+++
writing new private key to 'tls.key'
-----


[root@K8S-PROD-MASTER-A1 traefik]# ls -l tls.*
-rw-r--r-- 1 root root 1099 May  5 10:50 tls.crt
-rw-r--r-- 1 root root 1704 May  5 10:50 tls.key
```



```
[root@K8S-PROD-MASTER-A1 traefik]# kubectl -n kube-system create secret tls traefik-cert --key=tls.key --cert=tls.crt

secret/traefik-cert created
```

2.12.4 部署 Traefik

```
[root@K8S-PROD-MASTER-A1 traefik]# vi traefik-deamonset.yaml

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: traefik-ingress-controller
  namespace: kube-system
---
kind: DaemonSet
apiVersion: extensions/v1beta1
metadata:
  name: traefik-ingress-controller
  namespace: kube-system
  labels:
    k8s-app: traefik-ingress-lb
spec:
  template:
    metadata:
      labels:
        k8s-app: traefik-ingress-lb
        name: traefik-ingress-lb
    spec:
      serviceAccountName: traefik-ingress-controller
      terminationGracePeriodSeconds: 60
      hostNetwork: true
      restartPolicy: Always
      volumes:
       - name: ssl
         secret:
          secretName: traefik-cert
       - name: traefik-config
         configMap:
```

```
        name: traefik-conf
    containers:
    - image: traefik:v1.7
      imagePullPolicy: IfNotPresent
      name: traefik-ingress-lb
      ports:
      - name: http
        containerPort: 80
        hostPort: 80
      - name: https
        containerPort: 443
        hostPort: 443
      - name: admin
        containerPort: 8080
        hostPort: 8080
      volumeMounts:
      - mountPath: "/ssl"
        name: "ssl"
      - mountPath: "/etc/traefik"
        name: "traefik-config"
      securityContext:
        capabilities:
          drop:
          - ALL
          add:
          - NET_BIND_SERVICE
      args:
      - --configfile=/etc/traefik/traefik.toml
      - --api
      - --kubernetes
      - --logLevel=INFO
    tolerations:
    - key: "edgenode"
      operator: "Equal"
      value: "true"
      effect: "NoSchedule"
    nodeSelector:
      edgenode: "true"
---
kind: Service
apiVersion: v1
metadata:
  name: traefik-ingress-service
```

```
  namespace: kube-system
spec:
  selector:
    k8s-app: traefik-ingress-lb
  ports:
    - protocol: TCP
      port: 80
      name: web
    - protocol: TCP
      port: 443
      name: https
    - protocol: TCP
      port: 8080
      targetPort: 8080
      name: admin


[root@K8S-PROD-MASTER-A1 traefik]# kubectl  apply -f traefik-deamonset.yaml
serviceaccount/traefik-ingress-controller created
daemonset.extensions/traefik-ingress-controller created
service/traefik-ingress-service created



[root@K8S-PROD-MASTER-A1 traefik]# kubectl  get pod,svc -n kube-system
NAME                                        READY   STATUS    RESTARTS   AGE
pod/coredns-756d6db49-s6x84                 1/1     Running   2          2d6h
pod/coredns-756d6db49-sf9wj                 1/1     Running   2          2d6h
pod/kubernetes-dashboard-5974995975-8tlgv   1/1     Running   2          2d6h
pod/nginx-test2-65c869854f-h8lxb            1/1     Running   1          2d4h
pod/traefik-ingress-controller-rkvk2        1/1     Running   0          25s

NAME                              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
             AGE
service/coredns                   ClusterIP   10.99.110.110   <none>        53/UDP,53/
TCP,9153/TCP    4d21h
service/kubernetes-dashboard      NodePort    10.99.255.171   <none>        443:32279
/TCP            4d15h
service/nginx-test2               NodePort    10.99.201.93    <none>        80:31001/T
CP              2d4h
service/traefik-ingress-service   ClusterIP   10.99.89.99     <none>        80/TCP,44
3/TCP,8181/TCP   27s
```

```
service/traefik-web-ui           ClusterIP   10.99.229.113   <none>        80/TCP
              2d5h
```

其中 traefik 监听 node 的 80 和 443 端口，80/443 提供 http/https 服务，8080 是其自带的 UI 界面。 由于 traefik1.7.11 已经丢弃了以下参数，生产环境建议对 8080 端口做限制。

```
   --web     (Deprecated) Enable Web backend with default settings (default "false")
   --web.address (Deprecated) Web administration port  (default ":8080")
```

2.12.5 部署 UI

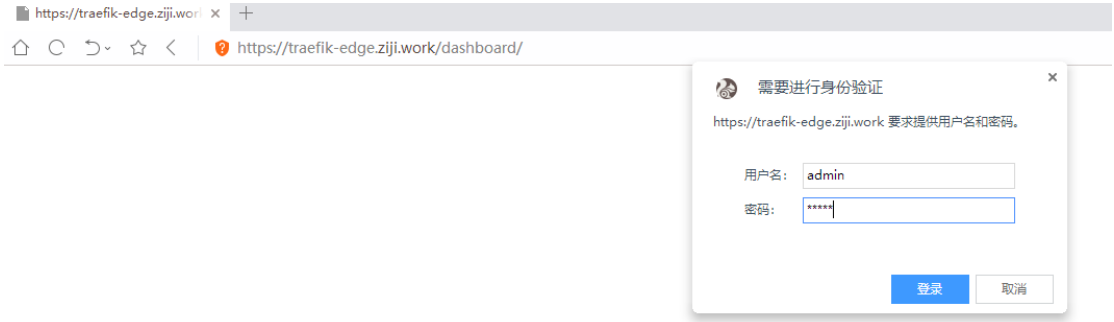
```
#生成 https 证书
[root@K8S-PROD-MASTER-A1 traefik]# openssl req -x509 -nodes -days 365 -newkey rsa:20
48 -keyout traefik-edge.ziji.work.key -out traefik-edge.ziji.work.crt -subj "/CN=tra
efik-edge.ziji.work"

[root@K8S-PROD-MASTER-A1 traefik]# kubectl -n kube-system create secret tls traefik-
cert --key=traefik-edge.ziji.work.key --cert=traefik-edge.ziji.work.crt



#为 traefik 创建 ingress 代理
[root@K8S-PROD-MASTER-A1 traefik]# vi  traefik-web-ui.yaml

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: traefik
    #301 http to https
    ingress.kubernetes.io/ssl-redirect: "true"
    # Forces the frontend to redirect to SSL  but by sending a 302 instead of a 301.
    #ingress.kubernetes.io/ssl-temporary-redirect: "true"
    #优先级
    ingress.kubernetes.io/priority: "100"
    #限制 IP
    ingress.kubernetes.io/whitelist-x-forwarded-for: "true"
    ingress.kubernetes.io/whitelist-source-range: "192.168.1.0/24, 10.211.18.0/24"
    #设置后端 service 权重值
    ingress.kubernetes.io/service-weights: |
```

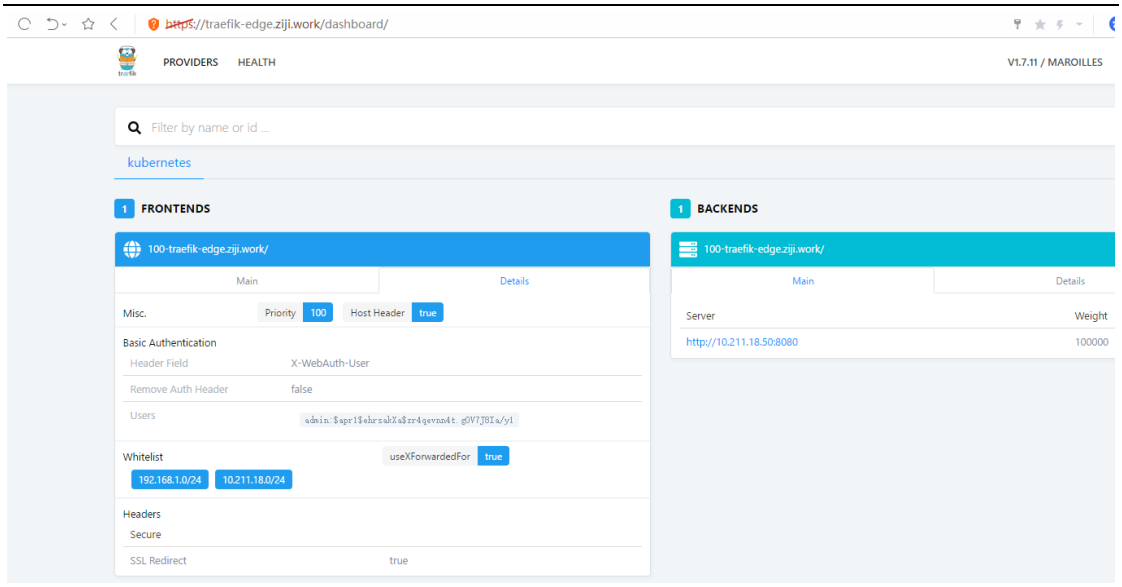


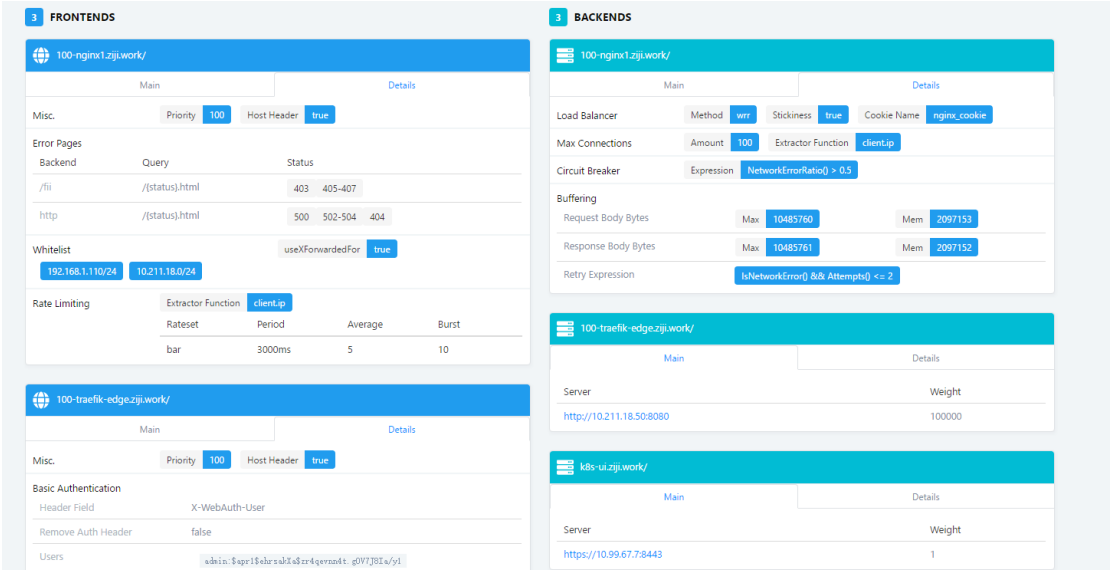
```
      traefik-web-ui: 100%
    #设置认证
    ingress.kubernetes.io/auth-type: "basic"
    ingress.kubernetes.io/auth-secret: "ingress-auth"
  name: traefik-web-ui
  namespace: kube-system
spec:
  rules:
  - host: traefik-edge.ziji.work
    http:
      paths:
      - path: /
        backend:
          serviceName: traefik-ingress-service
          servicePort: 8080
  tls:
   - secretName: traefik-edge.ziji.work

[root@K8S-PROD-MASTER-A1 traefik]# kubectl  apply -f traefik-web-ui.yaml
ingress.extensions/traefik-web-ui created



[root@K8S-PROD-MASTER-A1 traefik]# kubectl  get  ingress -n kube-system
NAME             HOSTS                    ADDRESS   PORTS     AGE
traefik-web-ui   traefik-edge.ziji.work             80, 443   50s
```

将 traefik-edge.ziji.work 域名解析到边缘节点浮动 IP(VIP)， 访问域名测试是否正常

输入认证信息： admin/admin







2.12.6 部署 kubernetes dashboard

步骤 1 生成证书文件

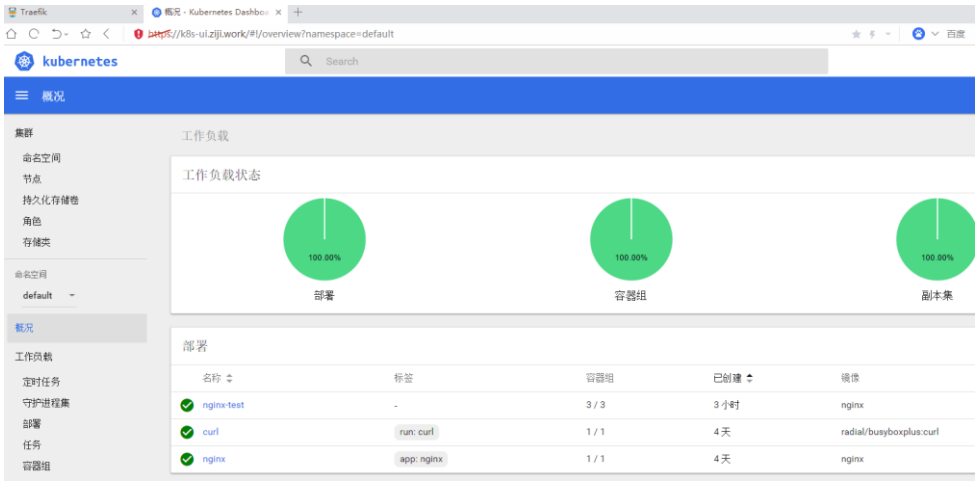
```
[root@K8S-PROD-MASTER-A1 traefik]# openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout k8s-ui.ziji.work.key -out k8s-ui.ziji.work.crt -subj "/CN= k8s-ui.ziji.work"
```

```
[root@K8S-PROD-MASTER-A1 traefik]# kubectl  create  secret  tls  k8s-ui.ziji.work -n
 kube-system --key=k8s-ui.ziji.work.key  --cert=k8s-ui.ziji.work.crt
secret/k8s-ui.ziji.work created
```

步骤 2　应用编排文件

```
[root@K8S-PROD-MASTER-A1 traefik]# vi k8s-ui.ziji.work.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  annotations:
    #ingress.kubernetes.io/ssl-passthrough: "true"
    kubernetes.io/ingress.class: traefik
    #强制跳转到 https
    ingress.kubernetes.io/ssl-redirect: "true"
spec:
  rules:
  - host: k8s-ui.ziji.work
    http:
      paths:
      - path: /
        backend:
          serviceName: kubernetes-dashboard
          servicePort: 443

  tls:
   - secretName: k8s-ui.ziji.work

[root@K8S-PROD-MASTER-A1 traefik]# kubectl  apply -f k8s-ui.ziji.work.yaml
ingress.extensions/kubernetes-dashboard configured
```

解析 k8s-ui.ziji.work 域名到 ingress 节点



2.12.7 部署 Keepalived

- Traefik 以 DaemonSet 的方式启动
- 通过 nodeSelector 选择边缘节点
- 通过 hostPort 暴露端口
- 当前 VIP 漂移到了 10.211.18.100 上，Traefik 根据访问的 host 和 path 配置，将流量转发到相应的 service 上

注意：

在 LB1 和 LB2 主机修改 kube-proxy 代理模式， 将 IPVS 改为 iptables --proxy-mode=iptables

```
[root@K8S-PROD-LB-A1 keepalived]# cat dr-start.sh
#!/bin/bash
#description : start realserver


VIP=10.211.18.100
#/etc/rc.d/init.d/functions

case "$1" in
start)
echo " start LVS of REALServer"
/sbin/ifconfig lo:0 $VIP broadcast $VIP netmask 255.255.255.255 up
route add -host $VIP dev lo:0
echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
```

```
echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
;;

stop)
/sbin/ifconfig lo:0 down
route del -host $VIP dev lo:0
echo "close LVS Directorserver"
echo "0" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "0" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "0" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "0" >/proc/sys/net/ipv4/conf/all/arp_announce
;;
*)
echo "Usage: $0 {start|stop}"
exit 1
esac
```



```
[root@K8S-PROD-LB-A1 ~]#  yum install keepalived ipvsadm -y
```

LB1 主机配置文件`/etc/keepalived/keepalived.conf`文件内容如下：

```
! Configuration File for keepalived

global_defs {
    notification_email {
      root@localhost
    }
    notification_email_from kaadmin@localhost
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id     ingress-edge
}


vrrp_instance ingress-edge-web {
    state MASTER
    interface ens33
```

```
    #lvs_sync_daemon_interface  ens33
    nopreempt
    garp_master_delay  5
    virtual_router_id  100
    priority  100
    advert_int  1
    authentication  {
       auth_type  PASS
       auth_pass  1111
    }

    virtual_ipaddress  {
         10.211.18.100  label  ens33:0
    }
}

virtual_server  10.211.18.100  80  {
          delay_loop  3
          #lb_algo  loadbalance
          lb_algo  wrr
          lb_kind  DR
          persistence_timeout  0
          protocol  TCP

          real_server  10.211.18.50  80  {
              weight  100
              TCP_CHECK  {
                 connect_timeout  3
                 connect_port  80
              }
          }

           real_server  10.211.18.51  80  {
              weight  100
              TCP_CHECK  {
                 connect_timeout  3
                 connect_port  80
              }
      }
}
```



```
virtual_server 10.211.18.100 443 {
          delay_loop 3
          #lb_algo loadbalance
          lb_algo wrr
          lb_kind DR
          persistence_timeout 0
          protocol TCP

          real_server 10.211.18.50 443 {
              weight 100
              TCP_CHECK {
                 connect_timeout 3
                 connect_port 443
              }
          }

          real_server 10.211.18.51 443 {
              weight 100
              TCP_CHECK {
                 connect_timeout 3
                 connect_port 443
              }
      }
}
```

LB2 主机 keepalived 配置内容如下：

```
! Configuration File for keepalived

global_defs {
    notification_email {
       root@localhost
    }
    notification_email_from kaadmin@localhost
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id     ingress-edge
}
```

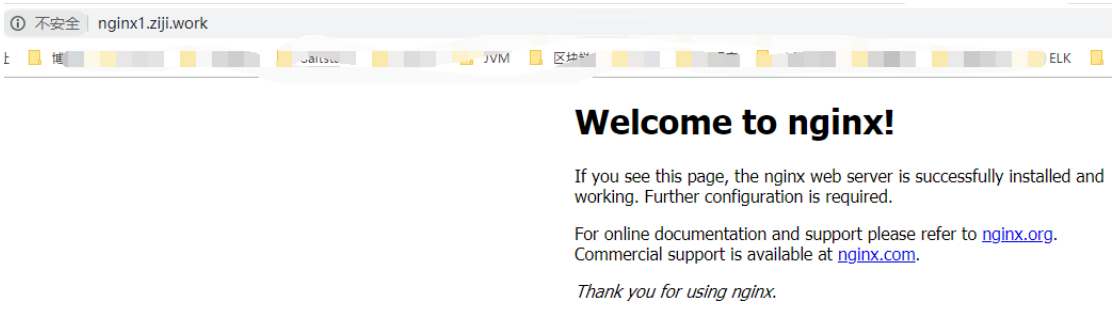


```
vrrp_instance ingress-edge-web {
    state MASTER
    interface ens33
    #lvs_sync_daemon_interface ens33
    #nopreempt
    garp_master_delay 5
    virtual_router_id 100
    priority 99
    advert_int 1
    authentication {
       auth_type PASS
       auth_pass 1111
    }

    virtual_ipaddress {
         10.211.18.100 label ens33:0
    }
}

virtual_server 10.211.18.100 80 {
          delay_loop 3
          #lb_algo loadbalance
          lb_algo wrr
          lb_kind DR
          persistence_timeout 0
          protocol TCP

          real_server 10.211.18.50 80 {
              weight 100
              TCP_CHECK {
                 connect_timeout 3
                 connect_port 80
              }
          }

           real_server 10.211.18.51 80 {
              weight 100
              TCP_CHECK {
                 connect_timeout 3
                 connect_port 80
              }
```

```
    }
}



virtual_server 10.211.18.100 443 {
        delay_loop 3
        #lb_algo loadbalance
        lb_algo wrr
        lb_kind DR
        persistence_timeout 0
        protocol TCP

        real_server 10.211.18.50 443 {
            weight 100
            TCP_CHECK {
              connect_timeout 3
              connect_port 443
            }
        }

        real_server 10.211.18.51 443 {
            weight 100
            TCP_CHECK {
              connect_timeout 3
              connect_port 443
            }
    }
}
```

启动 keepalived

```
systemctl start keepalived
```



2.13 部署 Ceph(数据持久化)

2.13.1 部署 rbd-provisioner

kubernetes 在创建使用 ceph rbd pv/pvc 时没任何问题，但使用 dynamic provisioning 自动管理存储生命周期时会报错。提示`"rbd: create volume failed, err: failed to create rbd image: executable file not found in $PATH:"`

首先得在 kubernetes 集群中安装 rbd-provisioner，github 仓库链接 https://github.com/kubernetes-incubator/external-storage

```
[root@K8S-PROD-MASTER-A1 ceph]# git clone
https://github.com/kubernetes-incubator/external-storage.git
[root@K8S-PROD-MASTER-A1 ceph]# cd external-
storage/ceph/rbd/deploy
[root@K8S-PROD-MASTER-A1 deploy]# NAMESPACE=kube-system
[root@K8S-PROD-MASTER-A1 deploy]# sed -r -i "s/namespace:
[^ ]+/namespace:
$NAMESPACE/g" ./rbac/clusterrolebinding.yaml ./rbac/rolebinding.
yaml
[root@K8S-PROD-MASTER-A1 deploy]# kubectl -n $NAMESPACE apply -
f ./rbac
```

2.13.2 创建 storageclass

部署完 rbd-provisioner，还需要创建 StorageClass。创建 SC 前，我们还需要创建相关用户的 secret；

```
[root@K8S-PROD-MASTER-A1 ceph]# vi ceph-secrets.yaml
apiVersion: v1
kind: Secret
metadata:
  name: ceph-admin-secret
  namespace: kube-system
```

```
type: "kubernetes.io/rbd"
data:
  # ceph auth get-key client.admin | base64
  key: QVFEdXc5eGNSWEVhQlJBQXhCTi9Ya1JIc2syb0VLYURqNmJXdUE9PQ==

---
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
  namespace: kube-system
type: "kubernetes.io/rbd"
data:
  # ceph auth add client.kube mon 'allow r' osd 'allow rwx
pool=kube'
  # ceph auth get-key client.kube | base64
  key: QVFEdXc5eGNSWEVhQlJBQXhCTi9Ya1JIc2syb0VLYURqNmJXdUE9PQ==

---
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
type: "kubernetes.io/rbd"
data:
  # ceph auth add client.kube mon 'allow r' osd 'allow rwx
pool=kube'
```

```
  # ceph auth get-key client.kube | base64
```

```
  key: QVFEdXc5eGNSWEVhQlJBQXhCTi9Ya1JIc2syb0VLYURqNmJXdUE9PQ==


[root@K8S-PROD-MASTER-A1 ceph]#  kubectl create -f ceph-secrets.yaml
```

```
[root@K8S-PROD-MASTER-A1 ceph]# vi storageclass.yaml

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: dynamic-cephfs
   annotations:
     storageclass.beta.kubernetes.io/is-default-class: "true"
provisioner: ceph.com/rbd
parameters:
    monitors: 10.211.18.5:6789,10.211.18.8:6789,10.211.18.9:6789
    adminId: admin
    adminSecretName: ceph-admin-secret
    adminSecretNamespace: kube-system
    pool: rbd
    userId: admin
    userSecretName: ceph-admin-secret
    fsType: xfs
    imageFormat: "2"
    imageFeatures: "layering"

[root@K8S-PROD-MASTER-A1 ceph]#  kubectl create -f  ceph-rbd-sc.yaml
```

- 其他设置和普通的 ceph rbd StorageClass 一致，但 provisioner 需要设置为 `ceph.com/rbd`，不是默认的 `kubernetes.io/rbd`，这样 rbd 的请求将由 rbd-provisioner 来处理；
- 考虑到兼容性，建议尽量关闭 rbd image feature，并且 kubelet 节点的 ceph-common 版本尽量和 ceph 服务器端保持一致

2.13.3 测试 ceph rbd 自动分配

```
[root@K8S-PROD-MASTER-A1 ceph]# vi test-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1
spec:
  containers:
  - name: ceph-busybox
    image: busybox
    command: ["sleep", "60000"]
    volumeMounts:
    - name: ceph-vol1
      mountPath: /mnt
      readOnly: false
  volumes:
  - name: ceph-vol1
    persistentVolumeClaim:
      claimName: ceph-claim
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim

spec:

  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

检查 pv 和 pvc 的创建状态，是否创建成功

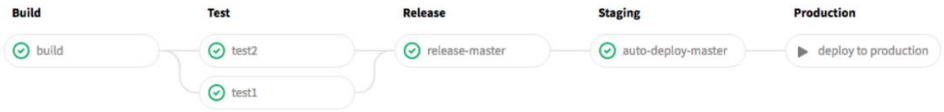
[root@K8S-PROD-MASTER-A1 ceph]# kubectl get pv,pvc

```
[root@K8S-PROD-MASTER-A1 ceph]# kubectl  get pv,pvc
NAME                                                        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                 STORAGECLASS
persistentvolume/pvc-f0a9a07d-77ba-11e9-bb24-fa163efceeb2   2Gi        RWO            Delete           Bound    default/ceph-claim    dynamic-cephfs

NAME                                  STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS     AGE
persistentvolumeclaim/ceph-claim      Bound    pvc-f0a9a07d-77ba-11e9-bb24-fa163efceeb2   2Gi        RWO            dynamic-cephfs   6d22h
[root@K8S-PROD-MASTER-A1 ceph]#
```



```
[root@K8S-PROD-MASTER-A1 ceph]# kubectl  describe  pvc ceph-claim
Name:          ceph-claim
Namespace:     default
StorageClass:  dynamic-cephfs
Status:        Bound
Volume:        pvc-f0a9a07d-77ba-11e9-bb24-fa163efceeb2
Labels:        <none>
Annotations:   kubectl.kubernetes.io/last-applied-configuration:
                 {"apiVersion":"v1","kind":"PersistentVolumeClaim","metadata":{"
               pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner: ceph.com/rbd
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      2Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Events:        <none>
Mounted By:    <none>
```

3 CI/CD 自动化构建



4 容器日志采集

5 弹性伸缩

5.1.1 部署 metrics-server 0.3.1

kubernetes 从 1.8 版本开始，CPU、内存等资源的 metrics 信息可以通过 Metrics API 来获取，用户可以直接获取这些 metrics 信息（例如通过执行 kubect top 命令），HPA（HorizontalPodAutoscaling）使用这些 metics 信息来实现动态伸缩。 Heapster 在 Kubernetes1.11 后已经被废弃了，后续版本中会使用 metrics-server 代替 Heapster

步骤 1 生成证书文件

```
[root@K8S-PROD-MASTER-A1 ~]# mkdir metrics-server/ && cd metrics-server
[root@K8S-PROD-MASTER-A1 metrics-server]# mkdir ssl/ && cd ssl/

# 证书文件主要用在 Metrics API aggregator 上
[root@K8S-PROD-MASTER-A1 ssl]#  cat >> metrics-proxy-ca-csr.json << EOF
{
    "CN": "kubernetes",
    "key": {
        "algo": "rsa",
        "size": 2048
    }
}
EOF

[root@K8S-PROD-MASTER-A1 ssl]# cat >> metrics-proxy-client-csr.json << EOF
{
    "CN": "metrics-proxy-client",
    "key": {
        "algo": "rsa",
        "size": 2048
    }
}
EOF
```

```
[root@K8S-PROD-MASTER-A1 ssl]# cfssl gencert   -initca metrics-proxy-ca-csr.json | c
fssljson -bare metrics-proxy-ca

[root@K8S-PROD-MASTER-A1 k8s-ssl]# cfssl gencert   -ca=/data/apps/kubernetes/pki/kub
e-proxy.pem  \
-ca-key=/data/apps/kubernetes/pki/kube-proxy-key.pem \
-config=/root/k8s-ssl/ca-config.json   \
-profile=kubernetes   metrics-proxy-client-csr.json | cfssljson -bare metrics-proxy-
client


[root@K8S-PROD-MASTER-A1 ssl]# ll *.pem
-rw------- 1 root root 1679 May  7 10:38 metrics-proxy-ca-key.pem
-rw-r--r-- 1 root root 1143 May  7 10:38 metrics-proxy-ca.pem
-rw------- 1 root root 1675 May  7 10:40 metrics-proxy-client-key.pem
-rw-r--r-- 1 root root 1326 May  7 10:40 metrics-proxy-client.pem
```

步骤 2 拷贝证书文件到所有节点

```
[root@K8S-PROD-MASTER-A1 ssl]# scp -r *.pem  10.211.18.11:/data/apps/kubernetes/pki
root@10.211.18.11's password:
metrics-proxy-ca-key.pem
                                                      100% 1679   825.4KB/s
00:00
metrics-proxy-ca.pem
                                                     100% 1143   696.9KB/s   0
0:00
metrics-proxy-client-key.pem
                                                      100% 1675     1.0MB/s
00:00
metrics-proxy-client.pem
                                                      100% 1326     1.0MB/s
00:00
```

步骤 3 修改 kube-apiserver 集群配置文件

```
[root@K8S-PROD-MASTER-A1 ssl]# vim /data/apps/kubernetes/etc/apiserver
--requestheader-client-ca-file=/data/apps/kubernetes/pki/metrics-proxy-ca.pem \
--requestheader-allowed-names=aggregator \
--requestheader-extra-headers-prefix=X-Remote-Extra- \
--requestheader-group-headers=X-Remote-Group \
--requestheader-username-headers=X-Remote-User \
```

```
--proxy-client-cert-file=/data/apps/kubernetes/pki/metrics-proxy-client.pem \
--proxy-client-key-file=/data/apps/kubernetes/pki/metrics-proxy-client-key.pem


[root@K8S-PROD-MASTER-A1 ssl]# systemctl  restart kube-apiserver
```

步骤 4 通过 yaml 文件创建对应的资源

```
[root@K8S-PROD-MASTER-A1 ssl]# cd ../
[root@K8S-PROD-MASTER-A1 metrics-server]#  for file in auth-delegator.yaml auth-read
er.yaml metrics-apiservice.yaml metrics-server-deployment.yaml metrics-server-servic
e.yaml resource-reader.yaml; do wget https://raw.githubusercontent.com/kubernetes/ku
bernetes/release-1.13/cluster/addons/metrics-server/$file;done
```

```
[root@K8S-PROD-MASTER-A1 metrics-server]# ls -l
total 24
-rw-r--r-- 1 root root  398 May  6 16:09 auth-delegator.yaml
-rw-r--r-- 1 root root  419 May  6 16:09 auth-reader.yaml
-rw-r--r-- 1 root root  393 May  6 16:09 metrics-apiservice.yaml
-rw-r--r-- 1 root root 3663 May  6 18:47 metrics-server-deployment.yaml
-rw-r--r-- 1 root root  336 May  6 16:09 metrics-server-service.yaml
-rw-r--r-- 1 root root  817 May  6 16:11 resource-reader.yaml
```

步骤 5 修改 resource-reader.yaml 增加 nodes/stats 字段

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - nodes
  - nodes/stats
  - namespaces
  verbs:
  - get
  - list
  - watch
```

步骤 6 将 metrics-server-deployment.yaml 修改为以下内容

```
[root@K8S-PROD-MASTER-A1 metrics-server]# vim metrics-server-deployment.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metrics-server
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: metrics-server-config
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: EnsureExists
data:
  NannyConfiguration: |-
    apiVersion: nannyconfig/v1alpha1
    kind: NannyConfiguration
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: metrics-server-v0.3.1
  namespace: kube-system
  labels:
    k8s-app: metrics-server
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    version: v0.3.1
spec:
  selector:
    matchLabels:
      k8s-app: metrics-server
      version: v0.3.1
  template:
    metadata:
```

```
      name: metrics-server
      labels:
        k8s-app: metrics-server
        version: v0.3.1
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
        seccomp.security.alpha.kubernetes.io/pod: 'docker/default'
    spec:
      priorityClassName: system-cluster-critical
      serviceAccountName: metrics-server
      containers:
      - name: metrics-server
        image: gcr.azk8s.cn/google_containers/metrics-server-amd64:v0.3.1
        imagePullPolicy: IfNotPresent
        volumeMounts:
        - mountPath: /etc/kubernetes/ssl/
          name: ca-ssl
        command:
         # tail -F /var/log/*
          - /metrics-server
          - --metric-resolution=30s
          # These are needed for GKE, which doesn't support secure communication yet.
          # Remove these lines for non-GKE clusters, and when GKE supports token-base
d auth.
          #- --kubelet-port=10255
          - --kubelet-insecure-tls
          #- --deprecated-kubelet-completely-insecure=true
          - --kubelet-preferred-address-types=InternalIP,Hostname,InternalDNS,Extern
alDNS,ExternalIP
          - --requestheader-client-ca-file=/etc/kubernetes/ssl/metrics-proxy-client.
pem
        ports:
        - containerPort: 443
          name: https
          protocol: TCP
      volumes:
       - name: ca-ssl
         hostPath:
           path: /data/apps/kubernetes/pki
```

应用部署 kubectl apply -f .

```
[root@K8S-PROD-MASTER-A1 metrics-server]# kubectl  apply -f .
```



```
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator cr
eated
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
serviceaccount/metrics-server created
configmap/metrics-server-config created
deployment.extensions/metrics-server-v0.3.1 created
service/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
```

查看 Pod 运行是否正常

```
[root@K8S-PROD-MASTER-A1 metrics-server]# kubectl  get pod -n kube-system
NAME                                  READY   STATUS    RESTARTS   AGE
coredns-756d6db49-s6x84                 1/1     Running   5          4d2h
coredns-756d6db49-sf9wj                 1/1     Running   5          4d2h
kubernetes-dashboard-5974995975-qx8mk    1/1     Running   0          15h
metrics-server-v0.3.1-689bf6598f-nfcvd   1/1     Running   0          68s
traefik-ingress-controller-242z5         1/1     Running   1          42h
```

5.1.2 测试 创建 HPA

```
[root@K8S-PROD-MASTER-A1 metrics-server]# kubectl  top node
NAME          CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
10.211.18.11   62m          6%     455Mi           37%
10.211.18.4    139m         13%    964Mi           85%
10.211.18.5    156m         15%    1055Mi          86%
10.211.18.50   36m          3%     388Mi           40%
10.211.18.6    189m         18%    1038Mi          85%



[root@K8S-PROD-MASTER-A1 metrics-server]# kubectl get --raw "/apis/metrics.k8s.io/v1
beta1/nodes" | jq
```

测试 HPA 是否正常工作

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
```

```
  name: nginx-test-hpa
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: nginx-test
  minReplicas: 1
  maxReplicas: 3
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 10
  - type: Resource
    resource:
      name: memory
      targetAverageValue: 10Mi

[root@K8S-PROD-MASTER-A1 metrics-server]# kubectl  apply -f hpa-nginx.yaml
horizontalpodautoscaler.autoscaling/nginx-hpa created
```



```
[root@K8S-PROD-MASTER-A1 metrics-server]# kubectl  get hpa
```

```
[root@K8S-PROD-MASTER-A1 metrics]# kubectl  get hpa
NAME             REFERENCE                TARGETS                MINPODS   MAXPODS   REPLICAS   AGE
nginx-test-hpa   Deployment/nginx-test1   2224128/10Mi, 0%/10%   1         3         1          61m
nginx-test2      Deployment/nginx-test2   0%/50%                 2         5         2          53m
[root@K8S-PROD-MASTER-A1 metrics]#
[root@K8S-PROD-MASTER-A1 metrics]#
```

使用 ab 对 nginx-test pod 进行压力测试，观察 HPA 是否会对 pod 进行伸缩

```
[root@K8S-PROD-NODE-A2 yum.repos.d]# ab -c100 -n 10000 http://10.99.84.211/index.html
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 10.99.84.211 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        nginx/1.15.12
Server Hostname:        10.99.84.211
Server Port:            80

Document Path:          /index.html
Document Length:        612 bytes
```

```
[root@K8S-PROD-MASTER-A1 metrics]# kubectl  get hpa
```

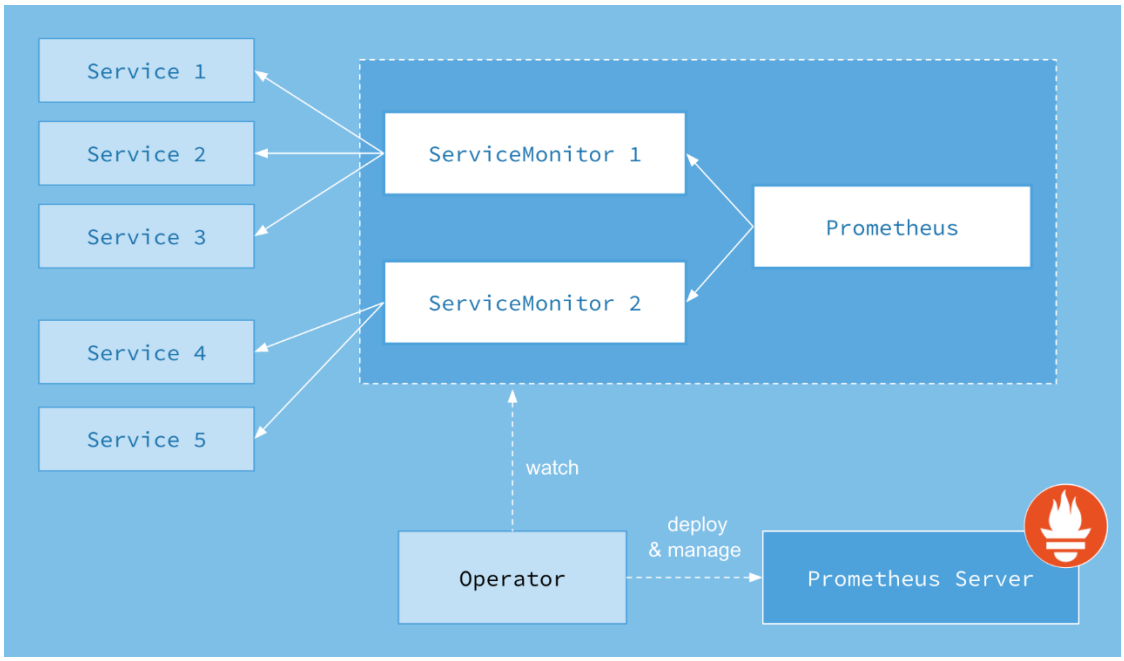
通过 kubectl describe hpa nginx-test-hpa,可以看到 Pod 已经自动完成缩放

```
[root@K8S-PROD-MASTER-A1 metrics]# kubectl  get hpa
NAME             REFERENCE                TARGETS                      MINPODS   MAXPODS   REPLICAS   AGE
nginx-test-hpa   Deployment/nginx-test1   2456234666m/10Mi, 0%/10%   1         3         3          69m
nginx-test2      Deployment/nginx-test2   0%/50%                       2         5         2          61m
[root@K8S-PROD-MASTER-A1 metrics]#
```

6 监控告警

6.1.1 什么是 Prometheus?

Prometheus 是由 SoundCloud 开发的开源监控报警系统和时序列数据库(TSDB)。Prometheus 使用 Go 语言开发，是 Google BorgMon 监控系统的开源版本。Prometheus 和 Heapster(Heapster 是 K8S 的一个子项目，用于获取集群的性能数据。)相比功能更完善、更全面。Prometheus 性能也足够支撑上万台规模的集群。



6.1.2 安装 Prometheus

- 下载二进制包

```
wget
https://github.com/prometheus/prometheus/releases/download/v2.6.1/prometheus-2.6.1.linux-amd64.tar.gz
tar zxfv prometheus-2.6.1.linux-amd64.tar.gz
mv prometheus-2.6.1.linux-amd64  /data/apps/prometheus
```

- 配置系统服务

```
vi /usr/lib/systemd/system/prometheus.service

[Unit]
Description=prometheus
After=network.target

[Service]
Type=simple
User=prometheus
ExecStart=/data/apps/prometheus/prometheus --
config.file=/data/apps/prometheus/prometheus.yml \
--storage.tsdb.path=/data/apps/prometheus/data \
--web.console.libraries=/data/apps/prometheus/console_libraries \
--web.console.templates=/data/apps/prometheus/consoles \
--web.enable-lifecycle

Restart=on-failure

[Install]

WantedBy=multi-user.target
```

- 创建用户

```
groupadd prometheus
```

```
useradd -g prometheus -m -d /data/apps/prometheus/data -s /sbin/nologin
prometheus
```

- 验证安装

```
[root@ceph1-10-211-18-5 prometheus]# systemctl  start prometheus
[root@ceph1-10-211-18-5 prometheus]# systemctl  enable  prometheus

[root@ceph1-10-211-18-5 prometheus]# /data/apps/prometheus/prometheus --
version
prometheus, version 2.6.1 (branch: HEAD, revision:
b639fe140c1f71b2cbad3fc322b17efe60839e7e)
  build user:       root@4c0e286fe2b3
  build date:       20190115-19:12:04
  go version:       go1.11.4


[root@ceph1-10-211-18-5 prometheus]# curl localhost:9090
<a href="/graph">Found</a>.
```

- 配置 prometheus 修改 `prometheus.yml`

```
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds.
Default is every 1 minute.

  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default
is every 1 minute.

  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
```

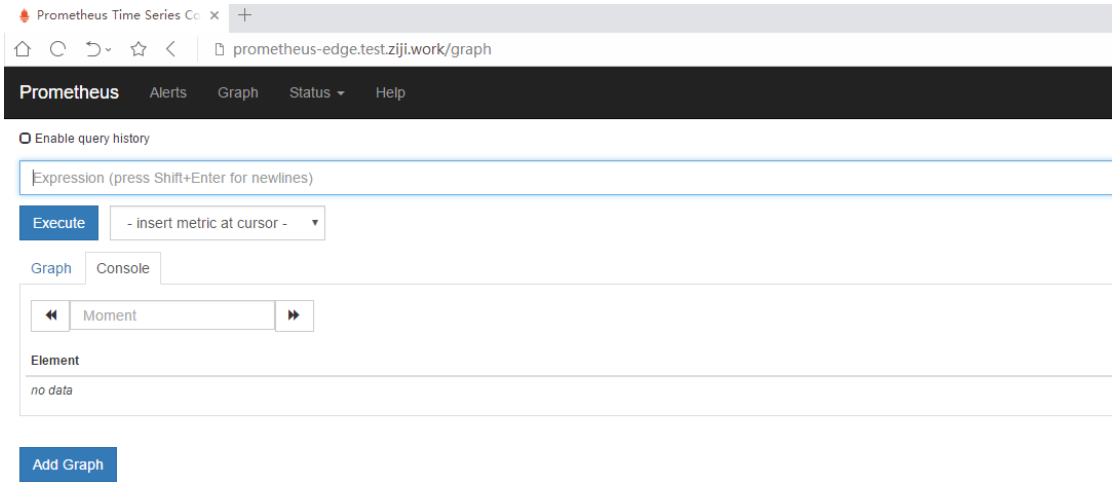


```
    - targets:
       - 127.0.0.1:9093

# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
rule_files:
  - "rules/*.rules"
  #- "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries
scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['alertmanager-edge.test.ziji.work:9090']


  - job_name: 'kubernetes-nodes'
    static_configs:
      - targets:
        - 10.211.18.7:9100
        - 10.211.18.3:9100
        - 10.211.18.14:9100
        - 10.211.18.17:9100
        - 10.211.18.6:9100

        - 10.211.18.10:9100


  - job_name: 'ceph'
    static_configs:
      - targets: ['127.0.0.1:9128']
        labels:
          instance: 10.211.18.5

  - job_name: 'kube-state-metrics'
```

```
  static_configs:
    - targets: ['kube-edge.test.ziji.work']
      labels:
        instance: kube-state-metrics
        k8scluster: test-k8s-cluster

- job_name: kubernetes-nodes-kubelet
  static_configs:
    - targets:
      - 10.211.18.7:10250
      - 10.211.18.3:10250
      - 10.211.18.14:10250
      - 10.211.18.17:10250
      - 10.211.18.6:10250
      labels:
        k8scluster: test-k8s-cluster
  kubernetes_sd_configs:
  - role: node
  relabel_configs:
  - action: labelmap
    regex: __meta_kubernetes_node_label_(.+)
  scheme: https
  tls_config:
    ca_file: /data/apps/prometheus/rbac/ca.crt
    insecure_skip_verify: true
  bearer_token_file: /data/apps/prometheus/rbac/token

- job_name: kubernetes-nodes-cadvisor
  static_configs:
    - targets:

      - 10.211.18.7:10250

      - 10.211.18.3:10250
      - 10.211.18.14:10250
      - 10.211.18.17:10250
      - 10.211.18.6:10250
      labels:
        k8scluster: test-k8s-cluster
  kubernetes_sd_configs:
  - role: node
  relabel_configs:
```

```
  - action: labelmap
    regex: __meta_kubernetes_node_label_(.+)
  - target_label: __metrics_path__
    replacement: /metrics/cadvisor
  scheme: https
  tls_config:
    ca_file: /data/apps/prometheus/rbac/ca.crt
    insecure_skip_verify: true
  bearer_token_file: /data/apps/prometheus/rbac/token
```

- 访问自带 Web
- http://ip:9090



6.1.3 安装 node_exporter

```
wget
https://github.com/prometheus/node_exporter/releases/download/v0.17.0/nod
e_exporter-0.17.0.linux-amd64.tar.gz

tar zxf node_exporter-0.17.0.linux-amd64.tar.gz
mv node_exporter-0.17.0.linux-amd64 /data/apps/node_exporter
```

- 创建 Systemd 服务

```
[Unit]
Description=https://prometheus.io

[Service]
Restart=on-failure
ExecStart=/data/apps/node_exporter/node_exporter --collector.systemd --
collector.systemd.unit-whitelist=(docker|kubelet|kube-
proxy|flanneld).service

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

- 启动 Node exporter

```
# systemctl  enable node_exporter
# systemctl  start node_exporter
```

Node Exporter 默认的抓取地址为 http://IP:9100/metrics

6.1.4 安装 grafana

- 下载 rpm 包

```
wget https://dl.grafana.com/oss/release/grafana-6.2.0-1.x86_64.rpm
yum localinstall grafana-6.2.0-1.x86_64.rpm
```

- 启动 grafana 服务

```
systemctl  enable grafana-server
systemctl start grafana-server
```

6.1.5 部署 kube-state-metrics

```
[root@K8S-PROD-MASTER-A1 ~]# mkdir prometheus
[root@K8S-PROD-MASTER-A1 ~]# cd prometheus

创建 yaml 文件
[root@K8S-PROD-MASTER-A1 ~]# vi  kube-state-metrics-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-state-metrics
  namespace: kube-system
  labels:
    k8s-app: kube-state-metrics
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    version: v1.3.0
spec:
  selector:
    matchLabels:
      k8s-app: kube-state-metrics

      version: v1.3.0

  replicas: 1
  template:
    metadata:
      labels:
        k8s-app: kube-state-metrics
        version: v1.3.0
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      priorityClassName: system-cluster-critical
      serviceAccountName: kube-state-metrics
      containers:
      - name: kube-state-metrics
        image: lizhenliang/kube-state-metrics:v1.3.0
        ports:
        - name: http-metrics
```

```
        containerPort: 8080
      - name: telemetry
        containerPort: 8081
      readinessProbe:
        httpGet:
          path: /healthz
          port: 8080
        initialDelaySeconds: 5
        timeoutSeconds: 5
    - name: addon-resizer
      image: lizhenliang/addon-resizer:1.8.3
      resources:
        limits:
          cpu: 100m
          memory: 30Mi
        requests:
          cpu: 100m
          memory: 30Mi
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
      command:
        - /pod_nanny
        - --config-dir=/etc/config
        - --container=kube-state-metrics
        - --cpu=100m
        - --extra-cpu=1m
        - --memory=100Mi
        - --extra-memory=2Mi
        - --threshold=5
        - --deployment=kube-state-metrics
```

```
      volumes:
        - name: config-volume
          configMap:
            name: kube-state-metrics-config
---
# Config map for resource configuration.
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-state-metrics-config
  namespace: kube-system
  labels:
    k8s-app: kube-state-metrics
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
data:
  NannyConfiguration: |-
    apiVersion: nannyconfig/v1alpha1
    kind: NannyConfiguration
```

创建 RBAC 文件

```
[root@K8S-PROD-MASTER-A1 prometheus]# vi kube-state-metrics-rbac.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-state-metrics
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kube-state-metrics
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
```

```
rules:
- apiGroups: [""]
  resources:
  - configmaps
  - secrets
  - nodes
  - pods
  - services
  - resourcequotas
  - replicationcontrollers
  - limitranges
  - persistentvolumeclaims
  - persistentvolumes
  - namespaces
  - endpoints
  verbs: ["list", "watch"]
- apiGroups: ["extensions"]
  resources:
  - daemonsets
  - deployments
  - replicasets
  verbs: ["list", "watch"]
- apiGroups: ["apps"]
  resources:
  - statefulsets
  verbs: ["list", "watch"]
- apiGroups: ["batch"]
  resources:
  - cronjobs
  - jobs
  verbs: ["list", "watch"]
- apiGroups: ["autoscaling"]
  resources:
  - horizontalpodautoscalers
  verbs: ["list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kube-state-metrics-resizer
```

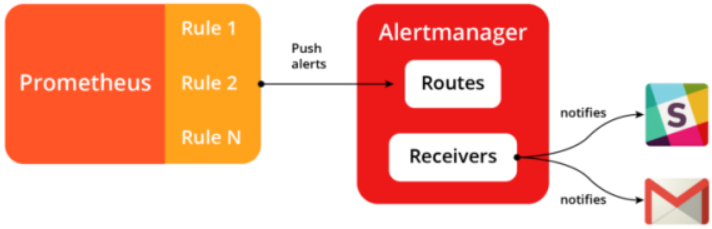
```
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
rules:
- apiGroups: [""]
  resources:
  - pods
  verbs: ["get"]
- apiGroups: ["extensions"]
  resources:
  - deployments
  resourceNames: ["kube-state-metrics"]
  verbs: ["get", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kube-state-metrics

  labels:

    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kube-state-metrics
subjects:
- kind: ServiceAccount
  name: kube-state-metrics
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kube-state-metrics
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
roleRef:
```

```
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kube-state-metrics-resizer
subjects:
- kind: ServiceAccount
  name: kube-state-metrics
  namespace: kube-system
```

创建 kube-state-metrics-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: kube-state-metrics
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "kube-state-metrics"
  annotations:
    prometheus.io/scrape: 'true'
spec:
  ports:
  - name: http-metrics
    port: 8080
    targetPort: http-metrics
    protocol: TCP
  - name: telemetry
    port: 8081
    targetPort: telemetry
    protocol: TCP
  selector:
    k8s-app: kube-state-metrics
```

6.1.6 安装 alertmanager



- 下载二进制包

```
Wget
https://github.com/prometheus/alertmanager/releases/download/v0.16.2/alertmanager-0.16.2.linux-amd64.tar.gz

tar zxvf alertmanager-0.16.2.linux-amd64.tar.gz
mv alertmanager-0.16.2.linux-amd64 /data/apps/alertmanager
```

- 创建系统服务

```
vi /usr/lib/systemd/system/alertmanager.service
[Unit]
Description=Alertmanager
After=network.target

[Service]
Type=simple
User=alertmanager
ExecStart=/data/apps/alertmanager/alertmanager --
config.file=/data/apps/alertmanager/alertmanager.yml --
storage.path=/data/apps/alertmanager/data
```



```
Restart=on-failure

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

- 启动服务

```
# systemctl  enable alertmanager
# systemctl  start alertmanager
```

6.1.7 配置 alertmanager 告警通知

```
global:
  resolve_timeout: 5m
  smtp_smarthost: 'smtp.139.com:25'
  smtp_from: 'zijiwork@139.com'
  smtp_auth_username: 'username'
  smtp_auth_password: 'password'
  smtp_require_tls: false

route:
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 1h
  receiver: 'mail'
receivers:
- name: 'mail'
  #webhook_configs:
  email_configs:
  #- url: 'http://127.0.0.1:5001/'
  - to: 'admin@ziji.work'
    send_resolved: true
```

6.1.8 配置 rules 规则

```
mkdir rules  && cd rules


vi general.rules
groups:
- name: general.rules
  rules:

  # Alert for any instance that is unreachable for >5 minutes.
  - alert: InstanceDown
    expr: up == 0

    for: 1m

    labels:
      severity: error
    annotations:
      summary: "Instance {{ $labels.instance }} down"
      description: "{{ $labels.instance }} of job {{ $labels.job }} has
been down for more than 1 minutes."




vi kubernetes.rules
groups:
- name: general.rules
  rules:

  # Alert for any instance that is unreachable for >5 minutes.
  - alert: InstanceDown
    expr: up == 0
    for: 1m
    labels:
      severity: error
    annotations:
      summary: "Instance {{ $labels.instance }} down"
```

```
      description: "{{ $labels.instance }} of job {{ $labels.job }} has
been down for more than 1 minutes."
[root@ceph1-10-211-18-5 rules]# cat kubernetes.rules
groups:
- name: kubernetes
  rules:
  - alert: PodDown
    expr: kube_pod_status_phase{phase="Unknown"} == 1 or
kube_pod_status_phase{phase="Failed"} == 1
    for: 1m
    labels:
      severity: error
      service: prometheus_bot
      receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"

    annotations:

      summary: Pod Down
      k8scluster: "{{ $labels.k8scluster}}"
      namespace: "{{ $labels.namespace }}"
      pod: "{{ $labels.pod }}"
      container: "{{ $labels.container }}"

  - alert: PodRestart
    expr: changes(kube_pod_container_status_restarts_total{pod !~
"analyzer.*"}[10m]) > 0
    for: 1m
    labels:
      severity: error
      service: prometheus_bot
      receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
    annotations:
      summary: Pod Restart
      k8scluster: "{{ $labels.k8scluster}}"
      namespace: "{{ $labels.namespace }}"
      pod: "{{ $labels.pod }}"
      container: "{{ $labels.container }}"

  - alert: NodeUnschedulable
    expr: kube_node_spec_unschedulable == 1
    for: 5m
    labels:
```

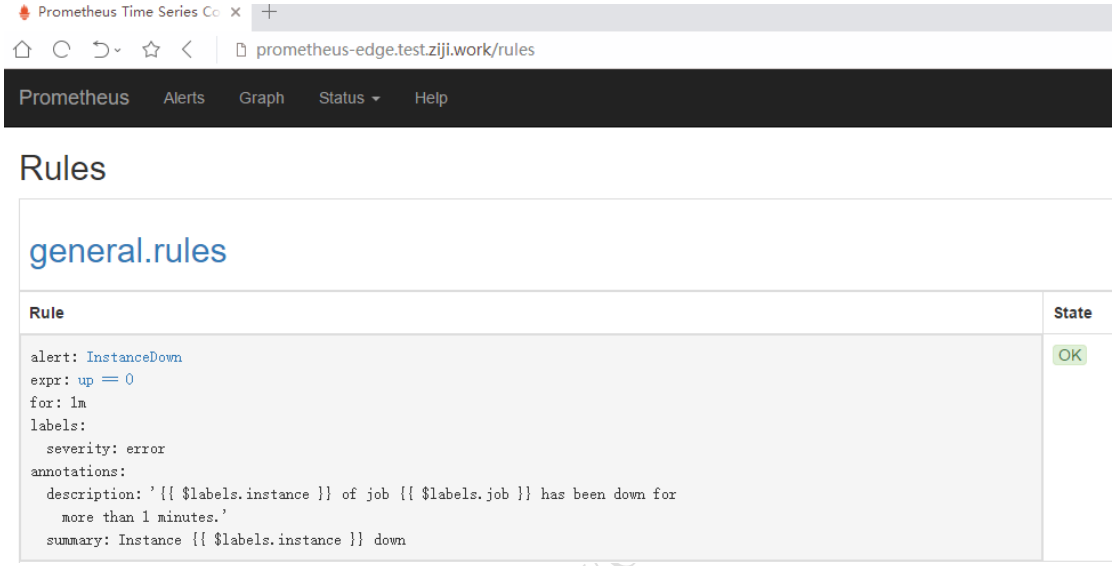
```
      severity: error
      service: prometheus_bot
      receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
    annotations:
      summary: Node Unschedulable
      k8scluster: "{{ $labels.k8scluster}}"
      node: "{{ $labels.node }}"

  - alert: NodeStatusError
    expr: kube_node_status_condition{condition="Ready", status!="true"}
== 1
    for: 5m
    labels:
```

```
      severity: error
```

```
      service: prometheus_bot
      receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
    annotations:
      summary: Node Status Error
      k8scluster: "{{ $labels.k8scluster}}"
      node: "{{ $labels.node }}"

  - alert: DaemonsetUnavailable
    expr: kube_daemonset_status_number_unavailable > 0
    for: 5m
    labels:
      severity: error
      service: prometheus_bot
      receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
    annotations:
      summary: Daemonset Unavailable
      k8scluster: "{{ $labels.k8scluster}}"
      namespace: "{{ $labels.namespace }}"
      daemonset: "{{ $labels.daemonset }}"

  - alert: JobFailed
    expr: kube_job_status_failed == 1
    for: 5m
    labels:
      severity: error
      service: prometheus_bot
```

```
      receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
    annotations:
      summary: Job Failed
      k8scluster: "{{ $labels.k8scluster}}"
      namespace: "{{ $labels.namespace }}"
      job: "{{ $labels.exported_job }}"



vi node.rules
groups:
- name: node.rules

  rules:

  - alert: NodeFilesystemUsage
    expr: 100 - (node_filesystem_free_bytes{fstype=~"ext4|xfs"} /
node_filesystem_size_bytes{fstype=~"ext4|xfs"} * 100) > 80
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: "{{$labels.instance}}: {{$labels.mountpoint }} 分区使用过高"
      description: "{{$labels.instance}}: {{$labels.mountpoint }} 分区使用
大于 80% (当前值: {{ $value }})"
  - alert: NodeMemoryUsage
    expr: 100 -
(node_memory_MemFree_bytes+node_memory_Cached_bytes+node_memory_Buffers_b
ytes) / node_memory_MemTotal_bytes * 100 > 80
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: "{{$labels.instance}}: 内存使用过高"
      description: "{{$labels.instance}}: 内存使用大于 80% (当前值:
{{ $value }})"
  - alert: NodeCPUUsage
    expr: 100 - (avg(irate(node_cpu_seconds_total{mode="idle"}[5m])) by
(instance) * 100) > 80
    for: 2m
    labels:
      severity: warning
```

```
    annotations:
      summary: "{{$labels.instance}}: CPU 使用过高"
      description: "{{$labels.instance}}: CPU 使用大于 80% (当前值:
{{ $value }})"
```

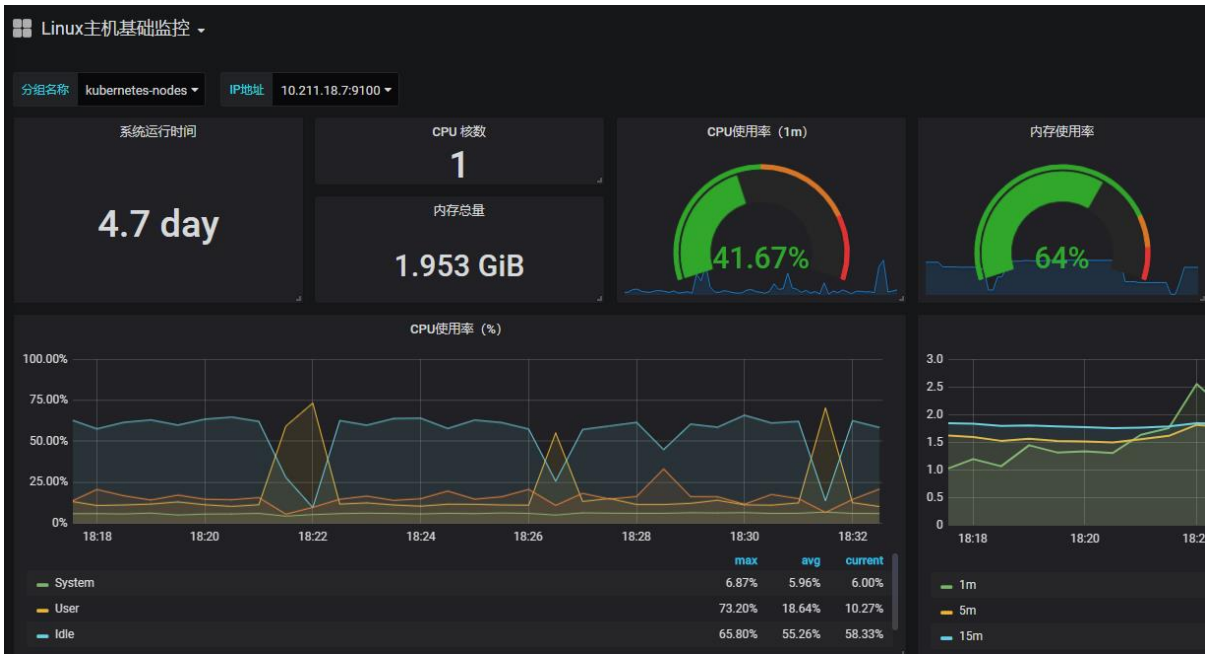


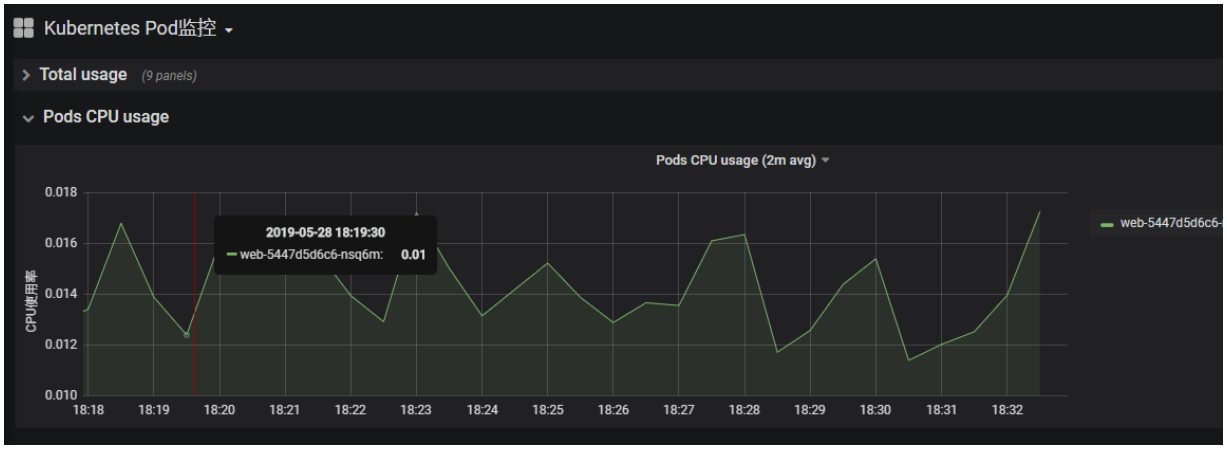
打开 grafana 页面，导入以下模板

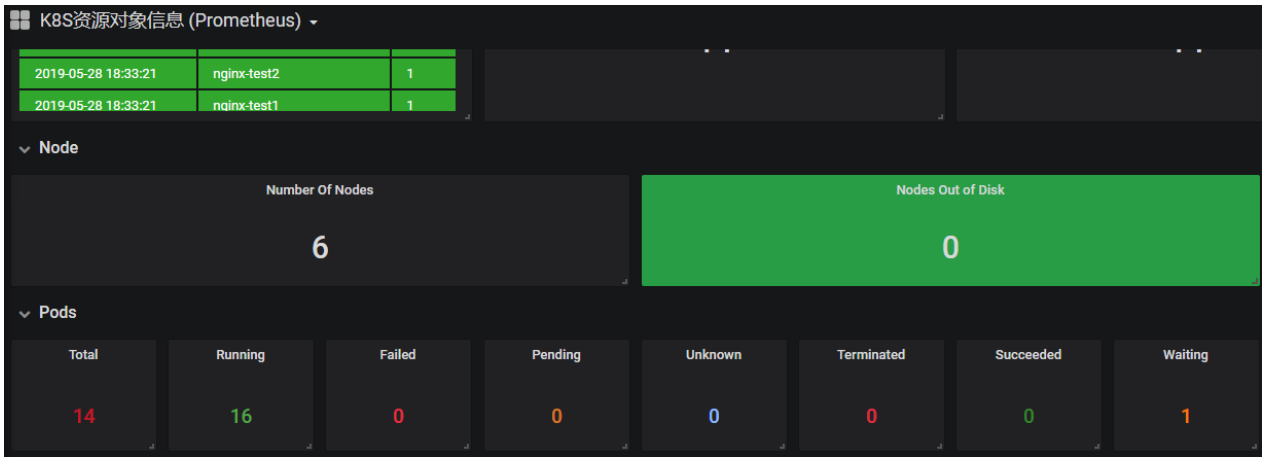
9276　Linux 主机基础监控

3119　Kubernetes Pod 监控

6417　K8S 资源对象信息


Linux主机基础监控
分组名称
kubernetes-nodes
IP地址
10.211.18.7:9100
系统运行时间
4.7 day
CPU 核数
1
内存总量
1.953 GiB
CPU使用率（1m）
41.67%
内存使用率
64%
CPU使用率（%）
100.00%
75.00%
50.00%
25.00%
0%
18:18
18:20
18:22
18:24
18:26
18:28
18:30
18:32
max
avg
current
System 6.87% 5.96% 6.00%
User 73.20% 18.64% 10.27%
Idle 65.80% 55.26% 58.33%
3.0
2.5
2.0
1.5
1.0
0.5
0
1m
5m
15m


Kubernetes Pod监控
Total usage (9 panels)
Pods CPU usage
Pods CPU usage (2m avg)
2019-05-28 18:19:30
web-5447d5d6c6-nsq6m: 0.01
CPU使用率
0.018
0.016
0.014
0.012
0.010
18:18
18:19
18:20
18:21
18:22
18:23
18:24
18:25
18:26
18:27
18:28
18:29
18:30
18:31
18:32


K8S资源对象信息 (Prometheus)
2019-05-28 18:33:21
nginx-test2
1
2019-05-28 18:33:21
nginx-test1
1
Node
Number Of Nodes
6
Nodes Out of Disk
0
Pods
Total
14
Running
16
Failed
0
Pending
0
Unknown
0
Terminated
0
Succeeded
0
Waiting
1

View In AlertManager

[3] Firing

Labels
alertname = InstanceDown
instance = kube-state-metrics
job = kube-state-metrics
k8scluster = test-k8s-cluster
severity = error
Annotations
description = kube-state-metrics of job kube-state-metrics has been down for more than 1 minutes.
summary = Instance kube-state-metrics down
Source

Labels
alertname = InstanceDown
instance = 10.211.18.6:10250
job = kubernetes-nodes-cadvisor
k8scluster = test-k8s-cluster
severity = error
Annotations
description = 10.211.18.6:10250 of job kubernetes-nodes-cadvisor has been down for more than 1 minutes.
summary = Instance 10.211.18.6:10250 down
Source

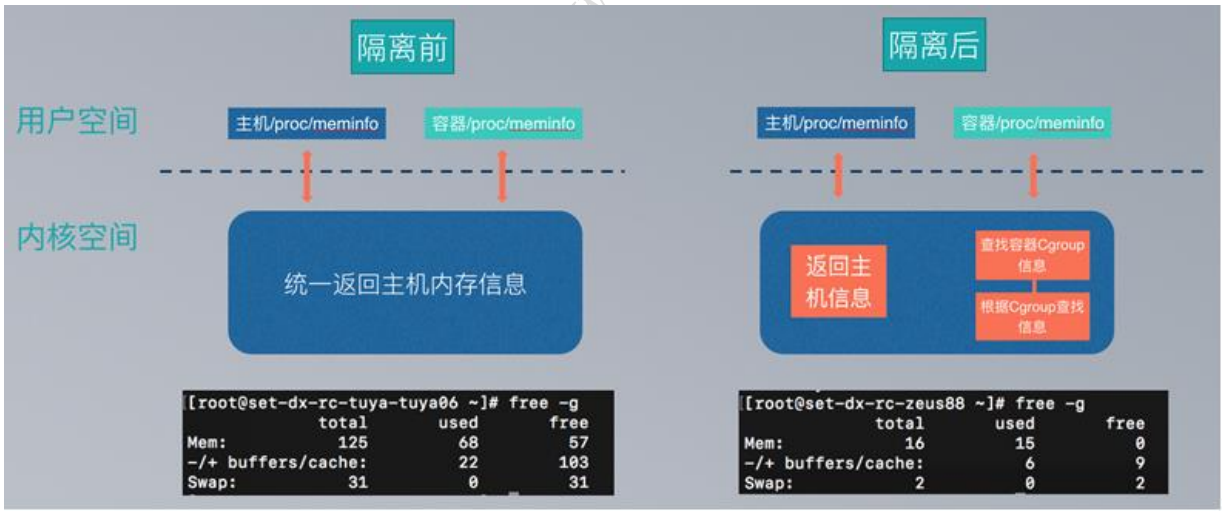
Labels
alertname = InstanceDown
instance = 10.211.18.6:10250
job = kubernetes-nodes-kubelet
k8scluster = test-k8s-cluster
severity = error
Annotations
description = 10.211.18.6:10250 of job kubernetes-nodes-kubelet has been down for more than 1 minutes.
summary = Instance 10.211.18.6:10250 down
Source

7 资源隔离

社区中常见的做法是利用 lxcfs 来提供容器中的资源可见性。lxcfs 是一个开源的 FUSE（用户态文件系统）实现来支持 LXC 容器，它也可以支持 Docker 容器。

LXCFS 通过用户态文件系统，在容器中提供下列 procfs 的文件。

```
/proc/cpuinfo
/proc/diskstats
/proc/meminfo
/proc/stat
/proc/swaps
/proc/uptime
```



所有安装 lxcfs 的 RPM 包

```
[root@K8S-PROD-MASTER-A1 lxcfs]# wget https://copr-be.cloud.fedoraproject.org/results/ganto/lxd/epel-7-x86_64/00486278-lxcfs/lxcfs-2.0.5-3.el7.centos.x86_64.rpm

[root@K8S-PROD-MASTER-A1 lxcfs]#  yum localinstall lxcfs-2.0.5-3.el7.centos.x86_64.rpm
```

启动 lxcfs

```
[root@K8S-PROD-MASTER-A1 lxcfs]# systemctl  enable lxcfs
Created symlink from /etc/systemd/system/multi-user.target.wants/lxcfs.service to /u
sr/lib/systemd/system/lxcfs.service.

[root@K8S-PROD-MASTER-A1 lxcfs]# systemctl  start lxcfs
[root@K8S-PROD-MASTER-A1 lxcfs]# systemctl  status  lxcfs
● lxcfs.service - FUSE filesystem for LXC
   Loaded: loaded (/usr/lib/systemd/system/lxcfs.service; enabled; vendor preset: di
sabled)
   Active: active (running) since Mon 2019-05-20 18:07:27 CST; 6s ago
     Docs: man:lxcfs(1)
 Main PID: 15148 (lxcfs)
    Tasks: 3
   Memory: 312.0K
   CGroup: /system.slice/lxcfs.service
           └─15148 /usr/bin/lxcfs /var/lib/lxcfs/
```

Kubernetes 提供了 Initializer 扩展机制，可以用于对资源创建进行拦截和注入处理，

我们可以借助它优雅地完成对 lxcfs 文件的自动化挂载。

1、 在集群 kube-apiserver 配置文件中添加如下参数， 并重启 kube-apiserver

```
--enable-admission-plugins=Initializers --runtime-
config=admissionregistration.k8s.io/v1alpha1
```

下载 LXCFS

```
[root@K8S-PROD-MASTER-A1 lxcfs]# git clone https://github.com/denverdino/lxcfs-initi
alizer
Cloning into 'lxcfs-initializer'...
remote: Enumerating objects: 1583, done.
remote: Total 1583 (delta 0), reused 0 (delta 0), pack-reused 1583
Receiving objects: 100% (1583/1583), 4.11 MiB | 1.92 MiB/s, done.
Resolving deltas: 100% (520/520), done.
```

通过如下命令在所有集群节点上自动安装、部署完成 lxcfs

```
[root@K8S-PROD-MASTER-A1 lxcfs-initializer]# kubectl  apply -f lxcfs-initializer.yaml



# 查看 pod 运行是否正常
[root@K8S-PROD-MASTER-A1 lxcfs-initializer]# kubectl  get pod
NAME                                READY   STATUS    RESTARTS   AGE
ceph-pod1                           1/1     Running   5          4d1h
curl-66959f6557-8h4ll               1/1     Running   0          6d2h
lxcfs-initializer-769d7fb857-p6p45  1/1     Running   2          15m
nginx-test1-6d7fd56775-m2rzc        1/1     Running   0          111m
nginx-test2-95c548cd4-68gzc         1/1     Running   0          112m
nginx-test2-95c548cd4-ds9hf         1/1     Running   0          112m
```

测试：

kubectl apply -f web.yaml   #其他 pod 如果需要隔离资源， 请确保 lxcfs 值为 true

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    "initializer.kubernetes.io/lxcfs": "true"
  labels:
    app: web
  name: web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: httpd:2.4.32
          imagePullPolicy: Always
          resources:
            requests:
              memory: "256Mi"
              cpu: "500m"
            limits:
              memory: "256Mi"
              cpu: "500m"
```


8 维护命令

8.1 主机维护

步骤 1 当需要对主机进行维护升级时，首先将节点主机设置成不可调度模式

```
kubectl cordon NODE [options]


[root@K8S-PROD-MASTER-A1 ~]# kubectl  cordon  10.211.18.50
node/10.211.18.50 cordoned


[root@K8S-PROD-MASTER-A1 ~]# kubectl  get node
NAME          STATUS  ROLES    AGE   VERSION
10.211.18.11   Ready  node     49d   v1.13.5
10.211.18.4    Ready, master   49d   v1.13.5
10.211.18.5    Ready, master   49d   v1.13.5
10.211.18.50   Ready,SchedulingDisabled   LB      49d
v1.13.5
10.211.18.6    Ready   master   49d   v1.13.5
```

步骤 2 然后需要将主机上正在运行的容器驱赶到其它可用节点

```
kubectl drain [nodeid]

[root@K8S-PROD-MASTER-A1 ~]# kubectl drain 10.211.18.50
node/10.211.18.50 already cordoned
```

给予 900 秒宽限期优雅的调度

```
kubectl drain 10.211.18.50 --grace-period=120
```

当容器迁移完毕后，运维人员可以对该主机进行操作，配置升级性能参数调优等等。当对主机的维护操作完毕后， 再将主机设置成可调度模式：

```
kubectl uncordon [nodeid]
```


8.2 滚动更新

8.3 蓝绿发布

9 集群故障

必备工具

- kubectl：用于查看 Kubernetes 集群以及容器的状态，如 kubectl describe pod <pod-name>
- journalctl：用于查看 Kubernetes 组件日志，如 journalctl -u kubelet -l
- iptables 和 ebtables：用于排查 Service 是否工作，如 iptables -t nat -nL 查看 kube-proxy 配置的 iptables 规则是否正常
- tcpdump：用于排查容器网络问题，如 tcpdump -nn host 10.240.0.8
- perf：Linux 内核自带的性能分析工具，常用来排查性能问题，如 Container Isolation Gone Wrong 问题的排查

9.1 集群异常处理

按照不同的组件来说，具体的原因可能包括

- kube-apiserver 无法启动会导致
  - 集群不可访问
  - 已有的 Pod 和服务正常运行（依赖于 Kubernetes API 的除外）
- etcd 集群异常会导致
  - kube-apiserver 无法正常读写集群状态，进而导致 Kubernetes API 访问出错
  - kubelet 无法周期性更新状态
- kube-controller-manager/kube-scheduler 异常会导致
  - 复制控制器、节点控制器、云服务控制器等无法工作，从而导致 Deployment、Service 等无法工作，也无法注册新的 Node 到集群中来
  - 新创建的 Pod 无法调度（总是 Pending 状态）
- Node 本身宕机或者 Kubelet 无法启动会导致
  - Node 上面的 Pod 无法正常运行
  - 已在运行的 Pod 无法正常终止

- 网络分区会导致 Kubelet 等与控制平面通信异常以及 Pod 之间通信异常

9.1.1 查看 Node 状态

一般来说，可以首先查看 Node 的状态，确认 Node 本身是不是 Ready 状态

```
kubectl get nodes
kubectl describe node <node-name>
```

如果是 NotReady 状态，则可以执行 kubectl describe node <node-name> 命令来查看当前 Node 的事件。这些事件通常都会有助于排查 Node 发生的问题。

9.1.2 查看日志

一般来说，Kubernetes 的主要组件有两种部署方法

- 直接使用 systemd 等启动控制节点的各个服务
- 使用 Static Pod 来管理和启动控制节点的各个服务

使用 systemd 等管理控制节点服务时，查看日志必须要首先 SSH 登录到机器上，然后查看具体的日志文件。如

```
journalctl -l -u kube-apiserver
journalctl -l -u kube-controller-manager
journalctl -l -u kube-scheduler
journalctl -l -u kubelet
journalctl -l -u kube-proxy
journalctl -l -u etcd
journalctl -l -u flanneld
```

或者直接查看日志文件

9.2 证书过期

将集群设置为维护状态， 替换证书文件。 重启业务进程。

9.3 Pods 异常处理

一般来说，无论 Pod 处于什么异常状态，都可以执行以下命令来查看 Pod 的状态

- kubectl get pod <pod-name> -o yaml 查看 Pod 的配置是否正确
- kubectl describe pod <pod-name> 查看 Pod 的事件
- kubectl logs <pod-name> [-c <container-name>] 查看容器日志

这些事件和日志通常都会有助于排查 Pod 发生的问题。

Pod 一直处于 Pending 状态

Pending 说明 Pod 还没有调度到某个 Node 上面。可以通过 kubectl describe pod <pod-name> 命令查看到当前 Pod 的事件，进而判断为什么没有调度。

```
$ kubectl describe pod mypod
...
Events:
  Type     Reason            Age                From               Message
  ----     ------            ----               ----               -------
  Warning  FailedScheduling  12s (x6 over 27s)  default-scheduler  0/4
nodes are available: 2 Insufficient cpu
```

可能的原因包括

- 资源不足，集群内所有的 Node 都不满足该 Pod 请求的 CPU、内存、GPU 或者临时存储空间等资源。解决方法是删除集群内不用的 Pod 或者增加新的 Node。
- HostPort 端口已被占用，通常推荐使用 Service 对外开放服务端口

Pod 一直处于 Waiting 或 ContainerCreating 状态

通过 kubectl describe pod <pod-name> 命令查看到当前 Pod 的事件

可能的原因有以下几种

- 镜像拉取失败，比如
  - 配置了错误的镜像
  - Kubelet 无法访问镜像（国内环境访问 gcr.io 需要特殊处理）
  - 私有镜像的密钥配置错误
  - 镜像太大，拉取超时（可以适当调整 kubelet 的 --image-pull-progress-deadline 和 --runtime-request-timeout 选项）
- CNI 网络错误，一般需要检查 CNI 网络插件的配置，比如
  - 无法配置 Pod 网络
  - 无法分配 IP 地址
- 容器无法启动，需要检查是否打包了正确的镜像或者是否配置了正确的容器参数

Pod 处于 ImagePullBackOff 状态

这通常是镜像名称配置错误或者私有镜像的密钥配置错误导致。这种情况可以使用 docker pull <image> 来验证镜像是否可以正常拉取。

如果是私有镜像，需要首先创建一个 docker-registry 类型的 Secret

```
kubectl create secret docker-registry my-secret --docker-
server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-
password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
```

然后在容器中引用这个 Secret

```
spec:
  containers:

  - name: private-reg-container

    image: <your-private-image>
  imagePullSecrets:
  - name: my-secret
```


Pod 一直处于 CrashLoopBackOff 状态

CrashLoopBackOff 状态说明容器曾经启动了，但又异常退出了。此时 Pod 的 RestartCounts 通常是大于 0 的，可以先查看一下容器的日志

```
kubectl describe pod <pod-name>
kubectl logs <pod-name>
kubectl logs --previous <pod-name>
```

这里可以发现一些容器退出的原因，比如

- 容器进程退出
- 健康检查失败退出
- OOMKilled

```
$ kubectl describe pod mypod
...
Containers:
  sh:
    Container ID:
docker://3f7a2ee0e7e0e16c22090a25f9b6e42b5c06ec049405bc34d3aa183060eb4906
    Image:         alpine
    Image ID:      docker-
pullable://alpine@sha256:7b848083f93822dd21b0a2f14a110bd99f6efb4b838d499d
f6d04a49d0debf8b
    Port:          <none>
    Host Port:     <none>
    State:          Terminated
      Reason:       OOMKilled
      Exit Code:    2

    Last State:     Terminated

      Reason:       OOMKilled
      Exit Code:    2
    Ready:          False
    Restart Count:  3
    Limits:
      cpu:     1
      memory:  1G
    Requests:
```

```
      cpu:        100m
      memory:     500M
...
```

如果此时如果还未发现线索，还可以到容器内执行命令来进一步查看退出原因

```
kubectl exec cassandra -- cat /var/log/cassandra/system.log
```

如果还是没有线索，那就需要 SSH 登录该 Pod 所在的 Node 上，查看 Kubelet 或者 Docker 的日志进一步排查了

```
# Query Node
kubectl get pod <pod-name> -o wide

# SSH to Node
ssh <username>@<node-name>
```


Pod 处于 Error 状态

通常处于 Error 状态说明 Pod 启动过程中发生了错误。常见的原因包括

- 依赖的 ConfigMap、Secret 或者 PV 等不存在
- 请求的资源超过了管理员设置的限制，比如超过了 LimitRange 等
- 违反集群的安全策略，比如违反了 PodSecurityPolicy 等
- 容器无权操作集群内的资源，比如开启 RBAC 后，需要为 ServiceAccount 配置角色绑定

Pod 处于 Terminating 或 Unknown 状态

- 从集群中删除该 Node。使用公有云时，kube-controller-manager 会在 VM 删除后自动删除对应的 Node。而在物理机部署的集群中，需要管理员手动删除 Node（如 kubectl delete node <node-name>。

- Node 恢复正常。Kubelet 会重新跟 kube-apiserver 通信确认这些 Pod 的期待状态，进而再决定删除或者继续运行这些 Pod。

- 用户强制删除。用户可以执行 kubectl delete pods <pod> --grace-period=0 --force 强制删除 Pod。除非明确知道 Pod 的确处于停止状态（比如 Node 所在 VM 或物理机已经关机），否则不建议使用该方法。特别是 StatefulSet 管理的 Pod，强制删除容易导致脑裂或者数据丢失等问题。

处于 Terminating 状态的 Pod 在 Kubelet 恢复正常运行后一般会自动删除。但有时也会出现无法删除的情况，并且通过 kubectl delete pods <pod> --grace-period=0 --force 也无法强制删除。此时一般是由于 finalizers 导致的，通过 kubectl edit 将 finalizers 删除即可解决。

```
"finalizers": [
  "foregroundDeletion"
]
```

Pod 行为异常

这里所说的行为异常是指 Pod 没有按预期的行为执行，比如没有运行 podSpec 里面设置的命令行参数。这一般是 podSpec yaml 文件内容有误，可以尝试使用 --validate 参数重建容器，比如

```
kubectl delete pod mypod
kubectl create --validate -f mypod.yaml
```

也可以查看创建后的 podSpec 是否是对的，比如

```
kubectl get pod mypod -o yaml
```


9.4 网络排错

介绍各种常见的网络问题以及排错方法，包括 Pod 访问异常、Service 访问异常以及网络安全策略异常等。

- Pod 访问容器外部网络
- 从容器外部访问 Pod 网络
- Pod 之间相互访问

当然，以上每种情况还都分别包括本地访问和跨主机访问两种场景，并且一般情况下都是通过 Service 间接访问 Pod。

网络异常可能的原因比较多，常见的有

- CNI 网络插件配置错误，导致多主机网络不通，比如
  - IP 网段与现有网络冲突
  - 插件使用了底层网络不支持的协议
  - 忘记开启 IP 转发等
    - sysctl net.ipv4.ip_forward
    - sysctl net.bridge.bridge-nf-call-iptables
- Pod 网络路由丢失，比如
  - kubenet 要求网络中有 podCIDR 到主机 IP 地址的路由，这些路由如果没有正确配置会导致 Pod 网络通信等问题
  - 在公有云平台上，kube-controller-manager 会自动为所有 Node 配置路由，但如果配置不当（如认证授权失败、超出配额等），也有可能导致无法配置路由
- 主机内或者云平台的安全组、防火墙或者安全策略等阻止了 Pod 网络，比如
  - 非 Kubernetes 管理的 iptables 规则禁止了 Pod 网络
  - 公有云平台的安全组禁止了 Pod 网络（注意 Pod 网络有可能与 Node 网络不在同一个网段）
  - 交换机或者路由器的 ACL 禁止了 Pod 网络

Service 无法访问

访问 Service ClusterIP 失败时，可以首先确认是否有对应的 Endpoints

```
kubectl get endpoints <service-name>
```

如果该列表为空，则有可能是该 Service 的 LabelSelector 配置错误，可以用下面的方法确认一下

查询 Service 的 LabelSelector

```
kubectl get svc <service-name> -o jsonpath='{.spec.selector}'
```

查询匹配 LabelSelector 的 Pod

```
kubectl get pods -l key1=value1,key2=value2
```

如果 Endpoints 正常，可以进一步检查

- Pod 的 containerPort 与 Service 的 containerPort 是否对应
- 直接访问 podIP:containerPort 是否正常

再进一步，即使上述配置都正确无误，还有其他的原因会导致 Service 无法访问，比如

- Pod 内的容器有可能未正常运行或者没有监听在指定的 containerPort 上
- CNI 网络或主机路由异常也会导致类似的问题
- kube-proxy 服务有可能未启动或者未正确配置相应的 iptables 规则，比如正常情况下名为 hostnames 的服务会配置以下 iptables 规则

9.5 Volume 异常处理

持久化存储异常（PV、PVC、StorageClass 等）的排错方法。

> 一般来说，无论 PV 处于什么异常状态，都可以执行 kubectl describe pv/pvc <pod-name> 命令来查看当前 PV 的事件。这些事件通常都会有助于排查 PV 或 PVC 发生的问题。

```
kubectl get pv
kubectl get pvc
kubectl get sc

kubectl describe pv <pv-name>
kubectl describe pvc <pvc-name>
kubectl describe sc <storage-class-name>
```